IBM

IBM Power Systems Performance Guide

Implementing and Optimizing

- Leverages IBM Power virtualization
- Helps maximize system resources
- Provides sample scenarios

Dino Quintero
Sebastien Chabrolles
Chi Hui Chen
Murali Dhandapani
Talor Holloway
Chandrakant Jadhav
Sae Kee Kim
Sijo Kurian
Bharath Raj
Ronan Resende
Bjorn Roden
Niranjan Srinivasan
Richard Wale
William Zanatta
Zhi Zhang

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Power Systems Performance Guide: Implementing and Optimizing

February 2013

SG24-8080-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (February 2013)

This edition applies to IBM POWER 750 FW AL730-095, VIO 2.2.2.0 & 2.2.1.4, SDDPCM 2.6.3.2 HMC v7.6.0.0, nmem version 2.0, netperf 1.0.0.0, AIX 7.1 TL2, SDDPCM 2.6.3.2, ndisk version 5.9, IBM SAN24B-4 (v6.4.2b), IBM Storwize V7000 2076-124 (6.3.0.1)

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Active Memory™
AIX®
alphaWorks®
DB2®
developerWorks®
DS6000™
DS8000®
Easy Tier®
EnergyScale™
eServer™
FDPR®
HACMP™
IBM Systems Director Active Energy Manager™
IBM®
Informix®
Jazz™
Micro-Partitioning®
Power Systems™
POWER6+™
POWER6®
POWER7+™
POWER7®
PowerHA®
PowerPC®
PowerVM®
POWER®
pSeries®
PureFlex™
PureSystems™
Rational®
Redbooks®
Redbooks (logo) ®
RS/6000®
Storwize®
System p®
System Storage®
SystemMirror®
Tivoli®
WebSphere®
XIV®
z/VM®
zSeries®

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication addresses performance tuning topics to help leverage the virtualization strengths of the POWER® platform to solve clients' system resource utilization challenges, and maximize system throughput and capacity. We examine the performance monitoring tools, utilities, documentation, and other resources available to help technical teams provide optimized business solutions and support for applications running on IBM POWER systems' virtualized environments.

The book offers application performance examples deployed on IBM Power Systems™ utilizing performance monitoring tools to leverage the comprehensive set of POWER virtualization features: Logical Partitions (LPARs), micro-partitioning, active memory sharing, workload partitions, and more. We provide a well-defined and documented performance tuning model in a POWER system virtualized environment to help you plan a foundation for scaling, capacity, and optimization.

This book targets technical professionals (technical consultants, technical support staff, IT Architects, and IT Specialists) responsible for providing solutions and support on IBM POWER systems, including performance tuning.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is an IBM Senior Certified IT Specialist with the ITSO in Poughkeepsie, NY. His areas of knowledge include enterprise continuous availability, enterprise systems management, system virtualization, and technical computing and clustering solutions. He is currently an Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

Sebastien Chabrolles is an IT Specialist at the Product and Solution Support Center in Montpellier, France. His main activity is to perform pre-sales customer benchmarks on Power Systems in the European Benchmark Center. He graduated from a Computer Engineering school (ESIEA) and has 10 years of experience in AIX® and Power Systems. His areas of expertise include IBM Power Systems, PowerVM®, AIX, and Linux.

Chi Hui Chen is a Senior IT Specialist at the IBM Advanced Technical Skills (ATS) team in China. He has more than eight years of experience in IBM Power Systems. He provides AIX support to GCG ISVs in the areas of application design, system performance tuning, problem determination, and application benchmarks. He holds a degree in Computer Science from University of Science and Technology of China.

Murali Dhandapani is a Certified IT Specialist in Systems Management in IBM India. He is working for the IBM India Software Lab Operations team, where he is a technical lead for IBM Rational® Jazz™ products infrastructure, high availability, and disaster recovery deployment. His areas of expertise include Linux, AIX, IBM POWER virtualization, PowerHA® SystemMirror®, System Management, and Rational tools. Murali has a Master of Computer Science degree. He is an IBM developerWorks® Contributing Author, IBM Certified Specialist

in System p® administration and an IBM eServer™ Certified Systems Expert - pSeries® High Availability Cluster Multi-Processing (IBM HACMP™).

Talor Holloway is a senior technical consultant working for Advent One, an IBM business partner in Melbourne, Australia. He has worked extensively with AIX and Power Systems and System p for over seven years. His areas of expertise include AIX, NIM, PowerHA, PowerVM, IBM Storage, and IBM Tivoli® Storage Manager.

Chandrakant Jadhav is an IT Specialist working at IBM India. He is working for the IBM India Software Lab Operations team. He has over five years of experience in System P, Power Virtualization. His areas of expertise include AIX, Linux, NIM, PowerVM, IBM Storage, and IBM Tivoli Storage Manager.

Sae Kee Kim is a Senior Engineer at Samsung SDS in Korea. He has 13 years of experience in AIX Administration and five years of Quality Control in the ISO20000 field. He holds a Bachelor's degree in Electronic Engineering from Dankook University in Korea. His areas of expertise include IBM Power Systems and IBM AIX administration.

Sijo Kurian is a Project Manager in IBM Software Labs in India. He has seven years of experience in AIX and Power Systems. He holds a Masters degree in Computer Science. He is an IBM Certified Expert in AIX, HACMP and Virtualization technologies.His areas of expertise include IBM Power Systems, AIX, PowerVM, and PowerHA.

Bharath Raj is a Performance Architect for Enterprise Solutions from Bangalore, India. He works with the software group and has over five years of experience in the performance engineering of IBM cross-brand products, mainly in WebSphere® Application Server integration areas. He holds a Bachelor of Engineering degree from the University of RVCE, Bangalore, India. His areas of expertise include performance benchmarking IBM products, end-to-end performance engineering of enterprise solutions, performance architecting, designing solutions, and sizing capacity for solutions with IBM product components. He wrote many articles that pertain to performance engineering in developerWorks and in international science journals.

Ronan Resende is a System Analyst at Banco do Brasil in Brazil. He has 10 years of experience with Linux and three years of experience in IBM Power Systems. His areas of expertise include IBM AIX, Linux in pSeries, and zSeries® (z/VM®).

Bjorn Roden is a Systems Architect for IBM STG Lab Services and is part of the IBM PowerCare Teams working with High End Enterprise IBM Power Systems for clients. He has co-authored seven other IBM Redbooks publications, been speaker at IBM Technical events. Bjorn holds MSc, BSc and DiplSSc in Informatics from Lund University in Sweden, and BCSc and DiplCSc in Computer Science from Malmo University in Sweden. He also has certifications as IBM Certified Infrastructure Systems Architect (ISA), Certified TOGAF Architect, Certified PRINCE2 Project Manager, and Certified IBM Advanced Technical Expert, IBM Specialist and IBM Technical Leader since 1994. He has worked with designing, planning, implementing, programming, and assessing high availability, resiliency, security, and high performance systems and solutions for Power/AIX since AIX v3.1 1990.

Niranjan Srinivasan is a software engineer with the client enablement and systems assurance team.

Richard Wale is a Senior IT Specialist working at the IBM Hursley Lab, UK. He holds a B.Sc. (Hons) degree in Computer Science from Portsmouth University, England. He has over 12 years of experience supporting AIX. His areas of expertise include IBM Power Systems, PowerVM, AIX, and IBM i.

William Zanatta is an IT Specialist working in the Strategic Outsourcing Delivery at IBM Brazil. He holds a B.S. degree in Computer Engineering from Universidade Metodista de Sao Paulo, Brazil. He has over 10 years of experience in supporting different UNIX platforms, and his areas of expertise include IBM Power Systems, PowerVM, PowerHA, AIX and Linux.

Zhi Zhang is an Advisory Software Engineer in IBM China. He has more than 10 years of experience in the IT field. He is a certified DB2® DBA. His areas of expertise include IBM AIX, DB2 and WebSphere Application Performance Tuning. He is currently working in the IBM software group as performance QA.

Thanks to the following people for their contributions to this project:

Ella Buslovich, Richard Conway, Octavian Lascu, Ann Lund, Alfred Schwab, and Scott Vetter
International Technical Support Organization, Poughkeepsie Center

Gordon McPheeters, Barry Knapp, Bob Maher and Barry Spielberg
IBM Poughkeepsie

Mark McConaughy, David Sheffield, Khalid Filali-Adib, Rene R Martinez, Sungjin Yook, Vishal C Aslot, Bruce Mealey, Jay Kruemcke, Nikhil Hedge, Camilla McWilliams, Calvin Sze, and Jim Czenkusch
IBM Austin

Stuart Z Jacobs, Karl Huppler, Pete Heyrman, Ed Prosser
IBM Rochester

Linda Flanders
IBM Beaverton

Rob Convery, Tim Dunn and David Gorman
IBM Hursley

Nigel Griffiths and Gareth Coates
IBM UK

Yaoqing Gao
IBM Canada

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

IBM Power Systems and performance tuning

The following topics are discussed in this chapter:

- Introduction
- IBM Power Systems
- Overview of this publication
- Regarding performance

1.1 Introduction

To plan the journey ahead, you must understand the available options and where you stand today. It also helps to know some of the history.

Power is performance redefined. Everyone knows what performance meant for IT in the past: processing power and benchmarks. Enterprise Systems, entry systems and Expert Integrated Systems built on the foundation of a POWER processor continue to excel and extend industry leadership in these traditional benchmarks of performance.

Let us briefly reflect on where we are today and how we arrived here.

1.2 IBM Power Systems

Over the years, the IBM Power Systems family has grown, matured, been innovated and pushed the boundaries of what clients expect and demand from the harmony of hardware and software.

With the advent of the POWER4 processor in 2001, IBM introduced logical partitions (LPARs) outside of their mainframe family to another audience. What was seen as radical then, has grown into the expected today. The term *virtualization* is now common-place across most platforms and operating systems. However, what options a given platform or hypervisor provides greatly varies. Many hypervisors provide a number of options to achieve the same end result. The availability of such options provides choices to fulfill the majority of client requirements. For general workloads the difference between the various implementations may not be obvious or apparent. However, for the more demanding workloads or when clients are looking to achieve virtualization or utilization goals, the different approaches need to be understood.

As an example, PowerVM can virtualize storage to an LPAR through a number of routes. Each option delivers the required storage, but the choice is dictated by the expectations for that storage. Previously the requirement was simply for storage, but today the requirement could also include management, functionality, resilience, or quality of service.

We cannot stress enough the importance of understanding your requirements and your workload requirements. These complimentary factors provide you, the consumer, with enough knowledge to qualify what you require and expect from your environment. If you are not familiar with the range of options and technologies, then that is where your IBM sales advisor can help.

POWER processor-based servers can be found in three product families: IBM Power Systems servers, IBM Blade servers and IBM PureSystems™. Each of these three families is positioned for different types of client requirements and expectations.

In this book we concentrate on the Power Systems family. This is the current incarnation of the previous System p, pSeries and RS/6000® families. It is the traditional Power platform for which clients demand performance, availability, resilience, and security, combined with a broad, differentiated catalogue of capabilities to suit requirements from the entry level to the enterprise. As an example, Table 1-1 on page 3 summarizes the processor sizings available across the range.

Table 1-1 Power Systems servers processor configurations

Power Systems	Max socket per CEC	Max core per socket	Max CEC per system	Max core per system
Power 710	1	8	1	8
Power 720	1	8	1	8
Power 730	2	8	1	16
Power 740	2	8	1	16
Power 750	4	8	1	32
Power 755	4	8	1	32
Power 770	4	4	4	64
Power 780	4	8	4	128
Power 795	4	8	8	256

Note: The enterprise-class models have a modular approach: allowing a single system to be constructed from one or more enclosures or Central Electronic Complexes (CECs). This building-block approach provides an upgrade path to increase capacity without replacing the entire system.

The smallest configuration for a Power 710 is currently a single 4-core processor with 4 GB of RAM. There are configuration options and combinations from this model up to a Power 795 with 256 cores with 16 TB of RAM. While Table 1-1 may suggest similarities between certain models, we illustrate later in 2.3, “Performance consequences for I/O mapping and adapter placement” on page 26 some of the differences between models.

IBM Power Systems servers are not just processors and memory. The vitality of the platform comes from its virtualization component, that is, PowerVM, which provides a secure, scalable virtualization environment for AIX, IBM i and Linux applications. In addition to hardware virtualization for processor, RAM, network, and storage, PowerVM also delivers a broad range of features for availability, management, and administration.

For a complete overview of the PowerVM component, refer to *IBM PowerVM Getting Started Guide*, REDP-4815.

1.3 Overview of this publication

The chapters in our book are purposely ordered. Chapters 2, 3 and 4 discuss the three foundational layers on which every Power Systems server is implemented:

- Hardware
- Hypervisor
- Operating system

Configuration and implementation in one layer impacts and influences the subsequent layers. It is important to understand the dependencies and relationships between layers to appreciate the implications of decisions.

Note: The focus of this book is on topics concerning PowerVM and AIX. Some of the hardware and hypervisor topics are equally applicable when hosting IBM i or Linux LPARs. There are, however, specific implications and considerations relative to IBM i and Linux LPARs. Unfortunately, doing suitable justice to these in addition to AIX is beyond the scope of this book.

In these four initial chapters, the subtopics are grouped and ordered for consistency in the following sequence:

1. Processor
2. Memory
3. Storage
4. Network

The first four chapters are followed by a fifth that describes how to investigate and analyze given components when you think you may have a problem, or just want to verify that everything is normal. Databases grow, quantities of users increase, networks become saturated. Like cars, systems need regular checkups to ensure everything is running as expected. So where applicable we highlight cases where it is good practice to regularly check a given component.

1.4 Regarding performance

The word performance was previously used to simply describe and quantify. It is the fastest or the best; the most advanced; in some cases the biggest and typically most expensive.

However, today's IT landscape brings new viewpoints and perspectives to familiar concepts. Over the years performance has acquired additional and in some cases opposing attributes.

Today quantifying performance relates to more than just throughput. To illustrate the point, consider the decision-making process when buying a motor vehicle. Depending on your requirements, one or more of the following may be important to you:

- Maximum speed
- Speed of acceleration
- Horsepower

These three fall into the traditional ideals of what performance is. Now consider the following additional attributes:

- Fuel economy
- Number of seats

- Wheel clearance
- Storage space
- Safety features

All are elements that would help qualify how a given vehicle would perform, for a given requirement.

For example, race car drivers would absolutely be interested in the first three attributes. However, safety features would also be high on their requirements. Even then, depending on the type of race, the wheel clearance could also be of key interest.

Whereas a family with two children is more likely to be more interested in safety, storage, seats and fuel economy, whereas speed of acceleration would be less of a concern.

Turning the focus back to performance in the IT context and drawing a parallel to the car analogy, traditionally one or more of the following may have been considered important:

- Processor speed
- Number of processors
- Size of memory

Whereas today's perspective could include these additional considerations:

- Utilization
- Virtualization
- Total cost of ownership
- Efficiency
- Size

Do you need performance to be fastest or just fast enough? Consider, for example, any health, military or industry-related applications. Planes need to land safety, heartbeats need to be accurately monitored, and everyone needs electricity. In those cases, applications cannot underperform.

If leveraging virtualization to achieve server consolidation is your goal, are you wanting performance in efficiency? Perhaps you need your server to perform with regard to its power and physical footprint? For some clients, resilience and availability may be more of a performance metric than traditional data rates.

Throughout this book we stress the importance of understanding your requirements and your workload.

2

Hardware implementation and LPAR planning

To get all the benefits from your POWER7® System, it is really important to know the hardware architecture of your system and to understand how the POWER hypervisor assigns hardware to the partitions.

In this chapter we present the following topics:

- Hardware migration considerations
- Performance consequences for processor and memory placement
- Performance consequences for I/O mapping and adapter placement
- Continuous availability with CHARM
- Power management

2.1 Hardware migration considerations

In section 2.2.2 of *Virtualization and Clustering Best Practices Using IBM System p Servers*, SG24-7349, we discussed a range of points to consider when migrating workloads from POWER4 to POWER5 hardware. While much has changed in the six years since that publication was written, many of the themes remain relevant today.

In the interim years, the IBM POWER Systems product family has evolved through POWER6® and onto POWER7 generations. The range of models has changed based on innovation and client demands to equally cater from entry-level deployment to the large-scale enterprise. PowerVM has continued to mature by adding new virtualization features and refining the abilities of some of the familiar components.

So with the advent of POWER7+™, these key areas should be evaluated when considering or planning an upgrade:

- Understanding your workload and its requirements and dependencies. This is crucial to the decision-making process. Without significant understanding of these areas, an informed decision is not possible. Assumptions based on knowledge of previous hardware generations may not lead to the best decision.
- One size does not fit all. This is why IBM offers more than one model. Consider what you need today, and compare that to what you might need tomorrow. Some of the models have expansion paths, both with and without replacing the entire system. Are any of your requirements dependant on POWER7+ or is POWER7 equally an option? If you are looking to upgrade or replace servers from both POWER and x86 platforms, would an IBM PureFlex™ System deployment be an option? Comparison of all the models with the IBM POWER Systems catalogue is outside the scope of this publication. However, the various sections in this chapter should provide you with a range of areas that need to be considered in the decision-making process.
- Impact on your existing infrastructure. If you already have a Hardware Management Console (HMC), is it suitable for managing POWER7 or POWER7+? Would you need to upgrade or replace storage, network or POWER components to take full advantage of the new hardware? Which upgrades would be required from day one and which could be planned and staggered?
- Impact on your existing deployments. Are the operating systems running on your existing servers supported on the new POWER7/POWER7+ hardware? Do you need to accommodate and schedule upgrades? If upgrades are required, do you also need new software licenses for newer versions of middleware?
- Optional PowerVM features. There are a small number of POWER7 features that are not included as part of the standard PowerVM tiers. If you are moving up to POWER7 for the first time, you may not appreciate that some features are enabled by separate feature codes. For example, you might be interested in leveraging Versioned WPARs or Active Memory™ Expansion (AME); both of these are provided by separate codes.
- If you are replacing existing hardware, are there connectivity options or adapters that you need to preserve from your legacy hardware? For example, do you require adapters for tape support? Not all options that were available on previous System p or POWER Systems generations are available on the current POWER7 and POWER7+ family. Some have been depreciated, replaced or superseded. For example, it is not possible to connect an IBM Serial Storage Architecture (SSA) disk array to POWER7; however, new storage options have been introduced since SSA such as SAN and SSD. If you are unsure whether a given option is available for or supported on the current generation, contact your IBM representative.

Aside from technological advancements, external factors have added pressure to the decision-making process:

- Greener data centers. Increased electricity prices, combined with external expectations result in companies proactively retiring older hardware in favour of newer, more efficient, models.
- Higher utilization and virtualization. The challenging economic climate means that companies have fewer funds to spend on IT resources. There is a trend for increased efficiency, utilization and virtualization of physical assets. This adds significant pressure to make sure assets procured meet expectations and are suitably utilized. Industry average is approximately 40% virtualization and there are ongoing industry trends to push this higher.

Taking these points into consideration, it is possible that for given configurations, while the initial cost might be greater, the total cost of ownership (TCO) would actually be significantly less over time.

For example, a POWER7 720 (8205-E4C) provides up to eight processor cores and has a quoted maximum power consumption of 840 watts. While a POWER7 740 (8205-E6C) provides up to 16 cores with a quoted maximum power consumption of 1400 watts; which is fractionally less than the 1680 watts required for two POWER7 720 servers to provide the same core quantity.

Looking higher up the range, a POWER7+ 780 (9117-MHD) can provide up to 32 cores per enclosure. An enclosure has a quoted maximum power consumption of 1900 watts. Four POWER 720 machines would require 3360 watts to provide 32 cores.

A POWER 780 can also be upgraded with up to three additional enclosures. So if your requirements could quickly outgrow the available capacity of a given model, then considering the next largest model might be beneficial and cheaper in the longer term.

Note: In the simple comparison above we are just comparing core quantity with power rating. The obvious benefit of the 740 over the 720 (and the 780 over the 740) is maximum size of LPAR. We also are not considering the difference in processor clock frequency between the models or the benefits of POWER7+ over POWER7.

In 2.1.12 of *Virtualization and Clustering Best Practices Using IBM System p Servers*, SG24-7349, we summarized that the decision-making process was far more complex than just a single metric. And that while the final decision might be heavily influenced by the most prevalent factor, other viewpoints and considerations must be equally evaluated. While much has changed in the interim, ironically the statement still stands true.

2.2 Performance consequences for processor and memory placement

As described in Table 1-1 on page 3, the IBM Power Systems family are all multiprocessor systems. Scalability in a multiprocessor system has always been a challenge and become even more challenging in this *multicore era*. Add on top of that the hardware virtualization, and you will have an *amazing puzzle game* to solve when you are facing performance issues.

In this section, we give the key to better understanding your Power Systems hardware. We do not go into detail to all the available features, but we try to give you the main concepts and best practices to take the best decision to size and create your Logical Partitions (LPARs).

Note: More detail about a specific IBM Power System can be found here:

http://pic.dhe.ibm.com/infocenter/powersys/v3r1m5/index.jsp

2.2.1 Power Systems and NUMA effect

Symmetric multiprocessing (SMP) architecture allows a system to scale beyond one processor. Each processor is connected to the same bus (also known as crossbar switch) to access the main memory. But this computation scaling is not infinite due to the fact that each processor needs to share the same memory bus, so access to the main memory is serialized. With this limitation, this kind of architecture can scale up to four to eight processors only (depending on the hardware).

Note: A multicore processor chip can be seen as an *SMP system in a chip.* All the cores in the same chip share the same memory controller (Figure 2-1).

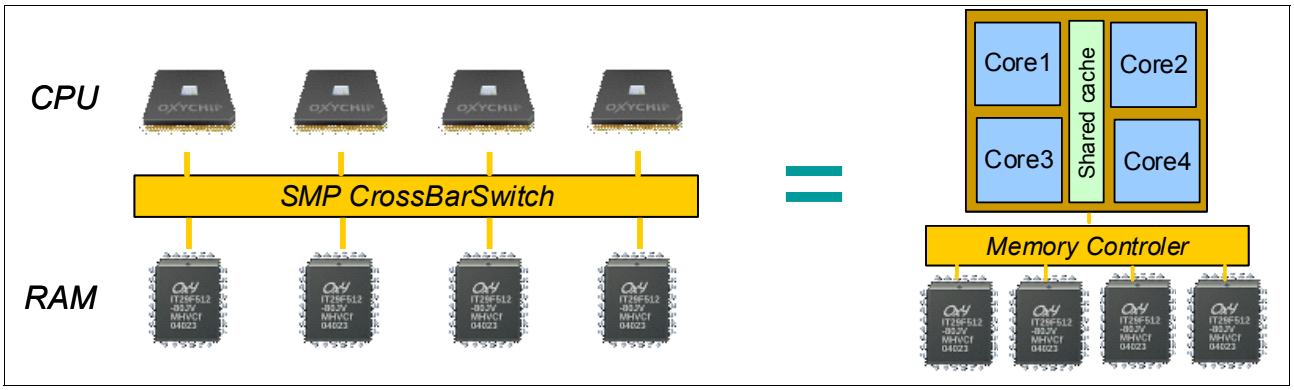


Figure 2-1 SMP architecture and multicore

The Non-Uniform Memory Access (NUMA) architecture is a way to partially solve the SMP scalability issue by reducing pressure on the memory bus.

As opposed to the SMP system, NUMA adds the notion of a multiple memory subsystem called *NUMA node*:

- Each node is composed of processors sharing the same bus to access memory (a node can be seen as an SMP system).
- NUMA nodes are connected using a special “interlink bus” to provide processor data coherency across the entire system.

Each processor can have access to the entire memory of a system; but access to this memory is not uniform (Figure 2-2 on page 11):

- Access to memory located in the same node (local memory) is direct with a very low latency.
- Access to memory located in another node is achieved through the interlink bus with a higher latency.

By limiting the number of processors that directly access the entire memory, performance is improved compared to an SMP because of the much shorter queue of requests on each memory domain.

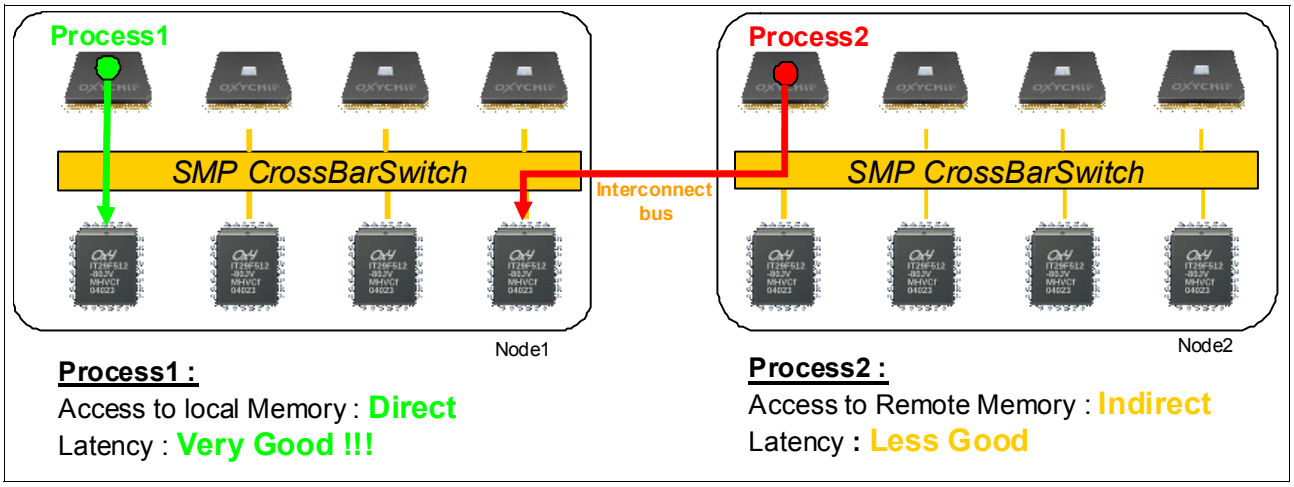


Figure 2-2 NUMA architecture concept

The architecture design of the Power platform is mostly NUMA with three levels:

- Each POWER7 chip has its own memory dimms. Access to these dimms has a very low latency and is named *local*.
- Up to four POWER7 chips can be connected to each other in the same CEC (or node) by using X, Y, Z buses from POWER7. Access to memory owned by another POWER7 chip in the same CEC is called *near* or *remote*. Near or remote memory access has a higher latency compared than local memory access.
- Up to eight CECs can be connected through A, B buses from a POWER7 chip (only on high-end systems). Access to memory owned by another POWER7 in another CEC (or node) is called *far* or *distant*. Far or distant memory access has a higher latency than remote memory access.

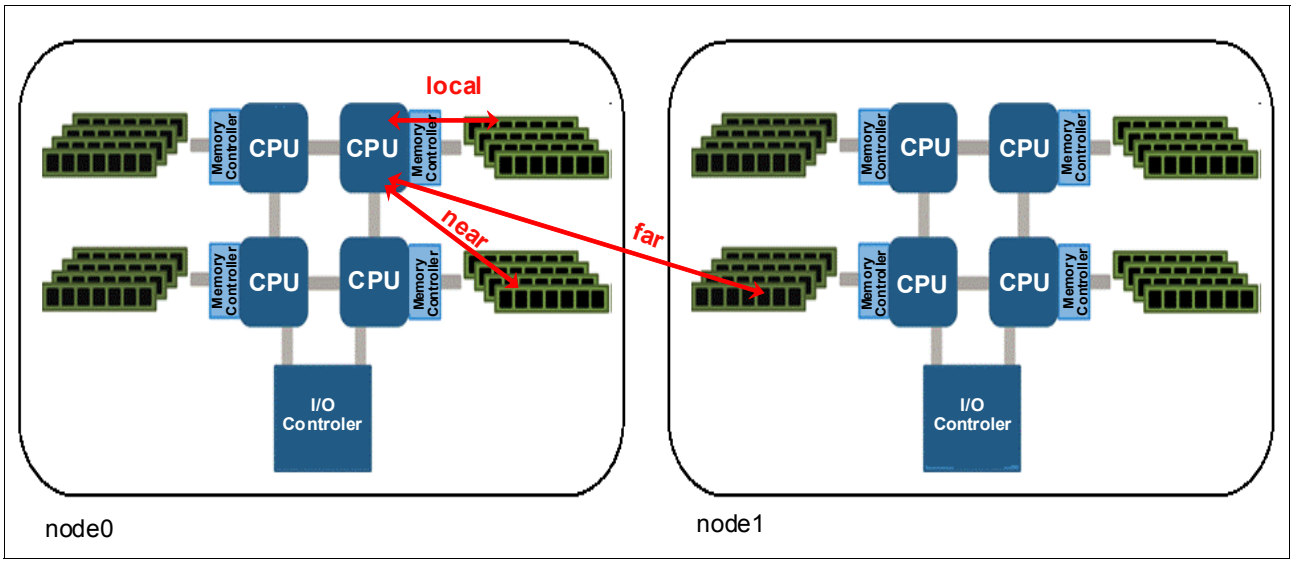


Figure 2-3 Power Systems with local, near, and far memory access

Summary: Power Systems can have up to three different latency memory accesses (Figure 2-3). This memory access time depends on the memory location relative to a processor.

Latency access time (from lowest to highest): local → near or remote → far or distant.

Many people focus on the latency effect and think NUMA is a problem, which is wrong. Remember that NUMA is attempting to solve the scalability issue of the SMP architecture. Having a system with 32 cores in two CECs performs better than 16 cores in one CEC; check the system performance document at:

http://www.ibm.com/systems/power/hardware/reports/system_perf.html

2.2.2 PowerVM logical partitioning and NUMA

You know now that the hardware architecture of the IBM Power Systems is based on NUMA. But compared to other systems, Power System servers offer the ability to create several LPARs, thanks to PowerVM.

The PowerVM hypervisor is an abstraction layer that runs on top of the hardware. One of its roles is to assign cores and memory to the defined logical partitions (LPARs). The POWER7 hypervisor was improved to maximize partition performance through processor and memory affinity. It optimizes the assignment of processor and memory to partitions based on system topology. This results in a balanced configuration when running multiple partitions on a system. The *first time* an LPAR gets activated, the hypervisor allocates processors as close as possible to where allocated memory is located in order to reduce remote and distant memory access. This processor and memory placement is preserved across LPAR reboot (even after a shutdown and reactivation of the LPAR profile) to keep consistent performance and prevent fragmentation of the hypervisor memory.

For shared partitions, the hypervisor assigns a home node domain, the chip where the partition's memory is located. The entitlement capacity (EC) and the amount of memory determine the number of home node domains for your LPAR. The hypervisor dispatches the shared partition's virtual processors (VP) to run on the home node domain whenever possible. If dispatching on the home node domain is not possible due to physical processor overcommitment of the system, the hypervisor dispatches the virtual processor temporarily on another chip.

Let us take some example to illustrate the hypervisor resource placement for virtual processors.

In a POWER 780 with four drawers and 64 cores (Example 2-1), we create one LPAR with different EC/VP configurations and check the processor and memory placement.

Example 2-1 POWER 780 configuration

```
{D-PW2k2-lpar2:root}/home/2bench # prtconf
System Model: IBM,9179-MHB
Machine Serial Number: 10ADA0E
Processor Type: PowerPC_POWER7
Processor Implementation Mode: POWER 7
Processor Version: PV_7_Compat
Number Of Processors: 64
Processor Clock Speed: 3864 MHz
CPU Type: 64-bit
Kernel Type: 64-bit
LPAR Info: 4 D-PW2k2-lpar2
Memory Size: 16384 MB
Good Memory Size: 16384 MB
Platform Firmware level: AM730_095
Firmware Version: IBM,AM730_095
```

```
Console Login: enable
Auto Restart: true
Full Core: false
```

- D-PW2k2-lpar2 is created with EC=6.4, VP=16, MEM=4 GB. Because of EC=6.4, the hypervisor creates one HOME domain in one chip with all the VPs (Example 2-2).

Example 2-2 Number of HOME domains created for an LPAR EC=6.4, VP=16

```
D-PW2k2-lpar2:root}/ # lssrad -av
REF1   SRAD        MEM      CPU
0
          0    3692.12      0-63
```

Note: The **lssrad** command detail is explained in Example 2-6 on page 16.

- D-PW2k2-lpar2 is created with EC=10, VP=16, MEM=4 GB. Because of EC=10, which is greater than the number of cores in one chip, the hypervisor creates two HOME domains in two chips with VPs spread across them (Example 2-3).

Example 2-3 Number of HOME domain created for an LPAR EC=10, VP=16

```
{D-PW2k2-lpar2:root}/ # lssrad -av
REF1   SRAD        MEM      CPU
0
          0    2464.62      0-23 28-31 36-39 44-47 52-55 60-63
          1    1227.50      24-27 32-35 40-43 48-51 56-59
```

- Last test with EC=6.4, VP=64 and MEM=16 GB; just to verify that number of VP has no influence on the resource placement made by the hypervisor.

EC=6.4 < 8 cores so it can be contained in one chip, even if the number of VPs is 64 (Example 2-4).

Example 2-4 Number of HOME domains created for an LPAR EC=6.4, VP=64

```
D-PW2k2-lpar2:root}/ # lssrad -av
REF1   SRAD        MEM      CPU
0
          0   15611.06      0-255
```

Of course, it is obvious that 256 SMT threads (64 cores) cannot really fit in one POWER7 8-core chip. **lssrad** only reports the VPs in front of their *preferred* memory domain (called home domain).

On LPAR activation, the hypervisor allocates only one memory domain with 16 GB because our EC can be store within a chip (6.4 EC < 8 cores), and there is enough free cores in a chip and enough memory close to it. During the workload, if the need in physical cores goes beyond the EC, the POWER hypervisor tries to dispatch VP on the same chip (home domain) if possible. If not, VPs are dispatched on another POWER7 chip with free resources, and memory access will not be local.

Conclusion

If you have a large number of LPARs on your system, we suggest that you create and start your *critical LPARs* first, from the biggest to the smallest. This helps you to get a better affinity

configuration for these LPARs because it makes it more possible for the POWER hypervisor to find resources for optimal placement.

Tip: If you have LPARs with a virtualized I/O card that depend on resources from a VIOS, but you want them to boot before the VIOS to have a better affinity, you can:

1. Start LPARs the *first time* (most important LPARs first) in “open firmware” or “SMS” mode to let the PowerVM hypervisor assign processor and memory.
2. When all your LPARs are up, you can boot the VIOS in normal mode.
3. When the VIOS are ready, you can reboot all the LPARs in normal mode. The order is not important here because LPAR placement is already optimized by PowerVM in step 1.

Even if the hypervisor optimizes your LPAR processor and memory affinity on the very first boot and tries to keep this configuration persistent across reboot, you must be aware that some operations can change your affinity setup, such as:

- Reconfiguration of existing LPARs with new profiles
- Deleting and recreating LPARs
- Adding and removing resources to LPARs dynamically (dynamic LPAR operations)

In the next chapter we show how to determine your LPAR processor memory affinity, and how to *re-optimize* it.

2.2.3 Verifying processor memory placement

You need now to find a way to verify whether the LPARs created have an “optimal” processor and memory placement, which is achieved when, for a given LPAR definition (number of processors and memory), the partition uses the minimum number of sockets and books to reduce remote and distant memory access to the minimum. The information about your system, such as the number of cores per chip, memory per chip and per book, are critical to be able to make this estimation.

Here is an example for a system with 8-core POWER7 chips, 32 GB of memory per chip, two books (or nodes), and two sockets per node.

- An LPAR with six cores, 24 GB of memory is optimal if it can be contained in one chip (only local memory access).
- An LPAR with 16 cores, 32 GB of memory is optimal if it can be contained in two chips within the same book (local and remote memory access). This is the best processor and memory placement you can have with this number of cores. You must also verify that the memory is well balanced across the two chips.
- An LPAR with 24 cores, 48 GB memory is optimal if it can be contained in two books with a balanced memory across the chips. Even if you have some distant memory access, this configuration is optimal because you do not have another solution to satisfy the 24 required cores.
- An LPAR with 12 cores, 72 GB of memory is optimal if it can be contained in two books with a balanced memory across the chips. Even if you have some distant memory access, this configuration is optimal because you do not have another solution to satisfy the 72 GB of memory.

As explained in the shaded box on page 14, some operations such as dynamic LPAR can “fragment” your LPAR configuration, which gives you some nonoptimal placement for some

LPARs. As described in Figure 2-4, avoid having a system with “fragmentation” in the LPAR processor and memory assignment.

Also, be aware that the more LPARs you have, the harder it is to have all your partitions defined with an optimal placement. Sometimes you have to take a decision to choose which LPARs are more critical, and give them a better placement by starting them (the first time) before the others (as explained in 2.2.2, “PowerVM logical partitioning and NUMA” on page 12).

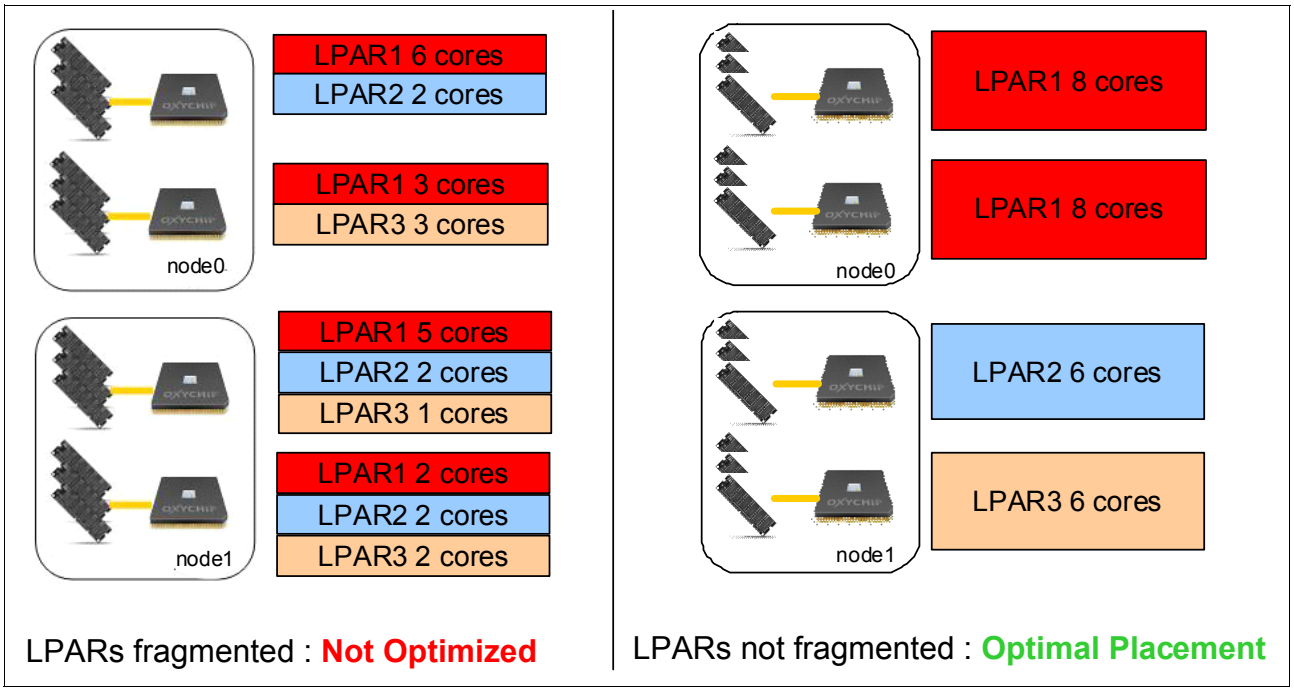


Figure 2-4 Example of optimal versus fragmented LPAR placement

Verifying LPAR resource placement in AIX

In an AIX partition, the **`lparstat -i`** command shows how many processors and how much memory are defined in your partition (Example 2-5).

Example 2-5 Determining LPAR resource with lparstat

```
{D-PW2k2-lpar1:root}/ # lparstat -i
Node Name                                  : D-PW2k2-lpar1
Partition Name                             : D-PW2k2-lpar1
Partition Number                           : 3
Type                                       : Dedicated-SMT-4
Mode                                       : Capped
Entitled Capacity                          : 8.00
Partition Group-ID                         : 32771
Shared Pool ID                             : -
Online Virtual CPUs                        : 8
Maximum Virtual CPUs                       : 16
Minimum Virtual CPUs                       : 1
Online Memory                              : 32768 MB
Unallocated I/O Memory entitlement         : -
Memory Group ID of LPAR                    : -
Desired Virtual CPUs                       : 8
Desired Memory                             : 32768 MB
```

```
....
```

From Example 2-5 on page 15, we know that our LPAR has eight *dedicated* cores with SMT4 (8x4=32 logical cpu) and 32 GB of memory.

Our system is a 9179-MHB (POWER 780) with four nodes, two sockets per node, each socket with eight cores and 64 GB of memory. So, the best resource placement for our LPAR would be one POWER7 chip with eight cores and 32 GB of memory next to this chip.

To check your processor and memory placement, you can use the **lssrad -av** command from your AIX instance, as shown in Example 2-6.

Example 2-6 Determining resource placement with lssrad

```
{D-PW2k2-lpar1:root}/ # lssrad -av
REF1   SRAD        MEM CPU
0
          0   15662.56      0-15
1
          1   15857.19      16-31
```

- REF1 (first hardware-provided reference point) represents a drawer of our POWER 780. For a POWER 795, this represents a book. Systems other than POWER 770, POWER 780, or POWER 795, do not have a multiple drawer configuration (Table 1-1 on page 3) so they cannot have several REF1s.
- Scheduler Resource Allocation Domain (SRAD) represents a socket number. In front of each socket, there is an amount of memory attached to our partition. We also find the logical processor number attached to this socket.

Note: The number given by REF1 or SRAD does not represent the real node number or socket number on the hardware. All LPARs will report a REF1 0 and a SRAD 0. They just represent a logical number inside the operating system instance.

From Example 2-6, we can conclude that our LPAR is composed of two sockets (SRAD 0 and 1) with four cores on each (0-15 = 16-31 = 16 lcpu SMT4 = 4 cores) and 16 GB of memory attached to each socket. These two sockets are located in two different nodes (REF1 0 and 1).

Compared to our expectation (which was: only one socket with 32 GB of memory means only local memory access), we have two different sockets in two different nodes (high potential of distant memory access). The processor and memory resource placement for this LPAR is not optimal and performance could be degraded.

LPAR processor and memory placement impact

To demonstrate the performance impact, we performed the following experiment: We created an LPAR (eight dedicated cores, 32 GB of memory) on a POWER 780 (four drawers, eight sockets). We generated an OnLine Transactional Processing (OLTP) load on an Oracle database with 200 concurrent users and measured the number of transactions per second (TPS). Refer to the “Oracle SwingBench” on page 342.

Test 1: The first test was done with a nonoptimal resource placement: eight dedicated cores spread across two POWER7 chips, as shown in Example 2-6,

Test 2: The second test was done with an optimal resource placement: eight dedicated cores on the same chip with all the memory attached as shown in Example 2-7 on page 17.

Example 2-7 Optimal resource placement for eight cores and 32 GB of memory

```
{D-PW2k2-lpar1:root}/ # lssrad -av
REF1   SRAD        MEM      CPU
0
          0   31519.75      0-31
```

Test results

During the two tests, the LPAR processor utilization was 100%. We waited 5 minutes during the steady phase and took the average TPS as result of the experiment (Table 2-1 on page 18). See Figure 2-5, and Figure 2-6.

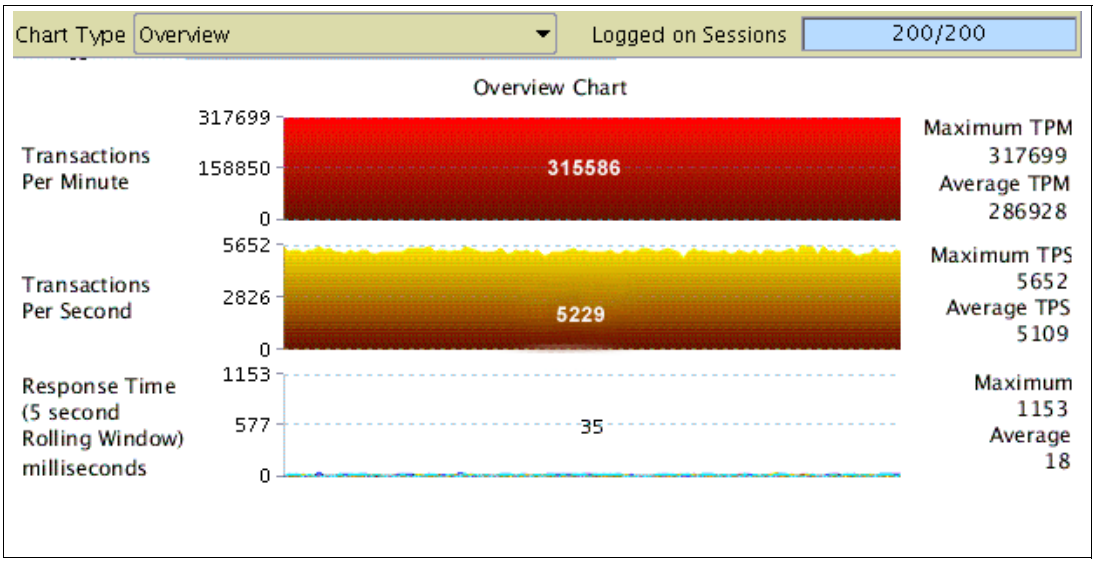


Figure 2-5 Swinbench results for test1 (eight cores on two chips: nonoptimal resource placement)

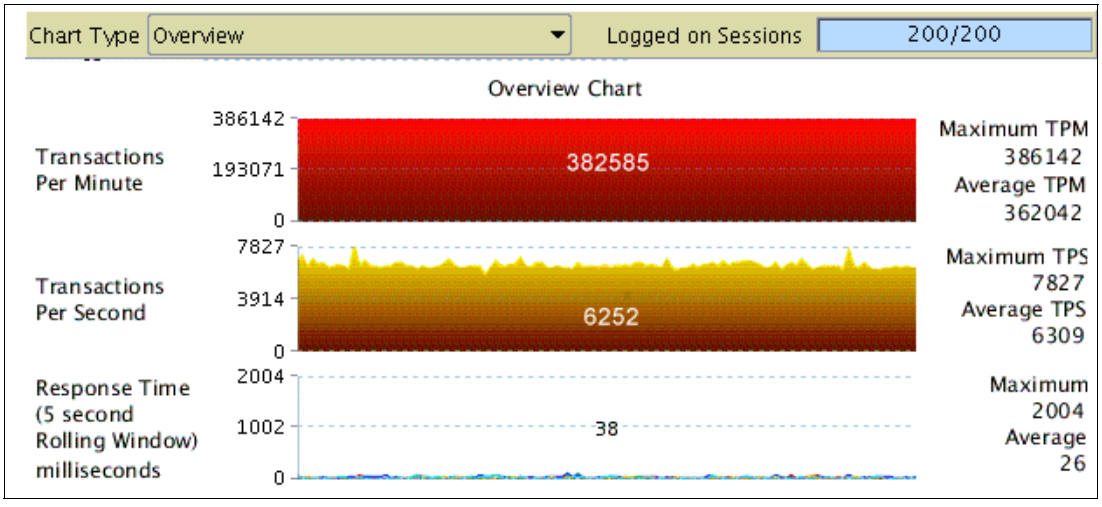


Figure 2-6 Swingbench results for Test 2 (eight cores on one chip: optimal resource placement)

This experiment shows 24% improvement in TPS when most of the memory accesses are local compared to a mix of 59% local and 41% distant. This is confirmed by a higher Cycle per Instruction (CPI) in test 1 (CPI=7.5) compared to test 2 (CPI=4.8). This difference can be explained by a higher memory latency for 41% of the access in test 1, which causes some

additional *empty* processor cycle when waiting for data from the distant memory to complete the instruction.

Table 2-1 Result table of resource placement impact test on an Oracle OLTP workload

Test name	Resource placement	Access to local memory[a]	CPI	Average TPS	Performance ratio
Test 1	non Optimal (local + distant)	59%	7.5	5100	1.00
Test 2	Optimal (only local)	99.8%	4.8	6300	1.24

a. Results given by the AIX **hpmstat** command in “Using hpmstat to identify LSA issues” on page 134.

Notice that 59% local access is not so bad with this “half local/ half distant” configuration. This is because the AIX scheduler is aware of the processor and memory placement in the LPAR, and has enhancements to reduce the NUMA effect as shown in 6.1, “Optimizing applications with AIX features” on page 280.

Note: These results are from experiments based on a load generation tool named Swingbench; results may vary depending on the characteristics of your workload. The purpose of this experiment is to give you an idea of the potential gain you can get if you take care of your resource placement.

2.2.4 Optimizing the LPAR resource placement

As explained in the previous section, processor and memory placement can have a direct impact on the performance of an application. Even if the PowerVM hypervisor optimizes the resource placement of your partitions on the first boot, it is still not “clairvoyant.” It cannot know by itself which partitions are more important than others, and cannot anticipate what will be the next changes in our “production” (creation of new “critical production” LPARs, deletion of old LPARs, dynamic LPAR, and so on). You can help the PowerVM hypervisor to cleanly place the partitions by sizing your LPAR correctly and using the proper PowerVM option during your LPAR creation.

Do not oversize your LPAR

Realistic sizing of your LPAR is really important to get a better processor memory affinity. Try not to give to a partition more processor than needed.

If a partition has nine cores assigned, cores and memory are spread across two chips (best scenario). If, during peak time, this partition consumes only seven cores, it would have been more efficient to assign seven or even eight cores to this partition only to have the cores and the memory within the same POWER7 chip.

For a virtualized processor, a good Entitlement Capacity (EC) is really important. Your EC must fit with the average need of processor power of your LPAR during a regular load (for example, the day only for a typical day’s OLTP workload, the night only for typical night “batch” processing). This gives you a resource placement that better fits the needs of your partition. As for dedicated processor, try not to oversize your EC across domain boundaries (cores per chip, cores per node). A discussion regarding how to efficiently size your virtual processor resources is available in 3.1, “Optimal logical partition (LPAR) sizing” on page 42.

Memory follows the same rule. If you assign to a partition more memory than can be found behind a socket or inside a node, you will have to deal with some remote and distant memory access. This is not a problem if you really need this memory, but if you do not use it totally, this situation could be avoided with a more realistic memory sizing.

Affinity groups

This option is available with PowerVM Firmware level 730. The primary objective is to give hints to the hypervisor to place multiple LPARs within a single domain (chip, drawer, or book). If multiple LPARs have the same `affinity_group_id`, the hypervisor places this group of LPARs as follows:

- Within the same chip if the total capacity of the group does not exceed the capacity of the chip
- Within the same drawer (node) if the capacity of the group does not exceed the capacity of the drawer

The second objective is to give a different priority to one or a group of LPARs. Since Firmware level 730, when a server frame is rebooted, the hypervisor places all LPARs before their activation. To decide which partition (or group of partitions) should be placed first, it relies on `affinity_group_id` and places the highest number first (from 255 to 1).

The following Hardware Management Console (HMC) CLI command adds or removes a partition from an affinity group:

```
chsyscfg -r prof -m <system_name> -i
name=<profile_name>,lpar_name=<partition_name>,affinity_group_id=<group_id>
```

where `group_id` is a number between 1 and 255, `affinity_group_id=none` removes a partition from the group.

The command shown in Example 2-8 sets the `affinty_group_id` to 250 to the profile named Default for the `795_1_AIX1` LPAR.

Example 2-8 Modifying the affinity_group_id flag with the HMC command line

```
hscroot@hmc24:~> chsyscfg -r prof  -m HAUTBRION -i
name=Default,lpar_name=795_1_AIX1,affinity_group_id=250
```

You can check the `affinity_group_id` flag of all the partitions of your system with the **`lssyscfg`** command, as described in Example 2-9.

Example 2-9 Checking the affinity_group_id flag of all the partitions with the HMC command line

```
hscroot@hmc24:~> lssyscfg -r lpar -m HAUTBRION -F name,affinity_group_id
p24n17,none
p24n16,none
795_1_AIX1,250
795_1_AIX2,none
795_1_AIX4,none
795_1_AIX3,none
795_1_AIX5,none
795_1_AIX6,none
```

POWER 795 SPPL option and LPAR placement

On POWER 795, there is an option called Shared Partition Processor Limit (SPPL). Literally, this option limits the processor capacity of an LPAR. By default, this option is set to 24 for

POWER 795 with six-core POWER7 chip or 32 for the eight-core POWER7 chip. If your POWER 795 has three processor books or more, you can set this option to maximum to remove this limit. This change can be made on the Hardware Management Console (HMC).

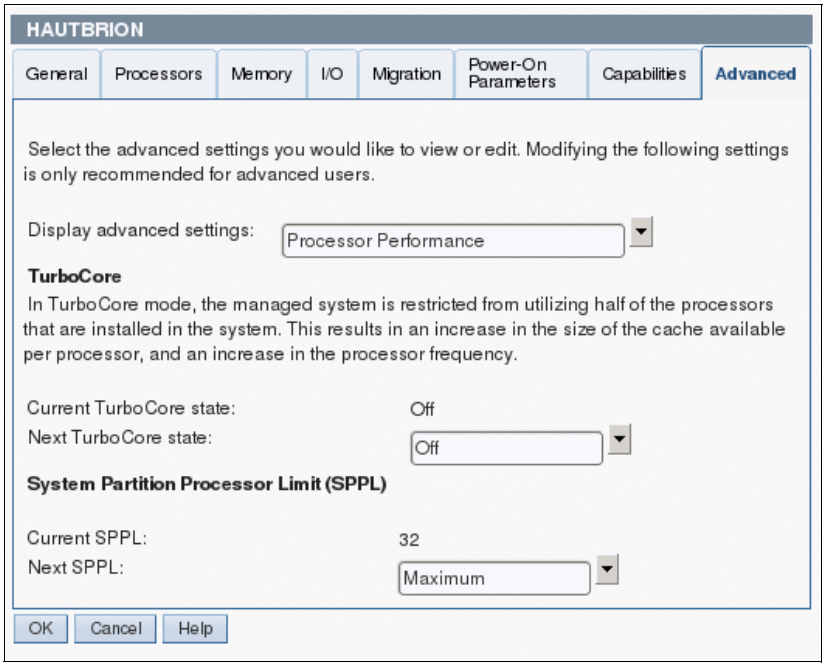


Figure 2-7 Changing POWER 795 SPPL option from the HMC

Changing SPPL on the Hardware Management Console (HMC): Select your **POWER 795** → **Properties** → **Advanced** → **change Next SPPL to maximum** (Figure 2-7). After changing the SPPL value, you need to stop all your LPARs and restart the POWER 795.

The main objective of the SPPL is not to limit the processor capacity of an LPAR, but to influence the way the PowerVM hypervisor assigns processor and memory to the LPARs.

- When SPPL is set to 32 (or 24 if six-core POWER7), then the PowerVM hypervisor allocates processor and memory in the same processor book, if possible. This reduces access to distant memory to improve memory latency.
- When SPPL is set to maximum, there is no limitation to the number of desired processors in your LPAR. But large LPAR (more than 24 cores) will be spread across several books to use more memory DIMMs and maximize the interconnect bandwidth. For example, a 32-core partition in SPPL set to maximum will spread across two books compared to only one if SPPL is set to 32.

SPPL maximum improves memory bandwidth for large LPARs, but reduces locality of the memory. This can have a direct impact on applications that are more latency sensitive compared to memory bandwidth (for example, databases for most of the client workload).

To address this case, a flag can be set on the profile of each large LPAR to signal the hypervisor to try to allocate processor and memory in a minimum number of books (such as SPPL 32 or 24). This flag is `lpar_placement` and can be set with the following HMC command (Example 2-10 on page 21):

```
chsyscfg -r prof -m <managed-system> -i
name=<profile-name>,lpar_name=<lpar-name>,lpar_placement=1
```

Example 2-10 Modifying the lpar_placement flag with the HMC command line

This command sets the *lpar_placement* to 1 to the profile named *default* for 795_1_AIX1 LPAR:

```
hscroot@hmc24:~> chsyscfg -r prof -m HAUTBRION -i
name=Default,lpar_name=795_1_AIX1,lpar_placement=1
```

You can use the **lsyscfg** command to check the current lpar_placement value for all the partitions of your system:

```
hscroot@hmc24:~> lssyscfg -r lpar -m HAUTBRION -F name,lpar_placement
p24n17,0
p24n16,0
795_1_AIX1,1
795_1_AIX2,0
795_1_AIX4,0
795_1_AIX3,0
795_1_AIX5,0
795_1_AIX6,0
```

Table 2-2 describes in how many books an LPAR is spread by the hypervisor, depending on the number of processors of this LPAR, SPPL value, and `lpar_placement` value.

Table 2-2 Number of books used by LPAR depending on SPPL and the lpar_placement value

Number of processors	**Number of books** (SPPL=32)	**Number of books** (SPPL=maximum, lpar_placement=0)	**Number of books** (SPPL=maximum, lpar_placement=1)
8	1	1	1
16	1	1	1
24	1	1	1
32	1	2	1
64	not possible	4	2

Note: The **lpar_placement=1** flag is only available for PowerVM Hypervisor eFW 730 and above. In the 730 level of firmware, **lpar_placement=1** was only recognized for dedicated processors and non-TurboCore mode (MaxCore) partitions when SPPL=MAX.

Starting with the 760 firmware level, **lpar_placement=1** is also recognized for shared processor partitions with SPPL=MAX or systems configured to run in TurboCore mode with SPPL=MAX.

Force hypervisor to re-optimize LPAR resource placement

As explained in 2.2.2, “PowerVM logical partitioning and NUMA” on page 12, the PowerVM hypervisor optimizes resource placement on the first LPAR activation. But some operations, such as dynamic LPAR, may result in memory fragmentation causing LPARs to be spread across multiple domains. Because the hypervisor keeps track of the placement of each LPAR, we need to find a way to re-optimize the placement for some partitions.

Note: In this section, we give you some ways to force the hypervizor to re-optimize processor and memory affinity. Most of these solutions are workarounds based on the new PowerVM options in Firmware level 730.

In Firmware level 760, IBM gives an official solution to this problem with *Dynamic Platform Optimizer* (DPO). If you have Firmware level 760 or above, go to “Dynamic Platform Optimizer” on page 23.

There are three ways to re-optimize LPAR placement, but they can be disruptive:

- You can shut down all your LPARs, and restart your system. When PowerVM hypervisor is restarted, it starts to place LPARs starting from the higher `group_id` to the lower and then place LPARs without `affinity_group_id`.
- Shut down all your LPARs and create a new partition in an *all-resources* mode and activate it in open firmware. This frees all the resources from your partitions and re-assigns them to this new LPAR. Then, shut down the *all-resources* and delete it. You can now restart your partitions. They will be re-optimized by the hypervisor. Start with the most critical LPAR to have the best location.
- There is a way to force the hypervisor to forget placement for a specific LPAR. This can be useful to get processor and memory placement from noncritical LPARs, and force the hypervisor to re-optimize a critical one. By freeing resources before re-optimization, your critical LPAR will have a chance to get a better processor and memory placement.
  - Stop critical LPARs that should be re-optimized.
  - Stop some noncritical LPARs (to free the most resources possible to help the hypervisor to find a better placement for your critical LPARs).
  - Freeing resources from the *non-activated* LPARs with the following HMC commands. You need to remove all memory and processors (Figure 2-8):

    ```
    chhwres -r mem -m <system_name> -o r -q <num_of_Mbytes> --id <lp_id>
    chhwres -r proc -m <system_name> -o r --procunits <number> --id <lp_id>
    ```

    You need to remove all memory and processor as shown in Example 2-11.

    You can check the result from the HMC. All resources “Processing Units” and Memory should be 0, as shown in Figure 2-9 on page 23.
  - Restart your critical LPAR. Because all processors and memory are removed from your LPAR, the PowerVM hypervisor is forced to re-optimize the resource placements for this LPAR.
  - Restart your noncritical LPAR.

Name	ID	Status	Processing Units	Memory (GB)	Active Profile	Environment
p24n16	8	Running	1	4	p24n16_vios	Virtual I/O Server
p24n17	9	Running	1	4	p24n17_vios	Virtual I/O Server
750_1_AIX1	10	Not Activated	1	8	Default	AIX or Linux
750_1_AIX2	11	Running	3	8	seb	AIX or Linux

Figure 2-8 HMC screenshot before freeing 750_1_AIX1 LPAR resources

Example 2-11 HMC command line to free 750_1_AIX1 LPAR resources

```
hscroot@hmc24:~> chhwres -r mem -m 750_1_SN106011P -o r -q 8192 --id 10
hscroot@hmc24:~> chhwres -r proc -m 750_1_SN106011P -o r --procunits 1 --id 10
```

Name	ID	Status	Processing Units	Memory (GB)	Active Profile	Environment
p24n16	8	Running	1	4	p24n16_vios	Virtual I/O Server
p24n17	9	Running	1	4	p24n17_vios	Virtual I/O Server
750_1_AIX1	10	Not Activated	0	0	Default	AIX or Linux
750_1_AIX2	11	Running	3	8	seb	AIX or Linux

Figure 2-9 HMC screenshot after freeing 750_1_AIX1 LPAR resources

In Figure 2-9, notice that Processing Units and Memory of the 750_1_AIX1 LPAR are now set to 0. This means that processor and memory placement for this partition will be re-optimized by the hypervisor on the next profile activation.

Dynamic Platform Optimizer

Dynamic Platform Optimizer (DPO) is a new capability of IBM Power Systems introduced with Firmware level 760, and announced in October 2012. It is free of charge, available since October 2012 for POWER 770+, POWER 780+, and since November 2012 for POWER 795. Basically it allows the hypervisor to dynamically optimize the processor and memory resource placement of LPARs while they are running (Figure 2-10). This operation needs to be run manually from the HMC command line.

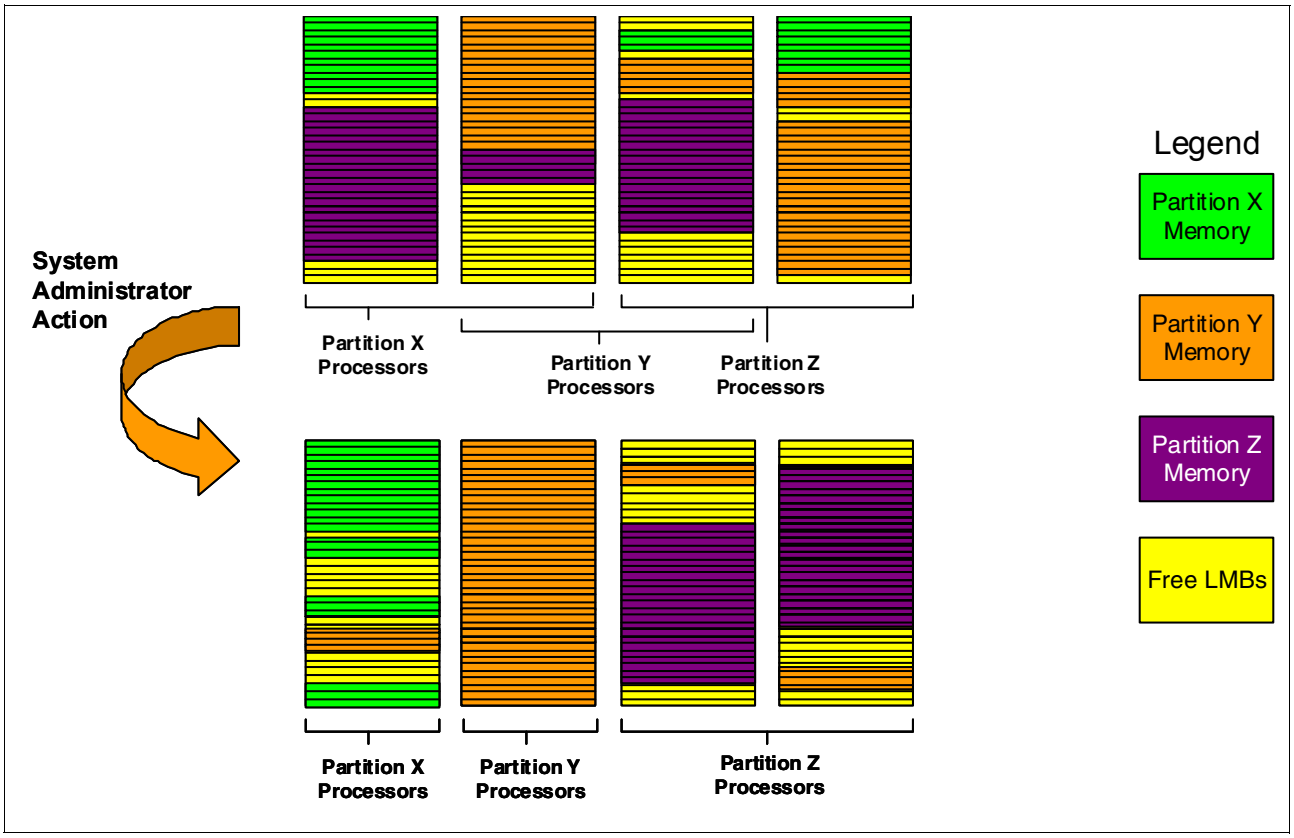


Figure 2-10 DPO defragmentation illustration

To check whether your system supports DPO: Select your system from your HMC graphical interface:

Properties → **Capabilities** → **Check Capabilities for DPO** (Figure 2-11 on page 24).

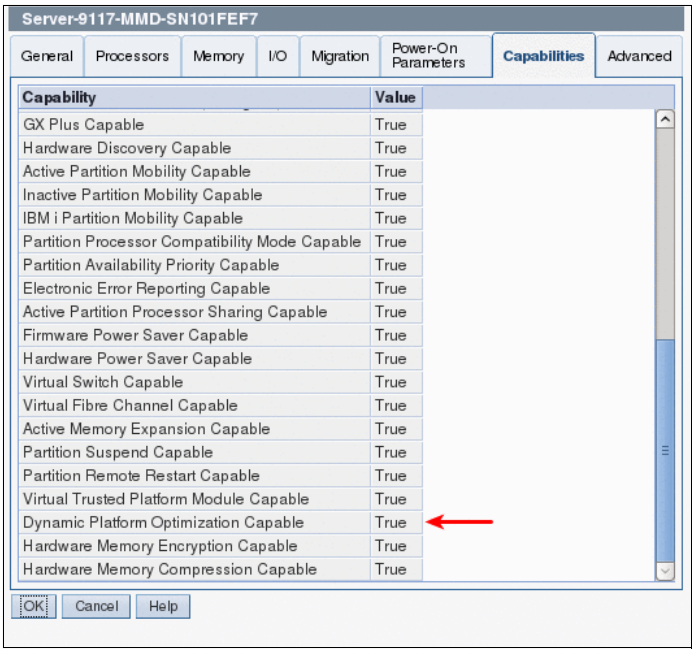


Figure 2-11 Checking for DPO in system capabilities

Dynamic Platform Optimizer is able to give you a score (from 0 to 100) of the actual resource placement in your system based on hardware characteristics and partition configuration. A score of 0 means poor affinity and 100 means perfect affinity.

Note: Depending on the system topology and partition configuration, perfect affinity is not always possible.

On the HMC, the command line to get this score is:

```
lsmemopt -m <system_name> -o currscore
```

In Example 2-12, you can see that our system affinity is 75.

Example 2-12 HMC DPO command to get system affinity current score

```
hscroot@hmc56:~> lsmemopt -m Server-9117-MMD-SN101FEF7 -o currscore
curr_sys_score=75
```

From the HMC, you can also ask for an evaluation of what the score on your system would be after affinity was optimized using the Dynamic Platform Optimizer.

Note: The predicted affinity score is an estimate, and may not match the actual affinity score after DPO has been run.

The HMC command line to get this score is:

```
lsmemopt -m <system_name> -o calcscore
```

In Example 2-13 on page 25, you can see that our current system affinity score is 75, and 95 is predicted after re-optimization by DPO.

Example 2-13 HMC DPO command to evaluate affinity score after optimization

```
hscroot@hmc56:~> lsmemopt -m Server-9117-MMD-SN101FEF7 -o calcscore
curr_sys_score=75,predicted_sys_score=95,"requested_lpar_ids=5,6,7,9,10,11,12,13,1
4,15,16,17,18,19,20,21,22,23,24,25,27,28,29,30,31,32,33,34,35,36,37,38,41,42,43,44
,45,46,47,53",protected_lpar_ids=none
```

When DPO starts the optimization procedure, it starts with LPARs from the highest `affinity_group_id` from id 255 to 0 (see “Affinity groups” on page 19), then it continues with the biggest partition and finishes with the smallest one.

To start affinity optimization with DPO for all partitions in your system, use the following HMC command:

```
optmem -m <system_name> -o start -t affinity
```

Note: The **optmem** command-line interface supports a set of “requested” partitions, and a set of “protected” partitions. The requested ones are prioritized highest. The protected ones are not touched. Too many protected partitions can adversely affect the affinity optimization, since their resources cannot participate in the system-wide optimization.

To perform this operation, DPO needs free memory available in the system and some processor cycles. The more resources available for DPO to perform its optimization, the faster it will be. But be aware that this operation can take time and performance could be degraded; so this operation should be planned during low activity periods to reduce the impact on your production environment.

Note: Some functions such as dynamic LPAR and Live Partition Mobility cannot run concurrently with the optimizer.

Here are a set of examples to illustrate how to start (Example 2-14), monitor (Example 2-15) and stop (Example 2-16) a DPO optimization.

Example 2-14 Starting DPO on a system

```
hscroot@hmc56:~> optmem -m Server-9117-MMD-SN101FEF7 -o start -t affinity
```

Example 2-15 Monitoring DPO task progress

```
hscroot@hmc56:~> lsmemopt -m Server-9117-MMD-SN101FEF7
in_progress=0,status=Finished,type=affinity,opt_id=1,progress=100,
requested_lpar_ids=non,protected_lpar_ids=none,”impacted_lpar_ids=106,110”
```

Example 2-16 Forcing DPO to stop before the end of optimization

```
hscroot@hmc56:~> optmem -m Server-9117-MMD-SN101FE_F7 -o stop
```

Note: Stopping DPO before the end of the optimization process can result in poor affinity for some partitions.

Partitions running on AIX 7.1 TL2, AIX 6.1 TL8, and IBM i 7.1 PTF MF56058 receive notification from DPO when the optimization completes, and whether affinity has been changed for them. This means that all the tools such as **lssrad** (Example 2-6 on page 16) can report the changes automatically; but more importantly, the scheduler in these operating systems is now aware of the new processor and memory affinity topology.

For older operating systems, the scheduler will not be aware of the affinity topology changes. This could result in performance degradation. You can exclude these partitions from the DPO optimization to avoid performance degradation by adding them to the protected partition set on the **optmem** command, or reboot the partition to refresh the processor and memory affinity topology.

2.2.5 Conclusion of processor and memory placement

As you have seen, processor and memory placement is really important when talking about *raw* performance. Even if the hypervisor optimizes the affinity, you can influence it and help it by:

- Knowing your system hardware architecture and planning your LPAR creation.
- Creating and starting critical LPARs first.
- Giving a priority order to your LPAR and correctly setting the `affinity_group` HMC parameter.
- Setting the parameter `lpar_placement` when running on POWER 795 with SSPL=maximum.
- If possible, use DPO to defragment your LPAR placement and optimize your system.

If you want to continue this Processor and Memory Affinity subject at the AIX operating system level, refer to 6.1, “Optimizing applications with AIX features” on page 280.

2.3 Performance consequences for I/O mapping and adapter placement

In this section, we provide adapter placement, which is an important step to consider when installing or expanding a server.

Environments change all the time, in every way, from a simple hardware upgrade to entire new machines, and many details come to our attention.

Different machines have different hardware, even when talking about different models of the same type. The POWER7 family has different types and models from the entry level to high-end purposes and the latest POWER7 models may have some advantages over their predecessors. If you are upgrading or acquiring new machines, the same plan used to deploy the first may not be the optimal for the latest.

Systems with heavy I/O workloads demand machines capable of delivering maximum throughput and that depends on several architectural characteristics and not only on the adapters themselves. Buses, chipsets, slots—all of these components must be considered to achieve maximum performance. And due to the combination of several components, maximum performance may not be the same that some of them offer individually.

These technologies are always being improved to combine the maximum throughput, but one of the key points for using them efficiently and to achieve good performance is how they are combined, and sometimes this can be a difficult task.

Each machine type has a different architecture and an adequate setup is required to obtain the best I/O throughput from that server. Because the processors and chipsets are connected through different buses, the way adapters are distributed across the slots on a server directly affects the I/O performance of the environment.

Connecting adapters in the wrong slots may not give you the best performance because you do not take advantage of the hardware design, and worse, it may even cause performance degradation. Even different models of the same machine type may have important characteristics for those who want to achieve the highest performance and take full advantage of their environments.

Important: Placing the wrong card in the wrong slot may not only not bring you the performance that you are expecting but can also degrade the performance of your existing environment.

To illustrate some of the differences that can affect the performance of the environment, we take a look at the design of two POWER 740 models. In the sequence, we make a brief comparison between the POWER 740 and the POWER 770, and finally between two models of the POWER 770.

Note: The next examples are only intended to demonstrate how the designs and architectures of the different machines can affect the system performance. For other types of machines, make sure you read the *Technical Overview and Introduction* documentation for that specific model, available on the IBM Hardware Information Center at:

`http://pic.dhe.ibm.com/infocenter/powersys/v3r1m5/index.jsp`

2.3.1 POWER 740 8205-E6B logical data flow

Figure 2-12 on page 28 presents the logical data flow of the 8205-E6B (POWER 740), one of the smaller models of the POWER7 family.

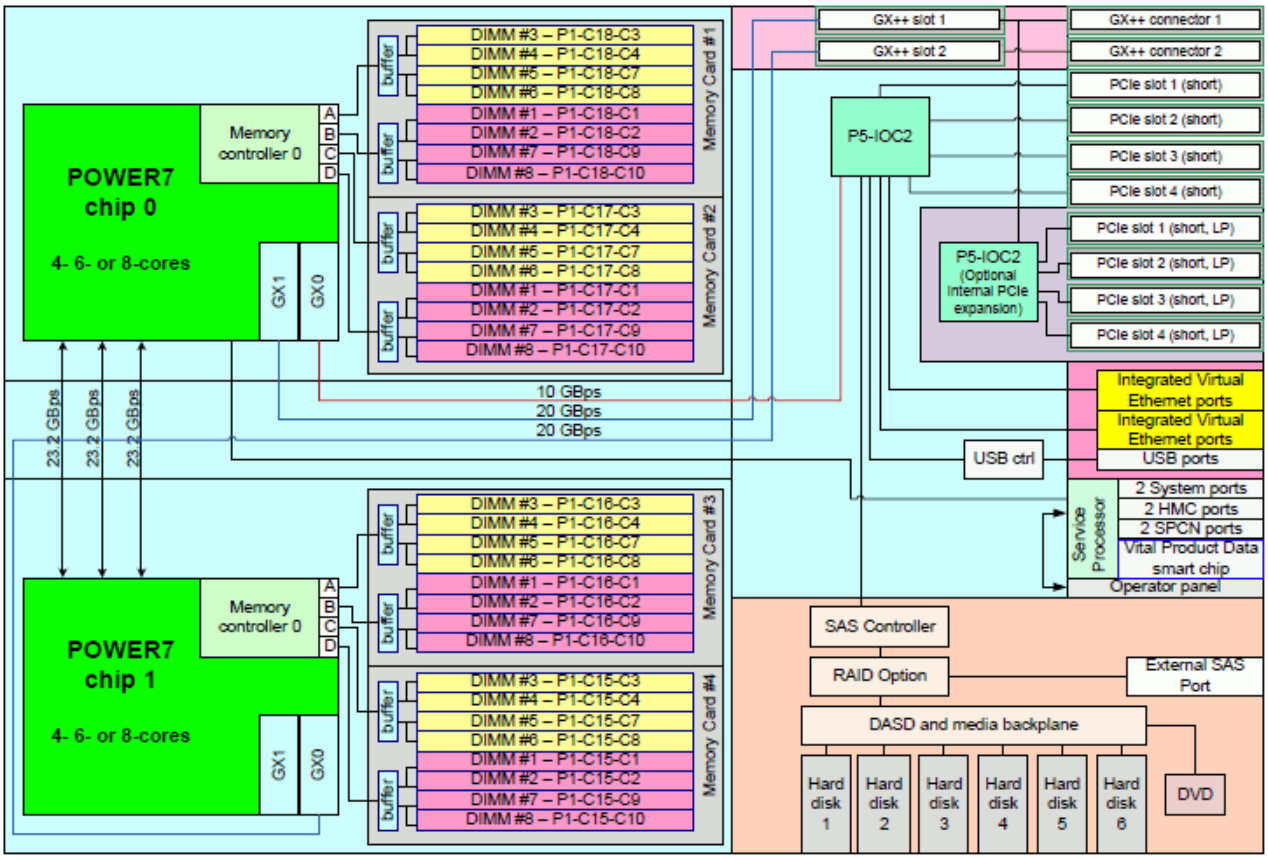


Figure 2-12 POWER 740 8205-E6B - Logic unit flow

The 8205-E6B has two POWER7 chips, each with two GX controllers (GX0 and GX1) connected to the GX++ slots 1 and 2 and to the internal P5-IOC2 controller. The two GX++ bus slots provide 20 Gbps bandwidth for the expansion units while the P5-IOC2 limits the bandwidth of the internal slots to 10 GBps only. Further, the GX++ Slot 1 can connect both an additional P5-IOC2 controller and external expansion units at the same time, sharing its bandwidth, while GX++ Slot 2 only connects to external expansion units.

Notice that this model also has the Integrated Virtual Ethernet (IVE) ports connected to the P5-IOC2, sharing the same bandwidth with all the other internal adapters and SAS disks. It also has an External SAS Port, allowing for more weight over the 10 Gbps chipset.

2.3.2 POWER 740 8205-E6C logical data flow

This is the same picture as seen in the previous section but this time, we have the successor model 8205-E6C, as shown in Figure 2-13 on page 29.

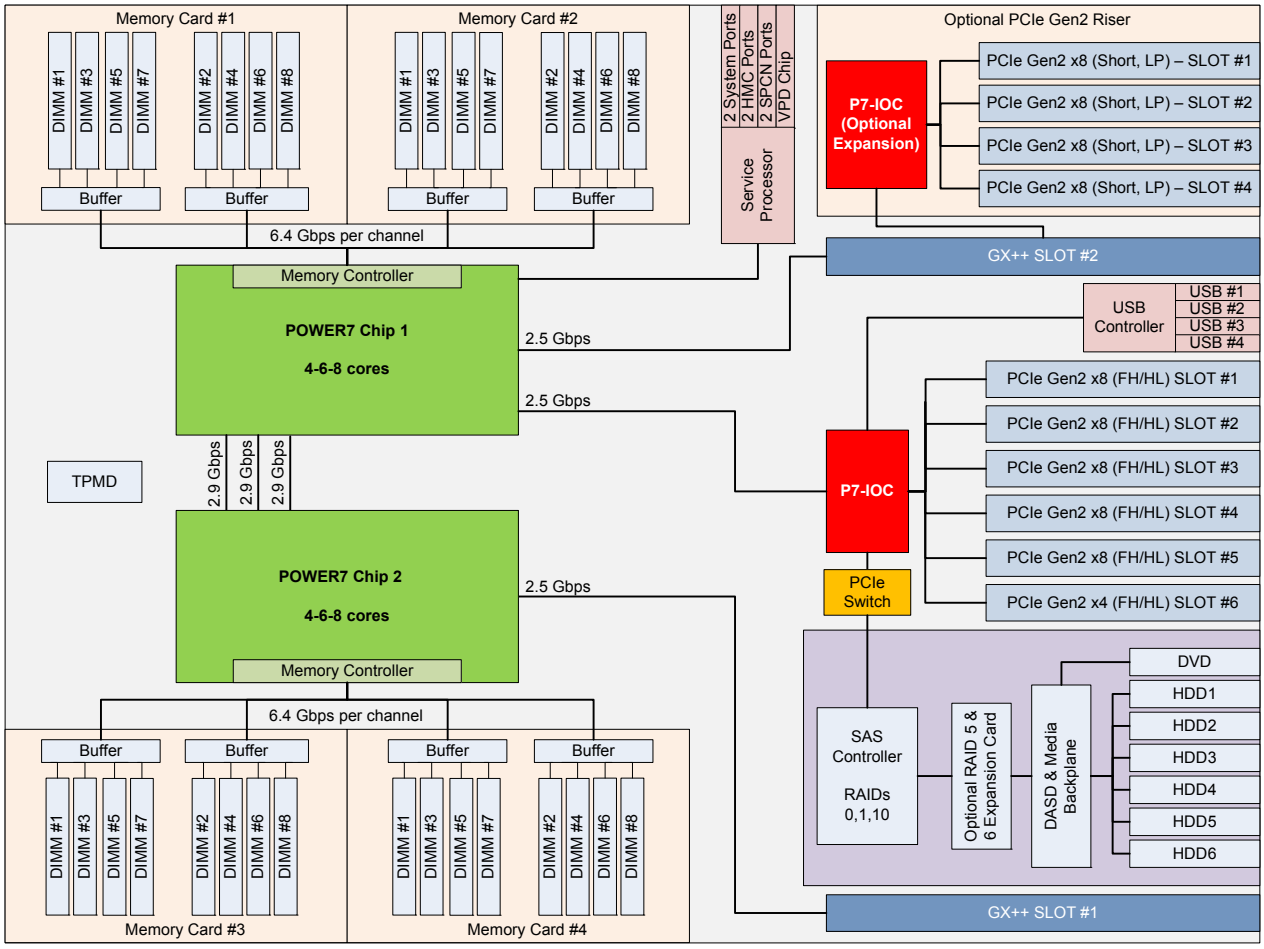


Figure 2-13 POWER 740 8205-E6C - Logic unit flow

Although the general organization of the picture is a bit different, the design itself is still the same, but in this version there a few enhancements to notice. First, the 1.25 GHz P5-IOC from the 8205-E6B has been upgraded to the new P7-IOC with a 2.5 GHz clock on 8205-E6C, raising the channel bandwidth from 10 Gbps to 20 Gbps. Second, the newest model now comes with PCIe Gen2 slots, which allows peaks of 4 Gbps for x8 interfaces instead of the 2 Gbps of their predecessors.

Another interesting difference in this design is that the GX++ slots have been flipped. The GX slot connected to the POWER7 Chip 2 now is the GX++ Slot 1, which provides dedicated channel and bandwidth to the expansion units. And the POWER7 Chip 1 keeps the two separate channels for the P7-IOC and the GX++ Slot 2, which can still connect an optional P7-IOC bus with the riser card.

Finally, beyond the fact that the P7-IOC has an increased bandwidth over the older P5-IOC2, the 8205-E6C does not come with the External SAS Port (although it is allowed through adapter expansion), neither the Integrated Virtual Ethernet (IVE) ports, constraining the load you can put on that bus.

Important: To take full advantage of the PCIe Gen2 slot's characteristics, compatible PCIe2 cards must be used. Using the old PCIe Gen1 cards on Gen2 slots is supported, but although a slight increase of performance may be observed due to the several changes along the bus, the PCIe Gen1cards are still limited to their own speed by design and should not be expected to achieve the same performance as the PCIe Gen2 cards.

2.3.3 Differences between the 8205-E6B and 8205-E6C

Table 2-3 gives a clear view of the differences we mentioned.

Table 2-3 POWER 740 comparison - Models 8205-E6B and 8205-E6C

POWER 740	8205-E6B	8205-E6C
GX++ Slots	2x 20 Gbps	2x 20 Gbps
Internal I/O chipset	P5-IOC2 (10 Gbps)	P7-IOC (20 Gbps)
PCIe Version	Gen1 (2 Gbps)	Gen2 (4 Gbps), Gen1-compatible
Primary GX++ Slot	POWER7 Chip 1 (shared bandwidth)	POWER7 Chip 2 (dedicated bandwidth)
Secondary GX++ Slot	POWER7 Chip2 (dedicated bandwidth)	POWER7 Chip 1 (shared bandwidth)
Expansion I/O w/ Riser Card	POWER7 Chip 1	POWER7 Chip 1

The purpose of this quick comparison is to illustrate important differences that exist among the different machines that may go unnoticed when the hardware is configured. You may eventually find out that instead of placing an adapter on a CEC, you may take advantage of attaching it to an internal slot, if your enclosure is not under a heavy load.

2.3.4 POWER 770 9117-MMC logical data flow

Now we take a brief look to another machine type, and you notice that several differences exist between the POWER 740 and the POWER 770.

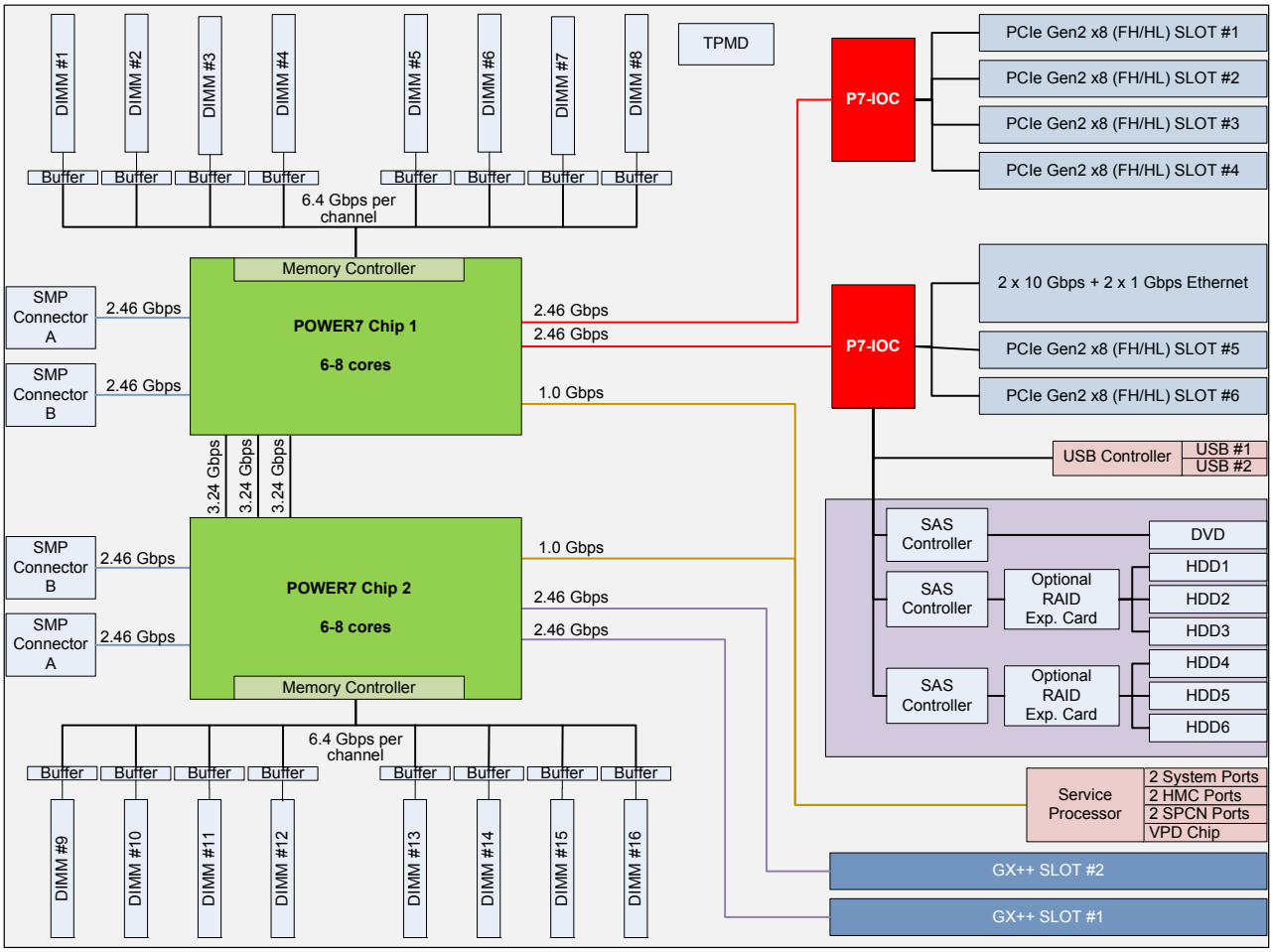


Figure 2-14 POWER 770 9117-MMC - Logic data flow

As shown in Figure 2-14, the 9117-MMC has a completely different design of the buses. It now has two P7-IOC chipsets by default and one of them is dedicated to four PCIe Gen2 slots. Moreover, both P7-IOCs are exclusively connected to the POWER7 Chip1 while both GX++ slots are exclusively connected to the POWER7 Chip2.

2.3.5 POWER 770 9117-MMD logical data flow

As of the time of this writing, the 9117-MMD model featuring the new POWER7+ processors has recently been announced and once again, along with new features and improvements, a new design has come, as we can see in Figure 2-15 on page 32.

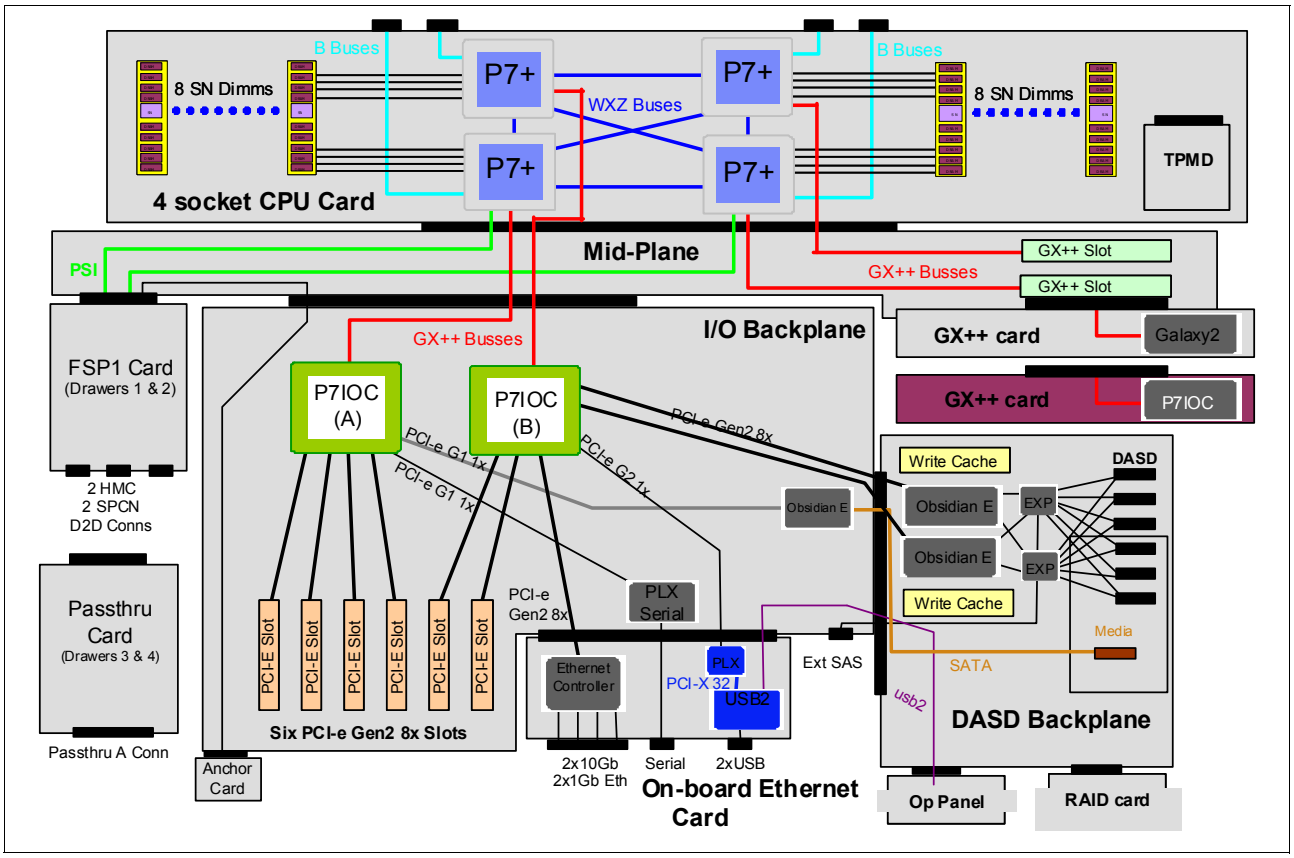


Figure 2-15 POWER 770 9117-MMD - Logic data flow

The 9117-MMD now includes four POWER7+ sockets and for the sake of this section, the major improvement in the design of this model is that the bus sharing has been reduced for the P7-IOC chipsets and the GX++ controllers by connecting each of these controllers to a different socket.

Table 2-4 POWER 770 comparison - Models 9117-MMC and 9117-MMD

POWER 770	9117-MMC	9117-MMD
Sockets per card	2	4
I/O Bus Controller	2x P7-IOC (20 Gbps), sharing the same chip.	2x P7-IOC (20 Gbps), independent chips.
GX++ slots (primary and secondary)	2x GX++ channels, sharing the same chip.	2x GX++ channels, independent chips.

Once again, this comparison (Table 2-4) illustrates the hardware evolution in the same type of machines.

2.3.6 Expansion units

Expansion units, as the name says, allow for capacity expansion by providing additional adapters and disk slots. These units are connected to the GX++ slots by using SPCN and 12x cables in specific configurations not covered in this book. These types of connections are designed for minimum latency but still, the more distant the adapters are from the processors, the more latency is present.

Adapters, cables and enclosures are available with Single Data Rate (SDR) and Double Data Rate (DDR) capacity. Table 2-5 shows the bandwidth differences between these two types.

Table 2-5 InfiniBand bandwidth table

Connection type	Bandwidth	Effective
Single Data Rate (SDR)	2.5 Gbps	2 Gbps
Double Data Rate (DDR)	5 Gbps	4 Gbps

In order to take full advantage of DDR, the three components (GX++ adapter, cable, and unit) must be equally capable of transmitting data at that same rate. If any of the components is SDR only, then the communication channel is limited to SDR speed.

Note: Detailed information about the several available Expansion Units can be obtained on the IBM Hardware Information Center at:

http://pic.dhe.ibm.com/infocenter/powersys/v3r1m5/index.jsp

2.3.7 Conclusions

First, the workload requirements must be known and should be established. Defining the required throughput on an initial installation is quite hard because usually you deal with several aspects that can make such analysis more difficult, but when expanding a system, that is data that can be very useful.

Starting at the choice of the proper machine type and model, all of the hardware characteristics should be carefully studied, deciding on adapter placement to obtain optimal results in accordance with the workload requirements. Even proper cabling has its importance.

At the time of partition deployment, assuming that the adapters have been distributed in the optimal way, assigning the correct slots to the partitions based on their physical placement is another important step to match the proper workload. Try to establish which partitions are more critical in regard to each component (processor, memory, disk, and network) and with that in mind plan their distribution and placement of resources.

Note: Detailed information about each machine type and further adapter placement documentation can be obtained in the *Technical Overview and Introduction* and *PCI Adapter Placement* documents, available on the IBM Hardware Information Center at:

http://pic.dhe.ibm.com/infocenter/powersys/v3r1m5/index.jsp

2.4 Continuous availability with CHARM

IBM continues to introduce new and advanced continuous Reliability, Availability, and Serviceability (RAS) functions in the IBM Power Systems to improve overall system availability. With advanced functions in fault resiliency, recovery, and redundancy design, the impact to the system from hardware failure has been significantly reduced.

With these system attributes, Power Systems continue to be leveraged for server consolidation. For clients experiencing rapid growth in computing needs, upgrading hardware capacity incrementally with limited disruption becomes an important system capability.

In this section we provide a brief overview of the CEC level functions and operations, called CEC Hot Add and Repair Maintenance (CHARM) functions and the POWER 770/780/795 servers that these functions are supporting. The CHARM functions provide the ability to upgrade system capacity and repair the CEC, or the heart of a large computer, without powering down the system.

The CEC hardware includes the processors, memory, I/O hubs (GX adapters), system clock, service processor, and associated CEC support hardware. The CHARM functions consist of special hardware design, service processor, hypervisor, and Hardware Management Console (HMC) firmware. The HMC provides the user interfaces for the system administrator and System Service Representative (SSR) to perform the tasks for the CHARM operation.

2.4.1 Hot add or upgrade

In this section we describe how to work with nodes during add or upgrade operations.

Hot node add

This function allows the system administrator to add a node to a system to increase the processor, memory, and I/O capacity of the system. After the physical node installation, the system firmware activates and integrates the new node hardware into the system.

The new processor and memory resources are available for the creation of new partitions or to be dynamically added to existing partitions. After the completion of the node add operation, additional I/O expansion units can be attached to the new GX adapters in the new node in a separate concurrent I/O expansion unit add operation.

Hot node upgrade (memory)

This function enables an SSR to increase the memory capacity in a system by adding additional memory DIMMs to a node, or upgrading (exchanging) existing memory with higher-capacity memory DIMMs.

The system must have two or more nodes to utilize the hot node upgrade function. Since the node that is being upgraded is functional and possibly running workloads prior to starting the hot upgrade operation, the system administrator uses the “Prepare for Hot Repair or Upgrade” utility, system management tools and operating system (OS) tools to prepare the node for evacuation.

During the hot node upgrade operation, the system firmware performs the node evacuation by relocating the workloads from the target node to other nodes in the system and logically isolating the resources in the target node.

It then deactivates and electrically isolates the node to allow the removal of the node for the upgrade. After the physical node upgrade, the system firmware activates and integrates the node hardware into the system with additional memory.

After the completion of a hot node upgrade operation, the system administrator then restores the usage of processor, memory, and I/O resources, including redundant I/O configurations if present. The new memory resources are available for the creation of new partitions or to be dynamically added to existing partitions by the system administrator using dynamic LPAR operations.

Concurrent GX adapter add

This function enables an SSR to add a GX adapter to increase the I/O capacity of the system. The default settings allow one GX adapter to be added concurrently with a POWER 770/780

system and two GX adapters to be added concurrently to a POWER 795 system, without planning for a GX memory reservation.

After the completion of the concurrent GX adapter add operation, I/O expansion units can be attached to the new GX adapter through separate concurrent I/O expansion unit add operations.

2.4.2 Hot repair

In this section, we describe the hot repair functionality of the IBM POWER Systems servers.

Hot node repair

This function allows an SSR to repair defective hardware in a node of a system while the system is powered on.

The system must have two or more nodes to utilize the hot node repair function. Since the node that is being repaired may be fully or partially functional and running workloads prior to starting the hot repair operation, the system administrator uses the “Prepare for Hot Repair or Upgrade” utility, system management tools and OS tools to prepare the system. During the hot node repair operation, the system firmware performs the node evacuation by relocating workloads in the target node to other nodes in the system and logically isolating the resources in the target node.

It then deactivates and electrically isolates the node to allow the removal of the node for repair. After the physical node repair, the system firmware activates and integrates the node hardware back into the system.

Hot GX adapter repair

This function allows an SSR to repair a defective GX adapter in the system. The GX adapter may still be in use prior to starting the hot repair operation, so the system administrator uses the “Prepare for Hot Repair or Upgrade” utility and OS tools to prepare the system.

During the hot GX repair operation, the system firmware logically isolates the resources associated with or dependent on the GX adapter, then deactivates and electrically isolates the GX adapter to allow physical removal for repair.

After the completion of the hot repair operation, the system administrator restores the usage of I/O resources, including redundant I/O configurations if present.

Concurrent system controller repair

The concurrent System Controller (SC) repair function allows an SSR to concurrently replace a defective SC card. The target SC card can be fully or partially functional and in the primary or backup role, or it can be in a nonfunctional state.

After the repair, the system firmware activates and integrates the new SC card into the system in the backup role.

2.4.3 Prepare for Hot Repair or Upgrade utility

The Prepare for Hot Repair or Upgrade (PHRU) utility is a tool on the HMC used by the system administrator to prepare the system for a hot node repair, hot node upgrade, or hot GX adapter repair operation.

Among other things, the utility identifies resources that are in use by operating systems and must be deactivated or released by the operating system. Based on the PHRU utility's output, the system administrator may reconfigure or vary off impacted I/O resources using operating system tools, remove reserved processing units from shared processor pools, reduce active partitions' entitled processor and/or memory capacity using dynamic LPAR operations, or shut down low priority partitions.

This utility also runs automatically near the beginning of a hot repair or upgrade procedure to verify that the system is in the proper state for the procedure. The system firmware does not allow the CHARM operation to proceed unless the necessary resources have been deactivated and/or made available by the system administrator and the configuration supports it.

Table 2-6 summarizes the usage of the PHRU utility based on specific CHARM operations and expected involvement of the system administrator and service representative.

Table 2-6 PHRU utility usage

CHARM operation	***Minimum # of nodes to use operation***	***PHRU usage***	***System administrator***	***Service representative***
Hot Node Add	1	No	Planning only	Yes
Hot Node Repair	2	Yes	Yes	Yes
Hot Node Upgrade (memory)	2	Yes	Yes	Yes
Concurrent GX Adapter Add	1	No	Planning only	Yes
Hot GX Adapter Repair	1	Yes	Yes	Yes
Concurrent System Controller Repair	1	No	Planning only	Yes

2.4.4 System hardware configurations

To obtain the maximum system and partition availability benefits from the CEC Hot Add and Repair Maintenance functions, follow these best practices and guidelines for system hardware configuration:

- Request the free pre-sales “I/O Optimization for RAS” services offering to ensure that your system configuration is optimized to minimize disruptions when using CHARM.
- The system should have spare processor and memory capacity to allow a node to be taken offline for hot repair or upgrade with minimum impact and memory capacity to take the node offline.
- All critical I/O resources should be configured using an operating system multipath I/O redundancy configuration.
- Redundant I/O paths should be configured through different nodes and GX adapters.
- All logical partitions should have RMC network connections with the HMCs.
- The HMC should be configured with redundant service networks with both service processors in the CEC.

Note: Refer to the IBM POWER 770/780 and 795 Servers CEC Hot Add and Repair Maintenance Technical Overview at:

ftp://public.dhe.ibm.com/common/ssi/ecm/en/pow03058usen/POW03058USEN.PDF

2.5 Power management

One of the new features introduced with the POWER6 family of servers was Power Management. This feature is part of the IBM EnergyScale™ technology featured on the POWER6 processor. IBM EnergyScale provides functions to help monitor and optimize hardware power and cooling usage. While the complete set of IBM EnergyScale is facilitated through the IBM Systems Director Active Energy Manager™, the Power Management option can also be enabled in isolation from the HMC or ASMI interface.

The feature is known on the HMC GUI as Power Management and Power Saver mode on the ASMI GUI; while in IBM EnergyScale terminology it is referred to as Static Power Saver Mode (differentiating it from Dynamic Power Saver Mode). Using the HMC it is also possible to schedule a state change (to either enable or disable the feature). This would allow the feature to be enabled over a weekend, but disabled on the following Monday morning.

For a complete list of features provided by IBM EnergyScale on POWER6, refer to the white paper:

http://www-03.ibm.com/systems/power/hardware/whitepapers/energyscale.html

With the advent of the POWER7 platform, IBM EnergyScale was expanded adding increased granularity along with additional reporting and optimization features. For a complete list of features provided by IBM EnergyScale on POWER7, refer to this white paper:

http://www-03.ibm.com/systems/power/hardware/whitepapers/energyscale7.html

On both POWER6 and POWER7 platforms, the Power Management feature is disabled by default. Enabling this feature reduces processor voltage and therefore clock frequency; this achieves lower power usage for the processor hardware and therefore system as a whole. For smaller workloads the impact from the reduction in clock frequency will be negligible. However, for larger or more stressful workloads the impact would be greater.

The amount by which processor clock frequency is reduced differs with machine model and installed processor feature code. But for a given combination of model and processor the amount is fixed. For detailed tables listing all the support combinations refer to Appendix I in either of the previously mentioned white papers.

Note: When enabled, certain commands can be used to validate the state and impact of the change. Care must be taken to not misinterpret output from familiar commands.

The output from the **lparstat** command lists the current status, as shown at the end of the output in Example 2-17.

Example 2-17 Running lparstat

```
# lparstat -i
Node Name                                  : p750s1aix6
Partition Name                             : 750_1_AIX6
Partition Number                           : 15
Type                                       : Shared-SMT-4
```

```
Mode                                        : Uncapped
Entitled Capacity                           : 1.00
Partition Group-ID                          : 32783
Shared Pool ID                              : 0
Online Virtual CPUs                         : 8
Maximum Virtual CPUs                        : 8
Minimum Virtual CPUs                        : 1
Online Memory                               : 8192 MB
Maximum Memory                              : 16384 MB
Minimum Memory                              : 4096 MB
Variable Capacity Weight                    : 128
Minimum Capacity                            : 0.50
Maximum Capacity                            : 4.00
Capacity Increment                          : 0.01
Maximum Physical CPUs in system             : 16
Active Physical CPUs in system              : 16
Active CPUs in Pool                         : 10
Shared Physical CPUs in system              : 10
Maximum Capacity of Pool                    : 1000
Entitled Capacity of Pool                   : 1000
Unallocated Capacity                        : 0.00
Physical CPU Percentage                     : 12.50%
Unallocated Weight                          : 0
Memory Mode                                 : Dedicated
Total I/O Memory Entitlement                : -
Variable Memory Capacity Weight             : -
Memory Pool ID                              : -
Physical Memory in the Pool                 : -
Hypervisor Page Size                        : -
Unallocated Variable Memory Capacity Weight: -
Unallocated I/O Memory entitlement          : -
Memory Group ID of LPAR                     : -
Desired Virtual CPUs                        : 8
Desired Memory                              : 8192 MB
Desired Variable Capacity Weight            : 128
Desired Capacity                            : 1.00
Target Memory Expansion Factor              : -
Target Memory Expansion Size                : -
Power Saving Mode                           : Disabled
```

In the case where the feature is enabled from the HMC or ASMI, the difference in output from the same command is shown in Example 2-18.

Example 2-18 lparstat output with power saving enabled

```
Desired Virtual CPUs                        : 8
Desired Memory                              : 8192 MB
Desired Variable Capacity Weight            : 128
Desired Capacity                            : 1.00
Target Memory Expansion Factor              : -
Target Memory Expansion Size                : -
Power Saving Mode                           : Static Power Savings
```

Once enabled, the new operational clock speed is not always apparent from AIX. Certain commands still report the default clock frequency, while others report the actual current frequency. This is because some commands are simply retrieving the default frequency stored in the ODM. Example 2-19 represents output from the **lsconf** command.

Example 2-19 Running lsconf

```
# lsconf
System Model: IBM,8233-E8B
Machine Serial Number: 10600EP
Processor Type: PowerPC_Power7
Processor Implementation Mode: Power 7
Processor Version: PV_7_Compat
Number Of Processors: 8
Processor Clock Speed: 3300 MHz
```

The same MHz is reported as standard by the **pmcycles** command as shown in Example 2-20.

Example 2-20 Running pmcycles

```
# pmcycles
This machine runs at 3300 MHz
```

However, using the **-M** parameter instructs **pmcycles** to report the current frequency as shown in Example 2-21.

Example 2-21 Running pmcycles -M

```
# pmcycles -M
This machine runs at 2321 MHz
```

The **lparstart** command represents the reduced clock speed as a percentage, as reported in the %nsp (nominal speed) column in Example 2-22.

Example 2-22 Running lparstat with Power Saving enabled

```
# lparstat -d 2 5

System configuration: type=Shared mode=Uncapped smt=4 lcpu=32 mem=8192MB psize=16
ent=1.00

%user  %sys  %wait  %idle physc %entc  %nsp
----- ----- ------ ------ ----- ----- -----
 94.7   2.9    0.0    2.3  7.62 761.9    70
 94.2   3.1    0.0    2.7  7.59 759.1    70
 94.5   3.0    0.0    2.5  7.61 760.5    70
 94.6   3.0    0.0    2.4  7.64 763.7    70
 94.5   3.0    0.0    2.5  7.60 760.1    70
```

In this example, the current 2321 MHz is approximately 70% of the default 3300 MHz.

Note: It is important to understand how to query the status of Power Saving mode on a given LPAR or system. Aside from the reduced clock speed, it can influence or reduce the effectiveness of other PowerVM optimization features.

For example, enabling Power Saving mode also enables virtual processor management in a dedicated processor environment. If Dynamic System Optimizer (6.1.2, “IBM AIX Dynamic System Optimizer” on page 288) is also active on the LPAR, it is unable to leverage its cache and memory affinity optimization routines.

3

IBM Power Systems virtualization

In this chapter, we describe some of the features and tools to optimize virtualization on POWER Systems running AIX. This includes PowerVM components and AIX Workload Partitions (WPAR). Virtualization when deployed correctly provides the best combination of performance and utilization of a system. Relationships and dependencies between some of the components need to be understood and observed. It is critical that the different elements are correctly configured and tuned to deliver optimal performance.

We discuss the following topics in this chapter:

- Optimal logical partition (LPAR) sizing
- Active Memory Expansion
- Active Memory Sharing (AMS)
- Active Memory Deduplication (AMD)
- Virtual I/O Server (VIOS) sizing
- Using Virtual SCSI, Shared Storage Pools and N-Port Virtualization
- Optimal Shared Ethernet Adapter configuration
- AIX Workload Partition implications, performance and suggestions
- LPAR suspend and resume best practices

3.1 Optimal logical partition (LPAR) sizing

A common theme throughout this book is to understand your workload and size your logical partitions appropriately. In this section we focus on some of the processor and memory settings available in the LPAR profile and provide guidance on how to set them to deliver optimal performance.

This section is divided into two parts, processor and memory.

Processor

There are a number of processor settings available. Some have more importance than others in terms of performance. Table 3-1 provides a summary of the processor settings available in the LPAR profile, a description of each, and some guidance on what values to consider.

Table 3-1 Processor settings in LPAR profile

Setting	Description	Recommended value
Minimum Processing Units	This is the minimum amount of processing units that must be available for the LPAR to be activated. Using DLPAR, processing units can be removed to a minimum of this value.	This value should be set to the minimum number of processing units that the LPAR would realistically be assigned.
Desired Processing Units	This is the desired amount of processing units reserved for the LPAR; this is also known as the LPAR's entitled capacity (EC).	This value should be set to the average utilization of the LPAR during peak workload.
Maximum Processing Units	This is the maximum amount of processing units that can be added to the LPAR using a DLPAR operation.	This value should be set to the maximum number of processing units that the LPAR would realistically be assigned.
Minimum Virtual Processors	This is the minimum amount of virtual processors that can be assigned to the LPAR with DLPAR.	This value should be set to the minimum number of virtual processors that the LPAR would be realistically assigned.
Desired Virtual Processors	This is the desired amount of virtual processors that will be assigned to the LPAR when it is activated. This is also referred to as virtual processors (VPs).	This value should be set to the upper limit of processor resources utilized during peak workload.
Maximum Virtual Processors	This the maximum amount of virtual processors that can be assigned to the LPAR using a DLPAR operation.	This value should be set to the maximum number of virtual processors that the LPAR would be realistically assigned.

Setting	Description	Recommended value
Sharing Mode	Uncapped LPARs can use processing units that are not being used by other LPARs, up to the number of virtual processors assigned to the uncapped LPAR. Capped LPARs can use only the number of processing units that are assigned to them. In this section we focus on uncapped LPARs.	For LPARs that will consume processing units above their entitled capacity, it is recommended to have the LPAR configured as uncapped.
Uncapped Weight	When contending for shared resources with other LPARs, this is the priority that this logical partition has when contention for shared virtual resources exists.	This is the relative weight that the LPAR will have during resource contention. This value should be set based on the importance of the LPAR compared to other LPARs in the system. It is suggested that the VIO servers have highest weight.

There are situations where it is required in a Power system to have multiple shared processor pools. A common reason for doing this is for licensing constraints where licenses are by processor, and there are different applications running on the same system. When this is the case, it is important to size the shared processor pool to be able to accommodate the peak workload of the LPARs in the shared pool.

In addition to dictating the maximum number of virtual processors that can be assigned to an LPAR, the entitled capacity is a very important setting that must be set correctly. The best practice for setting this is to set it to the average processor utilization during peak workload. The sum of the entitled capacity assigned to all the LPARs in a Power system should not be more than the amount of physical processors in the system or shared processor pool.

The virtual processors in an uncapped LPAR dictate the maximum amount of idle processor resources that can be taken from the shared pool when the workload goes beyond the capacity entitlement. The number of virtual processors should not be sized beyond the amount of processor resources required by the LPAR, and it should not be greater than the total amount of processors in the Power system or in the shared processor pool.

Figure 3-1 on page 44 shows a sample workload with the following characteristics:

- The system begins its peak workload at 8:00 am.
- The system's peak workload stops at around 6:30 pm.
- The ideal entitled capacity for this system is 25 processors, which is the average utilization during peak workload.
- The ideal number of virtual processors is 36, which is the maximum amount of virtual processors used during peak workload.

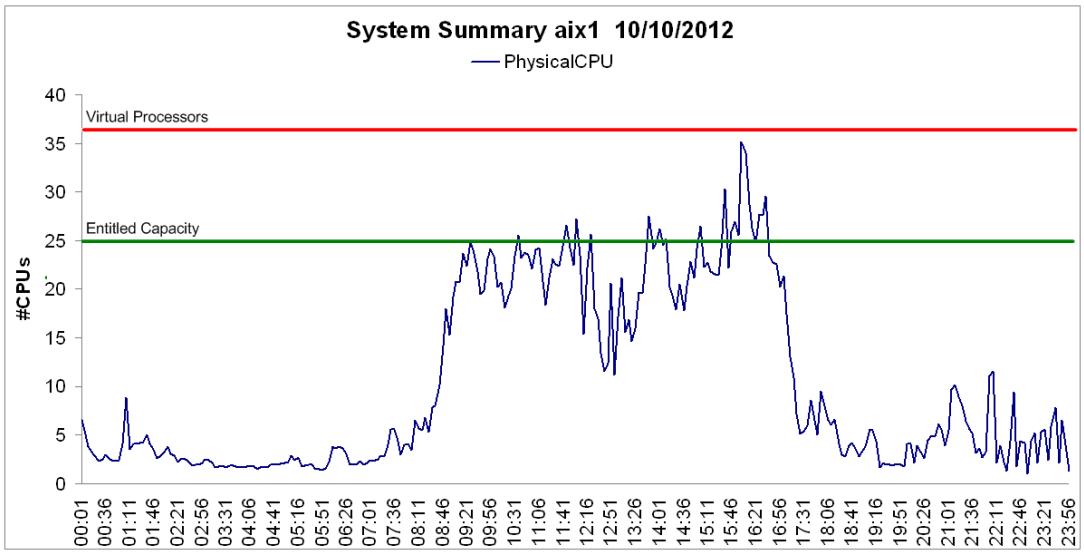


Figure 3-1 Graph of a workload over a 24-hour period

For LPARs with dedicated processors (these processors are not part of the shared processor pool), there is an option to enable this LPAR after it is activated for the first time to donate idle processing resources to the shared pool. This can be useful for LPARs with dedicated processors that do not always use 100% of the assigned processing capacity.

Figure 3-2 demonstrates where to set this setting in an LPAR's properties. It is important to note that sharing of idle capacity when the LPAR is not activated is enabled by default. However, the sharing of idle capacity when the LPAR is activated is not enabled by default.

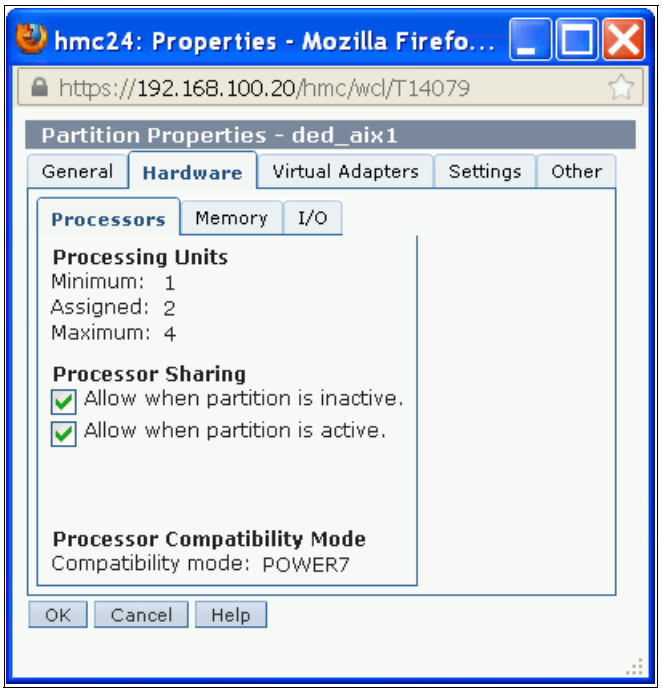


Figure 3-2 Dedicated LPAR sharing processing units

There are performance implications in the values you choose for the entitled capacity and the number of virtual processors assigned to the partition. These are discussed in detail in the following sections:

- “Optimizing the LPAR resource placement” on page 18.
- “Simultaneous multithreading (SMT)” on page 120 and “Processor folding” on page 123.

We were able to perform a simple test to demonstrate the implications of sizing the entitled capacity of an AIX LPAR. The first test is shown in Figure 3-3 and the following observations were made:

- The entitled capacity (EC) is 6.4 and the number of virtual processors is 64. There are 64 processors in the POWER7 780 that this test was performed on.
- When the test was executed, due to the time taken for the AIX scheduler to perform processor unfolding, the time taken for the workload to have access to the required cores was 30 seconds.

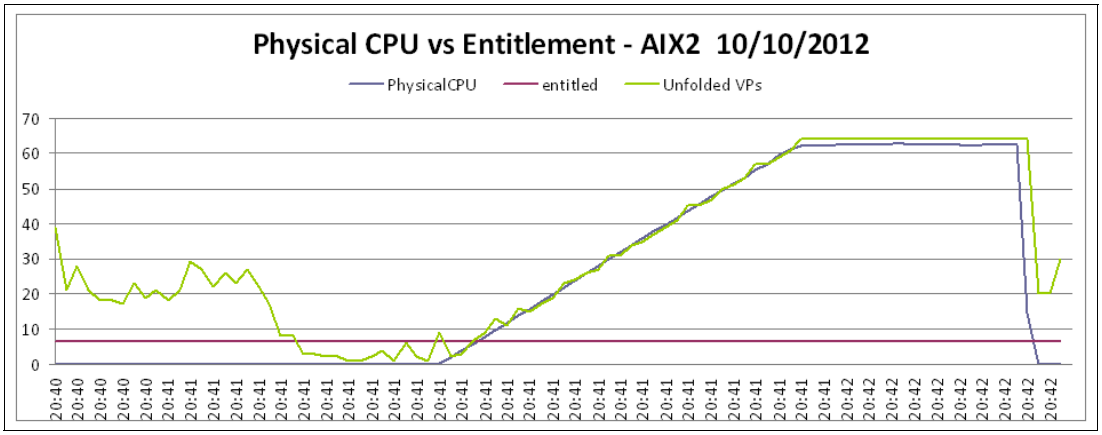


Figure 3-3 Folding effect with EC set too low

The same test was performed again, with the entitled capacity raised from 6.4 processing units to 50 processing units. The second test is shown in Figure 3-4 on page 46 and the following observations were made:

- The entitled capacity is 50 and the number of virtual processors is still 64.
- The amount of processor unfolding the hypervisor had to perform was significantly reduced.
- The time taken for the workload to access the processing capacity went from 30 seconds to 5 seconds.

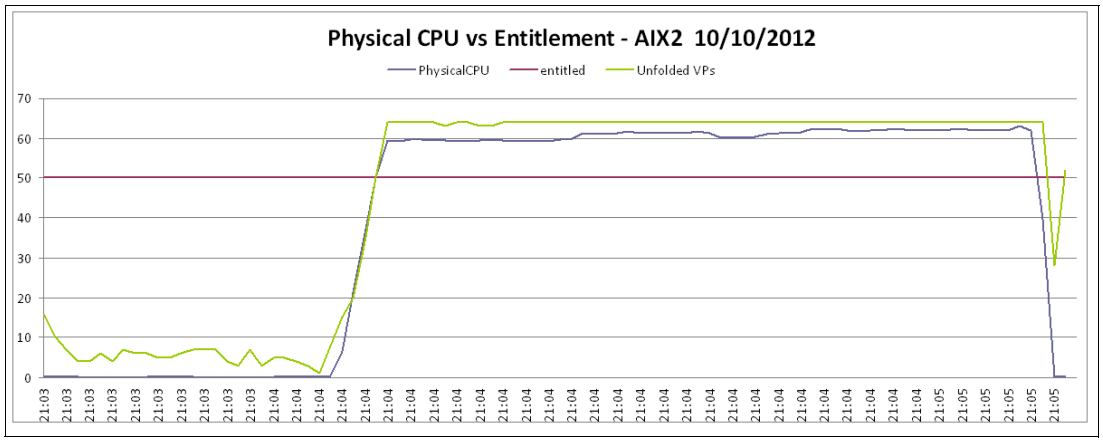


Figure 3-4 Folding effect with EC set higher; fasten your seat belts

The conclusion of the test: we found that tuning the entitled capacity correctly in this case provided us with a 16% performance improvement, simply due to the unfolding process. Further gains would also be possible related to memory access due to better LPAR placement, because there is an affinity reservation for the capacity entitlement.

Memory

Sizing memory is also an important consideration when configuring an AIX logical partition.

Table 3-2 provides a summary of the memory settings available in the LPAR profile.

Table 3-2 Memory settings in LPAR profile

Setting	Description	Recommended value
Minimum memory	This is the minimum amount of memory that must be available for the LPAR to be activated. Using DLPAR, memory can be removed to a minimum of this value.	This value should be set to the minimum amount of memory that the LPAR would realistically be assigned.
Desired memory	This is the amount of memory assigned to the LPAR when it is activated. If this amount is not available the hypervisor will assign as much available memory as possible to get close to this number.	This value should reflect the amount of memory that is assigned to this LPAR under normal circumstances.
Maximum memory	This is the maximum amount of memory that can be added to the LPAR using a DLPAR operation.	This value should be set to the maximum amount of memory that the LPAR would realistically be assigned.
AME expansion factor	See 3.2, “Active Memory Expansion” on page 48.	See 3.2, “Active Memory Expansion” on page 48

When sizing the desired amount of memory, it is important that this amount will satisfy the workload’s memory requirements. Adding more memory using dynamic LPAR can have an effect on performance due to affinity. This is described in 2.2.3, “Verifying processor memory placement” on page 14.

Another factor to consider is the maximum memory assigned to a logical partition. This affects the hardware page table (HPT) of the POWER system. The HPT is the amount of memory assigned from the memory reserved by the POWER hypervisor. If the maximum memory for an LPAR is set very high, the amount of memory required for the HPT increases, causing a memory overhead on the system.

On POWER5, POWER6 and POWER7 systems the HPT is calculated by the following formula, where the sum of all the LPAR's maximum memory is divided by a factor of 64 to calculate the HPT size:

```
HPT = sum_of_lpar_max_memory / 64
```

On POWER7+ systems the HPT is calculated using a factor of 64 for IBM i and any LPARs using Active Memory Sharing. However, for AIX and Linux LPARs the HPT is calculated using a factor of 128.

Example 3-1demonstrates how to display the default HPT ratio from the HMC command line for the managed system `750_1_SN106011P`, which is a POWER7 750 system.

Example 3-1 Display the default HPT ratio on a POWER7 system

```
hscroot@hmc24:~> lshwres -m 750_1_SN106011P -r mem --level sys -F default_hpt_ratios
1:64
hscroot@hmc24:~>
```

Figure 3-5 provides a sample of the properties of a POWER7 750 system. The amount of memory installed in the system is 256 GB, all of which is activated.

The memory allocations are as follows:

- 200.25 GB of memory is not assigned to any LPAR.
- 52.25 GB of memory is assigned to LPARs currently running on the system.
- 3.50 GB of memory is reserved for the hypervisor.

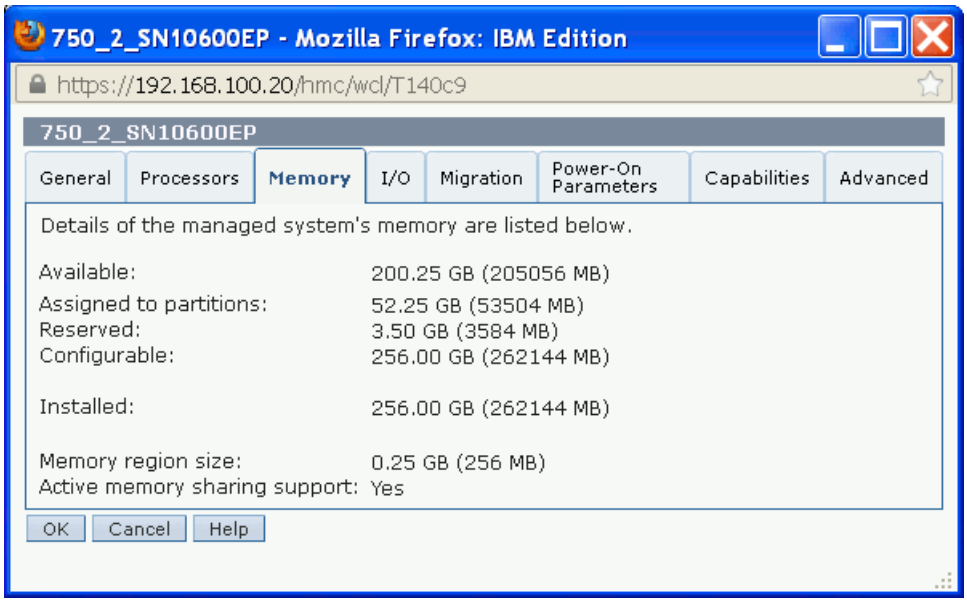


Figure 3-5 Memory assignments for a managed system

Important: Do not size your LPAR's maximum memory too large, because there will be an increased amount of reserved memory for the HPT.

3.2 Active Memory Expansion

Active Memory Expansion (AME) is an optional feature of IBM POWER7 and POWER7+ systems for expanding a system's effective memory capacity by performing memory compression. AME is enabled on a per-LPAR basis. Therefore, AME can be enabled on some or all of the LPARs on a Power system. POWER7 systems use LPAR processor cycles to perform the compression in software.

AME enables memory to be allocated beyond the amount that is physically installed in the system, where memory can be compressed on an LPAR and the memory savings can be allocated to another LPAR to improve system utilization, or compression can be used to oversubscribe memory to potentially improve performance.

AME is available on POWER7 and POWER7+ systems with AIX 6.1 Technology Level 4 and AIX 7.1 Service Pack 2 and above.

Active Memory Expansion is not ordered as part of any PowerVM edition. It is licensed as a separate feature code, and can be ordered with a new system, or added to a system at a later time. Table 3-3 provides the feature codes to order AME at the time of writing.

Table 3-3 AME feature codes

Feature code	Description
4795	Active Memory Expansion Enablement POWER 710 and 730
4793	Active Memory Expansion Enablement POWER 720
4794	Active Memory Expansion Enablement POWER 740
4792	Active Memory Expansion Enablement POWER 750
4791	Active Memory Expansion Enablement POWER 770 and 780[a]
4790	Active Memory Expansion Enablement POWER 795

a. This includes the Power 770+ and Power 780+ server models.

In this section we discuss the use of active memory expansion compression technology in POWER7 and POWER7+ systems. A number of terms are used in this section to describe AME. Table 3-4 provides a list of these terms and their meaning.

Table 3-4 Terms used in this section

Term	Meaning
LPAR true memory	The LPAR true memory is the amount of real memory assigned to the LPAR before compression.
LPAR expanded memory	The LPAR expanded memory is the amount of memory available to an LPAR after compression. This is the amount of memory an application running on AIX will see as the total memory inside the system.

Term	Meaning
Expansion factor	To enable AME, there is a single setting that must be set in the LPAR's profile. This is the expansion factor, which dictates the target memory capacity for the LPAR. This is calculated by this formula: LPAR_EXPANDED_MEM = LPAR_TRUE_MEM * EXP_FACTOR
Uncompressed pool	When AME is enabled, the operating system's memory is broken up into two pools, an uncompressed pool and a compressed pool. The uncompressed pool contains memory that is uncompressed and available to the application.
Compressed pool	The compressed pool contains memory pages that are compressed by AME. When an application needs to access memory pages that are compressed, AME uncompresses them and moves them into the uncompressed pool for application access. The size of the pools will vary based on memory access patterns and the memory compression factor.
Memory deficit	When an LPAR is configured with an AME expansion factor that is too high based on the compressibility of the workload. When the LPAR cannot reach the LPAR expanded memory target, the amount of memory that cannot fit into the memory pools is known as the memory deficit, which might cause paging activity. The expansion factor and the true memory can be changed dynamically, and when the expansion factor is set correctly, no memory deficit should occur.

Figure 3-6 on page 50 provides an overview of how AME works. The process of memory access is such that the application is accessing memory directly from the uncompressed pool. When memory pages that exist in the compressed pool are to be accessed, they are moved into the uncompressed pool for access. Memory that exists in the uncompressed pool that is no longer needed for access is moved into the compressed pool and subsequently compressed.

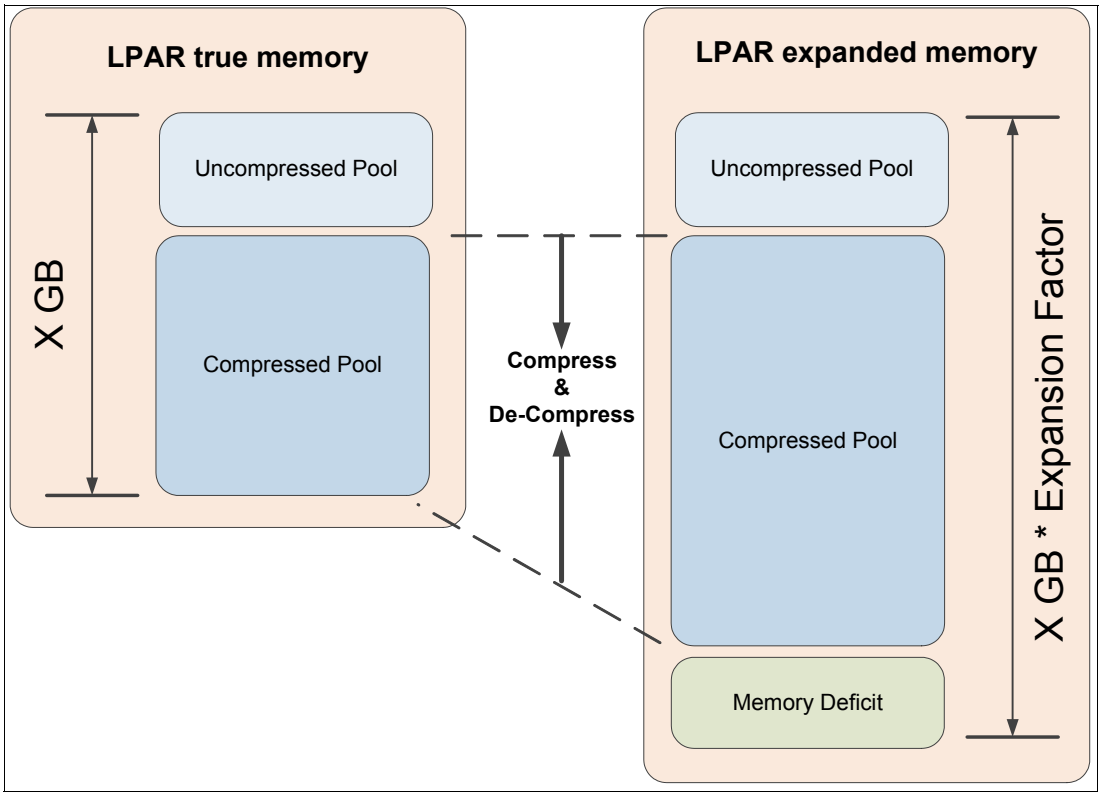


Figure 3-6 Active Memory Expansion overview

The memory gain from AME is determined by the expansion factor. The minimum expansion factor is 1.0 meaning no compression, and the maximum value is 10.0 meaning 90% compression.

Each expansion value has an associated processor overhead dependent on the type of workload. If the expansion factor is high, then additional processing is required to handle the memory compression and decompression. The kernel process in AIX is named **cmemd**, which performs the AME compression and decompression. This process can be monitored from topas or nmon to view its processor usage. The AME planning tool **amepat** covered in 3.2.2, “Sizing with the active memory expansion planning tool” on page 52 describes how to estimate and monitor the **cmemd** processor usage.

The AME expansion factor can be set in increments of 0.01. Table 3-5 gives an overview of some of the possible expansion factors to demonstrate the memory gains associated with the different expansion factors.

Note: These are only a subset of the expansion factors. The expansion factor can be set anywhere from 1.00 to 10.00 increasing by increments of 0.01.

Table 3-5 Sample expansion factors and associated memory gains

Expansion factor	Memory gain
1.0	0%
1.2	20%
1.4	40%

Expansion factor	Memory gain
1.6	60%
1.8	80%
2.0	100%
2.5	150%
3.0	200%
3.5	250%
4.0	300%
5.0	400%
10.0	900%

3.2.1 POWER7+ compression accelerator

A new feature in POWER7+ processors is the nest accelerator (NX). The nest accelerator contains accelerators also known as coprocessors, which are shared resources used by the hypervisor for the following purposes:

- Encryption for JFS2 Encrypted file systems
- Encryption for standard cryptography APIs
- Random number generation
- AME hardware compression

Each POWER7+ chip contains a single NX unit and multiple cores, depending on the model, and these cores all share the same NX unit. The NX unit allows some of the AME processing to be off-loaded to significantly reduce the amount of processor overhead involved in compression and decompression. Where multiple LPARs are accessing the NX unit for compression at once, the priority is on a first in first out (FIFO) basis.

As with the relationship between processor and memory affinity, optimal performance is achieved when the physical memory is in the same affinity domain as the NX unit. AIX creates compressed pools on affinity domain boundaries and makes the best effort to allocate from the local memory pool.

AIX automatically leverages hardware compression for AME when available. Configuring AME on POWER7+ is achieved by following exactly the same process as on POWER7. However, to leverage hardware compression AIX 6.1 Technology Level 8 or AIX 7.1 Technology Level 2 or later are required.

The active memory expansion planning tool **amepat** has also been updated as part of these same AIX Technology Levels to suggest compression savings and associated processing overhead using hardware compression. Example 3-4 on page 54 illustrates this **amepat** enhancement.

Figure 3-7 on page 52 demonstrates how to confirm that hardware compression is enabled on a POWER7+ system.

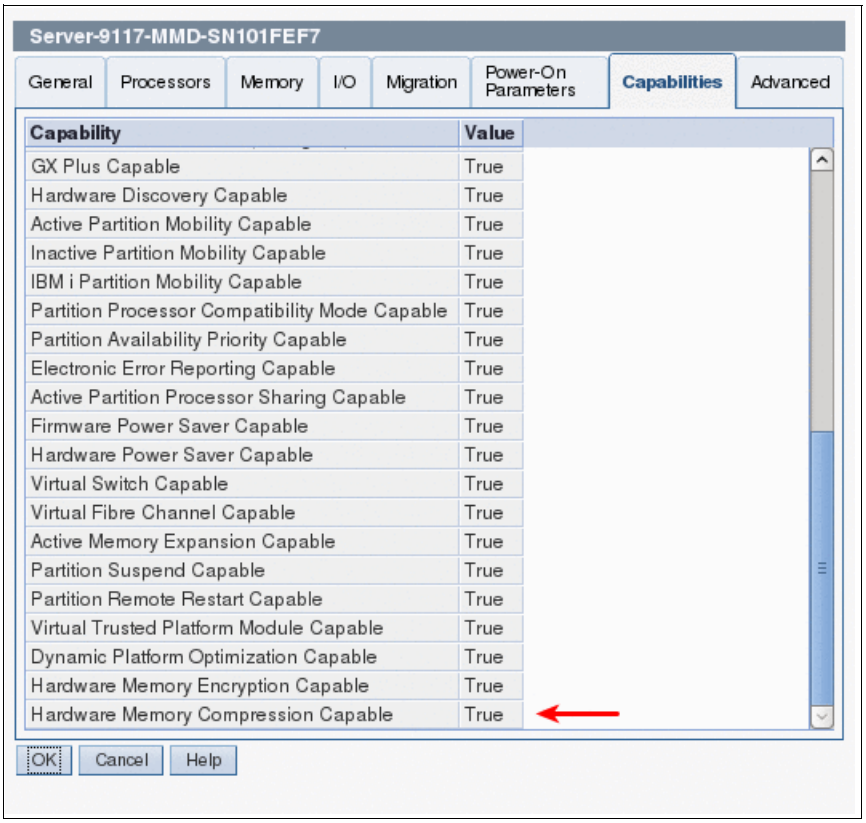


Figure 3-7 Confirming that hardware compression is available

Note: The compression accelerator only handles the compression of memory in the compressed pool. The LPAR's processor is still used to manage the moving of memory between the compressed and the uncompressed pool. The benefit of the accelerator is dependent on your workload characteristics.

3.2.2 Sizing with the active memory expansion planning tool

The active memory expansion planning tool **amepat** is a utility that should be run on the system on which you are evaluating the use of AME. When executed, **amepat** records system configuration and various performance metrics to provide guidance on possible AME configurations, and their processing impact. The tool should be run prior to activating AME, and run again after activating AME to continually evaluate the memory configuration.

The **amepat** tool provides a report with possible AME configurations and a recommendation based on the data it collected during the time it was running.

For best results, it is best to consider the following points:

- Run **amepat** during peak workload.
- Ensure that you run **amepat** for the full duration of the peak workload.
- The tool can be run in the foreground, or in recording mode.
- It is best to run the tool in recording mode, so that multiple configurations can be evaluated against the record file rather than running the tool in the foreground multiple times.

- Once the tool has been run once, it is reconnected running it again with a range of expansion factors to find the optimal value.
- Once AME is active, it is suggested to continue running the tool, because the workload may change resulting in a new expansion factor being recommended by the tool.

The amepat tool is available as part of AIX starting at AIX 6.1 Technology Level 4 Service Pack 2.

Example 3-2 demonstrates running **amepat** with the following input parameters:

- Run the report in the foreground.
- Run the report with a starting expansion factor of 1.20.
- Run the report with an upper limit expansion factor of 2.0.
- Include only POWER7 software compression in the report.
- Run the report to monitor the workload for 5 minutes.

Example 3-2 Running amepat with software compression

```
root@aix1:/ # amepat -e 1.20:2.0:0.1 -O proc=P7 5

Command Invoked                : amepat -e 1.20:2.0:0.1 -O proc=P7 5

Date/Time of invocation        : Tue Oct  9 07:33:53 CDT 2012
Total Monitored time           : 7 mins 21 secs
Total Samples Collected        : 3

System Configuration:
---------------------
Partition Name                 : aix1
Processor Implementation Mode  : Power7 Mode
Number Of Logical CPUs         : 16
Processor Entitled Capacity    : 2.00
Processor Max. Capacity        : 4.00
True Memory                    : 8.00 GB
SMT Threads                    : 4
Shared Processor Mode          : Enabled-Uncapped
Active Memory Sharing          : Disabled
Active Memory Expansion        : Enabled
Target Expanded Memory Size    : 8.00 GB
Target Memory Expansion factor : 1.00

System Resource Statistics:               Average             Min             Max
---------------------------               -----------       -----------       -----------
CPU Util (Phys. Processors)               1.41 [ 35%]       1.38 [ 35%]       1.46 [ 36%]
Virtual Memory Size (MB)                  5665 [ 69%]       5665 [ 69%]       5665 [ 69%]
True Memory In-Use (MB)                   5880 [ 72%]       5880 [ 72%]       5881 [ 72%]
Pinned Memory (MB)                        1105 [ 13%]       1105 [ 13%]       1105 [ 13%]
File Cache Size (MB)                       199 [  2%]        199 [  2%]        199 [  2%]
Available Memory (MB)                     2303 [ 28%]       2303 [ 28%]       2303 [ 28%]

AME Statistics:                           Average             Min             Max
---------------                           -----------       -----------       -----------
AME CPU Usage (Phy. Proc Units)           0.00 [  0%]       0.00 [  0%]       0.00 [  0%]
Compressed Memory (MB)                       0 [  0%]          0 [  0%]          0 [  0%]
Compression Ratio                          N/A

Active Memory Expansion Modeled Statistics        :
------------------------------------------
```

```
Modeled Implementation   : Power7
Modeled Expanded Memory Size :   8.00 GB
Achievable Compression ratio   :0.00

Expansion    Modeled True      Modeled             CPU Usage
Factor       Memory Size       Memory Gain         Estimate
---------    ------------      ------------------  -----------
     1.00          8.00 GB         0.00 KB [  0%]   0.00 [  0%] << CURRENT CONFIG
     1.28          6.25 GB         1.75 GB [ 28%]   0.41 [ 10%]
     1.40          5.75 GB         2.25 GB [ 39%]   1.16 [ 29%]
     1.46          5.50 GB         2.50 GB [ 45%]   1.54 [ 39%]
     1.53          5.25 GB         2.75 GB [ 52%]   1.92 [ 48%]
     1.69          4.75 GB         3.25 GB [ 68%]   2.68 [ 67%]
     1.78          4.50 GB         3.50 GB [ 78%]   3.02 [ 75%]
     1.89          4.25 GB         3.75 GB [ 88%]   3.02 [ 75%]
     2.00          4.00 GB         4.00 GB [100%]   3.02 [ 75%]

Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 6.25 GB and to configure a memory expansion factor
of 1.28.  This will result in a memory gain of 28%. With this
configuration, the estimated CPU usage due to AME is approximately 0.41
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 1.86 physical processors.

NOTE: amepat's recommendations are based on the workload's utilization level
during the monitored period. If there is a change in the workload's utilization
level or a change in workload itself, amepat should be run again.

The modeled Active Memory Expansion CPU usage reported by amepat is just an
estimate.  The actual CPU usage used for Active Memory Expansion may be lower
or higher depending on the workload.
```

Rather than running the report in the foreground each time you want to compare different AME configurations and expansion factors, it is suggested to run the tool in the background and record the statistics in a recording file for later use, as shown in Example 3-3.

Example 3-3 Create a 60-minute amepat recording to /tmp/ame.out

```
root@aix1:/ # amepat -R /tmp/ame.out 60
Continuing Recording through background process...
root@aix1:/ # ps -aef |grep amepat
    root 4587544       1   0 07:48:28  pts/0  0:25 amepat -R /tmp/ame.out 5
root@aix1:/ #
```

Once **amepat** has completed its recording, you can run the same **amepat** command as used previously in Example 3-2 on page 53 with the exception that you specify a `-P` option to specify the recording file to be processed rather than a time interval.

Example 3-4 demonstrates how to run **amepat** against a recording file, with the same AME expansion factor input parameters used in Example 3-2 on page 53 to compare software compression with hardware compression. The `-O proc=P7+` option specifies that **amepat** is to run the report using POWER7+ hardware with the compression accelerator.

Example 3-4 Running amepat against the record file with hardware compression

```
root@aix1:/ # amepat -e 1.20:2.0:0.1 -O proc=P7+ -P /tmp/ame.out
```

```
Command Invoked               : amepat -e 1.20:2.0:0.1 -O proc=P7+ -P /tmp/ame.out

Date/Time of invocation       : Tue Oct  9 07:48:28 CDT 2012
Total Monitored time          : 7 mins 21 secs
Total Samples Collected       : 3

System Configuration:
---------------------
Partition Name                : aix1
Processor Implementation Mode : Power7 Mode
Number Of Logical CPUs        : 16
Processor Entitled Capacity   : 2.00
Processor Max. Capacity       : 4.00
True Memory                   : 8.00 GB
SMT Threads                   : 4
Shared Processor Mode         : Enabled-Uncapped
Active Memory Sharing         : Disabled
Active Memory Expansion       : Enabled
Target Expanded Memory Size   : 8.00 GB
Target Memory Expansion factor : 1.00


System Resource Statistics:               Average             Min             Max
---------------------------               -----------       -----------       -----------
CPU Util (Phys. Processors)               1.41 [ 35%]       1.38 [ 35%]       1.46 [ 36%]
Virtual Memory Size (MB)                  5665 [ 69%]       5665 [ 69%]       5665 [ 69%]
True Memory In-Use (MB)                   5881 [ 72%]       5881 [ 72%]       5881 [ 72%]
Pinned Memory (MB)                        1105 [ 13%]       1105 [ 13%]       1106 [ 14%]
File Cache Size (MB)                       199 [  2%]        199 [  2%]        199 [  2%]
Available Memory (MB)                     2302 [ 28%]       2302 [ 28%]       2303 [ 28%]

AME Statistics:                           Average             Min             Max
---------------                           -----------       -----------       -----------
AME CPU Usage (Phy. Proc Units)           0.00 [  0%]       0.00 [  0%]       0.00 [  0%]
Compressed Memory (MB)                       0 [  0%]          0 [  0%]          0 [  0%]
Compression Ratio                           N/A

Active Memory Expansion Modeled Statistics        :
------------------------------------------
Modeled Implementation   : Power7+
Modeled Expanded Memory Size :   8.00 GB
Achievable Compression ratio   :0.00

Expansion    Modeled True      Modeled             CPU Usage
Factor       Memory Size       Memory Gain         Estimate
---------    -------------     ------------------  -----------
     1.00          8.00 GB         0.00 KB [  0%]   0.00 [  0%]
     1.28          6.25 GB         1.75 GB [ 28%]   0.14 [  4%]
     1.40          5.75 GB         2.25 GB [ 39%]   0.43 [ 11%]
     1.46          5.50 GB         2.50 GB [ 45%]   0.57 [ 14%]
     1.53          5.25 GB         2.75 GB [ 52%]   0.72 [ 18%]
     1.69          4.75 GB         3.25 GB [ 68%]   1.00 [ 25%]
     1.78          4.50 GB         3.50 GB [ 78%]   1.13 [ 28%]
     1.89          4.25 GB         3.75 GB [ 88%]   1.13 [ 28%]
     2.00          4.00 GB         4.00 GB [100%]   1.13 [ 28%]

Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
```

```
with a memory size of 5.50 GB and to configure a memory expansion factor
of 1.46.  This will result in a memory gain of 45%. With this
configuration, the estimated CPU usage due to AME is approximately 0.57
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.03 physical processors.

NOTE: amepat's recommendations are based on the workload's utilization level
during the monitored period. If there is a change in the workload's utilization
level or a change in workload itself, amepat should be run again.

The modeled Active Memory Expansion CPU usage reported by amepat is just an
estimate.  The actual CPU usage used for Active Memory Expansion may be lower
or higher depending on the workload.
```

Note: The **-O proc=value** option in **amepat** is available in AIX 6.1 TL8 and AIX 7.2 TL2 and later.

This shows that for an identical workload, POWER7+ enables a significant reduction in the processor overhead using hardware compression compared with POWER7 software compression.

3.2.3 Suitable workloads

Before enabling AME on an LPAR to benefit a given workload, some initial considerations need to be made to understand the workload's memory characteristics. This will affect the benefit that can be gained from the use of AME.

- The better a workload's data can be compressed, the higher the memory expansion factor that can be achieved with AME. The **amepat** tool can perform analysis. Data stored in memory that is not already compressed in any form is a good candidate for AME.
- Memory access patterns affect how well AME will perform. When memory is accessed, it is moved from the compressed pool to the uncompressed pool. If a small amount of the memory is frequently accessed, and a large amount is not frequently accessed, this type of workload will perform best with AME.
- Workloads that use a large portion of memory for a file system cache will not benefit substantially from AME, because file system cache memory will not be compressed.
- Workloads that have pinned memory or large pages may not experience the full benefit of AME because pinned memory and large memory pages cannot be compressed.
- Memory resource provided by AME cannot be used to create a RAMDISK in AIX.
- Compression of 64k pages is disabled by setting the tunable **vmm_mpsize_support** to **-1** by default. This can be changed to enable compression of 64k pages. However, the overhead of decompressing 64k pages (treated as 16 x 4k pages) outweighs the performance benefit of using medium 64k pages. It is in most cases not advised to compress 64k pages.

Note: Using the **amepat** tool provides guidance of the memory savings achievable by using Active Memory Expansion.

3.2.4 Deployment

Once you have run the **amepat** tool, and have an expansion factor in mind, to activate active memory for the first time you need to modify the LPAR's partition profile and reactivate the LPAR. The AME expansion factor can be dynamically modified after this step.

Figure 3-8 demonstrates how to enable active memory expansion with a starting expansion factor of 1.4. This means that there will be 8 GB of real memory, multiplied by 1.4 resulting in AIX seeing a total of 11.2 GB of expanded memory.

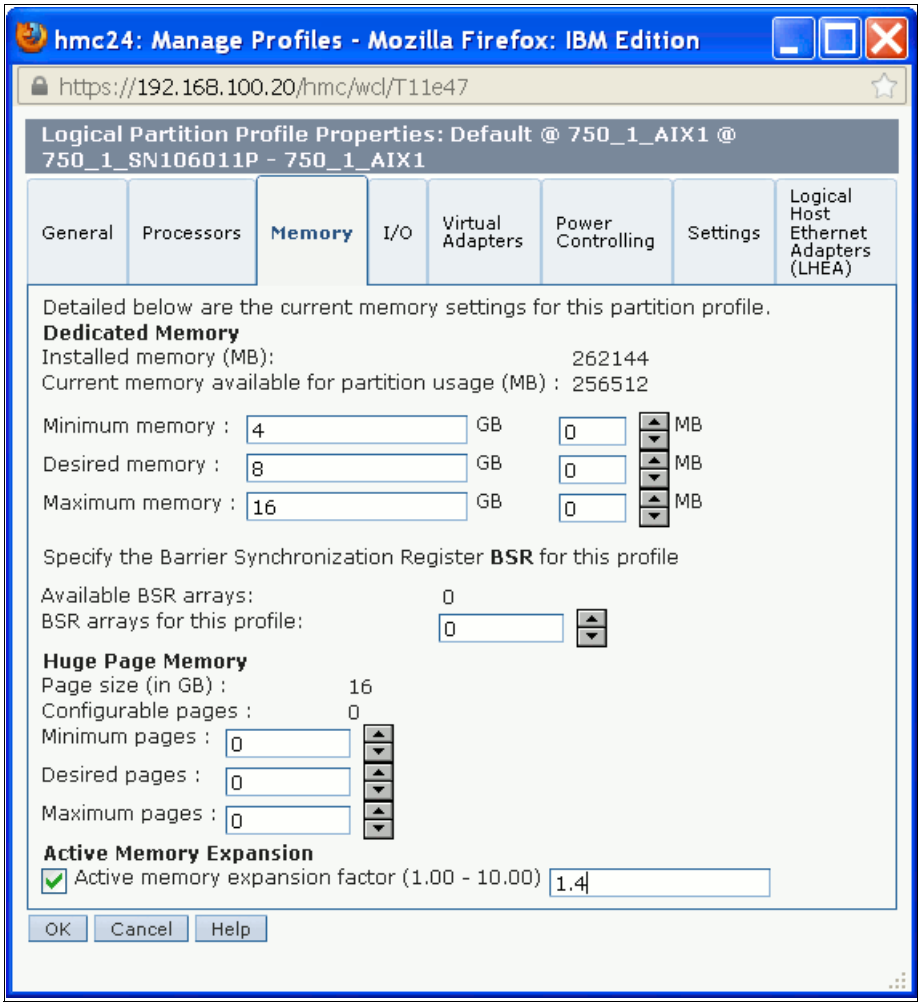


Figure 3-8 Enabling AME in the LPAR profile

Once the LPAR is re-activated, confirm that the settings took effect by running the **lparstat -i** command. This is shown in Example 3-5.

Example 3-5 Running lparstat -i

```
root@aix1:/ # lparstat -i
Node Name                                  : aix1
Partition Name                             : 750_2_AIX1
Partition Number                           : 20
Type                                       : Shared-SMT-4
Mode                                       : Uncapped
Entitled Capacity                          : 2.00
```

```
Partition Group-ID                            : 32788
Shared Pool ID                                : 0
Online Virtual CPUs                           : 4
Maximum Virtual CPUs                          : 8
Minimum Virtual CPUs                          : 1
Online Memory                                 : 8192 MB
Maximum Memory                                : 16384 MB
Minimum Memory                                : 4096 MB
Variable Capacity Weight                      : 128
Minimum Capacity                              : 0.50
Maximum Capacity                              : 8.00
Capacity Increment                            : 0.01
Maximum Physical CPUs in system               : 16
Active Physical CPUs in system                : 16
Active CPUs in Pool                           : 16
Shared Physical CPUs in system                : 16
Maximum Capacity of Pool                      : 1600
Entitled Capacity of Pool                     : 1000
Unallocated Capacity                          : 0.00
Physical CPU Percentage                       : 50.00%
Unallocated Weight                            : 0
Memory Mode                                   : Dedicated-Expanded
Total I/O Memory Entitlement                  : -
Variable Memory Capacity Weight               : -
Memory Pool ID                                : -
Physical Memory in the Pool                   : -
Hypervisor Page Size                          : -
Unallocated Variable Memory Capacity Weight: -
Unallocated I/O Memory entitlement            : -
Memory Group ID of LPAR                       : -
Desired Virtual CPUs                          : 4
Desired Memory                                : 8192 MB
Desired Variable Capacity Weight              : 128
Desired Capacity                              : 2.00
Target Memory Expansion Factor                : 1.25
Target Memory Expansion Size                  : 10240 MB
Power Saving Mode                             : Disabled
root@aix1:/ #
```

The output of Example 3-5 on page 57 tells the following:

- The memory mode is Dedicated-Expanded. This means that we are not using Active Memory Sharing (AMS), but we are using Active Memory Expansion (AME).
- The desired memory is 8192 MB. This is the true memory allocated to the LPAR.
- The AME expansion factor is 1.25.
- The size of the expanded memory pool is 10240 MB.

Once AME is activated, the workload may change, so it is suggested to run **amepat** regularly to see if the optimal expansion factor is currently set based on the **amepat** tool's recommendation. Example 3-6 shows a portion of the **amepat** output with the **amepat** tool's recommendation being 1.38.

Example 3-6 Running amepat after AME is enabled for comparison

```
Expansion    Modeled True      Modeled             CPU Usage
```

```
Factor       Memory Size       Memory Gain          Estimate
---------    -------------     ------------------   -----------
     1.25          8.00 GB          2.00 GB [ 25%]   0.00 [  0%] << CURRENT CONFIG
     1.30          7.75 GB          2.25 GB [ 29%]   0.00 [  0%]
     1.38          7.25 GB          2.75 GB [ 38%]   0.38 [ 10%]
     1.49          6.75 GB          3.25 GB [ 48%]   1.15 [ 29%]
     1.54          6.50 GB          3.50 GB [ 54%]   1.53 [ 38%]
     1.67          6.00 GB          4.00 GB [ 67%]   2.29 [ 57%]
     1.74          5.75 GB          4.25 GB [ 74%]   2.68 [ 67%]
     1.82          5.50 GB          4.50 GB [ 82%]   3.01 [ 75%]
     2.00          5.00 GB          5.00 GB [100%]   3.01 [ 75%]
```

To change the AME expansion factor once AME is enabled, this can be done by simply reducing the amount of true memory and increasing the expansion factor using Dynamic Logical Partitioning (DLPAR).

Figure 3-9 demonstrates changing the AME expansion factor to 1.38 and reducing the amount of real memory to 7.25 GB.

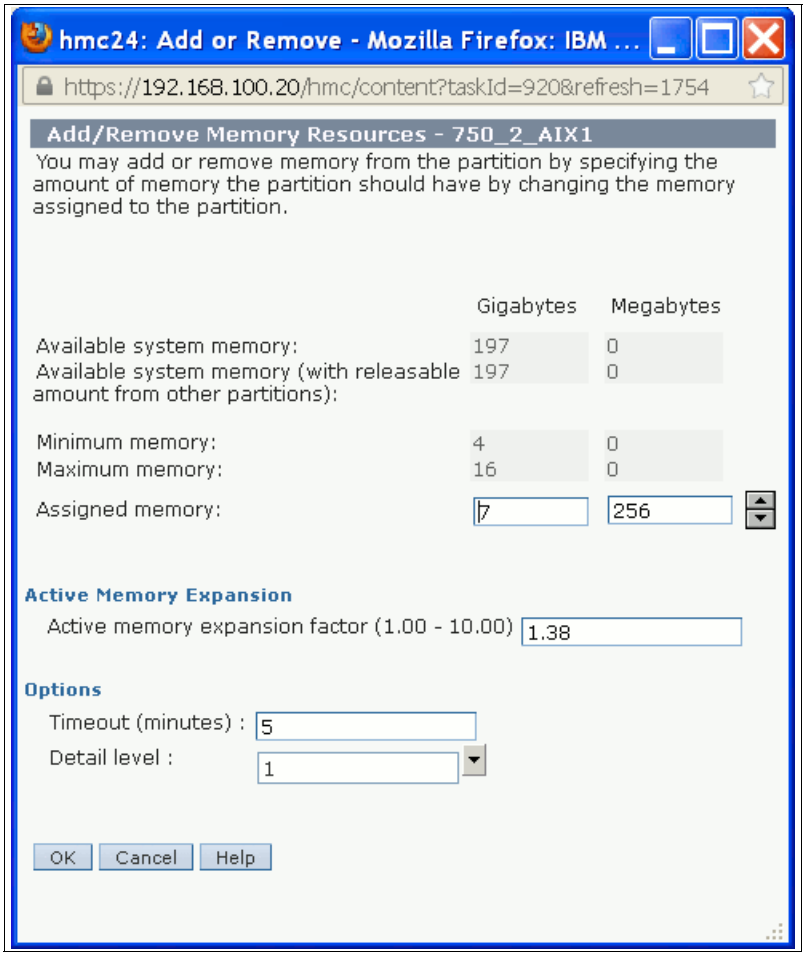


Figure 3-9 Dynamically modify the expansion factor and true memory

After the change, you can now see the memory configuration using the **`lparstat -i`** command as demonstrated in Example 3-5 on page 57. The **`lsattr`** and **`vmstat`** commands can also be used to display this information. This is shown in Example 3-7 on page 60.

Example 3-7 Using lsattr and vmstat to display memory size

```
root@aix1:/ # lsattr -El mem0
ent_mem_cap         I/O memory entitlement in Kbytes           False
goodsize       7424 Amount of usable physical memory in Mbytes False
mem_exp_factor 1.38 Memory expansion factor                    False
size           7424 Total amount of physical memory in Mbytes  False
var_mem_weight      Variable memory capacity weight            False
root@aix1:/ # vmstat |grep 'System configuration'
System configuration: lcpu=16 mem=10240MB ent=2.00
root@aix1:/ #
```

You can see that the true memory is 7424 MB, the expansion factor is 1.38, and the expanded memory pool size is 10240 MB.

Note: Additional information about AME usage can be found at:

ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/pow03037usen/POW03037USEN.PDF

3.2.5 Tunables

There are a number of tunables that can be modified using AME. Typically, the default values are suitable for most workloads, and these tunables should only be modified under the guidance of IBM support. The only value that would need to be tuned is the AME expansion factor.

Table 3-6 AME tunables

Tunable	Description
ame_minfree_mem	If processes are being delayed waiting for compressed memory to become available, increase `ame_minfree_mem` to improve response time. Note that the value use for `ame_minfree_mem` must be at least 257 kb less than `ame_maxfree_mem`.
ame_maxfree_mem	Excessive shrink and grow operations can occur if compressed memory pool size tends to change significantly. This can occur if a workload's working set size frequently changes. Increase this tunable to raise the threshold at which the VMM will shrink a compressed memory pool and thus reduce the number of overall shrink and grow operations.
ame_cpus_per_pool	Lower ratios can be used to reduce contention on compressed memory pools. This ratio is not the only factor used to determine the number of compressed memory pools (amount of memory and its layout are also considered), so certain changes to this ratio may not result in any change to the number of compressed memory pools.
ame_min_ucpool_size	If the compressed memory pool grows too large, there may not be enough space in memory to house uncompressed memory, which can slow down application performance due to excessive use of the compressed memory pool. Increase this value to limit the size of the compressed memory pool and make more uncompressed pages available.

Example 3-8 shows the default and possible values for each of the AME **vmo** tunables.

Example 3-8 AME tunables

```
root@aix1:/ # vmo -L ame_minfree_mem
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
--------------------------------------------------------------------------------
ame_minfree_mem           n/a    8M     8M     64K    4095M  bytes             D
     ame_maxfree_mem
--------------------------------------------------------------------------------
root@aix1:/ # vmo -L ame_maxfree_mem
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
--------------------------------------------------------------------------------
ame_maxfree_mem           n/a    24M    24M    320K   4G     bytes             D
     ame_minfree_mem
--------------------------------------------------------------------------------
root@aix1:/ # vmo -L ame_cpus_per_pool
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
--------------------------------------------------------------------------------
ame_cpus_per_pool         n/a    8      8      1      1K     processors        B
--------------------------------------------------------------------------------
root@aix1:/ # vmo -L ame_min_ucpool_size
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
--------------------------------------------------------------------------------
ame_min_ucpool_size       n/a    0      0      5      95     % memory          D
--------------------------------------------------------------------------------
root@aix1:/ #
```

3.2.6 Monitoring

When using active memory expansion, in addition to monitoring the processor usage of AME, it is also important to monitor paging space and memory deficit. Memory deficit is the amount of memory that cannot fit into the compressed pool as a result of AME not being able to reach the target expansion factor. This is caused by the expansion factor being set too high.

The **lparstat -c** command can be used to display specific information related to AME. This is shown in Example 3-9.

Example 3-9 Running lparstat -c

```
root@aix1:/ # lparstat -c 5 5

System configuration: type=Shared mode=Uncapped mmode=Ded-E smt=4 lcpu=64 mem=14336MB
tmem=8192MB psize=7 ent=2.00

%user  %sys  %wait  %idle physc %entc  lbusy   app  vcsw phint  %xcpu xphysc    dxm
----- ----- ------ ------ ----- ----- ------   --- ----- ----- ------ ------ ------
 66.3  13.4    8.8   11.5  5.10 255.1   19.9  0.00 18716  6078    1.3 0.0688      0
 68.5  12.7   10.7    8.0  4.91 245.5   18.7  0.00 17233  6666    2.3 0.1142      0
 69.7  13.2   13.1    4.1  4.59 229.5   16.2  0.00 15962  8267    1.0 0.0481      0
 73.8  14.7    9.2    2.3  4.03 201.7   34.6  0.00 19905  5135    0.5 0.0206      0
 73.5  15.9    7.9    2.8  4.09 204.6   28.7  0.00 20866  5808    0.3 0.0138      0
```

```
root@aix1:/ #
```

The items of interest in the **lparstat -c** output are the following:

mmode This is how the memory of our LPAR is configured. In this case **Ded-E** means the memory is dedicated, meaning AMS is not active, and AME is enabled.

mem This is the expanded memory size.

tmem This is the true memory size.

physc This is how many physical processor cores our LPAR is consuming.

%xcpu This is the percentage of the overall processor usage that AME is consuming.

xphysc This is the amount of physical processor cores that AME is consuming.

dxm This is the memory deficit, which is the number of 4 k pages that cannot fit into the expanded memory pool. If this number is greater than zero, it is likely that the expansion factor is too high, and paging activity will be present on the AIX system.

The **vmstat -sc** command also provides some information specific to AME. One is the amount of compressed pool pagein and pageout activity. This is important to check because it could be a sign of memory deficit and the expansion factor being set too high. Example 3-10 gives a demonstration of running the **vmstat -sc** command.

Example 3-10 Running vmstat -sc

```
root@aix1:/ # vmstat -sc
              5030471 total address trans. faults
                72972 page ins
                24093 page outs
                    0 paging space page ins
                    0 paging space page outs
                    0 total reclaims
              3142095 zero filled pages faults
                66304 executable filled pages faults
                    0 pages examined by clock
                    0 revolutions of the clock hand
                    0 pages freed by the clock
               132320 backtracks
                    0 free frame waits
                    0 extend XPT waits
                23331 pending I/O waits
                97065 start I/Os
                42771 iodones
             88835665 cpu context switches
               253502 device interrupts
              4793806 software interrupts
             92808260 decrementer interrupts
                68395 mpc-sent interrupts
                68395 mpc-receive interrupts
               528426 phantom interrupts
                    0 traps
             85759689 syscalls
                    0 compressed pool page ins
                    0 compressed pool page outs
root@aix1:/ #
```

The **vmstat -vc** command also provides some information specific to AME. This command displays information related to the size of the compressed pool and an indication whether AME is able to achieve the expansion factor that has been set. Items of interest include the following:

- Compressed pool size
- Percentage of true memory used for the compressed pool
- Free pages in the compressed pool (this is the mount of 4 k pages)
- Target AME expansion factor
- The AME expansion factor that is currently being achieved

Example 3-11 demonstrates running the **vmstat -vc** command.

Example 3-11 Running vmstat -vc

```
root@aix1:/ # vmstat -vc
              3670016 memory pages
              1879459 lruable pages
               880769 free pages
                    8 memory pools
               521245 pinned pages
                 95.0 maxpin percentage
                  3.0 minperm percentage
                 80.0 maxperm percentage
                  1.8 numperm percentage
                33976 file pages
                  0.0 compressed percentage
                    0 compressed pages
                  1.8 numclient percentage
                 80.0 maxclient percentage
                33976 client pages
                    0 remote pageouts scheduled
                    0 pending disk I/Os blocked with no pbuf
              1749365 paging space I/Os blocked with no psbuf
                 1972 filesystem I/Os blocked with no fsbuf
                 1278 client filesystem I/Os blocked with no fsbuf
                    0 external pager filesystem I/Os blocked with no fsbuf
               500963 Compressed Pool Size
                 23.9 percentage of true memory used for compressed pool
                61759 free pages in compressed pool (4K pages)
                  1.8 target memory expansion factor
                  1.8 achieved memory expansion factor
                 75.1 percentage of memory used for computational pages
root@aix1:/ #
```

Note: Additional information about AME performance can be found at:

ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/pow03038usen/POW03038USEN.PDF

3.2.7 Oracle batch scenario

We performed a test on an Oracle batch workload to determine the memory saving benefit of AME. The LPAR started with 120 GB of memory assigned and 24 virtual processors (VP) allocated.

Over the course of three tests, we increased the AME expansion factor and reduced the amount of memory with the same workload.

Figure 3-10 provides an overview of the three tests carried out.

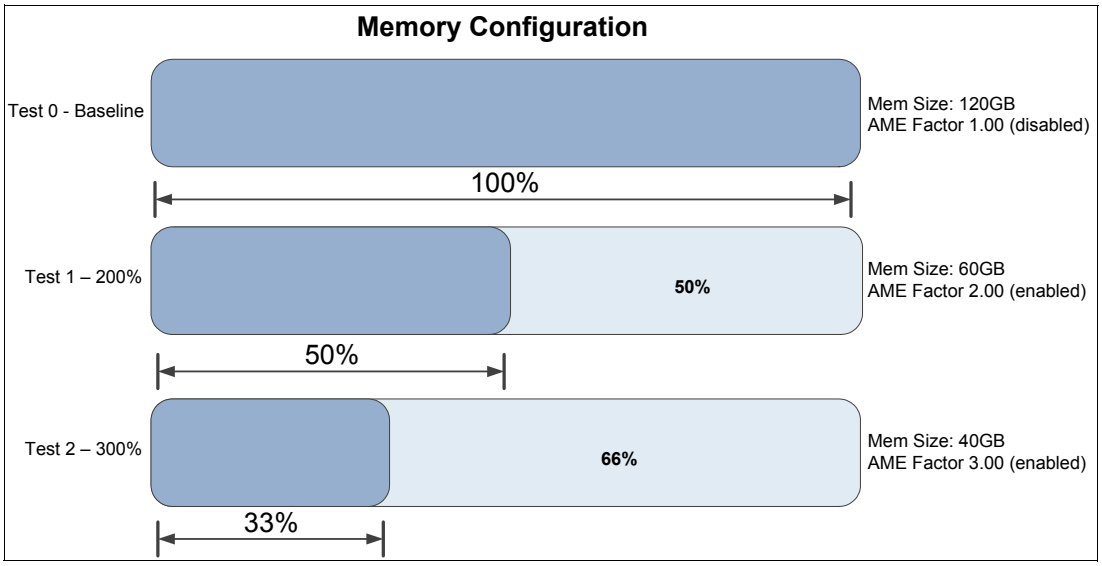


Figure 3-10 AME test on an Oracle batch workload

On completion of the tests, the first batch run completed in 124 minutes. The batch time grew slightly on the following two tests. However, the amount of true memory allocated was significantly reduced.

Table 3-7 provides a summary of the test results.

Table 3-7 Oracle batch test results

Test run	Processor	Memory assigned	Runtime	Avg. processor
Test 0	24	120 GB (AME disabled)	124 Mins	16.3
Test 1	24	60 GB (AME expansion 2.00)	127 Mins	16.8
Test 2	24	40 GB (AME expansion 3.00)	134 Mins	17.5

Conclusion: The impact of AME on batch duration is less than 10% with a processor overhead of 7% with *three times less memory.*

3.2.8 Oracle OLTP scenario

We performed a test on an Oracle OLTP workload in a scenario where the free memory on the LPAR with 100 users is less than 1%. By enabling active memory expansion we tested keeping the real memory the same, and increasing the expanded memory pool with active memory expansion to enable the LPAR to support additional users.

The objective of the test was to increase the number of users and TPS without affecting the application's response time.

Three tests were performed, first with AME turned off, the second with an expansion factor of 1.25 providing 25% additional memory as a result of compression, and a test with an expansion factor of 1.6 to provide 60% of additional memory as a result of compression. The amount of true memory assigned to the LPAR remained at 8 GB during all three tests.

Figure 3-11 provides an overview of the three tests.

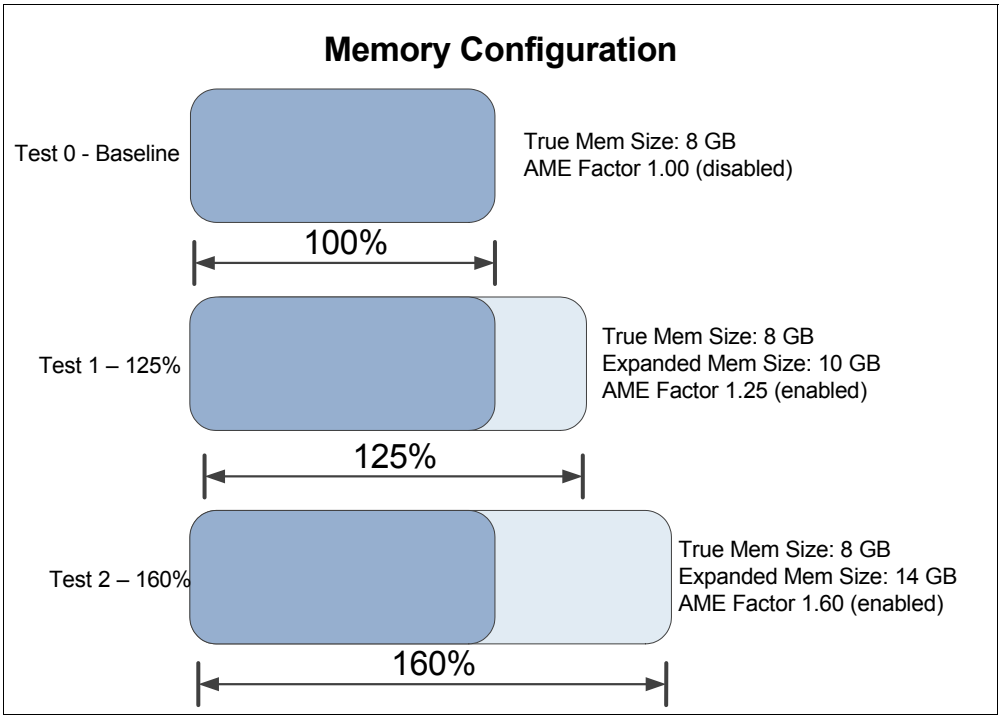


Figure 3-11 AME test on an Oracle OLTP workload

In the test, our LPAR had 8 GB of real memory and the Oracle SGA was sized at 5 GB.

With 100 concurrent users and no AME enabled, the 8 GB of assigned memory was 99% consumed. When the AME expansion factor was modified to 1.25 the amount of users supported was 300, with 0.1 processor cores consumed by AME overhead.

At this point of the test, we ran the **amepat** tool to identify the recommendation of **amepat** for our workload. Example 3-12 shows a subset of the **amepat** report, where our current expansion factor is 1.25 and the recommendation from **amepat** was a 1.54 expansion factor.

Example 3-12 Output of amepat during test 1

```
Expansion    Modeled True      Modeled             CPU Usage
Factor       Memory Size       Memory Gain         Estimate
---------    -------------     ------------------  -----------
     1.03          9.75 GB       256.00 MB [  3%]   0.00 [  0%]
     1.18          8.50 GB         1.50 GB [ 18%]   0.00 [  0%]
     1.25          8.00 GB         2.00 GB [ 25%]   0.01 [  0%] << CURRENT CONFIG
     1.34          7.50 GB         2.50 GB [ 33%]   0.98 [  6%]
     1.54          6.50 GB         3.50 GB [ 54%]   2.25 [ 14%]
     1.67          6.00 GB         4.00 GB [ 67%]   2.88 [ 18%]
     1.82          5.50 GB         4.50 GB [ 82%]   3.51 [ 22%]
     2.00          5.00 GB         5.00 GB [100%]   3.74 [ 23%]
```

It is important to note that the **amepat** tool's objective is to reduce the amount of real memory assigned to the LPAR by using compression based on the expansion factor. This explains the 2.25 processor overhead estimate of **amepat** being more than the 1.65 actual processor overhead that we experienced because we did not change our true memory.

Table 3-8 provides a summary of our test results.

Table 3-8 OLTP results

Test run	Processor	Memory assigned	TPS	No of users	Avg CPU
Test 0	VP = 16	8 GB (AME disabled)	325	100	1.7 (AME=0)
Test 1	VP = 16	8 GB (AME expansion 1.25)	990	300	4.3 (AME=0.10)
Test 2	VP = 16	8 GB (AME expansion 1.60)	1620	500	7.5 (AME=1.65)

Conclusion: The impact of AME on our Oracle OLTP workload enabled our AIX LPAR to have *5 times more users* and *5 times more TPS* with the *same memory footprint*.

3.2.9 Using amepat to suggest the correct LPAR size

During our testing with AME we observed cases where the recommendations by the **amepat** tool could be biased by incorrect LPAR size. We found that if the memory allocated to an LPAR far exceeded the amount consumed by the running workload, then the ratio suggested by **amepat** would actually be unrealistic. Such cases of concern become apparent when running through iterations of **amepat**—a suggested ratio will keep contradicting the previous result.

To illustrate this point, Example 3-13 lists a portion from the **amepat** output from a 5-minute sample of an LPAR running a WebSphere Message Broker workload. The LPAR was configured with 8 GB of memory.

Example 3-13 Initial amepat iteration for an 8 GB LPAR

```
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 1.00 GB and to configure a memory expansion factor
of 8.00.  This will result in a memory gain of 700%. With this
configuration, the estimated CPU usage due to AME is approximately 0.21
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.50 physical processors.
```

The LPAR was configured with 8 GB of memory and with an AME expansion ratio of 1.0. The LPAR was reconfigured based on the recommendation and reactivated to apply the change. The workload was restarted and **amepat** took another 5-minute sample. Example 3-14 lists the second recommendation.

Example 3-14 Second amepat iteration

```
WARNING: This LPAR currently has a memory deficit of 6239 MB.
A memory deficit is caused by a memory expansion factor that is too
high for the current workload.  It is recommended that you reconfigure
the LPAR to eliminate this memory deficit.  Reconfiguring the LPAR
with one of the recommended configurations in the above table should
```

```
eliminate this memory deficit.

The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 3.50 GB and to configure a memory expansion factor
of 2.29.  This will result in a memory gain of 129%. With this
configuration, the estimated CPU usage due to AME is approximately 0.00
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.25 physical processors.
```

The LPAR was once again reconfigured, reactivated, and the process repeated. Example 3-15 shows the third recommendation.

Example 3-15 Third amepat iteration

```
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 1.00 GB and to configure a memory expansion factor
of 8.00.  This will result in a memory gain of 700%. With this
configuration, the estimated CPU usage due to AME is approximately 0.25
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.54 physical processors.
```

We stopped this particular test cycle at this point. The LPAR was reconfigured to have 8 GB dedicated; the *active memory expansion factor* checkbox was unticked. The first **amepat** recommendation was now something more realistic, as shown in Example 3-16.

Example 3-16 First amepat iteration for the second test cycle

```
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 4.50 GB and to configure a memory expansion factor
of 1.78.  This will result in a memory gain of 78%. With this
configuration, the estimated CPU usage due to AME is approximately 0.00
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.28 physical processors.
```

However, reconfiguring the LPAR and repeating the process produced a familiar result, as shown in Example 3-17.

Example 3-17 Second amepat iteration for second test cycle

```
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 1.00 GB and to configure a memory expansion factor
of 8.00.  This will result in a memory gain of 700%. With this
configuration, the estimated CPU usage due to AME is approximately 0.28
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.56 physical processors.
```

The Message Broker workload being used had been intentionally configured to provide a small footprint; this was to provide an amount of load on the LPAR without being excessively demanding on processor or RAM. We reviewed the other sections in the **amepat** reports to see if there was anything to suggest why the recommendations were unbalanced.

Since the LPAR was originally configured with 8 GB of RAM, all the AME projections were based on that goal. However, from reviewing all the reports, we saw that the amount of RAM being consumed by the workload was not using near the 8 GB. The System Resource Statistics section details memory usage during the sample period. Example 3-18 on page 68 lists the details from the initial report, which was stated in part in Example 3-13 on page 66.

Example 3-18 Average system resource statistics from initial amepat iteration

```
System Resource Statistics:               Average
--------------------------               -----------
CPU Util (Phys. Processors)              1.82 [ 46%]
Virtual Memory Size (MB)                 2501 [ 31%]
True Memory In-Use (MB)                  2841 [ 35%]
Pinned Memory (MB)                       1097 [ 13%]
File Cache Size (MB)                      319 [  4%]
Available Memory (MB)                    5432 [ 66%]
```

From Example 3-18 we can conclude that only around a third of the allocated RAM is being consumed. However, in extreme examples where the LPAR was configured to have less than 2 GB of actual RAM, this allocation was too small for the workload to be healthily contained.

Taking the usage profile into consideration, the LPAR was reconfigured to have 4 GB of dedicated RAM (no AME). The initial **amepat** recommendations were now more realistic (Example 3-19).

Example 3-19 Initial amepat results for a 4-GB LPAR

```
System Resource Statistics:               Average
--------------------------               -----------
CPU Util (Phys. Processors)              1.84 [ 46%]
Virtual Memory Size (MB)                 2290 [ 56%]
True Memory In-Use (MB)                  2705 [ 66%]
Pinned Memory (MB)                       1096 [ 27%]
File Cache Size (MB)                      392 [ 10%]
Available Memory (MB)                    1658 [ 40%]

Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 2.50 GB and to configure a memory expansion factor
of 1.60.  This will result in a memory gain of 60%. With this
configuration, the estimated CPU usage due to AME is approximately 0.00
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.28 physical processors.
```

So the recommendation of 2.5 GB is still larger than the quantity actually consumed. But the amount of free memory is much more reasonable. Reconfiguring the LPAR and repeating the process now produced more productive results. Example 3-20 lists the expansion factor which **amepat** settled on.

Example 3-20 Final amepat recommendation for a 4-GB LPAR

```
Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 2.00 GB and to configure a memory expansion factor
of 1.50.  This will result in a memory gain of 50%. With this
configuration, the estimated CPU usage due to AME is approximately 0.13
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 2.43 physical processors.
```

Note: Once the LPAR has been configured to the size shown in Example 3-20 on page 68, the **amepat** recommendations were consistent for additional iterations. So if successive iterations with the **amepat** recommendations contradict themselves, we suggest reviewing the size of your LPAR.

3.2.10 Expectations of AME

When considering AME and using **amepat**, remember to take into consideration what data is used to calculate any recommendations:

- Memory size (allocated to LPAR)
- Memory usage
- Type of data stored in memory
- Processor usage

However, **amepat** has no concept of understanding of application throughput or time sensitivity. The recommendations aim to provide the optimal use of memory allocation at the expense of some processor cycles. **amepat** cannot base recommendations on optimal application performance, because it has no way to interpret such an attribute from AIX.

In the scenario discussed in 3.2.9, “Using amepat to suggest the correct LPAR size” on page 66, the concluding stable recommendation provided 96% of the application throughput of the original 4 GB dedicated configuration. However, an intermediate recommendation actually produced 101%, whereas the scenario discussed in 3.2.8, “Oracle OLTP scenario” on page 64 resulted in more throughput with the same footprint.

Different workloads will produce different results, both in terms of resource efficiency and application performance. Our previous sections illustrate some of the implications, which can be used to set expectations for your own workloads based on your requirements.

3.3 Active Memory Sharing (AMS)

Active Memory Sharing is a feature of Power systems that allows better memory utilization, similar to Shared Processor Partition (SPLPAR) processor optimization. Similar to what occurs with processors, instead of dedicating memory to partitions, the memory can be assigned as shared. The PowerVM hypervisor manages real memory across multiple AMS-enabled partitions, distributing memory from Share Memory Pool to partitions based on their workload demand.

AMS requirements:

- Enterprise version of PowerVM
- POWER6 AIX 6.1 TL 3 or later, VIOS 2.1.0.1-FP21
- POWER7 AIX 6.1 TL 4 or later, VIOS 2.1.3.10-FP23

For additional information about Active Memory Sharing, refer to *IBM PowerVM Virtualization Active Memory Sharing*, REDP-4470 at:

http://www.redbooks.ibm.com/redpapers/pdfs/redp4470.pdf

3.4 Active Memory Deduplication (AMD)

A system might have a considerable amount of duplicated information stored on its memory. Active Memory Deduplication allows the PowerVM hypervisor to dynamically map identical partition memory pages to a single physical memory page. AMD depends on the Active Memory Sharing (AMS) feature to be available, and relies on processor cycles to identify duplicated pages with hints taken directly from the operating system.

The Active Memory Deduplication feature requires the following minimum components:

- POWER7
- PowerVM Enterprise edition
- System firmware level 740
- AIX Version 6: AIX 6.1 TL7, or later
- AIX Version 7: AIX 7.1 TL1 SP1, or later

For more information about Active Memory Sharing, refer to:

http://www.redbooks.ibm.com/redpapers/pdfs/redp4827.pdf

3.5 Virtual I/O Server (VIOS) sizing

In this section we highlight some suggested sizing guidelines and tools to ensure that a VIOS on a POWER7 system is allocated adequate resources to deliver optimal performance.

It is essential to continually monitor the resource utilization of the VIOS and review the hardware assignments as workloads change.

3.5.1 VIOS processor assignment

The VIOS uses processor cycles to deliver I/O to client logical partitions. This includes running the VIO server's own instance of the AIX operating system, and processing shared Ethernet traffic as well as shared disk I/O traffic including virtual SCSI and N-Port Virtualization (NPIV).

Typically SEAs backed by 10 Gb physical adapters in particular consume a large amount of processor resources on the VIOS depending on the workload. High-speed 8 Gb Fibre Channel adapters in environments with a heavy disk I/O workload also consume large amounts of processor cycles.

In most cases, to provide maximum flexibility and performance capability, it is suggested that you configure the VIOS partition taking the settings into consideration described in Table 3-9.

Table 3-9 Suggested Virtual I/O Server processor settings

Setting	Suggestion
Processing mode	There are two options for the processing, shared or dedicated. In most cases the suggestion is to use shared to take advantage of PowerVM and enable the VIOS to take advantage of additional processor capacity during peak workloads.

Setting	Suggestion
Entitled capacity	The entitled capacity is ideally set to the average processing units that the VIOS partition is using. If your VIOS is constantly consuming beyond 100% of the entitled capacity, the suggestion is to increase the capacity entitlement to match the average consumption.
Desired virtual processors	The virtual processors should be set to the number of cores with some headroom that the VIOS will consume during peak workload.
Sharing mode	The suggested sharing mode is uncapped. This enables the VIOS partition to consume additional processor cycles from the shared pool when it is under load.
Weight	The VIOS partition is sensitive to processor allocation. When the VIOS is starved of resources, all virtual client logical partitions will be affected. The VIOS typically should have a higher weight than all of the other logical partitions in the system. The weight ranges from 0-255; the suggested value for the Virtual I/O server would be in the upper part of the range.
Processor compatibility mode	The suggested compatibility mode to configure in the VIOS partition profile to use the default setting. This allows the LPAR to run in whichever mode is best suited for the level of VIOS code installed.

Processor folding at the time of writing is not supported for VIOS partitions. When a VIOS is configured as uncapped, virtual processors that are not in use are “folded” to ensure that they are available for use by other logical partitions.

It is important to ensure that the entitled capacity and virtual processor are sized appropriately on the VIOS partition to ensure that there are no wasted processor cycles on the system.

When VIOS is installed from a base of 2.1.0.13-FP23 or later, processor folding is already disabled by default. If the VIOS has been upgraded or migrated from an older version, then processor folding may remain enabled.

The **schedo** command can be used to query whether processor folding is enabled, as shown in Example 3-21.

Example 3-21 How to check whether processor folding is enabled

```
$ oem_setup_env
# schedo -o vpm_fold_policy
vpm_fold_policy = 3
```

If the value is anything other than 4, then processor folding needs to be disabled.

Processor folding is discussed in 4.1.3, “Processor folding” on page 123.

Example 3-22 demonstrates how to disable processor folding. This change is dynamic, so that no reboot of the VIOS LPAR is required.

Example 3-22 How to disable processor folding

```
$ oem_setup_env
# schedo -p -o vpm_fold_policy=4
```

```
Setting vpm_fold_policy to 4 in nextboot file
Setting vpm_fold_policy to 4
```

3.5.2 VIOS memory assignment

The VIOS also has some specific memory requirements, which need to be monitored to ensure that the LPAR has sufficient memory.

The VIOS Performance Advisor, which is covered in 5.9, “VIOS performance advisor tool and the part command” on page 271, provides recommendations regarding sizing and configuration of a running VIOS.

However, for earlier VIOS releases or as a starting point, the following guidelines can be used to estimate the required memory to assign to a VIOS LPAR:

- 2 GB of memory for every 16 processor cores in the machine. For most VIOS this should be a workable base to start from.
- For more complex implementations, start with a minimum allocation of 768 MB. Then add increments based on the quantities of the following adapters:
  - For each Logical Host Ethernet Adapter (LHEA) add 512 MB, and an additional 102 MB per configured port.
  - For each non-LHEA 1 Gb Ethernet port add 102 MB.
  - For each non-LHEA 10 Gb port add 512 MB.
  - For each 8 Gb Fibre Channel adapter port add 512 MB.
  - For each NPIV Virtual Fibre Channel adapter add 140 MB.
  - For each Virtual Ethernet adapter add 16 MB.

In the cases above, even if a given adapter is idle or not yet assigned to an LPAR (in the case of NPIV), still base your sizing on the intended scaling.

3.5.3 Number of VIOS

Depending on your environment, the number of VIOS on your POWER7 system will vary. There are cases where due to hardware constraints only a single VIOS can be deployed, such non-HMC-managed systems using Integrated Virtualization Manager (IVM).

Note: With version V7R7.6.0.0 and later, an HMC can be used to manage POWER processor-based blades.

As a general best practice, it is ideal to deploy Virtual I/O servers in redundant pairs. This enables both additional availability and performance for the logical partitions using virtualized I/O. The first benefit this provides is the ability to be able to shut down one of the VIOS for maintenance; the second VIOS will be available to serve I/O to client logical partitions. This also caters for a situation where there may be an unexpected outage on a single VIOS and the second VIOS can continue to serve I/O and keep the client logical partitions running.

Configuring VIOS in this manner is covered in *IBM PowerVM Virtualization Introduction and Configuration,* SG24-7940-04.

In most cases, a POWER7 system has a single pair of VIOS. However, there may be situations where a second or even third pair may be required. In most situations, a single pair of VIO servers is sufficient.

Following are some situations where additional pairs of Virtual I/O servers may be a consideration on larger machines where there are additional resources available:

- Due to heavy workload, a pair of VIOS may be deployed for shared Ethernet and a second pair may be deployed for disk I/O using a combination of N-Port Virtualization (NPIV), Virtual SCSI, or shared storage pools (SSP).
- Due to different types of workloads, there may be a pair of VIOS deployed for each type of workload, to cater to multitenancy situations or situations where workloads must be totally separated by policy.
- There may be production and nonproduction LPARs on a single POWER7 frame with a pair of VIOS for production and a second pair for nonproduction. This would enable both workload separation and the ability to test applying fixes in the nonproduction pair of VIOS before applying them to the production pair. Obviously, where a single pair of VIOS are deployed, they can still be updated one at a time.

Note: Typically a single pair of VIOS per Power system will be sufficient, so long as the pair is provided with sufficient processor, memory, and I/O resources.

3.5.4 VIOS updates and drivers

On a regular basis, new enhancements and fixes are added to the VIOS code. It is important to ensure that your Virtual I/O servers are kept up to date. It is also important to check your IOS level and update it regularly. Example 3-23 demonstrates how to check the VIOS level.

Example 3-23 How to check your VIOS level

```
$ ioslevel
2.2.2.0
$
```

For optimal disk performance, it is also important to install the AIX device driver for your disk storage system on the VIOS. Example 3-24 illustrates where the storage device drivers are not installed. In this case AIX uses a generic device definition because the correct definition for the disk is not defined in the ODM.

Example 3-24 VIOS without correct device drivers installed

```
$ lsdev -type disk
name             status      description
hdisk0           Available   MPIO Other FC SCSI Disk Drive
hdisk1           Available   MPIO Other FC SCSI Disk Drive
hdisk2           Available   MPIO Other FC SCSI Disk Drive
hdisk3           Available   MPIO Other FC SCSI Disk Drive
hdisk4           Available   MPIO Other FC SCSI Disk Drive
hdisk5           Available   MPIO Other FC SCSI Disk Drive
$
```

In this case, the correct device driver needs to be installed to optimize how AIX handles I/O on the disk device. These drivers would include SDDPCM for IBM DS6000™, DS8000®,

V7000 and SAN Volume Controller. For other third-party storage systems, the device drivers can be obtained from the storage vendor such as HDLM for Hitachi or PowerPath for EMC.

Example 3-25 demonstrates verification of the SDDPCM fileset being installed for IBM SAN Volume Controller LUNs, and verification that the ODM definition for the disks is correct.

Example 3-25 Virtual I/O server with SDDPCM driver installed

```
$ oem_setup_env
# lslpp -l devices.fcp.disk.ibm.mpio.rte
  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  devices.fcp.disk.ibm.mpio.rte
                             1.0.0.23  COMMITTED  IBM MPIO FCP Disk Device
# lslpp -l devices.sddpcm*
  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  devices.sddpcm.61.rte      2.6.3.0  COMMITTED  IBM SDD PCM for AIX V61

Path: /etc/objrepos
  devices.sddpcm.61.rte      2.6.3.0  COMMITTED  IBM SDD PCM for AIX V61
# exit
$ lsdev -type disk
name             status      description
hdisk0           Available   MPIO FC 2145
hdisk1           Available   MPIO FC 2145
hdisk2           Available   MPIO FC 2145
hdisk3           Available   MPIO FC 2145
hdisk4           Available   MPIO FC 2145
hdisk5           Available   MPIO FC 2145
$
```

Note: IBM System Storage® device drivers are free to download for your IBM Storage System. Third-party vendors may supply device drivers at an additional charge.

3.6 Using Virtual SCSI, Shared Storage Pools and N-Port Virtualization

PowerVM and VIOS provide the capability to share physical resources among multiple logical partitions to provide efficient utilization of the physical resource. From a disk I/O perspective, different methods are available to implement this.

In this section, we provide a brief overview and comparison of the different I/O device virtualizations available in PowerVM. The topics covered in this section are as follows:

- Virtual SCSI
- Virtual SCSI using Shared Storage Pools
- N_Port Virtualization (NPIV)

Note that Live Partition Mobility (LPM) is supported on all three implementations and in situations that require it, combinations of these technologies can be deployed together, virtualizing different devices on the same machine.

Note: This section does not cover in detail how to tune disk and adapter devices in each scenario. This is covered in 4.3, “I/O device tuning” on page 140.

3.6.1 Virtual SCSI

Virtual SCSI describes the implementation of mapping devices allocated to one or more VIOS using the SCSI protocol to a client logical partition. Any device drivers required for the device such as a LUN are installed on the Virtual I/O server, and the client logical partition sees a generic virtual SCSI device.

In POWER5, this was the only way to share disk storage devices using VIO and is still commonly used in POWER6 and POWER7 environments.

The following are the advantages and performance considerations related to the use of Virtual SCSI:

Advantages

These are the advantages of using Virtual SCSI:

- It enables file-backed optical devices to be presented to a client logical partition as a virtual CDROM. This is mounting an ISO image residing on the VIO server to the client logical partition as a virtual CDROM.
- It does not require specific FC adapters or fabric switch configuration.
- It can virtualize internal disk.
- It provides the capability to map disk from a storage device not capable of a 520-byte format to an IBM i LPAR as supported generic SCSI disk.
- It does not require any disk device drivers to be installed on the client logical partitions, only the Virtual I/O server requires disk device drivers.

Performance considerations

The performance considerations of using Virtual SCSI are:

- Disk device and adapter tuning are required on both the VIO server and the client logical partition. If a tunable is set in VIO and not in AIX, there may be a significant performance penalty.
- When multiple VIO servers are in use, I/O cannot be load balanced between all VIO servers. A virtual SCSI disk can only be performing I/O operations on a single VIO server.
- If virtual SCSI CDROM devices are mapped to a client logical partition, all devices on that VSCSI adapter must use a block size of 256 kb (0x40000).

Figure 3-12 on page 76 describes a basic Virtual SCSI implementation consisting of four AIX LPARs and two VIOS. The process to present a storage Logical Unit (LUN) to the LPAR as a virtual disk is as follows:

1. Assign the storage LUN to both VIO servers and detect them using **`cfgdev`**.
2. Apply any tunables such as the queue depth and maximum transfer size on both VIOS.
3. Set the LUN’s reserve policy to **`no_reserve`** to enable I/O to enable both VIOS to map the device.

4. Map the device to the desired client LPAR.
5. Configure the device in AIX using **cfgmgr** and apply the same tunables as defined on the VIOS such as queue depth and maximum transfer size.

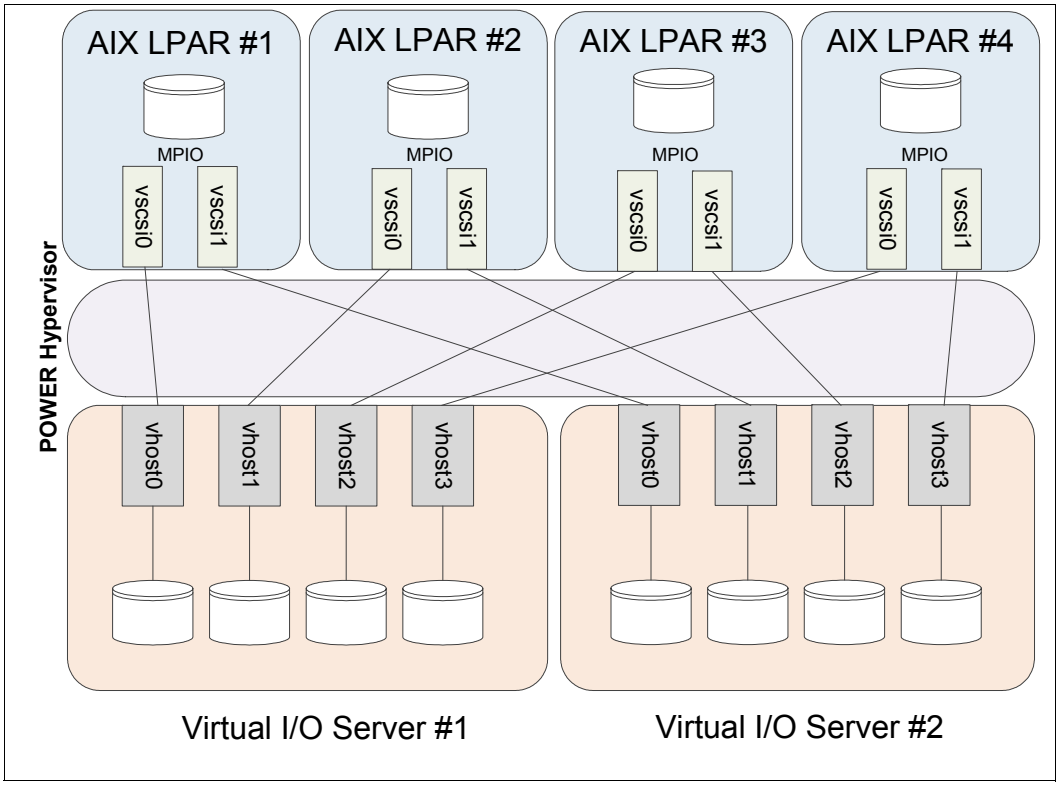


Figure 3-12 Virtual SCSI (VSCSI) overview

Note: This section does not cover how to configure Virtual SCSI. For details on the configuration steps, refer to *IBM PowerVM Virtualization Introduction and Configuration,* SG24-7590-03.

3.6.2 Shared storage pools

Shared storage pools are built on the virtual SCSI provisioning method, with the exception that the VIOS are added to a cluster, with one or more external disk devices (LUNs) assigned to the VIOS participating in the cluster. The LUNs assigned to the cluster must have some backend RAID for availability. Shared storage pools have been available since VIOS 2.2.1.3.

A shared storage pool is then created from the disks assigned to the cluster of VIO servers, and from there virtual disks can be provisioned from the pool.

Shared storage pools are ideal for situations where the overhead of SAN administration needs to be reduced for Power systems, and large volumes from SAN storage can simply be allocated to all the VIO servers. From there the administrator of the Power system can perform provisioning tasks to individual LPARs.

Shared storage pools also have thin provisioning and snapshot capabilities, which also may add benefit if you do not have these capabilities on your external storage system.

The advantages and performance considerations related to the use of shared storage pools are:

Advantages

- There can be one or more large pools of storage, where virtual disks can be provisioned from. This enables the administrator to see how much storage has been provisioned and how much is free in the pool.
- All the virtual disks that are created from a shared storage pool are striped across all the disks in the shared storage pool, reducing the likelihood of hot spots in the pool. The virtual disks are spread over the pool in 64 MB chunks.
- Shared storage pools use cluster-aware AIX (CAA) technology for the clustering, which is also used in IBM PowerHA, the IBM clustering product for AIX. This also means that a LUN must be presented to all participating VIO servers in the cluster for exclusive use as the CAA repository.
- Thin provisioning and snapshots are included in shared storage pools.
- The management of shared storage pools is simplified where volumes can be created and mapped from both the VIOS command line, and the Hardware Management Console (HMC) GUI.

Figure 3-13 on page 78 shows the creation of a virtual disk from shared storage pools. The following is a summary of our setup and the provisioning steps:

- Two VIOS, p24n16 and p24n17, are participating in the cluster.
- The name of the cluster is `bruce`.
- The name of the shared storage pool is `ssp_pool0` and it is 400 GB in size.
- The virtual disk we are creating is 100 GB in size and called `aix2_vdisk1`.
- The disk is mapped via virtual SCSI to the logical partition 750_2_AIX2, which is partition ID 21.
- 750_2_AIX2 has a virtual SCSI adapter mapped to each of the VIO servers, p24n16 and p24n17.
- The virtual disk is thin provisioned.

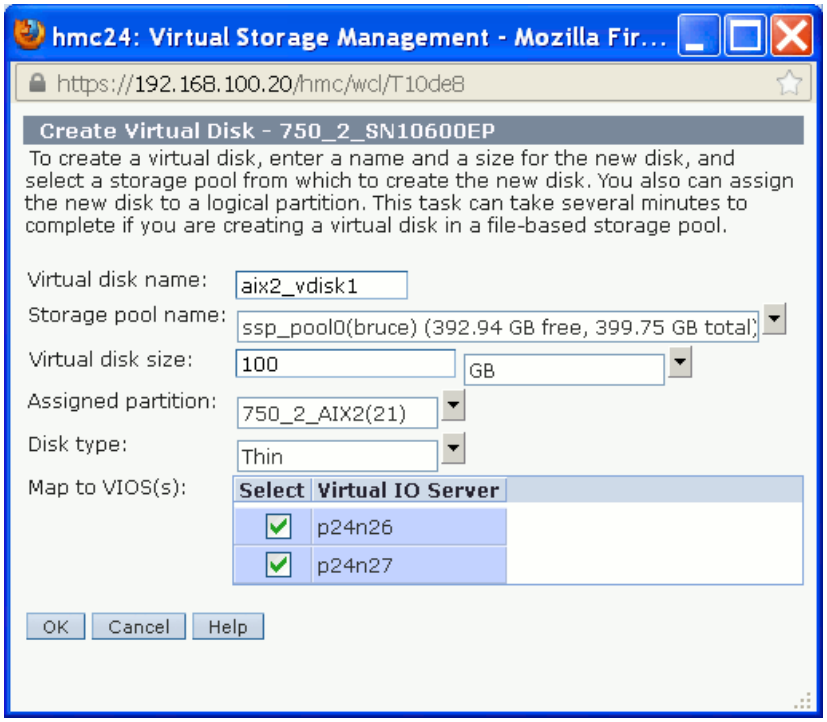


Figure 3-13 Shared storage pool virtual disk creation

Once **OK** is pressed in Figure 3-13, the logical partition `750_2_AIX2` will see a 100 GB virtual SCSI disk drive.

Performance considerations

The performance considerations related to the use of shared storage pools are:

- Ensure that the `max_transfer` and `queue_depth` settings are applied to each LUN in the shared storage pool before the pool is created, or you will need to either bring the pool offline to modify the hdisks in the pool or reboot each of the VIOS participating in the cluster one at a time after applying the change. This must be performed on all VIOS attached to the shared storage pool to ensure the configuration matches. These settings must be able to accommodate the `queue_depth` and `max_transfer` settings you apply on the AIX LPARs using the pool, so some planning is required prior to implementation.
- If the `queue_depth` or `max_transfer` for an hdisk device needs to be changed, all of the hdisk devices should be configured the same, and ideally of the same size on all VIO servers participating in the cluster. For an attribute change to be applied, the shared storage pool needs to be offline on the VIO server where the change is being applied. Ideally, each VIO server would be changed one at a time with the setting applied to take effect at the next reboot. The VIO servers would then be rebooted one at a time.
- Each hdisk device making up the shared storage pool will have its own `queue_depth`. If you find that there are performance issues where the queue is filling up on these disks, you may need to spread the load over more disks by adding more disks to the storage pool. Remember that ideally all disks in the pool will be of the same size, and you cannot resize a disk once it is assigned to the pool.
- There may be some processor overhead on the VIOS, so it is important to regularly monitor processor usage on the VIOS and adjust as needed.
- The `queue_depth` and `max_transfer` settings must still be set on the AIX LPAR. By default the `queue_depth` on a virtual SCSI disk is 3, which is insufficient in most cases.

- I/O cannot be load balanced between multiple VIOS. A virtual SCSI disk backed by a shared storage pool can only be performing I/O operations on a single VIOS.

Figure 3-14 demonstrates, at a high level, the concept of shared storage pools in a scenario with two VIOS and four AIX LPARs.

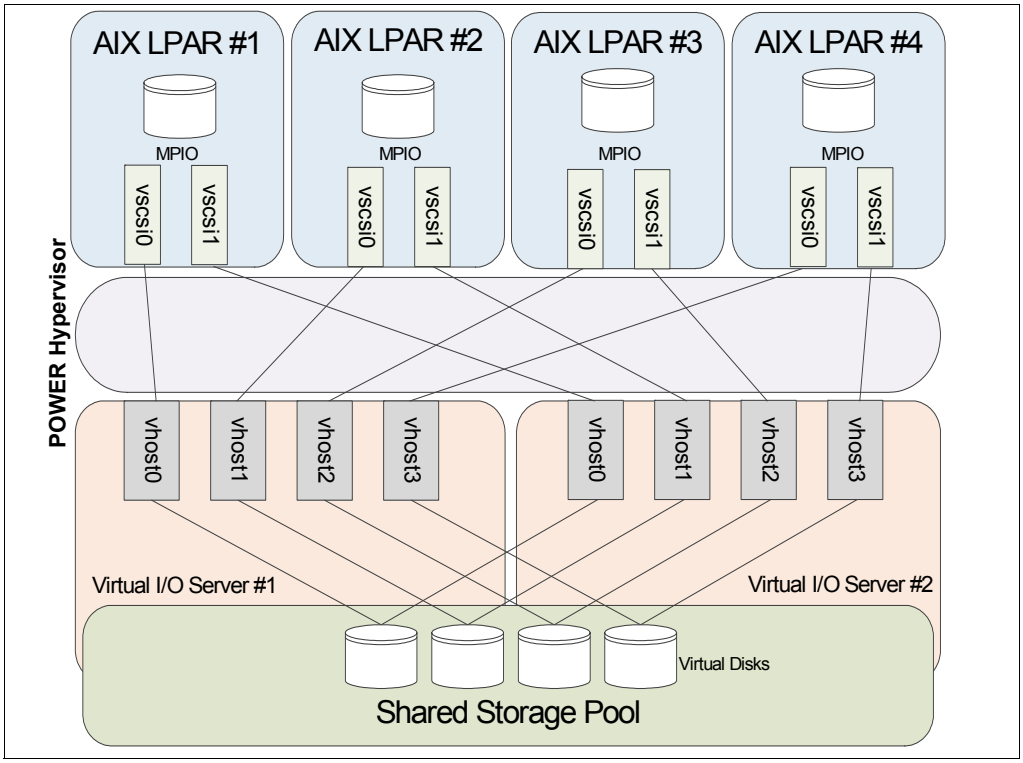


Figure 3-14 Shared storage pool (SSP) overview

Note: This section does not cover how to configure Shared Storage Pools. For details on the full configuration steps, refer to *IBM PowerVM Virtualization Managing and Monitoring,* SG24-7590-03.

3.6.3 N_Port Virtualization

N_Port Virtualization (NPIV) enables a single physical fiber channel port to appear as multiple distinct ports each with its own WWN as if it were a real physical port. VIOS provides the capability to have a single fiber channel port shared by up to 64 virtual fiber channel adapters.

NPIV is typically selected because it reduces administration on the VIOS because they are acting as passthrough devices from the physical fiber channel port on the VIOS to the client LPAR's virtual Fibre Channel adapter. During the initial configuration of NPIV some additional SAN zoning is required. Each virtual WWN belonging to a virtual Fibre Channel adapter needs to be zoned as if it belonged to a physical adapter card. Host connectivity on the storage system is required to be configured as if the client logical partition is a physical server with physical fiber channel adapters.

NPIV requires that the physical Fibre Channel adapter assigned to the VIOS for NPIV use is NPIV capable. At the time of writing only 8 Gb Fibre Channel adapters support NPIV, slower 4 Gb adapters do not.

It is also a requirement that the fabric switch supports NPIV. For Brocade fabric switches NPIV is enabled on a port by port basis, whereas on Cisco fabric switches NPIV needs to be enabled across the whole switch.

The advantages and performance considerations related to the use of NPIV are:

Advantages

- Once the initial configuration is complete, including virtual to physical port mapping on the VIOS, SAN zoning and storage presentation, there is no additional configuration required on the VIO servers. When disks are presented to client logical partitions they are not visible on the VIO server, they are mapped directly to the client logical partition. Once the initial configuration is complete, there is no additional configuration required at the VIOS level to present additional LUNs to a client LPAR.
- Where storage management tools are in use, it is simpler to monitor each client logical partition using NPIV as if it were a physical server. This provides simpler reporting and monitoring, whereas with Virtual SCSI, all the LUNs are mapped to the VIOS. It can be difficult to differentiate which disks are mapped to which client LPAR.
- Snapshot creation and provisioning is simpler on the storage side, because there is no need to map volumes to the VIOS and then map them to client LPARs. If any specific software is required to be installed on the client logical partition for snapshot creation and management, this can be greatly simplified using NPIV.
- When using NPIV the vendor-supplied multipathing drivers are installed on the client LPAR, because AIX will see a vendor-specific disk, not a virtual SCSI disk like in the case of virtual SCSI. This may provide additional capabilities for intelligent I/O queueing and load balancing across paths.

Performance considerations

- When configuring NPIV, the SAN fabric zoning must be correct. The physical WWN of the adapter belonging to the VIOS must not be in the same zone as a virtual WWN from a virtual Fibre Channel adapter.
- The queue depth (`num_cmd_elems`) and maximum transfer (`max_xfer_size`) configured on the virtual fiber channel adapter in AIX, *must* match what is configured on the VIOS.
- Up to 64 virtual clients can be connected to a single physical fiber channel port. This may cause the port to be saturated, so it is critical that there are sufficient ports on the VIOS to support the workload, and the client LPARs must be evenly distributed across the available ports.
- The correct vendor-supplied multipathing driver must be installed on the client logical partition. Any vendor-specific load balancing and disk configuration settings must also be applied.

Figure 3-15 on page 81 demonstrates the concept of NPIV in a scenario with two VIOS and four AIX LPARs.

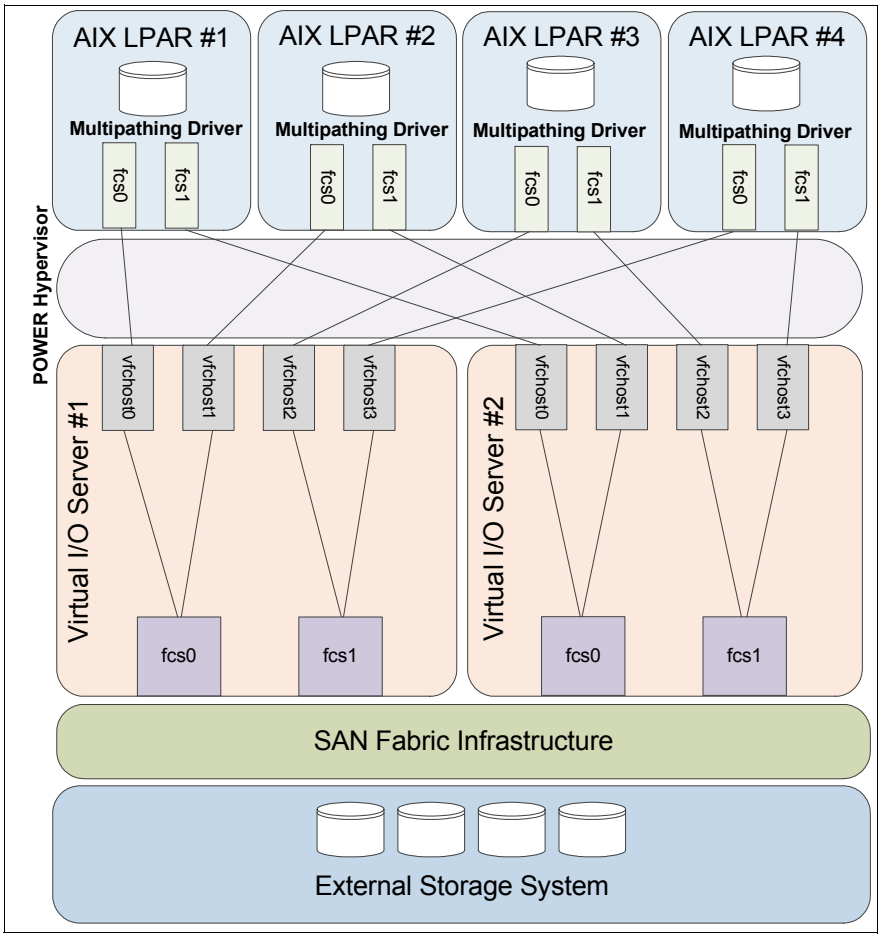


Figure 3-15 NPIV overview

Note: This section does not cover how to configure NPIV. For details on the configuration steps, refer to *IBM PowerVM Virtualization Introduction and Configuration*, SG24-7590-03.

It is also important to note that there are two WWPNs for each virtual fiber channel adapter. Both WWPNs for each virtual adapter need to be zoned to the storage for live partition mobility to work. Only one appears on the SAN fabric at any one time, so one of them needs to be added manually. The two WWPNs for one virtual fiber channel adapter can exist in the same zone. If they do not exist in the same zone, they must be zoned to the same target devices. The physical fiber channel port WWPN does not need to be included in the zone.

Figure 3-16 on page 82 shows the properties of a virtual fiber channel adapter belonging to the LPAR `aix1`.

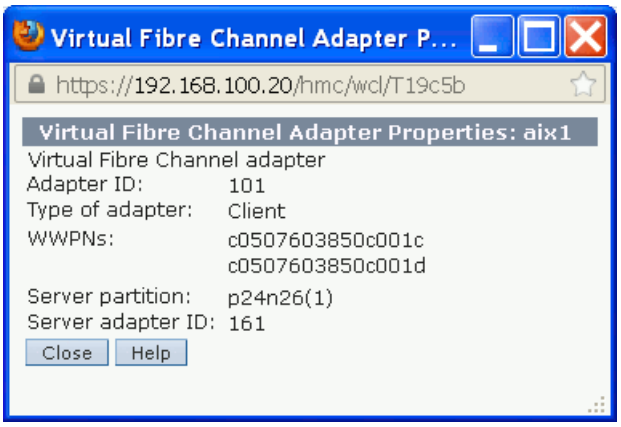


Figure 3-16 Displays WWPNs for a virtual fiber channel adapter

3.6.4 Conclusion

There are different reasons for using each type of disk virtualization method, and in some cases there may be a need to use a combination of the two.

For example, Virtual SCSI provides a virtual disk on the client LPAR using native AIX MPIO drivers. In the event that third party storage is used, it may be beneficial to use NPIV for the non-rootvg disks for performance. However, the rootvg may be presented via virtual SCSI to enable third-party disk device driver updates to be performed without having to reboot the system.

Conversely, you may want to have all of the AIX storage management performed by the VIOS using shared storage pools to reduce SAN and storage administration and provide features such as thin provisioning and snapshots, which may not be present on the external storage system you are using.

If your storage system provides a Quality of Service (QoS) capability, then since client logical partitions using NPIV are treated as separate entities as if they were physical servers on the storage system, it is possible to apply a QoS performance class to them.

From a performance perspective, NPIV typically delivers the best performance on a high I/O workload because it behaves like an LPAR using dedicated I/O adapters with the benefit of virtualization providing enhanced load balancing capabilities.

3.7 Optimal Shared Ethernet Adapter configuration

PowerVM offers the capability to provide a private network using the hypervisor between client LPARs. This isolated network can be bridged to share physical Ethernet resources assigned to a VIOS to allow client LPARs to access an external network. This sharing is achieved with a Shared Ethernet Adapter (SEA).

In this section we provide an overview of different SEA scenarios. We additionally discuss in detail some of the tuning options available to tune performance on the hypervisor network, and SEAs.

In 4.5.1, “Network tuning on 10 G-E” on page 186 we discuss in more detail the performance tuning of 10 gigabit Ethernet adapters.

Refer to *IBM PowerVM Best Practises,* SG24-8062-00, where a number of topics relating to shared Ethernet adapters are discussed that are only briefly covered in this section.

The scenarios in this chapter are illustrated for the purpose of highlighting that there is no single best configuration for a shared network setup. For instance with SEA failover or sharing the configuration is simple and you have VLAN tagging capability. However, you may have a situation where one VIOS is handling the majority or all of the network traffic. Likewise with network interface backup (NIB) there is additional management complexity and configuration. However, you can balance I/O from a single VLAN across both VIOS on an LPAR basis.

3.7.1 SEA failover scenario

Figure 3-17 on page 84 demonstrates four AIX LPARs, using SEA failover between two VIOS. Two LPARs are sending packets tagged with one VLAN ID, while the other pair are tagged with another VLAN ID. In this instance all of the traffic will flow through the first VIOS because it has the lowest bridge priority, and no traffic will go through the second VIOS unless a failover condition occurs.

The advantage of this is that the setup is very simple and enables VLAN tagging. The disadvantage is that all of the traffic will be sent through only one of the VIOS at a time, causing all the processor, memory and I/O load to be only on one VIOS at a time.

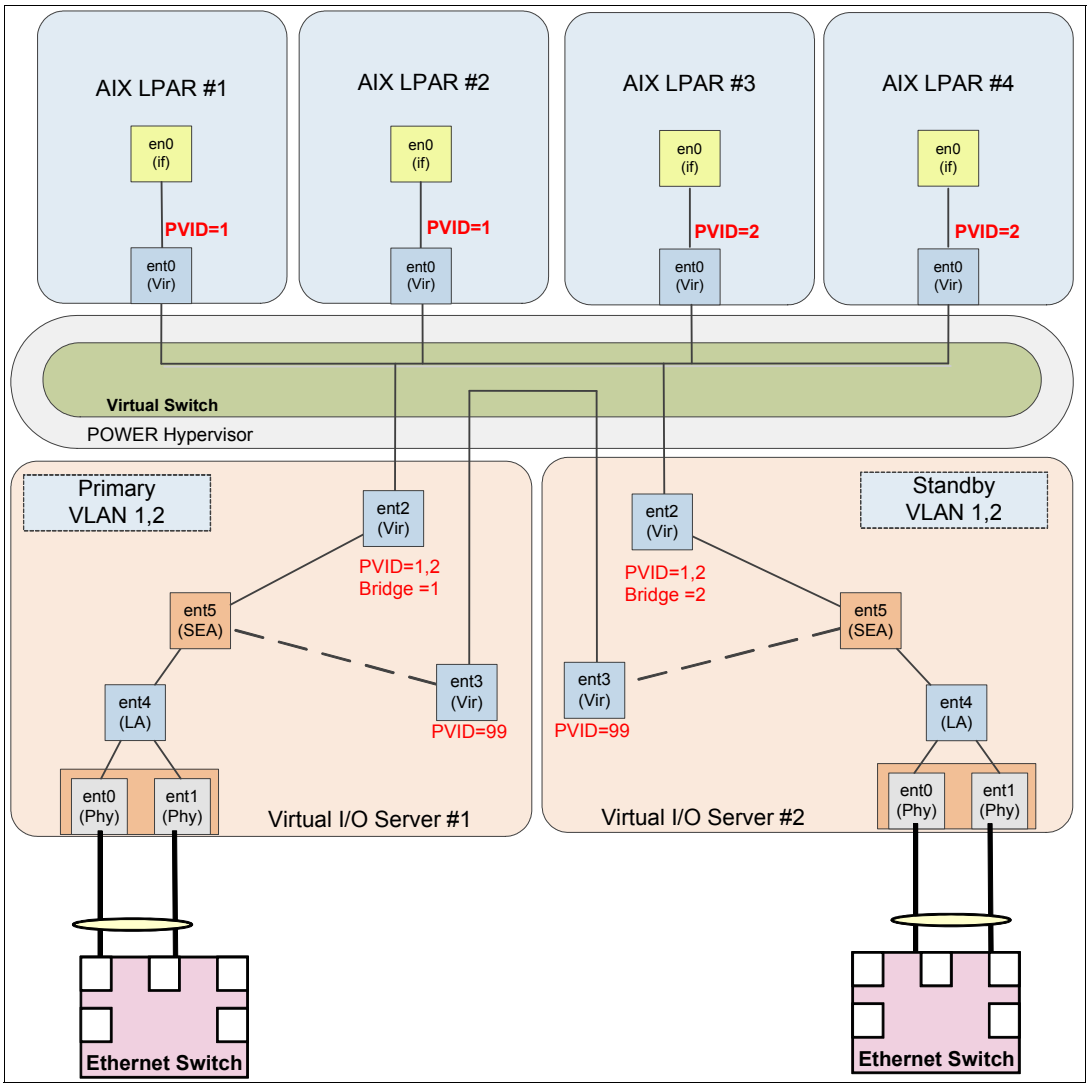


Figure 3-17 SEA failover configuration

Note: This is the simplest way to configure Shared Ethernet and is suitable in most cases. No special configuration is required on the client LPAR side.

3.7.2 SEA load sharing scenario

Figure 3-18 on page 85 demonstrates a feature known as SEA load sharing. This is in effect when there are multiple VLANs and the attribute `ha_mode=sharing` is enabled on the SEA on both VIOS. This option is available in VIOS 2.2.1.0 or later.

It is important to note that VLANs, not packets, are balanced between VIOS. This means that in a two VLAN scenario, one VLAN is active on one VIOS, while the other VLAN is active on the other VIOS. If one VLAN consists of the majority of network traffic, it is important to understand that the VIOS that this VLAN is active on will still be handling the majority of the network traffic.

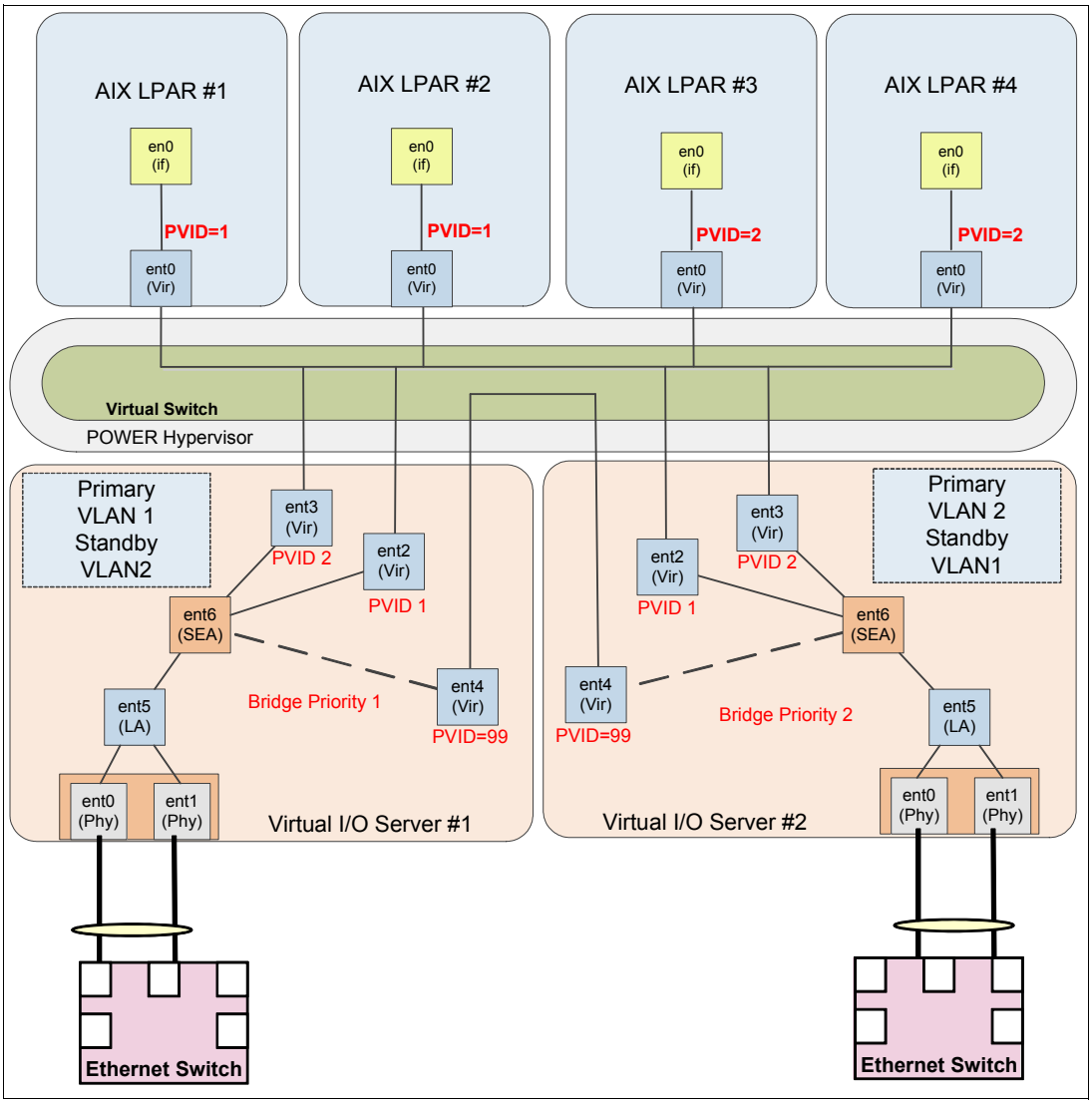


Figure 3-18 SEA load balancing scenario

Note: This method is suitable in cases where multiple VLANs are in use on a POWER System. This method is simple because no special configuration is required on the client LPAR side.

3.7.3 NIB with an SEA scenario

Figure 3-19 on page 86 shows a sample environment using Network Interface Backup (NIB). Typically, in this scenario all of the LPARs are using a single VLAN to the outside network, while internally they have two virtual Ethernet adapters, each with a different VLAN ID. This is used to send network packets to one of two separate SEA adapters, one per VIOS.

With NIB to load balance you could have half of the logical partitions sending traffic primarily to the first VIOS and in the event of a failover the second VIOS would be used, and the reverse is true for the other half of the logical partitions.

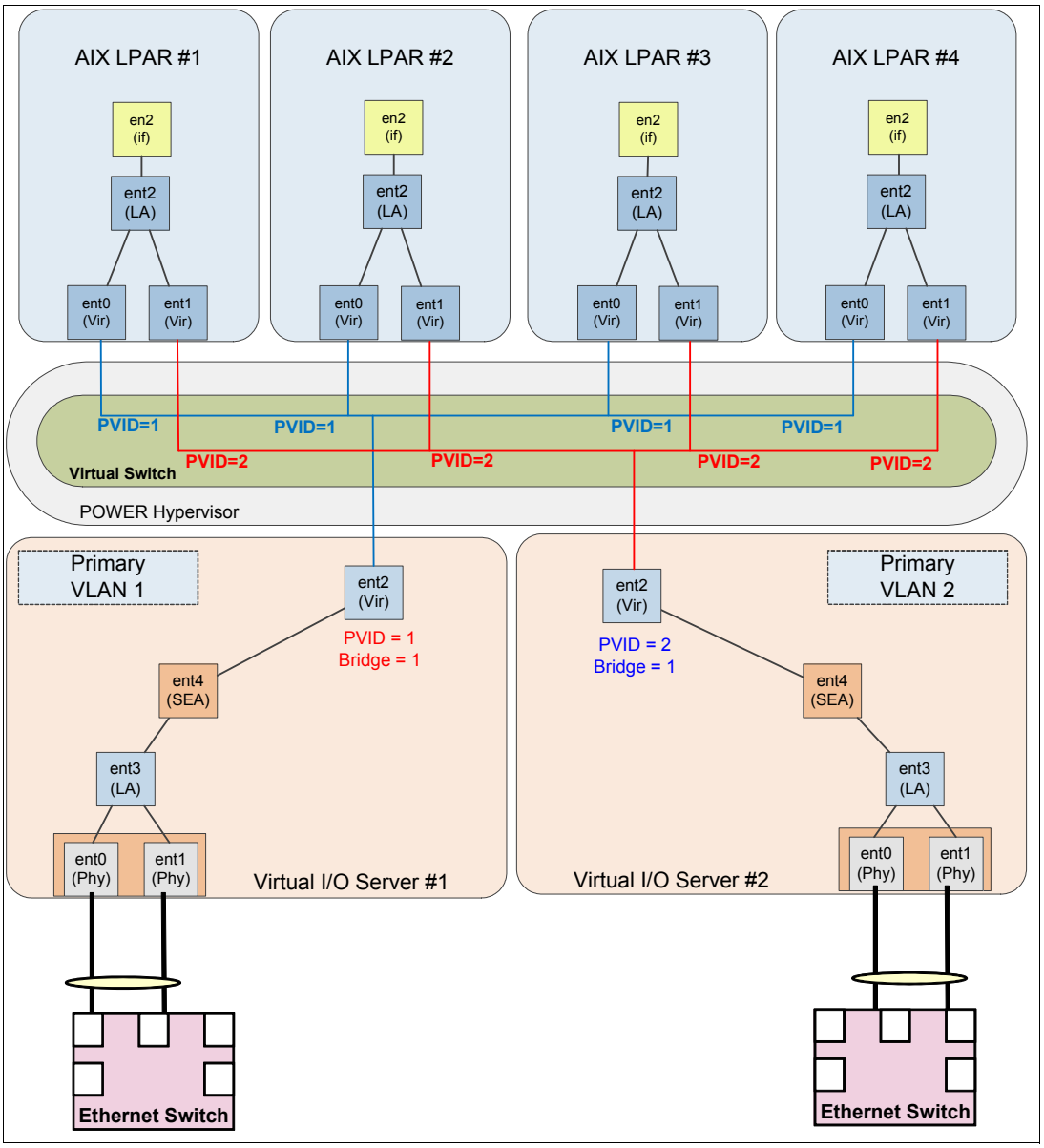


Figure 3-19 NIB scenario

Note: Special configuration is required on the client LPAR side. See 3.7.5, “Etherchannel configuration for NIB” on page 87 for details. VLAN tagging is also not supported in this configuration.

3.7.4 NIB with SEA, VLANs and multiple V-switches

Figure 3-20 on page 87 shows a sample environment using NIB with multiple virtual switches and VLAN tagging. In this scenario, there are two independent shared Ethernet adapters, each configured on a different virtual switch.

With NIB to load balance you could have half of the logical partitions sending traffic primarily to the first virtual switch and in the event of a failover the second virtual switch would be used, and the reverse for the other half of the logical partitions.

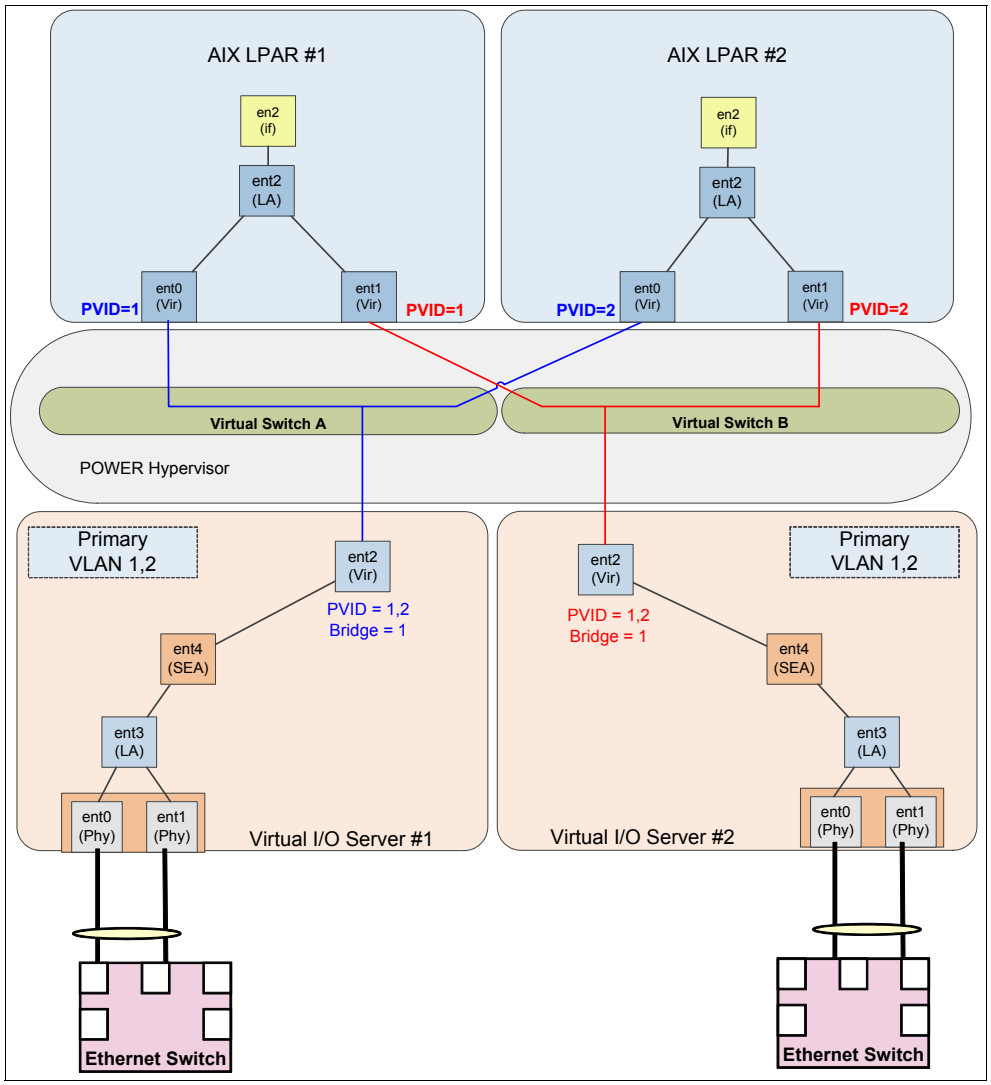


Figure 3-20 NIB with SEA, VLANs and multiple V-switches

Note: Special configuration is required on the client LPAR side. See 3.7.5, “Etherchannel configuration for NIB” on page 87 for details. This is the most complex method of configuring shared Ethernet.

3.7.5 Etherchannel configuration for NIB

In the event that NIB is used on the AIX LPAR using virtual Ethernet, there are some considerations to be made:

- There *must* be an IP address to ping that is outside the Power system. This ensures that the Etherchannel actually fails over.
- The Etherchannel will not fail back to the main channel automatically when the primary adapter recovers. It will only fail back when the backup adapter fails.
- When performing an installation via NIM, the IP address will be on one of the virtual adapters, and no Etherchannel device will exist. The adapters will need to be reconfigured in an Etherchannel configuration.

- It is suggested to balance the traffic per VIO server by having half of the VIO servers using one VIO server as the primary adapter, and the other half using the other VIO server as the backup adapter.

Example 3-26 demonstrates how to configure this feature with four AIX LPARs.

Example 3-26 Configuring NIB Etherchannel in AIX

```
root@aix1:/ #mkdev -c adapter -s pseudo -t ibm_ech -a adapter_names=ent0 -a
backup_adapter=ent1 -a netaddr=192.168.100.1
ent2 Available
root@aix2:/ # mkdev -c adapter -s pseudo -t ibm_ech -a adapter_names=ent1 -a
backup_adapter=ent0 -a netaddr=192.168.100.1
ent2 Available
root@aix3:/ # mkdev -c adapter -s pseudo -t ibm_ech -a adapter_names=ent0 -a
backup_adapter=ent1 -a netaddr=192.168.100.1
ent2 Available
root@aix4:/ # mkdev -c adapter -s pseudo -t ibm_ech -a adapter_names=ent1 -a
backup_adapter=ent0 -a netaddr=192.168.100.1
ent2 Available
```

3.7.6 VIO IP address assignment

For management purposes and the use of Live Partition Mobility (LPM), it is advised that each VIO server has an IP address assigned to it. There are multiple ways that an IP address can be configured:

- A dedicated adapter can be assigned to the VIO server for the management IP address.
- If you have a Host Ethernet Adapter in the Power System, an LHEA port can be used.
- The IP address can be put on the SEA.
- A separate virtual adapter can be assigned to the VIO server for the management IP address.

All of these methods are perfectly valid, and there are some implications and considerations when assigning the IP address.

If you are using LPM, it is suggested to have a separate adapter for LPM if possible. This ensures that the high network usage for LPM does not affect any traffic on the SEA. If this is not possible, ensure that the mover service partitions you are using for the LPM operation are not the VIO servers that are acting as primary in an SEA failover configuration.

From the testing we performed, there was no increase in latency and no decrease in throughput having the IP address on the SEA. This actually gave us two distinct advantages:

- We were able to use the **entstat** command on the SEA.
- We were able to use **topas -E** and collect more detailed SEA statistics with the **-O** option in the **nmon** recordings.

The only downside we found having the IP address for the VIO server on the SEA was that in the event of a failure on the SEA, the VIO server will not be reachable, so a terminal window from the HMC had to be used. However, having the VIO server acting as a client by having the IP address on a separate virtual adapter allowed us to take advantage of the SEA failover and maintain access to our VIO server.

There is no wrong answer for where to configure the IP address on the VIO server. However, depending on your environment there may be some advantages based on where you place the IP address.

3.7.7 Adapter choices

Choosing the number and type of Ethernet adapters for a shared network infrastructure is dependent on the server you have, and the workload you are placing on it.

There are a number of items that should be considered:

- What switch infrastructure exists on the network (1 Gb, 10 Gb)?
- How many gigabit or 10 gigabit adapters are required to supply the workload with sufficient bandwidth?
- Will dual VIOS be employed? If so, will each VIOS have sufficient resources to serve the entire workload in the event that one of the VIOS becomes unavailable?
- Will each VIOS require adapter redundancy? Where multiple adapters are placed in a link aggregation an increased throughput and ability to handle more packets will be gained.
- Is the workload sensitive to latency? If so, what is the latency of the network?
- What quantity and type of adapter slots are present in the server?

It is important to understand how your hardware will be configured to ensure that you will have sufficient resources for your workload. See Chapter 2, “Hardware implementation and LPAR planning” on page 7 for further details.

3.7.8 SEA conclusion

There are multiple ways to configure shared Ethernet on Power systems to provide both performance and redundancy for virtual networks. It is important to consider the method of shared Ethernet to implement, and the VLAN requirements for your environment.

It is also important to ensure that sufficient processor, memory and adapter bandwidth resources are available to your shared Ethernet implementation.

Table 3-10 provides a summary of the different shared Ethernet implementation methods, and when they could be used.

Table 3-10 SEA implementation method summary

Implementation Method	When to use
SEA failover	SEA failover is the typical way to implement shared Ethernet when you have an environment with one or more VLANs with dual VIO servers, and you do not want to have any special configuration on the client LPARs. This is the preferred method when you have a single VLAN. The presumed downside is that one VIO server handles all the traffic. However, you also know that if that VIO server fails, the other VIO server with identical configuration will handle all of the network traffic without degradation so this may not be a downside.

Implementation Method	When to use
SEA failover with load sharing	SEA failover with load sharing is the preferred method when you have two or more VLANs. There is no special configuration required on the client LPAR side and VLANs are evenly balanced across the VIO servers. This balancing is based on the number of VLANs, not on the amount of traffic per VLAN. To force VLANs to use a specific SEA or VIO server, it may be required to use SEA failover with multiple SEA adapters with rotating bridge priorities between the VIO servers for each SEA and different VLANs assigned to each SEA. Where multiple SEAs are in use, it is strongly suggested to have each SEA on a different Vswitch.
NIB with no VLAN tagging	The VIO server configuration for this method is very straightforward because no control channel needs to be configured. However, there is special Ether channel configuration required on the client side. When balancing LPARs between the VIO servers, it is important that no VIO server is busy beyond 50% because a single VIO server may not have enough resources to support all the network traffic. VLAN tagging is not supported using this method.
NIB with Multiple Vswitches and VLAN tagging	This configuration method is more complicated, because multiple virtual switches need to be configured on the Power system, to enable VLAN tagging. There is also a requirement to have Ether channel configured on the client LPAR side. The same sizing requirement applies to ensure that the VIO servers are not busy beyond 50% to ensure that a single VIO server has the resources to support all of the network load.

Note: This section provides guidance on where different SEA configurations can be used. Ensure that the method you choose meets your networking requirements.

3.7.9 Measuring latency

Appreciating the latency in your network, be that between physical machines or adjacent LPARs, can be key in time-sensitive environments.

There are tools such as **tcpdump** available in AIX that provide the capability to measure network latency. It is important to profile latency when there is background traffic on the network, in addition to observing peak load. This will provide you with the perspective to understand whether there is a bottleneck or not.

Example 3-27 shows a sample shell script that can be run on an AIX system to measure the average latency between itself and a routable destination.

It is suggested to use this or something similar to measure the latency between LPARs on the same Power system, to measure latency across the hypervisor and to hosts outside of the physical system, and to measure latency to another system.

Example 3-27 netlatency.sh

```
#!/bin/ksh
usage () {
        MESSAGE=$*
        echo
```

```
        echo "$MESSAGE"
        echo
        echo $0 -i INTERFACE -d dest_ip [ -c nb_packet ]
        exit 3
}

tcpdump_latency () {
        INTERFACE=$1
        DEST_HOST=$2
        COUNT=`echo "$3 * 2" | bc`

        tcpdump -c$COUNT -tti $INTERFACE icmp and host $DEST_HOST  2>/dev/null | awk '
                BEGIN { print "" }
                /echo request/ { REQUEST=$1 ; SEQUENCE=$12 }
                /echo reply/ && $12==SEQUENCE { COUNT=COUNT+1 ; REPLY=$1 ; LATENCY=(REPLY-REQUEST)*1000 ;
                 SUM=SUM+LATENCY ; print "Latency Packet " COUNT " : " LATENCY " ms"}
                END { print ""; print "Average latency (RTT): " SUM/COUNT " ms" ; print""}
                '  &
}
COUNT=10

while getopts ":i:d:c:" opt
do
        case $opt in
                i)    INTERFACE=${OPTARG} ;;
                d)              DEST_HOST=${OPTARG} ;;
                c)              COUNT=${OPTARG} ;;
                \?) usage USAGE
                                        return 1
        esac
done

#########################
# TEST Variable
[ -z "$INTERFACE" ] && usage "ERROR: specify INTERFACE"
[ -z "$DEST_HOST" ] && usage "ERROR: specify Host IP to ping"

#########################
# MAIN

tcpdump_latency $INTERFACE $DEST_HOST $COUNT
sleep 1

OS=`uname`
 case "$OS" in
        AIX) ping -f -c $COUNT -o $INTERFACE $DEST_HOST  > /dev/null ;;
        Linux) ping  -A -c$COUNT -I $INTERFACE $DEST_HOST  > /dev/null ;;
        \?) echo "OS $OS not supported" ;exit 1
esac
exit 0
```

The script output in Example 3-28 shows the round trip latency of each packet and the average latency across the 20 packets. The script was executed with the following parameters:

- **-i** is the interface that we will be sending the traffic out of, in this case ent0.
- **-d** is the target host or device that we are testing latency between. In this case it is another AIX system with the hostname aix2.
- **-c** is the amount of packets we are going to send, in this case 20 packets.

Example 3-28 Latency test

```
root@aix1:/usr/local/bin # ./netlatency.sh -i en0 -d aix2 -c 20

Latency Packet 1 : 0.194788 ms
Latency Packet 2 : 0.0870228 ms
Latency Packet 3 : 0.0491142 ms
Latency Packet 4 : 0.043869 ms
Latency Packet 5 : 0.0450611 ms
Latency Packet 6 : 0.0619888 ms
Latency Packet 7 : 0.0431538 ms
Latency Packet 8 : 0.0360012 ms
```

```
Latency Packet 9 : 0.0281334 ms
Latency Packet 10 : 0.0369549 ms
Latency Packet 11 : 0.043869 ms
Latency Packet 12 : 0.0419617 ms
Latency Packet 13 : 0.0441074 ms
Latency Packet 14 : 0.0400543 ms
Latency Packet 15 : 0.0360012 ms
Latency Packet 16 : 0.0448227 ms
Latency Packet 17 : 0.0398159 ms
Latency Packet 18 : 0.0369549 ms
Latency Packet 19 : 0.0441074 ms
Latency Packet 20 : 0.0491142 ms

Average latency (RTT): 0.0523448 ms
```

The latency between AIX systems or between an AIX system and a device differs depending on network configuration and load on that network.

3.7.10 Tuning the hypervisor LAN

The Power hypervisor is used for network connectivity between client LPARs, as well as client LPAR connectivity to a VIOS for SEA access.

Figure 3-21 gives a simplified example of how a Power system may be configured, and we look closely at the connectivity on VLAN 100, which is simply used for LPAR communications.

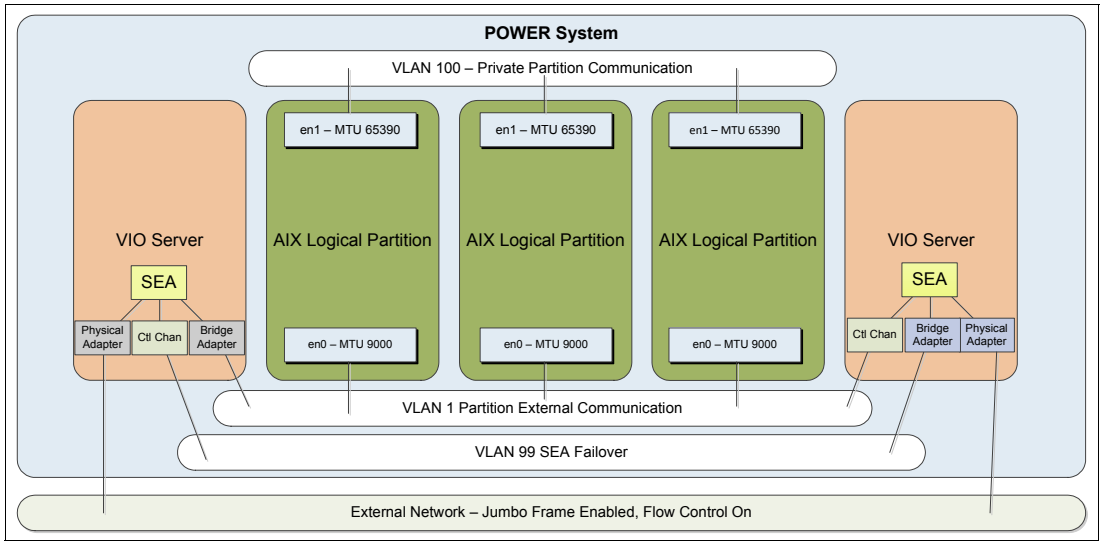


Figure 3-21 Sample configuration with separate VLAN for partition communication

The network may not be capable of accepting network packets with an MTU close to 64 k; so perhaps the VLAN for external communication on the Power system may have an MTU of 9000, and Jumbo Frames are enabled on the external network where we use a separate IP range on a different VLAN for partition communications. This can be particularly useful if one of the logical partitions on the Power system is a backup server LPAR, for example Tivoli Storage Manager (TSM) or a NIM server.

In Example 3-29 we can run a simple test using the **netperf** utility to perform a simple and repeatable bandwidth test between en0 in the aix1 LPAR and en0 on the aix2 LPAR in Figure 3-11 on page 65. The test duration will be 5 minutes.

Example 3-29 How to execute the netperf load

```
root@aix1:/ # netperf -H 192.168.100.12 -l 600
```

At this point in the example, all of the default AIX tunables are set. We can see in Figure 3-22 that the achieved throughput on this test was 202.7 megabytes per second.

```
Network
I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Re
    en0    2017.3  207611.9  39726.4 140491.8   52.0 1513.2     2281
    lo0       0.0       0.0      0.0      0.0    0.0    0.0    14954
  Total       2.0     202.7 in Mbytes/second   Overflow=0
I/F Name  MTU  ierror oerror collision Mbits/s Description
    en0   1500      0      0      0  10240 Standard Ethernet Network
    lo0  16896      0      0      0      0 Loopback Network Interfac
```

Figure 3-22 Network throughput with default tunables and a single netperf stream

For the next test, we changed some tunables on the en0 interface utilizing the hypervisor network and observed the results of the test.

Table 3-11 describes some of the tunables that were considered prior to performing the test.

Table 3-11 AIX network tunables considered

Tunable	Description	Value
mtu size	Media Transmission Unit (MTU) size is the largest packet that AIX will send. Increasing the mtu size will typically increase performance for streaming workloads. The value 64390 is the maximum value minus VLAN overhead.	65390 (for large throughput)
flow control	Flow control is a TCP technique which will match the transmission rate of the sender with the transmission rate of the receiver. This is enabled by default in AIX.	on
large send	The TCP large send offload option enables AIX to build a TCP message up to 64 KB in size for transmission.	on
large receive	The TCP large receive offload option enables AIX to aggregate multiple received packets into a larger buffer reducing the amount of packets to process.	on
rfc1323	This tunable, when set to 1, enables TCP window scaling when both ends of a TCP connection have rfc1323 enabled.	1
tcp send or receive space	These values specify how much data can be buffered when sending or receiving data. For most workloads the default of 16384 is sufficient. However, in high latency situations these values may need to be increased.	16384 is the default, increasing to 65536 may provide some increased throughput.

Tunable	Description	Value
checksum offload	This option allows the network adapter to compute the TCP checksum rather than the AIX system performing the computation. This is only valid for physical adapters.	yes
dcbflush_local	Data Cache Block Flush (dcbflush) is an attribute for a virtual Ethernet adapter that allows the virtual Ethernet device driver to flush the processor's data cache of any data after it has been received.	yes

These tunables are discussed in more detail in the AIX 7.1 Information Center at:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp

Note: It is important to try tuning each of these parameters individually and measuring the results. Your results may vary from the tests performed in this book. It is also expected that changes occur in processor and memory utilization as a result of modifying these tunables.

Example 3-30 demonstrates the tuning changes we made during the test. These changes included:

- Increasing the MTU size on both AIX LPARs from 1500 to 64390.
- Enabling largesend using the **mtu_bypass** option.
- Enabling Data Cache Block Flush with the **dcbflush_local** option. Note that the interface had to be down for this change to be applied.
- Enabling **rfc1323** to take effect and to be persistent across reboots.

Example 3-30 Apply tunables to AIX logical partitions

```
root@aix1:/ # chdev -l en0 -a mtu=65390
en0 changed
root@aix1:/ # chdev -l en0 -a mtu_bypass=on
en0 changed
root@aix1:/ # chdev -l en0 -a state=down
en0 changed
root@aix1:/ # chdev -l en0 -a state=detach
en0 changed
root@aix1:/ # chdev -l ent0 -a dcbflush_local=yes
ent0 changed
root@aix1:/ # chdev -l en0 -a state=up
en0 changed
root@aix1:/ # no -p -o rfc1323=1
Setting rfc1323 to 1
Setting rfc1323 to 1 in nextboot file
Change to tunable rfc1323, will only be effective for future connections
root@aix1:/ #

root@aix2:/ # chdev -l en0 -a mtu=65390
en0 changed
root@aix2:/ # chdev -l en0 -a mtu_bypass=on
en0 changed
root@aix2:/ # chdev -l en0 -a state=down
en0 changed
root@aix2:/ # chdev -l en0 -a state=detach
```

```
en0 changed
root@aix2:/ # chdev -l ent0 -a dcbflush_local=yes
ent0 changed
root@aix2:/ # chdev -l en0 -a state=up
en0 changed
root@aix2:/ # no -p -o rfc1323=1
Setting rfc1323 to 1
Setting rfc1323 to 1 in nextboot file
Change to tunable rfc1323, will only be effective for future connections
root@aix2:/ #
```

Note: Example 3-30 on page 94 requires that the en0 interface is down for some of the settings to be applied.

If the **mtu_bypass** option is not available on your adapter, run the tunable as follows instead; however, this change is not persistent across reboots. You need to add this to `/etc/rc.net` to ensure that **largesend** is enabled after a reboot (Example 3-31).

Example 3-31 Enable largesend with ifconfig

```
root@aix1:/ # ifconfig en0 largesend
root@aix1:/ #
root@aix2:/ # ifconfig en0 largesend
root@aix2:/ #
```

Figure 3-23 shows the next netperf test performed in exactly the same way as Example 3-29 on page 93. It is noticeable that this test delivered over a 7x improvement in throughput.

```
Network
I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Re
    en0     645.0 1544496.6  12700.9  25718.4   52.0 61495.5      64
    lo0       0.0       0.0      0.0      0.0    0.0    0.0        0
  Total       0.6    1508.3 in Mbytes/second   Overflow=0
I/F Name  MTU  ierror oerror collision Mbits/s Description
    en0  65390      0      0      0  10240 Standard Ethernet Network
    lo0  16896      0      0      0      0 Loopback Network Interfac
```

Figure 3-23 Network throughput with modified tunables and a single netperf stream

Figure 3-24 shows additional netperf load using additional streams to deliver increased throughput, demonstrating that the capable throughput is dependant on how network intensive the workload is.

```
Network
I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Re
    en0    1593.4 3820849.4  31377.1  64580.0   52.0 60584.5     159
    lo0       0.0       0.0      0.5      0.5   71.0   71.0        0
  Total       1.6    3731.3 in Mbytes/second   Overflow=0
I/F Name  MTU  ierror oerror collision Mbits/s Description
    en0  65390      0      0      0  10240 Standard Ethernet Network
    lo0  16896      0      0      0      0 Loopback Network Interfac
```

Figure 3-24 Network throughput with modified tunables again, but with additional netperf load

3.7.11 Dealing with dropped packets on the hypervisor network

When load on a virtual Ethernet adapter is heavy, there is a situation that will occur when two items become an issue:

- Latency increased across the hypervisor network between logical partitions. "Etherchannel configuration for NIB" on page 87 describes network latency. It is important to monitor the latency of the network.
- The virtual Ethernet adapter's receive buffers are exhausted and packets will be retransmitted and thoughput will decrease. This is shown in Example 3-32 where the aix1 LPAR is experiencing dropped packets.
  - **Packets Dropped** is the total amount of packets that could not be received by the aix1 LPAR.
  - **No Resource Errors** is the total number of times that the aix1 LPAR was unable to receive any more packets due to lack of buffer resources.
  - **Hypervisor Receive Failures** is the total number of packets the hypervisor could not deliver to because the receive queue was full.
  - **Hypervisor Send Failures** is the total number of times that a packet could not be sent due to a buffer shortage.

Example 3-32 The netstat -v output demonstrating dropped packets

```
root@aix1:/ # netstat -v ent0
-------------------------------------------------------------
ETHERNET STATISTICS (ent0) :
Device Type: Virtual I/O Ethernet Adapter (l-lan)
Hardware Address: 52:e8:7f:a2:19:0a
Elapsed Time: 0 days 0 hours 35 minutes 48 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 76773314                             Packets: 39046671
Bytes: 4693582873534                          Bytes: 45035198216
Interrupts: 0                                 Interrupts: 1449593
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 8184
                                              Bad Packets: 0
Max Packets on S/W Transmit Queue: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 14                         Broadcast Packets: 4474
Multicast Packets: 8                          Multicast Packets: 260
No Carrier Sense: 0                           CRC Errors: 0
DMA Underrun: 0                               DMA Overrun: 0
Lost CTS Errors: 0                            Alignment Errors: 0
Max Collision Errors: 0                       No Resource Errors: 8184
Late Collision Errors: 0                      Receive Collision Errors: 0
Deferred: 0                                   Packet Too Short Errors: 0
SQE Test: 0                                   Packet Too Long Errors: 0
Timeout Errors: 0                             Packets Discarded by Adapter: 0
Single Collision Count: 0                     Receiver Start Count: 0
Multiple Collision Count: 0
Current HW Transmit Queue Length: 0
```

```
General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 20000
Driver Flags: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        DataRateSet

Virtual I/O Ethernet Adapter (l-lan) Specific Statistics:
---------------------------------------------------------
RQ Length: 4481
Trunk Adapter: False
Filter MCast Mode: False
Filters: 255
  Enabled: 2  Queued: 0  Overflow: 0
LAN State: Operational

Hypervisor Send Failures: 2090
  Receiver Failures: 2090
  Send Errors: 0
Hypervisor Receive Failures: 8184

Invalid VLAN ID Packets: 0

ILLAN Attributes: 0000000000003002 [0000000000003002]

Port VLAN ID:     1
VLAN Tag IDs:  None


Switch ID: ETHERNET0

Hypervisor Information
  Virtual Memory
    Total (KB)                 79
  I/O Memory
    VRM Minimum (KB)          100
    VRM Desired (KB)          100
    DMA Max Min (KB)          128

Transmit Information
  Transmit Buffers
    Buffer Size             65536
    Buffers                    32
    History
      No Buffers                0
  Virtual Memory
    Total (KB)              2048
  I/O Memory
    VRM Minimum (KB)         2176
    VRM Desired (KB)        16384
    DMA Max Min (KB)        16384
```

```
Receive Information
  Receive Buffers
    Buffer Type              Tiny    Small   Medium    Large     Huge
    Min Buffers               512      512      128       24       24
    Max Buffers              2048     2048      256       64       64
    Allocated                 512      512      156       24       64
    Registered                511      512      127       24       18
    History
      Max Allocated           512      512      165       24       64
      Lowest Registered       511      510      123       22       12
  Virtual Memory
    Minimum (KB)              256     1024     2048      768     1536
    Maximum (KB)             1024     4096     4096     2048     4096
  I/O Memory
    VRM Minimum (KB)         4096     4096     2560      864     1632
    VRM Desired (KB)        16384    16384     5120     2304     4352
    DMA Max Min (KB)        16384    16384     8192     4096     8192

I/O Memory Information
  Total VRM Minimum (KB)    15524
  Total VRM Desired (KB)    61028
  Total DMA Max Min (KB)    69760

root@aix1:/ #
```

Under *Receive Information* in the **netstat -v** output in Example 3-32 on page 96, the type and number of buffers are listed. If at any point the Max Allocated under history reaches the max Buffers in the **netstat -v** output, it may be required to increase the buffer size to help overcome this issue.

Our **max_buf_huge** was exhausted due to the nature of the netperf streaming workload. The buffers which may require tuning are very dependant on workload and it is advisable to tune these only under the guidance of IBM support. Depending on the packet size and number of packets, different buffers may need to be increased. In our case it was large streaming packets, so only *huge buffers* needed to be increased.

Example 3-33 demonstrates how to increase the huge buffers for the ent0 interface. The en0 interface will need to be brought down for this change to take effect.

Example 3-33 How to increase the virtual Ethernet huge buffers

```
root@aix1:/ # chdev -l en0 -a state=down
en0 changed
root@aix1:/ # chdev -l en0 -a state=detach
en0 changed
root@aix1:/ # chdev -l ent0 -a min_buf_huge=64 -a max_buf_huge=128
ent0 changed
root@aix1:/ # chdev -l en0 -a state=up
en0 changed
root@aix1:/ #
```

Note: We suggest to review the processor utilization before making any changes to the virtual Ethernet buffer tuning. Buffers should only be tuned if the *allocated* buffers reaches the *maximum* buffers. If in doubt, consult with IBM support.

3.7.12 Tunables

Typically, VIOS are deployed in pairs, and when Ethernet sharing is in use each VIOS has a physical adapter that acts as a bridge for client LPARs to access the outside network.

Physical tunables

It is important to ensure that the physical resources that the shared Ethernet adapter is built on top of are configured for optimal performance. 4.5.1, “Network tuning on 10 G-E” on page 186 describes in detail how to configure physical Ethernet adapters for optimal performance.

EtherChannel tunables

When creating a Link Aggregation that a SEA is built on top of, it is important to consider the options available when configuring the EtherChannel device.

There are a number of options available when configuring aggregation; we suggest to consider the following:

- **mode** - This is the EtherChannel mode of operation. A suggested value is 8023ad.
- **use_jumbo_frame** - This enables Gigabit Ethernet Jumbo Frames.
- **hash_mode** - This determines how the outgoing adapter is chosen. A suggested value is src_dst_port.

Example 3-34 demonstrates how to create a link aggregation using these options.

Example 3-34 Creation of the link aggregation

```
$ mkvdev -lnagg ent1,ent2 -attr mode=8023ad hash_mode=src_dst_port
use_jumbo_frame=yes
ent5 Available
en5
et5
$
```

SEA tunables

When creating an SEA it is important to consider the options available to improve performance on the defined device.

Options that should be considered are:

- **jumbo_frames** - This enables gigabit Ethernet jumbo frames.
- **large_receive** - This enables TCP segment aggregation,
- **largesend** - This enables hardware transmit TCP resegmentation.

Example 3-35 demonstrates how to create a shared Ethernet adapter on top of the ent5 EtherChannel device using ent3 as the bridge adapter and ent4 as the control channel adapter.

Example 3-35 Creation of the shared Ethernet adapter

```
$ mkvdev -sea ent5 -vadapter ent3 -default ent3 -defaultid 1 -attr ha_mode=auto
ctl_chan=ent4 jumbo_frames=yes large_receive=yes largesend=1
ent6 Available
```

```
en6
et6
$
```

Monitoring and accounting

Accounting can be enabled for an SEA that at the time of writing is not enabled by default. It is suggested that this option be enabled to allow use of **seastat** to report SEA-related statistics.

Example 3-36 demonstrates how to enable SEA accounting.

Example 3-36 Enabling SEA accounting

```
$ lsdev -type adapter |grep "Shared Ethernet Adapter"
ent6             Available   Shared Ethernet Adapter
$ chdev -dev ent6 -attr accounting=enabled
ent6 changed
$ lsdev -dev ent6 -attr |grep accounting
accounting   enabled  Enable per-client accounting of network statistics
True
$
```

Any tunables that have been applied to the SEA on the VIOS, the adapter or devices it is defined onto, *must match the switch configuration*. This includes but is not limited to:

- EtherChannel mode; for example, 8023.ad
- Jumbo frames
- Flow control

3.8 PowerVM virtualization stack configuration with 10 Gbit

The PowerVM virtualization stack (Figure 3-25 on page 101) consists of the Virtual I/O Server with Shared Ethernet Adapter (SEA) backed by physical Ethernet adapters with or without link aggregation (Etherchannels), virtual Ethernet trunk adapters, and AIX or Linux or IBM i partitions with virtual Ethernet adapters.

Between the virtual Ethernet adapters are the hypervisor virtual switches.

Beyond the physical Ethernet adapters are the actual physical network, with switches, routers, and firewalls, all of which impact network throughput, latency and round trip times.

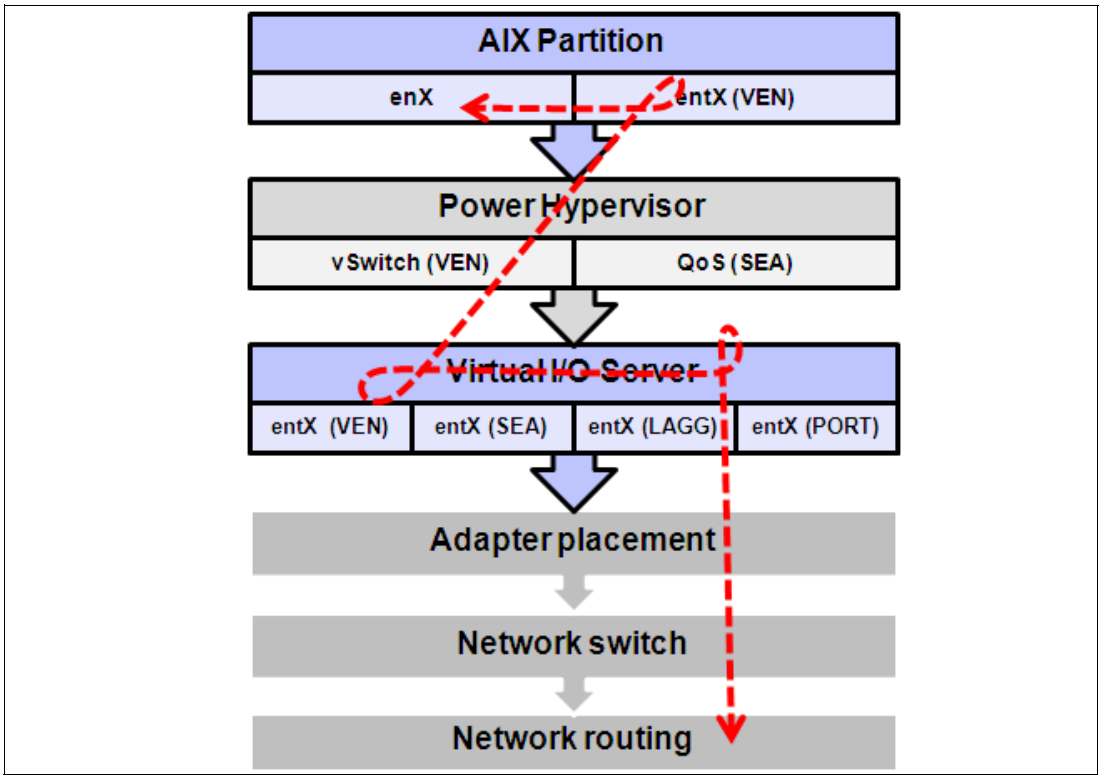


Figure 3-25 PowerVM virtualization stack overview

In some network environments, network and virtualization stacks, and protocol endpoint devices, other settings might apply.

Note: Apart from LRO, the configuration is also applicable for 1 Gbit.

Gigabit Ethernet and VIOS SEA considerations:

1. For optimum performance, ensure adapter placement according to Adapter Placement Guide, and size VIOS profile with sufficient memory and processing capacity to fit the expected workload, such as:
   - No more than one 10 Gigabit Ethernet adapter per I/O chip.
   - No more than one 10 Gigabit Ethernet port per two processors in a system.
   - If one 10 Gigabit Ethernet port is present per two processors in a system, no other 10 Gb or 1 Gb ports should be used.
2. Each switch port
   - Verify that flow control is enabled.
3. On each physical adapter port in the VIOS (ent).chksum_offload enabled (default)
   - flow_ctrl enabled (default)
   - large_send enabled (preferred)
   - large_receive enabled (preferred)
   - jumbo_frames enabled (optional)
   - Verify Adapter Data Rate for each physical adapter (entstat -d/netstat -v)
4. On the Link Aggregation in the VIOS (ent)
   - Load Balance mode (allow the second VIOS to act as backup)
   - hash_mode to src_dst_port (preferred)

- mode to 8023ad (preferred)
- use_jumbo_frame enabled (optional)
- Monitor each physical adapter port with entstat command to determine the selected hash_mode effectiveness in spreading the outgoing network load over the link aggregated adapters

5. On the SEA in the VIOS (ent)
   - largesend enabled (preferred)
   - jumbo_frames enabled (optional)
   - netaddr set for primary VIOS (preferred for SEA w/failover)
     - Use base VLAN (tag 0) to ping external network address (beyond local switch).
     - Do not use switch or router virtual IP address to ping (if its response time might fluctuate).
   - Consider disabling SEA thread mode for SEA only VIOS.
   - Consider implementing VLAN load sharing.
     - http://pic.dhe.ibm.com/infocenter/powersys/v3r1m5/index.jsp?topic=/p7hb1/iphb1_vios_scenario_sea_load_sharing.htm
     - http://www-01.ibm.com/support/docview.wss?uid=isg3T7000527
6. On the virtual Ethernet adapter in the VIOS (ent)
   - chksum_offload enabled (default)
   - Consider enabling dcbflush_local
   - In high load conditions, the virtual Ethernet buffer pool management of adding and reducing the buffer pools on demand can introduce latency of handling packets (and can result in drops of packets, “Hypervisor Receive Failures”).
     - Setting the “Min Buffers” to the same value as “Max Buffers” allowed will eliminate the action of adding and reducing the buffer pools on demand. However, this will use more pinned memory.
     - For VIOS in high end servers, you might also have to increase the max value to its maximum allowed, and then increase the min value accordingly. Check the maximum value with the **`lsattr`** command, such as: **`lsattr -Rl ent# -a max_buf_small`**
     - Max buffer sizes: Tiny (4096), Small (4096), Medium (2048), Large (256), Huge (128)
7. On the virtual Ethernet adapter in the virtual client/partition (ent)
   - chksum_offload enabled (default)
     - Monitoring utilization with enstat -d or netstat -v and if “Max Allocated” is higher than “Min Buffers”, increase to higher value than “Max Allocated” or to “Max Buffers”, for example: Increase the "Min Buffers“ to be greater than "Max Allocated" by increasing it up to the next multiple of 256 for "Tiny" and "Small" buffers, by the next multiple of 128 for "Medium" buffers, by the next multiple of 16 for "Large“ buffers, and by the next multiple of 8 for "Huge" buffers.
8. On the virtual network interface in the virtual client/partition (en)
   - mtu_bypass enabled
     - Is the largesend attribute for virtual Ethernet (AIX 6.1 TL7 SP1 or AIX7.1 SP1)
     - If not available, set with the **`ifconfig`** command after each partition boot in `/etc/rc.net` or equiv by init, for example: **`ifconfig enX largesend`**
   - Use the device driver built-in interface specific network options (ISNO)
     - ISNO is enabled by default (the no tunable use_isno).
     - Device drivers have default settings, leave the default values intact.
     - Check current settings with the **`ifconfig`** command.

- Change with the **chdev** command.
- Can override with the **ifconfig** command or setsockopt() options.

– Set mtu to 9000 if using jumbo frames (network support required)

- Default mtu is 1500 (Maximum Transmission Unit/IP)
- Default mss is 1460 (Maximum segment Size/TCP) with RFC1323 disabled
- Default mss is 1448 (Maximum segment Size/TCP) with RFC1323 enabled

– Consider enabling network interface thread mode (dog thread)

- Set with the **ifconfig** command, for example: **ifconfig enX thread**
- Check utilization with the **netstat** command: **netstat -s| grep hread**
- For partitions with dozens of VPs, review the no tunable ndogthreads
- http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/enable_thread_usage_lan_adapters.htm

A note on monitoring adapter port transmit statistics to determine how the actual workload spreads the network traffic over link aggregated (Etherchanneled) adapter ports.

Use the **entstat** command (or **netstat -v**) and summarize, as in Table 3-12. In this case we deploy an adapter port link aggregation in 8023ad mode using default hash_mode.

The **lsattr** command:

```
adapter_names   ent0,ent1,ent4,ent6 EtherChannel Adapters
hash_mode       default             Determines how outgoing adapter is chosen
mode            8023ad              EtherChannel mode of operation
```

The statistics in this case show that the majority of the outgoing (transmit) packets go out over ent6, and approximately 1/3 of the total packets go out over ent4, with ent0 and ent1 practically unused for outgoing traffic (the receive statistics are more related to load balancing from the network side, and switch MAC tables and trees).

Table 3-12 Etherchannel/Link Aggregation statistics with hash_mode default

Device	Transmit packets	% of total	Receive packets	% of total
ent0	811028335	3%	1239805118	12%
ent1	1127872165	4%	2184361773	21%
ent4	8604105240	28%	2203568387	21%
ent6	19992956659	65%	4671940746	45%
Total	29408090234	100%	8115314251	100%

3.9 AIX Workload Partition implications, performance and suggestions

Workload Partitions (WPARs) were introduced with AIX 6.1 in 2007. A WPAR is a software implementation of partitioning provided just by the operating system. The WPAR components have been regularly enhanced with subsequent AIX releases, including IPv6 and NPIV support. AIX 7.1 also introduced the notable addition of Versioned WPARs. This feature allows both AIX 5.2 and 5.3 to be hosted in a WPAR.

3.9.1 Consolidation scenario

This first scenario uses WPARs as a consolidation vehicle to collapse three LPARs into one larger LPAR. We demonstrate the sequence to migrate from LPAR to WPAR and discuss our observations about sizing and performance that should be considered for such implementations.

Our scenario begins with three AIX 7.1 TL02 LPARs hosted on a Power 750 server. Figure 3-26 provides a high-level view of this scenario.

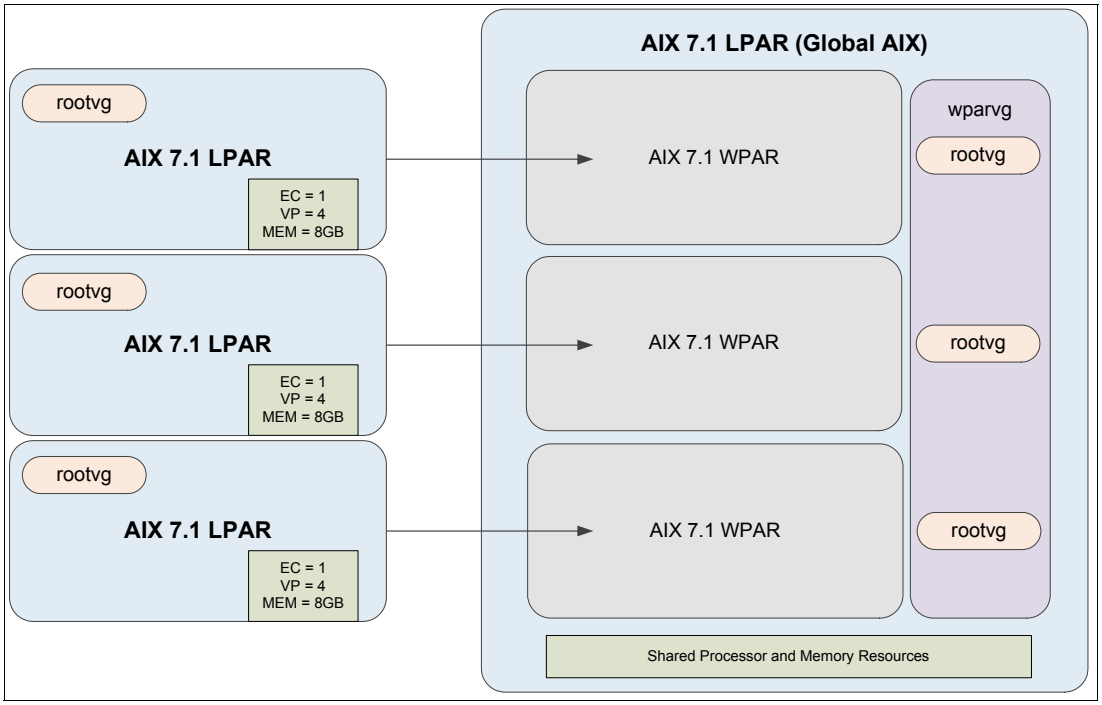


Figure 3-26 LPAR to WPAR scenario

The three LPARs were indentical in size. Table 3-13 details the LPAR resource configuration. The LPARs were generically defined for the test case—they were not sized based on their hosted workload footprint.

Table 3-13 LPAR configuration of consolidation candidates

CPU	RAM	Storage
4 VPs, EC 1.0, uncapped	8 GB (dedicated)	60 GB (via vSCSI)

Each LPAR hosted a deployment of our WebSphere Message Broker sample application. However, the application was configured differently on each LPAR to give a footprint; the sample application was configured to run with one application thread on LPAR1, two on LPAR2, and eight on LPAR4. So while the hosted application was the same, we were consolidating three different footprints.

A fourth LPAR was created with the same allocation of four VPs and 8 GB, but with additional storage. A secondary volume group was created of 120 GB to host the WPARs. This separation from rootvg was implemented to avoid any unnecessary contention or background noise.

For each LPAR, we ran the sample application for 10 minutes to obtain a baseline TPS. The applications were quiesced and a clean **mksysb** backup taken of each LPAR.

After transferring the mksysb files to the fourth LPAR, we used a new feature of the **mkwpar** command introduced with AIX 7.1 TL02. The additional functionality introduces a variant of a System WPAR called a System Copy WPAR. It allows a System WPAR to be created from a mksysb; so the feature is similar in operation to the creation of a Versioned WPAR.

Note: For reference the mksysb can be as old as AIX 4.3.3, but part of the deployment process requires the created WPAR to be synchronized to the level of the hosting Global system before it can be started.

Example 3-37 shows the **mkwpar** command used to create one of the System Copy WPARs.

Example 3-37 mkwpar command

```
# mkwpar -n WPAR1 -g wparvg -h p750s2aix2wp4 -N interface=en0
address=192.168.100.100 netmask=255.255.255.0 -A -s -t -B /export/mksysb_LPAR1
```

Parameters of interest are -g, which overrides the hosting volume group (the default is rootvg); -t, which informs the command to copy rootvg from a system backup specified by the subsequent -B flag.

The process was repeated to create a second and third WPAR. No resource controls were implemented on any of the WPARs. Example 3-38 shows the output from the **lswpar** command after all three were created.

Example 3-38 lswpar command

```
# lswpar
Name   State  Type  Hostname       Directory     RootVG WPAR
------------------------------------------------------------
WPAR1  A      S     p750s2aix2wp1  /wpars/WPAR1  no
WPAR2  A      S     p750s2aix2wp2  /wpars/WPAR2  no
WPAR3  A      S     p750s2aix2wp3  /wpars/WPAR3  no
```

The time required to create a WPAR from an mksysb will naturally vary depending on the size of your mksysb. In our case it took around 5 minutes per WPAR.

Having successfully created our WPARs, we verified that all required file systems and configurations were preserved from the mksysb. **mkwpar** had successfully deployed a WPAR from the given mksysb; file systems were intact and the Message Broker application restarted clean in all three cases.

Next we repeated the 10-minute WebSphere Message Broker workload, running individually in each WPAR in parallel; that is, all three WPARs were active and running their own workload at the same time. This gave an initial comparison of how the workloads performed compared to running in isolation on an LPAR. But it also demonstrated how the three workloads tolerated the initial sizing of the hosting LPAR.

Because this scenario is based around consolidation, for simplicity we will normalize the performance of the three WPARs as a percentage of the TPS obtained by the baseline LPARs. For example, with our initial configuration of 4VP, the three WPARs in parallel delivered approximately 78% of the combined baseline TPS. First impressions may suggest this is a worrying result; however, remember that the Global LPAR has a third of the processor

allocation compared to the original LPARs. The three LPARs combined had a total of 12 VPs, compared to the hosting LPAR, which had four. In context, 78% is actually quite encouraging.

We continued by amending the processor allocation and rerunning the workloads to profile the change in TPS. The LPAR was increased from 4VP in increments up to 12VP. We also tried a configuration of dedicated processor allocation as a comparison. Figure 3-27 illustrates the %TPS delivered by the five different configurations.

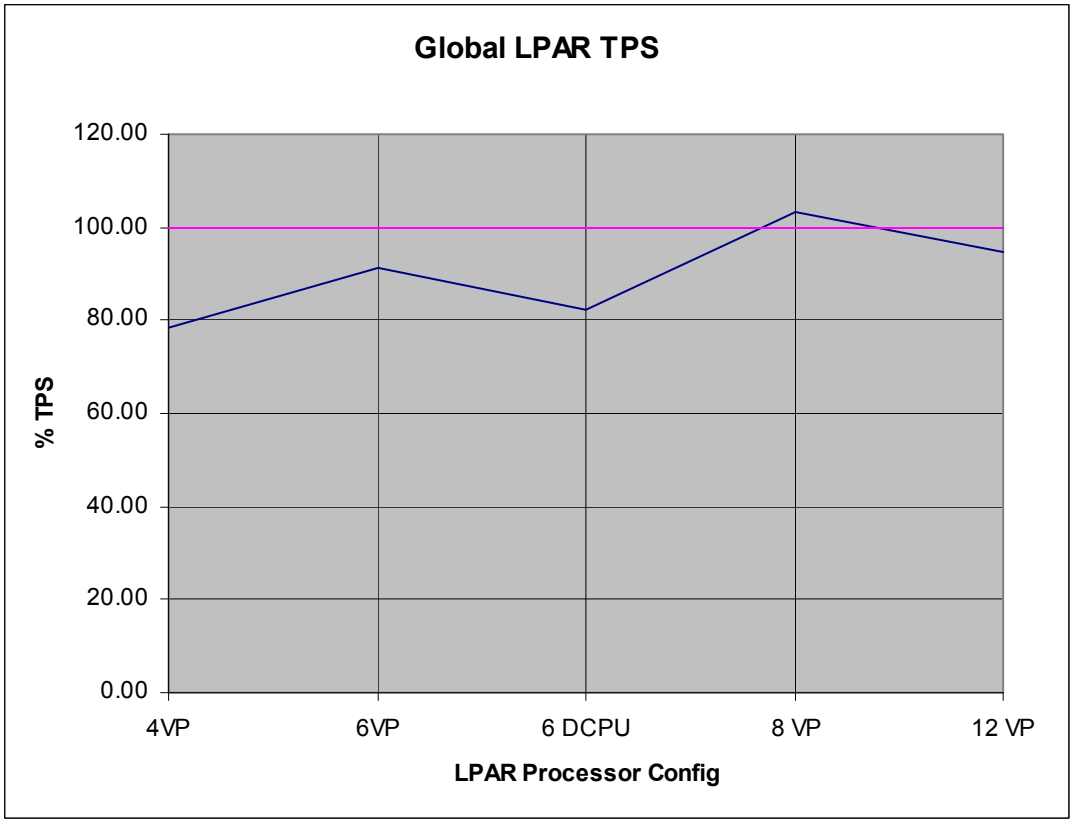


Figure 3-27 Global LPAR TPS

So for our scenario, when considering the combined workloads as a whole, 8VP proved to be the better configuration. Interestingly, the dedicated processor configuration was less efficient than a shared-processor LPAR of the same allocated size.

Illustrating the usage from another angle, Table 3-14 lists the average processor consumption during the 10-minute duration baseline on the original LPARs.

Table 3-14 LPAR CPU consumption

LPAR	LPAR1	LPAR2	LPAR3
Processor consumption	1.03	2.06	3.80

Almost 7.0 processor units were required for the three LPARs. Compare that to the results obtained for the 4VP configuration that consumed only 3.95 units. Another viewpoint is that approximately 57% of the processor resource produced 66% of the original throughput. It is important to consider the difference in consumed resource, compared to the combined throughput. The sustained consumption from the other configurations is listed in Table 3-15 on page 107.

Table 3-15 Global LPAR processor consumption

Virtual processor assignment	4VP	6VP	8VP	12VP
Processor consumption	3.95	5.50	7.60	9.30

The figures show that as the VPs increased, the utilization ultimately peaked and then dropped. The results in Table 3-15 conclude that 8VP was the better configuration for our tests, because 8VPs provided the best TPS of the tested configurations and the processor consumption was only marginally higher than the sum of the original LPARs. This suggested that the overhead for the Global LPAR was actually quite small. However, we were still concerned about the differences in observed TPS.

One thought was that Global LPAR hosting the WPARs was part of the cause. To rule this out we ran the workloads independently, with the Global LPAR in the 8VP configuration, with only one given WPAR active at once. Table 3-16 shows the percentage of throughput compared to the associated original LPAR; in each case more than 100% was achieved.

Table 3-16 Individual WPAR performance compared to individual LPAR

Application threads	Percentage
1	119%
2	116%
8	150%

Completing the analysis of this scenario, we compared the overhead of the original three LPARs and the hosting Global LPAR for the amount of hypervisor calls. We expected that a single LPAR should be less of an overhead than three; however, it was unclear from available documentation whether the use of WPARs would significantly increase calls to the hypervisor. We reran our 10-minute workloads and used **`lparstat`** to record the hypervisor call activity over the duration and provide a per-second average.

For our scenario we found the comparison between the sum of the LPARs and the Global LPAR quite surprising. The Global LPAR used 42% fewer hypervisor calls (per second) compared to the sum of the three LPARs. This is because the LPAR was containing some of the hosting overhead normally placed onto the hypervisor. It is important to appreciate the benefit of reducing unnessary load on the hypervisor; this frees up processor cycles for other uses such as shared processor and memory management, Virtual Ethernet operations, and dynamic LPAR overheads.

The difference in results between the original LPARs, compared to the various configurations of WPARs results from the contention of the primary SMT threads on each VP. Running isolation on an LPAR, there is no competition for the workload. Even when the host LPAR had the same resources as the combined three LPARs, there is enough contention between the workloads to result in the degradation of the smaller workloads. The larger workload actually benefits from there being more VPs to distribute work across.

When a workload test was in progress, we used **`nmon`** to observe the process usage across a given allocation. This allowed us to appreciate how the footprint of the whole Global LPAR changed as the resources were increased; **`nmon`** also allowed us to track the distribution and usage of SMT threads across the LPAR.

To complete the investigations on our consolidation scenario, we looked at memory. We used **`amepat`** to profile memory usage from the Global LPAR (configured with 8VP) and reconfigured the LPAR based on its recommendation. We subsequently reran the workloads

and reprofiled with **amepat** two further times to gain a stable recommendation. The stable recommendation reconfigured the LPAR from 8 GB down to 4 GB. However, we did record approximately 10% TPS reduction of the workloads.

We started with three LPARs, with a total of 12 VP, 24 GB RAM and 180 GB of disk. We demonstrated that with our given workload, the smaller cases suffered slightly due to scheduling competition between the WPARs, whereas the larger workload benefitted slightly from the implementation. The final LPAR configuration had 8 VP, 4 GB RAM and 180 GB of disk. Of the 120 GB allocated in the secondary volume group, only 82 GB were used to host the three WPARs. The final configuration had 75% of the original processor, 17% of the RAM and 45% of the storage. With that greatly reduced footprint, the one LPAR provided 79% of the original throughput. So throughput has been the trade-off for an increase in resource efficiency.

3.9.2 WPAR storage

There are a number of ways to present disk storage to an AIX Workload Partition (WPAR). Depending on the use case, the methods of disk presentation may be different, and have differing performance characteristics.

There are two types of WPARs described here:

- A rootvg WPAR - This is a WPAR built on an hdisk device that is dedicated to the WPAR. The WPAR has its own exclusive rootvg on this disk device. It is not possible to have a versioned WPAR built on a rootvg WPAR.
- A system WPAR - This is a WPAR that has its own root volume group which is built on file systems and logical volumes created inside the global environment. Where versioned WPARs are used, they must be of a system WPAR type.

This subsection discusses some different methods of storage presentation grouped into two areas: Presenting block storage (devices) and file storage (providing access to a file system).

Note: For further information related to WPARs, refer to *Exploiting IBM AIX Workload Partitions,* SG24-7955.

Block

Block storage presentation in this section refers to presenting LUNs, seen as hdisk devices to an AIX WPAR.

There are two methods to achieve this outcome:

- Taking a LUN (hdisk device) from the AIX global instance and presenting it to a WPAR using the **chwpar** command. This can be performed on a system or rootvg WPAR.
- Presenting one or more physical or NPIV fiber channel adapters from the global AIX instance to the WPAR, again using the **chwpar** command. It is not possible to present adapters to a rootvg or versioned WPAR. WPAR mobility is also not possible when mapping adapters to a WPAR.

Figure 3-28 on page 109 illustrates the different methods of presenting disks to a WPAR.

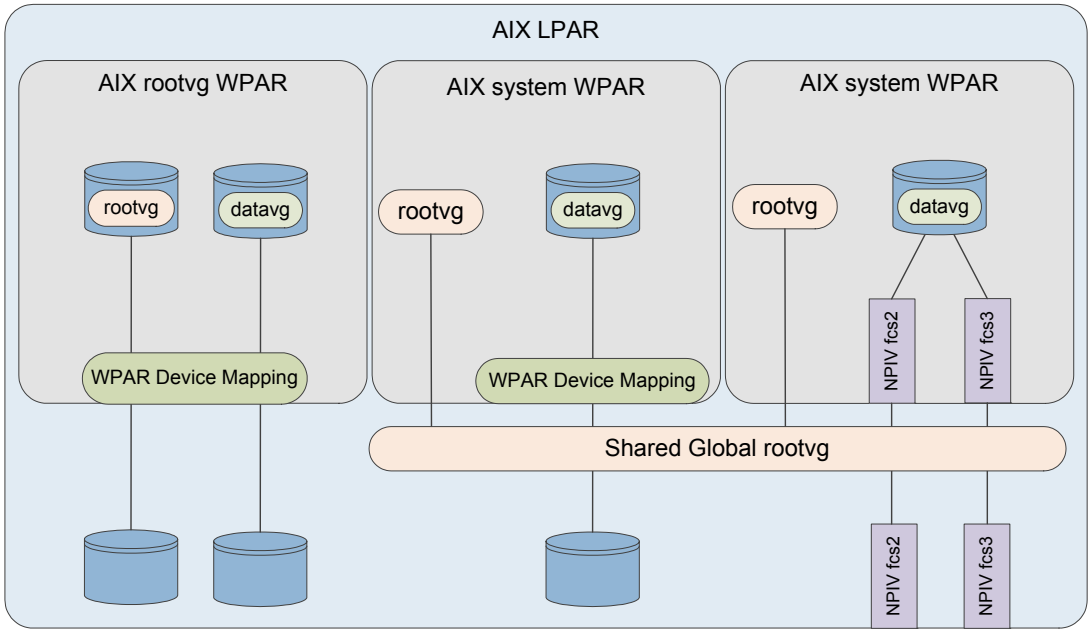


Figure 3-28 WPAR block storage access methods

When mapping a LUN (hdisk) device to a WPAR, the `queue_depth` and `max_transfer` settings can be applied as discussed in 4.3.2, “Disk device tuning” on page 143 with the exception of the `algorithm` attribute, which only supports **`fail_over`**.

Example 3-39 demonstrates how to take the device hdisk6 from the AIX global instance, and present it to a WPAR. Once the disk is exported, it is defined in the global AIX and available in the WPAR.

Example 3-39 WPAR disk device mapping

```
root@aix1global:/ # lsdev -Cc disk |grep hdisk6
hdisk6 Available 02-T1-01 MPIO FC 2145
root@aix1global:/ # chwpar -D devname=hdisk6 aix71wp
root@aix1global:/ # lsdev -Cc disk |grep hdisk6
hdisk6 Defined   02-T1-01 MPIO FC 2145
root@aix1global:/ # lswpar -D aix71wp
Name     Device Name      Type    Virtual Device  RootVG  Status
-------------------------------------------------------------------
aix71wp  hdisk6           disk                    no      EXPORTED
aix71wp  /dev/null        pseudo                          EXPORTED
aix71wp  /dev/tty         pseudo                          EXPORTED
aix71wp  /dev/console     pseudo                          EXPORTED
aix71wp  /dev/zero        pseudo                          EXPORTED
aix71wp  /dev/clone       pseudo                          EXPORTED
aix71wp  /dev/sad         clone                           EXPORTED
aix71wp  /dev/xti/tcp     clone                           EXPORTED
aix71wp  /dev/xti/tcp6    clone                           EXPORTED
aix71wp  /dev/xti/udp     clone                           EXPORTED
aix71wp  /dev/xti/udp6    clone                           EXPORTED
aix71wp  /dev/xti/unixdg  clone                           EXPORTED
aix71wp  /dev/xti/unixst  clone                           EXPORTED
aix71wp  /dev/error       pseudo                          EXPORTED
aix71wp  /dev/errorctl    pseudo                          EXPORTED
aix71wp  /dev/audit       pseudo                          EXPORTED
aix71wp  /dev/nvram       pseudo                          EXPORTED
```

```
aix71wp  /dev/kmem        pseudo                          EXPORTED
root@aix1global:/ # clogin aix71wp
*******************************************************************************
*                                                                             *
*                                                                             *
*  Welcome to AIX Version 7.1!                                                *
*                                                                             *
*                                                                             *
*  Please see the README file in /usr/lpp/bos for information pertinent to    *
*  this release of the AIX Operating System.                                  *
*                                                                             *
*                                                                             *
*******************************************************************************
Last unsuccessful login: Mon Oct  8 12:39:04 CDT 2012 on ssh from 172.16.253.14
Last login: Fri Oct 12 14:18:58 CDT 2012 on /dev/Global from aix1global

root@aix71wp:/ # lsdev -Cc disk
root@aix71wp:/ # cfgmgr
root@aix71wp:/ # lsdev -Cc disk
hdisk0 Available 02-T1-01 MPIO FC 2145
root@aix71wp:/ #
```

The other method of presenting block devices to AIX is to present physical adapters to the partition. These could also be NPIV. The method is exactly the same. It is important that any SAN zoning is completed prior to presenting the adapters, and device attributes discussed in 4.3.5, “Adapter tuning” on page 150 are configured correctly in the global AIX before the device is exported. These settings are passed through to the WPAR, and can be changed inside the WPAR if required after the device is presented.

Example 3-40 demonstrates how to present two NPIV fiber channel adapters (fcs2 and fcs3) to the WPAR. When the mapping is performed, the fcs devices change to a defined state in the global AIX instance, and become available in the WPAR. Any child devices such as a LUN (hdisk device) are available on the WPAR.

Example 3-40 WPAR NPIV mapping

```
root@aix1global:/ # chwpar -D devname=fcs2 aix71wp
fcs2 Available
fscsi2 Available
sfwcomm2 Defined
fscsi2 Defined
line = 0
root@aix1global:/ # chwpar -D devname=fcs3 aix71wp
fcs3 Available
fscsi3 Available
sfwcomm3 Defined
fscsi3 Defined
line = 0
root@aix1global:/ # lswpar -D aix71wp
Name     Device Name      Type     Virtual Device  RootVG  Status
-------------------------------------------------------------------
aix71wp  fcs3             adapter                          EXPORTED
aix71wp  fcs2             adapter                          EXPORTED
aix71wp  /dev/null        pseudo                           EXPORTED
aix71wp  /dev/tty         pseudo                           EXPORTED
aix71wp  /dev/console     pseudo                           EXPORTED
aix71wp  /dev/zero        pseudo                           EXPORTED
aix71wp  /dev/clone       pseudo                           EXPORTED
aix71wp  /dev/sad         clone                            EXPORTED
```

```
aix71wp  /dev/xti/tcp     clone                            EXPORTED
aix71wp  /dev/xti/tcp6    clone                            EXPORTED
aix71wp  /dev/xti/udp     clone                            EXPORTED
aix71wp  /dev/xti/udp6    clone                            EXPORTED
aix71wp  /dev/xti/unixdg  clone                            EXPORTED
aix71wp  /dev/xti/unixst  clone                            EXPORTED
aix71wp  /dev/error       pseudo                           EXPORTED
aix71wp  /dev/errorctl    pseudo                           EXPORTED
aix71wp  /dev/audit       pseudo                           EXPORTED
aix71wp  /dev/nvram       pseudo                           EXPORTED
aix71wp  /dev/kmem        pseudo                           EXPORTED
root@aix1global:/ # clogin aix71wp
*******************************************************************************
*                                                                             *
*                                                                             *
*  Welcome to AIX Version 7.1!                                                *
*                                                                             *
*                                                                             *
*  Please see the README file in /usr/lpp/bos for information pertinent to    *
*  this release of the AIX Operating System.                                  *
*                                                                             *
*                                                                             *
*******************************************************************************
Last unsuccessful login: Mon Oct  8 12:39:04 CDT 2012 on ssh from 172.16.253.14
Last login: Fri Oct 12 14:22:03 CDT 2012 on /dev/Global from p750s02aix1

root@aix71wp:/ # lsdev -Cc disk
root@aix71wp:/ # cfgmgr
root@aix71wp:/ # lsdev -Cc disk
hdisk0 Available 03-T1-01 MPIO FC 2145
root@aix71wp:/ # lspath
Enabled hdisk0 fscsi2
Enabled hdisk0 fscsi2
Enabled hdisk0 fscsi2
Enabled hdisk0 fscsi2
Enabled hdisk0 fscsi3
Enabled hdisk0 fscsi3
Enabled hdisk0 fscsi3
Enabled hdisk0 fscsi3
root@aix71wp:/ # lsdev -Cc adapter
fcs2 Available 03-T1 Virtual Fibre Channel Client Adapter
fcs3 Available 03-T1 Virtual Fibre Channel Client Adapter
root@aix71wp:/ #
```

Versioned WPARs can also have block storage assigned. However, at the time of this writing, NPIV is not supported. Example 3-41 demonstrates how to map disk to an AIX 5.2 Versioned WPAR. There are some important points to note:

- SDDPCM must not be installed in the Global AIX for 5.2 Versioned WPARs.
- Virtual SCSI disks are also supported, which can be LUNs on a VIO server or virtual disks from a shared storage pool.

Example 3-41 Mapping disk to an AIX 5.2 Versioned WPAR

```
root@aix1global:/ # chwpar -D devname=hdisk8 aix52wp
root@aix1global:/ # lslpp -l *sddpcm*
lslpp: 0504-132  Fileset *sddpcm* not installed.
root@aix1global:/ # clogin aix52wp
*******************************************************************************
```

```
*                                                                              *
*                                                                              *
*  Welcome to AIX Version 5.2!                                                 *
*                                                                              *
*                                                                              *
*  Please see the README file in /usr/lpp/bos for information pertinent to     *
*  this release of the AIX Operating System.                                   *
*                                                                              *
*                                                                              *
********************************************************************************
Last unsuccessful login: Thu Mar 24 17:01:03 EDT 2011 on ssh from 172.16.20.1
Last login: Fri Oct 19 08:08:17 EDT 2012 on /dev/Global from aix1global

root@aix52wp:/ # cfgmgr
root@aix52wp:/ # lspv
hdisk0          none                                None
root@aix52wp:/ # lsdev -Cc disk
hdisk0 Available 03-T1-01 MPIO IBM 2076 FC Disk
root@aix52wp:/ #
```

File

File storage presentation in this section refers to providing a WPAR access to an existing file system for I/O operations.

There are two methods for achieving this outcome:

- Creating an NFS export of the file system, and NFS mounting it inside the WPAR.
- Mounting the file system on a directory that is visible inside the WPAR.

Figure 3-29 on page 113 illustrates the different methods of providing file system access to a WPAR, which the examples in this subsection are based on.

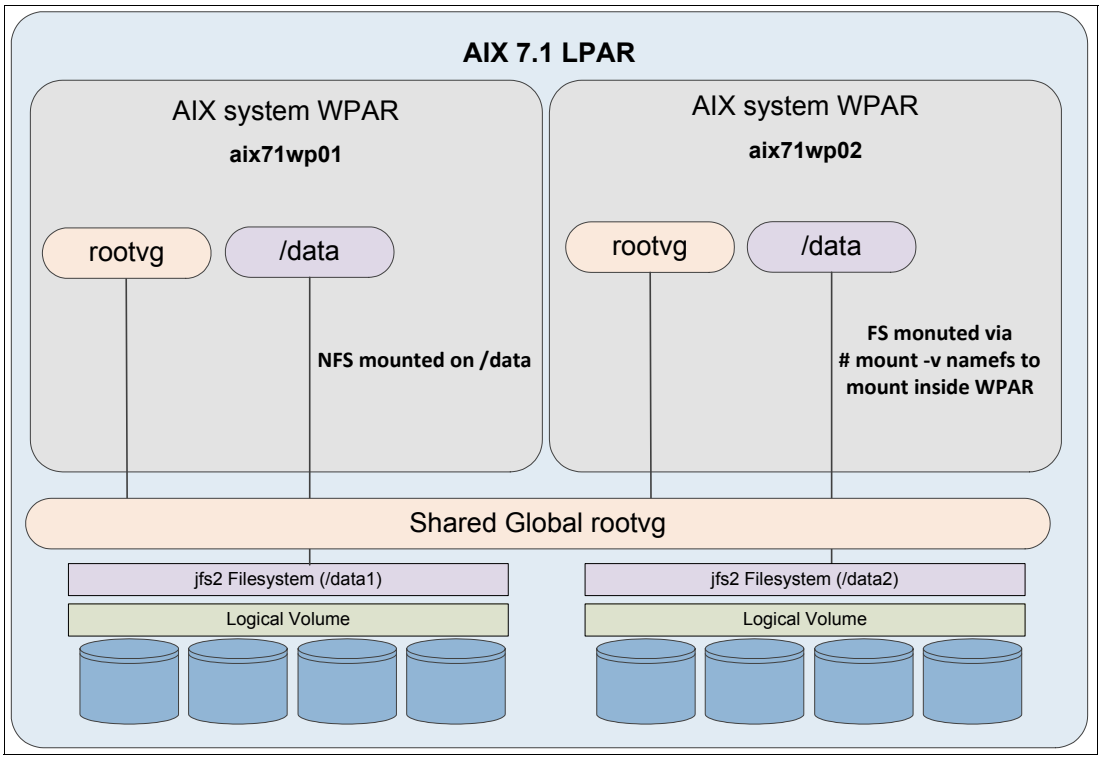


Figure 3-29 WPAR file access mappings

Example 3-42 is a scenario where we have an NFS export on the global AIX instance, and it is mounted inside the AIX WPAR.

Example 3-42 WPAR access via NFS

```
root@aix1global:/ # cat /etc/exports
/data1 -sec=sys,rw,root=aix71wp01
root@aix1global:/ # clogin aix71wp01
*******************************************************************************
*                                                                             *
*                                                                             *
*  Welcome to AIX Version 7.1!                                                *
*                                                                             *
*                                                                             *
*  Please see the README file in /usr/lpp/bos for information pertinent to    *
*  this release of the AIX Operating System.                                  *
*                                                                             *
*                                                                             *
*******************************************************************************
Last unsuccessful login: Mon Oct  8 12:39:04 CDT 2012 on ssh from 172.16.253.14
Last login: Fri Oct 12 14:13:10 CDT 2012 on /dev/Global from aix1global

root@aix71wp01:/ # mkdir /data
root@aix71wp01:/ # mount aix1global:/data1 /data
root@aix71wp01:/ # df -g /data
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
aix1global:/data1 80.00     76.37    5%       36     1% /data
root@aix71wp01:/ #
```

In the case that a file system on the global AIX instance requires WPAR access, the alternative is to create a mount point that is visible inside the WPAR rather than using NFS.

If our WPAR was created on for instance /wpars/aix71wp02, we could mount a file system on /wpars/aix71wp02/data2 and our WPAR would see only a /data2 mount point.

If the file system or directories inside the file system are going to be shared with multiple WPARs, it is good practice to create a Name File System (NameFS). This provides the function to mount a file system on another directory.

When the global AIX instance is started, it is important that the /wpars/.../ file systems are mounted first, before any namefs mounts are mounted. It is also important to note that namefs mounts are not persistent across reboots.

Example 3-43 demonstrates how to take the file system /data2 on the global AIX instance and mount it as /data2 inside the WPAR aix71wp02.

Example 3-43 WPAR access via namefs mount

```
root@aix1global:/ # df -g /data2
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
/dev/data2_lv     80.00     76.37    5%       36     1% /data
root@aix1global:/ # mkdir /wpars/aix71wp02/data
root@aix1global:/ # mount -v namefs /data2 /wpars/aix71wp02/data
root@aix1global:/ # df -g /wpars/aix71wp/data
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
/data2          80.00     76.37    5%       36     1% /wpars/aix71wp02/data
root@aix1global:/ # clogin aix71wp02
*******************************************************************************
*                                                                             *
*                                                                             *
*  Welcome to AIX Version 7.1!                                                *
*                                                                             *
*                                                                             *
*  Please see the README file in /usr/lpp/bos for information pertinent to    *
*  this release of the AIX Operating System.                                  *
*                                                                             *
*                                                                             *
*******************************************************************************
Last unsuccessful login: Mon Oct  8 12:39:04 CDT 2012 on ssh from 172.16.253.14
Last login: Fri Oct 12 14:23:17 CDT 2012 on /dev/Global from aix1global

root@aix71wp02:/ # df -g /data
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
Global            80.00     76.37    5%       36     1% /data
root@aix71wp02:/ #
```

To ensure that the NameFS mounts are recreated in the event that the global AIX is rebooted, there must be a process to mount them when the WPAR is started. To enable the mount to be created when the WPAR is started, it is possible to have a script run when the WPAR is started to perform this action.

Example 3-44 demonstrates how to use the **chwpar** command to have the **aix71wp** execute the script /usr/local/bin/wpar_mp.sh when the WPAR is started. The script must exist and be executable before modifying the WPAR.

Example 3-44 Modify the WPAR to execute a script when it starts

```
root@aix1global:/ # chwpar -u /usr/local/bin/wpar_mp.sh aix71wp
root@aix1global:/ #
```

Example 3-45 demonstrates how to confirm that the script will be executed the next time the WPAR is started.

Example 3-45 Confirming the WPAR will execute the script

```
root@aix1global:/ # lswpar -G aix71wp
================================================================
aix71wp - Active
================================================================
Type:                     S
RootVG WPAR:              no
Owner:                    root
Hostname:                 aix71wp
WPAR-Specific Routing:    no
Virtual IP WPAR:
Directory:                /wpars/aix71wp
Start/Stop Script:        /usr/local/bin/wpar_mp.sh
Auto:                     no
Private /usr:             yes
Checkpointable:           no
Application:

OStype:                   0
Cross-WPAR IPC:           no
Architecture:             none
UUID:                     1db4f4c2-719d-4e5f-bba8-f5e5dc789732
root@aix1global:/ #
```

Example 3-46 is a sample script to offer an idea of how this can be done. The script mounts `/data` on `/wpars/aix71wp/data` to provide the WPAR aix71wp access to the `/data` file system.

Example 3-46 Sample mount script wpar_mp.sh

```
#!/bin/ksh
#set -xv

WPARNAME=aix71wp
FS=/data # Mount point in global AIX to mount
WPARMP=/wpars/${WPARNAME}${FS}

# Check if the filesystem is mounted in the global AIX
if [ $(df -g |awk '{print $7}' |grep -x $FS |wc -l) -eq 0 ]
then
        echo "Filesystem not mounted in the global AIX... exiting"
        exit 1
else
        echo "Filesystem is mounted in the global AIX... continuing"
fi

# Check the WPAR mount point exists
if [ -d $WPARMP ]
then
        echo "Directory to mount on exists... continuing"
else
        echo "Creating directory $WPARMP"
        mkdir -p $WPARMP
fi
```

```
# Check if the namefs mount is already there
if [ $(df -g |awk '{print $7}' |grep -x $WPARMP |wc -l) -eq 1 ]
then
        echo "The namefs mount is already there... nothing to do"
        exit 0
fi

# Create the namefs mount
echo "Mounting $FS on $WPARMP..."
mount -v namefs $FS $WPARMP
if [ $? -eq 0 ]
then
        echo "ok"
        exit 0
else
        echo "Something went wrong with the namefs mount... investigation required."
        exit 99
fi
```

Example 3-47 demonstrates the WPAR being started, and the script being executed.

Example 3-47 Starting the WPAR and verifying execution

```
root@aix1global:/ # startwpar -v aix71wp
Starting workload partition aix71wp.
Mounting all workload partition file systems.
Mounting /wpars/aix71wp
Mounting /wpars/aix71wp/admin
Mounting /wpars/aix71wp/home
Mounting /wpars/aix71wp/opt
Mounting /wpars/aix71wp/proc
Mounting /wpars/aix71wp/tmp
Mounting /wpars/aix71wp/usr
Mounting /wpars/aix71wp/var
Mounting /wpars/aix71wp/var/adm/ras/livedump
Loading workload partition.
Exporting workload partition devices.
sfwcomm3 Defined
fscsi3 Defined
line = 0
sfwcomm2 Defined
fscsi2 Defined
line = 0
Exporting workload partition kernel extensions.
Running user script /usr/local/bin/wpar_mp.sh.
Filesystem is mounted in the global AIX... continuing
Directory to mount on exists... continuing
Mounting /data on /wpars/aix71wp/data...
ok
Starting workload partition subsystem cor_aix71wp.
0513-059 The cor_aix71wp Subsystem has been started. Subsystem PID is 34472382.
Verifying workload partition startup.
Return Status = SUCCESS.
root@aix1global:/ #
```

The case may also be that concurrent I/O is required inside the WPAR but not across the whole file system in the global AIX instance.

Using NameFS provides the capability to mount a file system or just a directory inside the file system with different mount points and optionally with Direct I/O (DIO) or Concurrent I/O (CIO). For examples using DIO and CIO, refer to 4.4.3, “File system best practice” on page 163.

Example 3-48 demonstrates how to mount the /data2 file system inside the global AIX instance, as /wpars/aix71wp02/data with CIO.

Example 3-48 NameFS mount with CIO

```
root@aix1global:/ # mount -v namefs -o cio /data2 /wpars/aix71wp02/data
root@aix1global:/ # clogin aix71wp02
*******************************************************************************
*                                                                             *
*                                                                             *
*  Welcome to AIX Version 7.1!                                                *
*                                                                             *
*                                                                             *
*  Please see the README file in /usr/lpp/bos for information pertinent to    *
*  this release of the AIX Operating System.                                  *
*                                                                             *
*                                                                             *
*******************************************************************************
Last unsuccessful login: Mon Oct  8 12:39:04 CDT 2012 on ssh from 172.16.253.14
Last login: Fri Oct 12 14:23:17 CDT 2012 on /dev/Global from aix1global

root@aix71wp02:/ # df -g /data
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
Global            80.00     76.37    5%       36     1% /data
root@aix71wp02:/ # mount |grep data
         Global           /data           namefs Oct 15 08:07 rw,cio
root@aix71wp02:/ #
```

Conclusion

There are multiple valid methods of presenting block or file storage to an AIX WPAR. From a performance perspective, our findings were as follows:

- For block access using NPIV provided better throughput, due to being able to take advantage of balancing I/O across all paths for a single LUN, and being able to queue I/O to the full queue depth of the fcs adapter device. From a management perspective, WPAR mobility was not possible and some additional zoning and LPAR configuration was required for NPIV to be configured. It is also important to note that if you are using Versioned WPARs, adapter mappings are not supported.
- For file access, if the file system exists on the global AIX instance mounting the file system on the /wpars/<wpar_name>/ directory or using NameFS provided better performance than NFS because we were able to bypass any TCP overhead of NFS and provide access to mount options such as DIO and CIO.

3.10 LPAR suspend and resume best practices

This PowerVM feature was introduced for POWER7-based servers with VIOS 2.2 Fixpack 24 Service Pack1. It utilizes elements of both Logical Partition Mobility (LPM) and Active Memory Sharing (AMS); those familiar with these technologies appreciate the similarities. Other sources of documentation highlight the use of this feature as a means to temporarily make processor and memory resource available, by suspending given LPARs. The method of suspend and resume is offered as a more preferable approach than a traditional shutdown

and restart route, because you do not need to shut down or restart your hosted applications. This saves actual administrator interaction and removes any associated application startup times. The feature is of similar concept to those found on the x86 platform, in that the operating system is quiesced and the running memory footprint is stored to disk and replayed during the resume activity.

We decided to investigate leveraging Suspend/Resume for another reason: to verify whether the feature could be used in the case where the physical hardware (CEC) needed power cycling. From looking at existing documentation, we could not conclude whether this was actually an applicable use of Suspend/Resume.

Note: Although perhaps obvious from the AMS reference above, it should be appreciated that only client LPARs are candidates for suspension. VIOS LPARs cannot be suspended and need to be shut down and rebooted as per normal.

In our test case, Suspend/Resume was configured to use a storage device provided by a pair of redundant VIOS LPARs. We performed a controlled shutdown of some hosted client LPARs and suspended others. Finally the pair of VIOS were shut down in a controlled manner and the CEC was powered off from the HMC.

After the CEC and VIOS LPARs were powered online, the LPARs we suspended were still listed as being in a suspended state—proving that the state survived the power cycle. We were able to successfully resume the LPARs, making them available in the previous state.

We observed that the LPAR does actually become available (in the sense a console will display login prompt) prior to the HMC completing the Resume activity. In our case, we could actually log in to the LPAR in question. However, we soon appreciated what was occurring when the system uptime suddenly jumped from seconds to days.

Note: While the LPAR may respond prior to the HMC completing the *resume* activities, do not attempt to use the LPAR until these activities have finished. The HMC will be in the process of replaying the saved state to the running LPAR.

The time required to suspend and resume a given LPAR depends on a number of factors. Larger and more busier LPARs take longer to suspend or resume. The speed of the storage hosting the paging device is also an obvious factor.

Our conclusions were that Suspend/Resume could successfully be leveraged for the power cycle scenario. Where clients host applications with significant startup and shutdown times it may be an attractive feature to consider.

4

Optimization of an IBM AIX operating system

In this chapter we discuss performance considerations on AIX. We also describe some basic general tuning for AIX. Some tuning may not apply to all workloads. It is important to check best practices documentation and guidelines for each type of workload (database, web server, fileserver) and follow the recommendations for each product.

We show parameters that can be changed in AIX and features that can be enabled:

- Processor folding, Active System Optimizer, and simultaneous multithreading
- Memory
- I/O device tuning
- AIX LVM and file systems
- Network

4.1 Processor folding, Active System Optimizer, and simultaneous multithreading

In this section we discuss some concepts related to processor performance in AIX.

4.1.1 Active System Optimizer

Active System Optimizer (ASO) is an AIX feature introduced in AIX 6.1 TL7 and AIX 7.1 TL1, which monitors and dynamically optimizes the system. ASO needs POWER7 hardware running in POWER7 mode. For more information, refer to 6.1.2, “IBM AIX Dynamic System Optimizer” on page 288.

4.1.2 Simultaneous multithreading (SMT)

First referred to as SMT with the introduction of POWER5. Prior to that, a previous incarnation of SMT was known as HMT (or hardware multithreading). There are many sources of information regarding SMT, some of which we reference later. This section complements these existing sources by highlighting some key areas and illustrating some scenarios when hosting them on the POWER7 platform.

POWER5, POWER6 and POWER7

The implementation of SMT has matured and evolved through each generation. It is important to understand which SMT component is provided by a given platform, but equally how it differs from the implementations in other generations. Such understanding can be critical when planning to migrate existing workloads from older hardware onto POWER7.

For example, both POWER5 and POWER6 provide SMT2 (although remember that prior to POWER7, it was just referred to as SMT). However, while there is no difference in naming to suggest otherwise, the implementation of SMT2 is significantly different between the two platforms. Similarly, SMT4 is not simply a parallel implementation of SMT2.

A confusion can easily arise between the acronyms of SMT and Symmetric Multi Processing (SMP). As we demonstrate in “SMT4” on page 120, inefficiencies can be introduced by confusing the two.

For a detailed comparison of the three implementations of SMT, refer to 2.14.1 of *IBM PowerVM Introduction and Configuration,* SG24-7940.

SMT4

An important characteristic of SMT4 is that the four hardware threads are ordered in a priority hierarchy. That is, for each core or virtual processor, there is one primary hardware thread, one secondary hardware thread, and two tertiary hardware threads in SMT4 mode. This means that work will not be allocated to the secondary threads until consumption of the primary threads exceeds a threshold (controlled by **schedo** options); similarly the tertiary threads will not have work scheduled to them until enough workload exists to drive the primary and secondary threads. This priority hierarchy provides best raw application throughput on POWER7 and POWER7+. Thus the default AIX dispatching behavior is to dispatch across primary threads and then pack across the secondary and tertiary threads.

However, it is possible to negate or influence the efficiency offered by SMT4, through suboptimal LPAR profile configuration. Also, the default AIX dispatching behavior can be changed via **schedo** options, which are discussed in 4.1.4, “Scaled throughput” on page 124.

Note: The following scenario illustrates how inefficiencies can be introduced. There are other elements of PowerVM such as processor folding, Active System Optimizer (ASO), and power saving features that can provide compensation against such issues.

An existing workload is hosted on a POWER6-based LPAR, running AIX 6.1 TL02. The uncapped LPAR is configured to have two virtual processors (VP) and 4 GB of RAM. The LPAR is backed up and restored to a new POWER7 server and the LPAR profile is recreated with the same uncapped/2 VP settings as before. All other processor settings in the new LPAR profile are default.

At a later date, the POWER7-based LPAR is migrated from AIX6.1 TL02 to AIX6.1 TL07. On reboot, the LPAR automatically switches from SMT2 to SMT4 due to the higher AIX level allowing the LPAR to switch from POWER6+™ to POWER7 compatibility mode.

To illustrate this we used a WMB workload. Figure 4-1 shows how the application is only using two of the eight available threads.

```
topas nmon--S=WLMsubclasses----Host=lpar2----------Refresh=2 secs---12:01.12
 CPU-Utilisation-Small-View ----------EntitledCPU=  0.20 UsedCPU=  0.872------
Logical  CPUs             0----------25----------50----------75----------100
CPU User%  Sys% Wait% Idle%|           |           |           |           |
  0  46.5   1.5   0.0  52.0|UUUUUUUUUUUUUUUUUUUUUUUU            >           |
  1   0.0   0.0   0.0 100.0|                     >                           |
  2   0.0   0.0   0.0 100.0|  >                                              |
  3   0.0   0.0   0.0 100.0|>                                                |
  4  50.0   4.5   0.0  45.5|UUUUUUUUUUUUUUUUUUUUUUUUUUss  >                  |
  5   0.0   0.0   0.0 100.0|     >                                           |
  6   0.0   0.0   0.0 100.0|   >                                             |
  7   0.0   0.0   0.0 100.0|   >                                             |
EntitleCapacity/VirtualCPU +-----------|-----------|-----------|-----------+
EC+  59.6   2.9   0.0  37.5|UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUsiiiiiiiiiiiiiiii-|
 VP  26.0   1.3   0.0  16.4|UUUUUUUUUUUUiiiiiiii----------------------------|
EC= 436.1%  VP=  43.6%     +--No Cap---|-----------|-SMT=4-----100% VP=2 CPU+
```

Figure 4-1 WMB workload with two VPs and SMT4

Reconfiguring the LPAR to have only a single VP (Figure 4-2) shows that the WMB workload is using the same amount of resource, but now more efficiently within one core. In our example, we were able to achieve a comparable throughput with one VP as with two VPs. AIX would only have to manage two idle threads, not six, so the resource allocation would be more optimal in that respect.

```
topas nmon--L-LongTerm-CPU-----Host=lpar2----------Refresh=2 secs---12:24.33
 CPU-Utilisation-Small-View ----------EntitledCPU=  0.20 UsedCPU=  0.812------
Logical  CPUs             0----------25----------50----------75----------100
CPU User%  Sys% Wait% Idle%|           |           |           |           |
  0  47.0   0.0   0.0  53.0|UUUUUUUUUUUUUUUUUUUUUUUU                       >
  1  51.0   2.0   0.0  47.0|UUUUUUUUUUUUUUUUUUUUUUUUUUUUs              >   |
  2   0.0   0.0   0.0 100.0|                              >                |
  3   0.0   0.0   0.0 100.0|                           >                   |
EntitleCapacity/VirtualCPU +-----------|-----------|-----------|-----------+
EC+  62.8   2.2   0.0  35.0|UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUsiiiiiiiiiiiiiiii|
 VP  51.0   1.8   0.0  28.4|UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUiiiiiiiiiiiiiii----------|
EC= 405.8%  VP=  81.2%     +--No Cap---|-----------|-SMT=4-----100% VP=1 CPU+
```

Figure 4-2 WMB workload with one VP and SMT4

Scaling the WMB workload to produce double the footprint in the same processor constraints again demonstrated similar efficient distribution. Figure 4-3 on page 122 illustrates the difference in consumption across the four SMT threads.

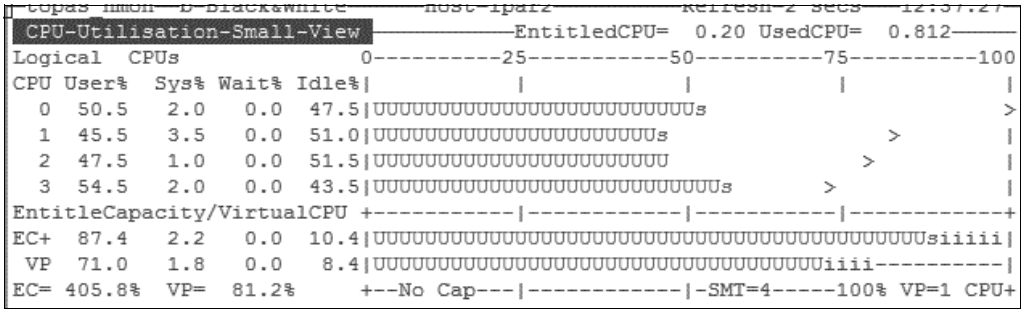


Figure 4-3 Increased WMB workload with one VP and SMT4

However, the throughput recorded using this larger footprint was around 90% less with one VP than with two VPs because the greater workload consumed the maximum capacity at some times. Remember that even if an LPAR is configured as uncapped, the amount of extra entitlement it can request is limited by the number of VPs. So one VP allows up to 1.0 units of allocation.

We observed that other smaller workloads could not take advantage of the larger number of SMT threads, thus it was more efficient to reconfigure the LPAR profile to have fewer VPs (potential example of our NIM server). Allocating what is required is the best approach compared to over-allocating based on a legacy viewpoint. Fewer idle SMT threads or VPs is less overhead for the hypervisor too. Just because your old POWER5-based LPAR had four dedicated processors, it does not always follow that your POWER7-based LPAR requires the same.

Where workloads or LPARs will be migrated from previous platform generations, spending time evaluating and understanding your workload footprint is important; investing time post-migrating is equally important. Regular monitoring of LPAR activity will help build a profile of resource usage to help assess the efficiency of your configuration and also will allow detection of footprint growth. While it is common for an LPAR to be allocated too many resources, it is also common for footprint growth to go undetected.

It is primarily beneficial in commercial environments where the speed of an individual transaction is not as important as the total number of transactions that are performed. Simultaneous multithreading is expected to increase the throughput of workloads with large or frequently changing working sets, such as database servers and web servers.

Workloads that do not benefit much from simultaneous multithreading are those in which the majority of individual software threads uses a large amount of any specific resource in the processor or memory. For example, workloads that are floating-point intensive are likely to gain little from simultaneous multithreading and are the ones most likely to lose performance.

AIX allows you to control the mode of the partition for simultaneous multithreading with the **smtctl** command. By default, AIX enables simultaneous multithreading.

In Example 4-1, in the **smtctl** output, we can see that SMT is enabled and the mode is SMT4. There are two virtual processors, proc0 and proc4, and four logical processors associated with each virtual one, giving a total of eight logical processors.

Example 4-1 Verifying that SMT is enabled and what the mode is

```
# smtctl

This system is SMT capable.
This system supports up to 4 SMT threads per processor.
```

```
SMT is currently enabled.
SMT boot mode is set to enabled.
SMT threads are bound to the same virtual processor.

proc0 has 4 SMT threads.
Bind processor 0 is bound with proc0
Bind processor 1 is bound with proc0
Bind processor 2 is bound with proc0
Bind processor 3 is bound with proc0


proc4 has 4 SMT threads.
Bind processor 4 is bound with proc4
Bind processor 5 is bound with proc4
Bind processor 6 is bound with proc4
Bind processor 7 is bound with proc4
```

4.1.3 Processor folding

On a shared-processor LPAR, AIX monitors the utilization of virtual processors. It watches each virtual processor and the LPAR as a whole. By default AIX will take action when the aggregate utilization drops below 49% (**schedo** option **vpm_fold_threshold**). When current load drops below this threshold, AIX will start folding away virtual processors to make more efficient use of fewer resources. The opposite reaction occurs when the workload increases and breaches the 49% threshold, in which case AIX dynamically unfolds virtual processors to accommodate the increased load.

The aim of this feature is to improve efficiency of thread and virtual processor usage within the LPAR. The folding and unfolding encourages the LPAR to make best use of its processing resources. Improved performance is achieved by attempting to reduce cache misses in the physical processors by efficiently distributing the processes.

Thus, processor folding is a feature introduced in AIX 5.3 ML3 that allows the kernel scheduler to dynamically increase and decrease the use of virtual processors. During low workload demand, some virtual processors are deactivated. Every second, the kernel scheduler evaluates the number of virtual processors that should be activated to accommodate the physical utilization of the partition.

When virtual processors are deactivated, they are not removed from the partition as with dynamic LPAR. The virtual processor is no longer a candidate to run on or receive unbound work; however, it can still run bound jobs. The number of online logical processors and online virtual processors that are visible to the user or applications does not change. There are no impacts to the middleware or the applications running on the system.

Some benefits of processor folding are:

- Better processor affinity
- Less overhead to the hypervisor due to lower number of context switches
- Less virtual processors being dispatched in physical processors implies more physical processors available to other partitions
- Improved energy resources consumption when the processors are idle

Processor folding is enabled by default. In specific situations where you do not want to have the system folding and unfolding all the time, the behavior can be controlled using the **schedo** command to modify the vpm_xvcpus tunable.

To determine whether or not processor folding is enabled, use the command shown in Example 4-2.

Example 4-2 How to check whether processor folding is enabled

```
# schedo -o vpm_xvcpus
vpm_xvcpus = 0
```

If vpm_xvcpus is greater than or equal to zero, processor folding is enabled. Otherwise, if it is equal to -1, folding is disabled. The command to enable is shown in Example 4-3.

Example 4-3 How to enable processor folding

```
# schedo -o vpm_xvcpus=0
Setting vpm_xvcpus to 0
```

Each virtual processor can consume a maximum of one physical processor. The number of virtual processors needed is determined by calculating the sum of the physical processor utilization and the value of the vpm_xvcpus tunable, as shown in the following equation:

Number of virtual processors needed = roundup (physical processor utilization) + vpm_xvcpus

If the number of virtual processors needed is less than the current number of enabled virtual processors, a virtual processor is disabled. If the number of virtual processors needed is greater than the current number of enabled virtual processors, a disabled virtual processor is enabled. Threads that are attached to a disabled virtual processor are still allowed to run on it.

Currently, there is no way to monitor the folding behavior on an AIX partition. The **nmon** tool does some attempt to track VP folding behavior based on the measured processor utilization but again, that is an estimation, not a value reported by any system component.

Important: Folding is available for both dedicated and shared mode partitions. On AIX 7.1, folding is *disabled* for *dedicated-mode* partitions and *enabled* for *shared-mode*.

4.1.4 Scaled throughput

This setting is an alternative dispatcher scheduling mechanism introduced with AIX 6.1 TL08 and AIX 7.1 TL02; the new logic affects how AIX utilizes SMT threads and directly dictates how and when folded VPs will be unfolded. This feature was added based on client requirements and is controlled by a **schedo** tunable. Therefore, it is enabled on an LPAR, by LPAR basis.

The implication of enabling this tunable is that AIX will utilize all SMT threads on a given VP before unfolding additional VPs. The characteristics we observed during tests are best described as a more scientific, controlled approach to what we achieved by forceably reducing VP allocation in “SMT4” on page 120.

The scaled_throughput_mode tunable has four settings: 0, 1, 2 and 4. A value of 0 is the default and disables the tunable. The three other settings enable the feature and dictate the desired level of SMT exploitation (that is SMT1, SMT2, or SMT4).

We tested the feature using our WebSphere Message Broker workload, running on an AIX 7.1 TL02 LPAR configured with four VPs. Two sizes of Message Broker workload were profiled to see what difference would be observed by running with two or four application threads.

Table 4-1 Message Broker scaled_throughput_mode results

	0	**1**	**2**	**4**
TPS for four WMB threads	409.46	286.44	208.08	243.06
Perf per core	127.96	149.18	208.08	243.06
TPS for two WMB threads	235.00	215.28	177.55	177.43
Perf per core	120.51	130.47	177.55	177.43

Table 4-1details the statistics from the eight iterations. In both cases the TPS declined as utilization increased. In the case of the 4-thread workload the trade-off was a 41% decline in throughput against an 89% increase in core efficiency. Whereas for the 2-thread workload it was a 25% decline in throughput against a 47% increase in core efficiency.

So the benefit of implementing this feature is increased core throughput, because AIX maximizes SMT thread utilization before dispatching to additional VPs. But this increased utilization is at the expense of overall performance. However, the tunable will allow aggressively dense server consolidation; another potential use case would be to implement this feature on low load, small footprint LPARs of a noncritical nature, reducing the hypervisor overhead for managing the LPAR and making more system resources available for more demanding LPARs.

Note: Use of the scaled_throughput_mode tunable should only be implemented after understanding the implications. While it is not a restricted **schedo** tunable, we strongly suggest only using it under the guidance of IBM Support.

4.2 Memory

Similar to other operating systems, AIX utilizes *virtual memory*. This allows the memory footprint of workloads to be greater than the physical memory allocated to the LPAR. This *virtual memory* is composed of several devices with different technology:

- **Real Memory -** Composed of physical memory DIMMs (SDRAM or DDRAM)
- **Paging device -** One or more devices hosted on storage (SATA, FC, SSD, or SAN)

  Size of virtual memory = size of real memory + size of paging devices

All memory pages allocated by processes are located in real memory. When the amount of free physical memory reaches a certain threshold, the virtual memory manager (VMM) through the page-replacement algorithm will search for some pages to be evicted from RAM and sent to paging devices (this operation is called *paging out*). If a program needs to access

a memory page located on a paging device (hard disk), this page needs to be first copied back to the real memory (*paging in*).

Because of the technology difference between real memory (RAM) and paging devices (hard disks), the time to access a page is much slower when it is located on paging space and needs a disk I/O to be paged in to the real memory. Paging activity is one of the most common reasons for performance degradation.

Paging activity can be monitored with **vmstat** (Example 4-4) or **nmon** (Figure 4-4).

Example 4-4 Monitoring paging activity with vmstat

```
{D-PW2k2-lpar1:root}/ #vmstat -w 2
kthr          memory                 page                      faults           cpu
------- --------------------- ------------------------------------ ------------------ -----------
  r   b        avm        fre    re    pi    po    fr     sr    cy    in     sy    cs us sy id wa
  1   0   12121859     655411     0     0     0     0      0     0     2  19588  3761  4  0 95  0
  2   0   12387502     389768     0     0     0     0      0     0     1  13877  3731  4  0 95  0
  1   0   12652613     124561     0    46     0     0      0     0    48  19580  3886  4  1 95  0
  3   9   12834625      80095     0    59 54301 81898 586695     0 13634   9323 14718  3 10 78  9
  2  13   12936506      82780     0    18 50349 53034  52856     0 16557    223 19123  2  6 77 16
  1  18   13046280      76018     0    31 49768 54040  53937     0 16139    210 20793  2  6 77 16
  2  19   13145505      81261     0    33 51443 48306  48243     0 16913    133 19889  1  5 77 17
```

With **vmstat**, the paging activity can be monitored by looking at the column **po** (number of pagings out per second) and **pi** (number of pagings in per second).

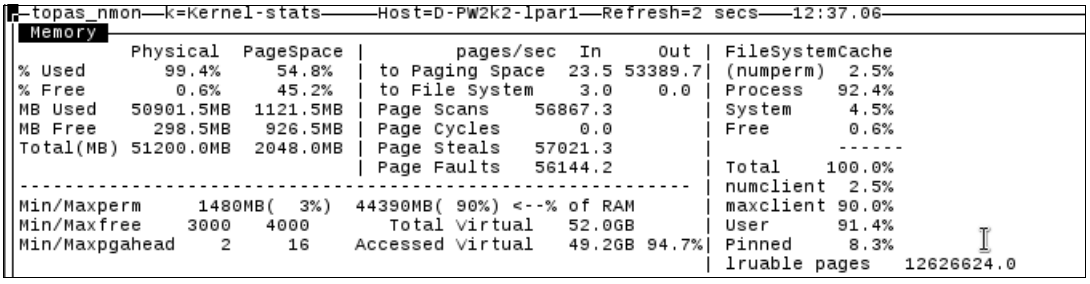


Figure 4-4 Monitoring paging activity with nmon

In Figure 4-4, we started **nmon** in interactive mode. We can monitor the number of paging in and out by looking at the values in *to Paging Space* in and out. This number is given in pages per second.

4.2.1 AIX vmo settings

The AIX virtual memory is partitioned into segments sized 256 MB (default segment size) or 1 TB. Note that 1 TB segment size is only used for 16 GB huge pages, and is similar but a different concept from what is mentioned in 4.2.3, “One TB segment aliasing” on page 129, which still uses a 256 MB segment size.

Depending on the backing storage type, the virtual memory segments can be classified into three types, as described in Table 4-2.

Table 4-2 VMM segments classification depending on backing storage

Segment type	Definition
Persistent	The pages of persistent segments have permanent storage locations on disk (JFS file systems). The persistent segments are used for file caching of JFS file systems.

Segment type	Definition
Client	The client segments also have permanent storage locations, which are backed by a JFS2, CD-ROM file system, or remote file systems such as NFS. The client segments are used for file caching of those file systems.
Working	Working segments are transitory and exist only during their use by a process. They have no permanent disk storage location and are stored on paging space when they are paged out. Typical working segments include process private segments (data, BSS, stack, u-block, heap), shared data segments (shmat or mmap), shared libary data segments, and so on. The kernel segments are also classified as working segments.

Computation memory, also known as computational pages, consists of the pages that belong to working segments or program text (executable files or shared libary files) segments.

File memory, also known as file pages or non-computation memory, consists of the remaining pages. These are usually pages belonging to client segments or persistent segments.

Some AIX tunable parameter can be modified via the **vmo** command to change the behavior of the VMM such as:

- Change the threshold to start or stop the page-replacement algorithm.
- Give more or less priority for the file pages to stay in physical memory compared to computational pages.

Since AIX 6.1, the default values of some vmo tunables were updated to fit most workloads. Refer to Table 4-3.

Table 4-3 vmo parameters: AIX 5.3 defaults vs. AIX 6.1 defaults

AIX 5.3 defaults	AIX 6.1/7.1 defaults
minperm% = 20	**minperm% = 3**
maxperm% = 80	maxperm% = 90
maxclient% = 80	maxclient% = 90
strict_maxperm = 0	strict_maxperm = 0
strict_maxclient = 1	strict_maxclient = 1
lru_file_repage = 1	**lru_file_repage = 0**
page_steal_methode = 0	**page_steal_methode = 1**

With these new parameters, VMM gives more priority to the computational pages to stay in the real memory and avoid paging. When the page replacement algorithm starts, it steals only the file pages as long as the percentage of file pages in memory is above **minperm%**, regardless of the repage rate. This is controlled by the vmo parameter **lru_file_repage=0** and it guarantees 97% memory (`minperm%=3`) for computational pages. If the percentage of file pages drops below `minperm%`, both file and computational pages might be stolen.

Note: In the new version of AIX 7.1, `lru_file_repage=0` is still the default, but this parameter disappears from the vmo tunables and cannot be changed any more.

The memory percentage used by the file pages can be monitored with **nmon** by looking at the value `numperm`, as shown in Figure 4-4 on page 126.

The `page_steal_method=1` specification improves the efficiency of the page replacement algorithm by maintaining several lists of pages (computational pages, file pages, and workload manager class). When used with `lru_file_repage=0`, the page replacement

algorithm can directly find file pages by looking at the corresponding list instead of searching in the entire page frame table. This reduces the number of scanned pages compared to freed pages (scan to free ratio).

The number of pages scanned and freed can be monitored in vmstat by looking at the sr column (pages scanned) and fr column (pages freed). In nmon, these values are reported by *Pages Scans* and *Pages Steals*. Usually, with `page_steal_method=1`, the ratio

Pages scanned to Pages freed should be between 1 and 2.

Conclusion: On new systems (AIX 6.1 and later), default parameters are usually good enough for the majority of the workloads. If you migrate your system from AIX 5.3, undo your old changes to vmo tunables indicated in `/etc/tunables/nextboot`, restart with the default, and change only if needed.

If you still have high paging activity, go through the perfpmr process (“Trace tools and PerfPMR” on page 316), and do not tune restricted tunables unless guided by IBM Support.

4.2.2 Paging space

Paging space or swap space is a special type of logical volume that is used to accommodate pages from RAM. This allows the memory footprint of workloads and processes to be greater than the physical memory allocated to the LPAR. When physical memory utilization reaches a certain threshold, the virtual memory manager (VMM) through the page-replacement algorithm will search for some pages to be evicted from RAM and sent to paging devices. This is called a page-out. When a program makes reference to a page, that page needs to be in real memory. If that page is on disk, a page-in must happen. This delays the execution of the program because it requires disk I/O, which is time-consuming. So it is important to have adequate paging devices.

The best situation is, where possible, to run the workload in main memory. However, it is important to have well dimensioned and good performing paging space to ensure that your system has the best performance when paging is inevitable. Note that some applications have a requirement on paging space, regardless of how much physical RAM is allocated. Therefore, performance of paging space is still valid today as it was previously.

Paging space size considerations can be found at:

`http://www.ibm.com/developerworks/aix/library/au-aix7memoryoptimize3/`

- Look for any specific recommendation from software vendors. Products such as IBM DB2 and Oracle have minimum requirements.
- Monitor your system frequently after going live. If you see that you are never approaching 50 percent of paging space utilization, do not add the space.
- A more sensible rule is to configure the paging space to be half the size of RAM plus 4 GB, with an upper limit of 32 GB.

Performance considerations for paging devices:

- Use multiple paging spaces.
- Use as many physical disks as possible.
- Avoid to use a heavily accessed disk.
- Use devices of the same size.

- Use a striped configuration with 4 KB stripe size.
- Use disks from your Storage Area Network (SAN).

4.2.3 One TB segment aliasing

One TB segment aliasing or Large Segment Aliasing (LSA) improves performance by using 1-TB segment translations for shared memory segments. 64-bit applications with large memory footprint and low spatial locality are likely to benefit from this feature. Both directed and undirected shared memory attachments are eligible for LSA promotion.

In this section, we introduce how 1-TB segment aliasing works, when to enable it, and how to observe the benefits of using it.

Introduction to 1-TB segment aliasing

To understand how LSA works, you need to get some knowledge about 64-bit memory addressing.

Virtual address space of 64-bit applications

64-bit architecture has an addressable range of 2**64, from 0x0000000000000000 to 0xFFFFFFFFFFFFFFFF, which is 16 exabytes (EB) in size. The address space is organized in segments, and there are 2**36 segments, each segment being 256 MB in size.

Table 4-4 shows the 64-bit effective address space.

Table 4-4 64-bit effective address space

Segment Number (hex)	Segment usage
0x0000_0000_0	System call tables, kernel text
0x0000_0000_1	Reserved for system use
0x0000_0000_2	Reserved for user mode loader (process private segment)
0x0000_0000_3 - 0x0000_0000_C	shmat or mmap use
0x0000_0000_D	Reserved for user mode loader
0x0000_0000_E	shmat or mmap use
0x0000_0000_F	Reserved for user mode loader
0x0000_0001_0 - 0x06FF_FFFF_F	Application text, data, BSS and heap
0x0700_0000_0 - 0x07FF_FFFF_F	Default application shmat and mmap area if 1-TB Segment Aliasing (LSA) is not enabled. Directed application shmat and mmap area if LSA is enabled.
0x0800_0000_0 - 0x08FF_FFFF_F	Application explicit module load area
0x0900_0000_0 - 0x09FF_FFFF_F	Shared library text and per-process shared library data
0x0A00_0000_0 - 0x0AFF_FFFF_F	Default (undirected) shmat and mmap area if LSA is enabled
0x0B00_0000_0 - 0x0EFF_FFFF_F	Reserved for future use
0x0F00_0000_0 - 0x0FFF_FFFF_F	Application primary thread stack

Segment Number (hex)	Segment usage
0x1000_0000_0 - 0XEFFF_FFFF_F	Reserved for future use
0xF000_0000_0 - 0xFFFF_FFFF_F	Additional kernel segments

64-bit hardware address resolution

Figure 4-5 gives an explanation of how the effective address of one process is translated to a virtual address, and finally the real hardware address in AIX.

As mentioned in “Virtual address space of 64-bit applications” on page 129, each 64 bit effective address uses the first 36 bits as the effective segment ID (ESID), and then it is mapped to a 52-bit virtual segment ID (VSID) using a segment lookaside buffer (SLB) or a segment table (STAB).

After the translation, we get a 52-bit VSID. Combine this VSID with the 16-bit page index, and we get a 68-bit virtual page number. Then the operating system uses TLB and other tables to translate the virtual page number into a real page number, which is combined with the 12-bit page offset to eventually form a 64-bit real address.

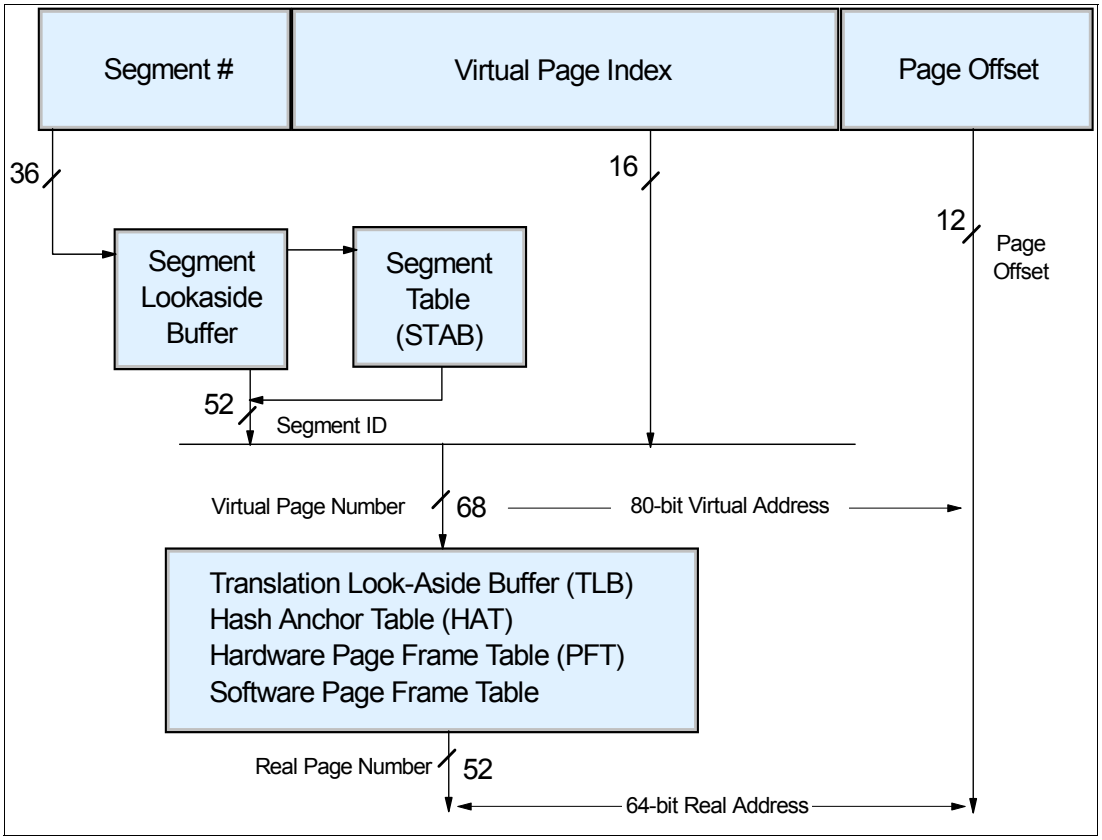


Figure 4-5 64-bit hardware address resolution

ESID and VSID mapping can be found with the **svmon** command, as shown in Example 4-5. Note that the VSID is unique in the operating system, while different processes may have the same ESID.

Example 4-5 VSID and ESID mapping in svmon

```
#svmon -P 9437198
--------------------------------------------------------------------------------
```

```
    Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
9437198 lsatest          24990     9968        0    24961      Y     N     N

    PageSize                Inuse        Pin       Pgsp    Virtual
    s    4 KB                11374          0          0      11345
    m   64 KB                  851        623          0        851

   Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
  20002         0 work kernel segment              m    671   620    0     671
 9d001d  90000000 work shared library text          m    175     0    0     175
  50005  9fffffd work shared library               sm   2544     0    0    2544
 9f001f  90020014 work shared library               s    166     0    0     166
 840fa4  70000004 work default shmat/mmap          sm    135     0    0     135
 890fc9  70000029 work default shmat/mmap          sm    135     0    0     135
 9d0efd  70000024 work default shmat/mmap          sm    135     0    0     135
 8c0f6c  70000012 work default shmat/mmap          sm    135     0    0     135
 9b0edb  70000008 work default shmat/mmap          sm    135     0    0     135
 980f38  7000000d work default shmat/mmap          sm    135     0    0     135
 8e0f0e  7000003b work default shmat/mmap          sm    135     0    0     135
 870ec7  70000036 work default shmat/mmap          sm    135     0    0     135
 9504b5  7000002d work default shmat/mmap          sm    135     0    0     135
```

Hardware limits on SLB entries and benefits of LSA

Now you know that SLB is used to translate ESID to VSID when doing address translation. Because SLB is in processor cache, the translation will be very efficient if we hit the SLB when accessing memory.

However, POWER6 and POWER7 processor has limited SLB entries, as follows:

- POWER6
  - SLB has 64 entries
  - 20 reserved for the kernel
  - 44 available for user processes, which yields 11 GB of accessible memory
  - Many client workloads do not fit into 11 GB
- POWER7
  - SLB has 32 entries; architectural trend towards smaller SLB sizes
  - 20 still reserved for the kernel
  - 12 available for user processes, which yields 3 GB of accessible memory
  - Potential for performance regression

As the SLB entries are limited, you can only address 3 GB of user memory directly from SLB in POWER7, which is usually not enough for most applications. And if you failed to address memory directly from SLB, the performance deteriorates.

This is why LSA is introduced in AIX. Through LSA, you can address 12 TB of memory using 12 SLB entries, and SLB faults should be rare. Because this is transparent to the application, you can expect an immediate performance boost for many applications that have a large memory footprint (Figure 4-6 on page 132).

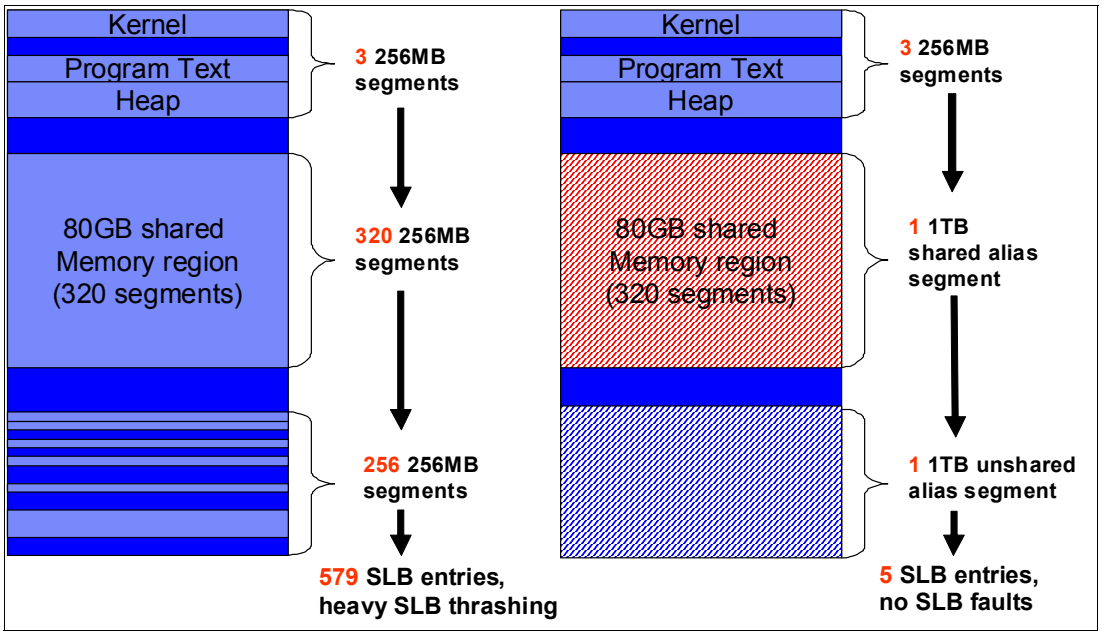


Figure 4-6 Process address space example with LSA

Enabling LSA and verification

In the following sections, we introduce how to enable LSA and check whether LSA has taken effect.

Enabling LSA

There are vmo options as well as environment variables available to enable LSA. For most cases, you need to set `esid_allocator=1` when in AIX 6.1, and do nothing in AIX 7.1 because the default is on already. You can also change the environment variables on a per process basis. The option details are as follows:

- esid_allocator, VMM_CNTRL=ESID_ALLOCATOR=[0,1]

  Default off (0) in AIX 6.1 TL06, on (1) in AIX 7.1. When on, indicates that the large segment aliasing effective address allocation policy is in use. This parameter can be changed dynamically but will only be effective for future shared memory allocations.

- shm_1tb_shared, VMM_CNTRL=SHM_1TB_SHARED=[0,4096]

  Default set to 12 (3 GB) on POWER7, 44 (11GB) on POWER6 and earlier. This is in accord with the hardware limit of POWER6 and POWER7. This parameter sets the threshold number of 256 MB segments at which a shared memory region is promoted to use a 1-TB alias.

- shm_1tb_unshared, VMM_CNTRL=SHM_1TB_UNSHARED=[0,4096]

  Default set to 256 (64 GB). This parameter controls the threshold number of 256 MB segments at which multiple homogeneous small shared memory regions will be promoted to an unshared alias. Use this parameter with caution because there could be performance degradation when there are frequent shared memory attaches and detaches.

- shm_1tb_unsh_enable

  Default set to on (1) in AIX 6.1 TL06 and AIX 7.1 TL01; Default set to off (0) in AIX 7.1 TL02 and later releases. When on, indicates unshared aliases are in use.

Note: Unshared aliases might degrade performance in case there are frequent shared memory attaches and detaches. We suggest you turn unshared aliasing off.

You can also refer to the Oracle Database and 1-TB Segment Aliasing in the following website for more information:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD105761

Verification of LSA

This section shows the LSA verification steps:

1. Get the process ID of the process using LSA, which can be any user process, for example a database process.
2. Use **svmon** to confirm that the shared memory regions are already allocated, as shown in Example 4-6.

Example 4-6 svmon -P <pid>

```
#svmon -P 3670250
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
 3670250 lsatest          17260    10000        0    17229      Y     N     N

     PageSize                Inuse        Pin       Pgsp    Virtual
     s    4 KB                3692          0          0       3661
     m   64 KB                 848        625          0        848

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
   20002         0 work kernel segment               m    671   622    0     671
  990019  90000000 work shared library text          m    172     0    0     172
   50005  9fffffd work shared library               sm   2541     0    0    2541
  9b001b  90020014 work shared library                s    161     0    0     161
  b70f37 f00000002 work process private               m      5     3    0       5
  fb0b7b  9001000a work shared library data          sm     68     0    0      68
  a10021  9fffffff clnt USLA text,/dev/hd2:4225       s     20     0    -       -
  e60f66  70000004 work default shmat/mmap           sm     14     0    0      14
  fd0e7d  70000023 work default shmat/mmap           sm     14     0    0      14
  ee0fee  70000007 work default shmat/mmap           sm     14     0    0      14
  ea0f6a  7000003f work default shmat/mmap           sm     14     0    0      14
  e50e65  70000021 work default shmat/mmap           sm     14     0    0      14
  d10bd1  7000001a work default shmat/mmap           sm     14     0    0      14
  fb0f7b  70000002 work default shmat/mmap           sm     14     0    0      14
  ff0fff  70000009 work default shmat/mmap           sm     14     0    0      14
  f20e72  7000003d work default shmat/mmap           sm     14     0    0      14
  e50fe5  70000028 work default shmat/mmap           sm     14     0    0      14
  f00e70  7000000e work default shmat/mmap           sm     14     0    0      14
  8a0c8a  7000001e work default shmat/mmap           sm     14     0    0      14
  f80f78  7000002f work default shmat/mmap           sm     14     0    0      14
```

3. Run **kdb** under root (Example 4-7).

Example 4-7 Running kdb

```
#kdb
            START              END <name>
0000000000001000 00000000058A0000 start+000FD8
```

```
F00000002FF47600 F00000002FFDF9C8 __ublock+000000
000000002FF22FF4 000000002FF22FF8 environ+000000
000000002FF22FF8 000000002FF22FFC errno+000000
F1000F0A00000000 F1000F0A10000000 pvproc+000000
F1000F0A10000000 F1000F0A18000000 pvthread+000000
read vscsi_scsi_ptrs OK, ptr = 0xF1000000C02D6380
(0)>
```

4. Run **tpid -d <pid>** in **kdb** to get the SLOT number of the related thread (Example 4-8).

Example 4-8 tpid -d <pid>

```
(0)> tpid -d 3670250
               SLOT NAME     STATE    TID PRI   RQ CPUID  CL  WCHAN

pvthread+019500  405!lsatest  RUN   1950075 071    4         0
```

5. Choose any of the thread SLOT numbers listed (only one available above), and run “user -ad <slot_number>” in **kdb**. As in Example 4-9, the LSA_ALIAS in the command output means LSA is activated for the shared memory allocation. If LSA_ALIAS flag does not exist, LSA is not in effect.

Example 4-9 user -ad <slot_number>

```
(0)> user -ad 405
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0001
shared alias thresh (U_shared_alias_thresh)...... 000C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00400401 SHMAT BIGSTAB LSA_ALIAS
```

Identify LSA issues

In the following sections, we introduce how to identify LSA issues using **hpmstat** and **tprof**.

Using hpmstat to identify LSA issues

The **hpmstat** command provides system-wide hardware performance counter information that can be used to monitor SLB misses. Refer to “The hpmstat and hpmcount utilities” on page 334 for more information about **hpmstat**. If there are a lot of SLB misses, then enabling LSA should help.

You can get the supported event groups from the **pmlist** command in AIX, as shown in Example 4-10 on page 135.

Example 4-10 Supported hardware performance event groups

```
#pmlist -g -1|pg
...
Group #10: pm_slb_miss
Group name: SLB Misses
Group description: SLB Misses
Group status: Verified
Group members:
Counter  1, event 77: PM_IERAT_MISS  : IERAT Reloaded  (Miss)
Counter  2, event 41: PM_DSLB_MISS  : Data SLB misses
Counter  3, event 89: PM_ISLB_MISS  : Instruction SLB misses
Counter  4, event 226: PM_SLB_MISS  : SLB misses
Counter  5, event  0: PM_RUN_INST_CMPL  : Run instructions completed
Counter  6, event  0: PM_RUN_CYC  : Run cycles
...
```

Group #10 is used for reporting SLB misses. Use **hpmstat** to monitor the SLB misses events as shown in Example 4-11. Generally you should further investigate the issue when the SLB miss rate per instruction is greater than 0.5%. The DSLB miss rate per instruction is 1.295%, and is not acceptable. You can enable LSA by setting `vmo -p -o esid_allocator=1` and seeing the effect.

Example 4-11 hpmstat before LSA is enabled

```
#hpmstat -r -g 10 20
 Execution time (wall clock time): 20.010013996 seconds

 Group: 10
 Counting mode: user+kernel+hypervisor+runlatch
 Counting duration: 160.115119955 seconds
  PM_IERAT_MISS (IERAT Reloaded  (Miss))                    :        20894033
  PM_DSLB_MISS (Data SLB misses)                            :        72329260
  PM_ISLB_MISS (Instruction SLB misses)                     :           15710
  PM_SLB_MISS (SLB misses)                                  :        72344970
  PM_RUN_INST_CMPL (Run instructions completed)             :      5584383071
  PM_RUN_CYC (Run cycles)                                   :     66322682987

 Normalization base: time

 Counting mode: user+kernel+hypervisor+runlatch

  Derived metric group: Translation

  [   ] % DSLB_Miss_Rate per inst                           :           1.295 %
  [   ] IERAT miss rate (%)                                 :           0.374 %
  [   ] % ISLB miss rate per inst                           :           0.000 %

  Derived metric group: General

  [   ] Run cycles per run instruction                      :          11.876
  [   ] MIPS                                                :          34.877
MIPS

 u=Unverified c=Caveat R=Redefined m=Interleaved
```

Example 4-12 shows the **hpmstat** output after we set esid_allocator=1 and restarted the application. You can see that the SLB misses are gone after LSA is activated.

Example 4-12 hpmstat output after LSA is enabled

```
#hpmstat -r -g 10 20
 Execution time (wall clock time): 20.001231826 seconds

 Group: 10
 Counting mode: user+kernel+hypervisor+runlatch
 Counting duration: 160.005281724 seconds
  PM_IERAT_MISS (IERAT Reloaded  (Miss))                    :          189529
  PM_DSLB_MISS (Data SLB misses)                            :           25347
  PM_ISLB_MISS (Instruction SLB misses)                     :           15090
  PM_SLB_MISS (SLB misses)                                  :           40437
  PM_RUN_INST_CMPL (Run instructions completed)             :      2371507258
  PM_RUN_CYC (Run cycles)                                   :     66319381743

 Normalization base: time

 Counting mode: user+kernel+hypervisor+runlatch

  Derived metric group: Translation

  [   ] % DSLB_Miss_Rate per inst                           :           0.001 %
  [   ] IERAT miss rate (%)                                 :           0.008 %
  [   ] % ISLB miss rate per inst                           :           0.001 %

  Derived metric group: General

  [   ] Run cycles per run instruction                      :          27.965
  [   ] MIPS                                                :          14.821
MIPS

 u=Unverified c=Caveat R=Redefined m=Interleaved
```

Using tprof to identify LSA issues

An **-E** option is available for monitoring such events. When there is a notable amount of SLB misses, you should be able to see a lot of kernel processor time spent in set_smt_pri_user_slb_found. In Example 4-13 you can see 13.92% set_smt_pri_user_slb_found, and you can find that `lsatest` caused the problem.

Example 4-13 tprof before LSA is enabled

```
#tprof -E -sku -x sleep 10
Configuration information
=========================
System: AIX 7.1 Node: p750s1aix2 Machine: 00F660114C00
Tprof command was:
    tprof -E -sku -x sleep 10
Trace command was:
    /usr/bin/trace -ad -M -L 1073741312 -T 500000 -j
00A,001,002,003,38F,005,006,134,210,139,5A2,5A5,465,2FF,5D8, -o -
Total Samples = 1007
Traced Time = 10.02s (out of a total execution time of 10.02s)
Performance Monitor based reports:
```

```
Processor name: POWER7
Monitored event: Processor cycles
Sampling interval: 10ms
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Process                                 Freq  Total Kernel   User Shared  Other
=======                                 ====  ===== ======   ==== ======  =====
lsatest                                    1  99.50  24.85  74.65   0.00   0.00
/usr/bin/sh                                2   0.20   0.10   0.00   0.10   0.00
gil                                        1   0.10   0.10   0.00   0.00   0.00
...
Total % For All Processes (KERNEL) = 25.25

Subroutine                                                %   Source
==========                                              ====== ======
set_smt_pri_user_slb_found                              13.92 noname
start                                                    8.05 low.s
.user_slb_found                                          1.79 noname
slb_stats_usr_point                                      0.80 noname
._ptrgl                                                  0.20 low.s
slb_user_tmm_fixup                                       0.10 noname
.enable                                                  0.10 misc.s
.tstart                                                  0.10 /kernel/proc/clock.c
.v_freexpt                                               0.10 rnel/vmm/v_xptsubs.c
```

After we enabled LSA, set_smt_pri_user_slb_found was gone (Example 4-14).

Example 4-14 tprof after LSA is enabled

```
#tprof -E -sku -x sleep 10
Configuration information
=========================
System: AIX 7.1 Node: p750s1aix2 Machine: 00F660114C00
Tprof command was:
    tprof -E -sku -x sleep 10
Trace command was:
    /usr/bin/trace -ad -M -L 1073741312 -T 500000 -j
00A,001,002,003,38F,005,006,134,210,139,5A2,5A5,465,2FF,5D8, -o -
Total Samples = 1007
Traced Time = 10.02s (out of a total execution time of 10.02s)
Performance Monitor based reports:
Processor name: POWER7
Monitored event: Processor cycles
Sampling interval: 10ms
...
Total % For All Processes (KERNEL) = 0.10

Subroutine                                                %   Source
==========                                              ====== ======
ovlya_addr_sc_ret                                        0.10 low.s
```

Sample program illustration

The sample program used in this section is in Appendix C, “Workloads” on page 341. In the sample scenario, we got about a 30% performance gain. Note that real workload benefits can vary.

4.2.4 Multiple page size support

In AIX, the virtual memory is split into pages, with a default page size of 4 KB. The POWER5+ processor supports four virtual memory page sizes: 4 KB (small pages), 64 KB (medium pages), 16 MB (large pages), and 16 GB (huge pages). The POWER6 processor also supports using 64 KB pages in segments with base page size of 4 KB. AIX uses this process to provide the performance benefits of 64 KB pages when useful or resorting to 4 KB pages where 64 KB pages would waste too much memory, such as allocated but not used by the application.

Using a larger virtual memory page size such as 64 KB for an application’s memory can improve the application's performance and throughput due to hardware efficiencies associated with larger page sizes. Using a larger page size can decrease the hardware latency of translating a virtual page address to a physical page address. This decrease in latency is due to improving the efficiency of hardware translation caches such as a processor’s translation lookaside buffer (TLB). Because a hardware translation cache only has a limited number of entries, using larger page sizes increases the amount of virtual memory that can be translated by each entry in the cache. This increases the amount of memory that can be accessed by an application without incurring hardware translation delays.

POWER6 supports mixing 4 KB and 64 KB page sizes. AIX 6.1 takes advantage of this new hardware capability automatically without user intervention. This AIX feature is called Dynamic Variable Page Size Support (DVPSS). To avoid backward compatibility issues, VPSS is disabled in segments that currently have an explicit page size selected by the user.

Some applications may require a configuration to take advantage of multiple page support, while others will take advantage by default. SAP, for example, needs some additional configuration to make use of 64 KB pages. Information regarding the required configuratin can be found in the “Improving SAP performance on IBM AIX: Modification of the application memory page size to improve the performance of SAP NetWeaver on the AIX operating system” whitepaper at:

`http://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/whitepaper/aix/sap/netweaver/performance`

Note: The use of multiple page support cannot be combined with Active Memory Sharing (AMS) or Active Memory Expansion (AME). Both only support 4 KB pages. AME can optionally support 64 K pages, but the overhead in enabling that support can cause poor performance.

Large pages

Large pages are intended to be used in specific environments. AIX does not automatically use these page sizes. AIX must be configured to do so, and the number of pages of each of these page sizes must also be configured. AIX cannot automatically change the number of configured 16 MB or 16 GB pages.

Not all applications benefit from using large pages. Memory-access-intensive applications such as databases that use large amounts of virtual memory can benefit from using large pages (16 MB). DB2 and Oracle require specific settings to use this. IBM Java can take

advantage of medium (64 K) and large page sizes. Refer to section 7.3, “Memory and page size considerations” in the *POWER7 and POWER7+ Optimization and Tuning Guide*, SG24-8079.

AIX maintains different pools for 4 KB and 16 MB pages. An application (at least WebSphere) configured to use large pages can still use 4 KB pages. However, other applications and system processes may not be able to use 16 MB pages. In this case, if you allocate too many large pages you can have contention for 4 KB and high paging activity.

AIX treats large pages as pinned memory and does not provide paging support for them. Using large pages can result in an increased memory footprint due to memory fragmentation.

Note: You should be extremely cautious when configuring your system for supporting large pages. You need to understand your workload before using large pages in your system.

Since AIX 5.3, the large page pool is dynamic. The amount of physical memory that you specify takes effect immediately and does not require a reboot.

Example 4-15 shows how to verify the available page sizes.

Example 4-15 Display the possible page sizes

```
# pagesize -a
4096
65536
16777216
17179869184
```

Example 4-16 shows how to configure two large pages dynamically.

Example 4-16 Configuring two large pages (16 MB)

```
# vmo  -o lgpg_regions=2 -o lgpg_size=16777216
Setting lgpg_size to 16777216
Setting lgpg_regions to 2
```

Example 4-17 shows how to disable large pages.

Example 4-17 Removing large page configuration

```
# vmo   -o lgpg_regions=0
Setting lgpg_regions to 0
```

The commands that can be used to monitor different page size utilization are **vmstat** and **svmon**. The flag **-P** of **vmstat** followed by the page size shows the information for that page size, as seen in Example 4-18. The flag **-P ALL** shows the overall memory utilization divided into different page sizes, as seen in Example 4-19 on page 140.

Example 4-18 vmstat command to verify large page utilization

```
# vmstat -P 16MB

System configuration: mem=8192MB

pgsz            memory                           page
----- -------------------------- ------------------------------------
           siz      avm      fre    re    pi    po    fr     sr    cy
```

```
  16M      200       49      136     0     0     0     0     0     0
```

Example 4-19 vmstat command to show memory utilization grouped by page sizes

```
# vmstat -P ALL

System configuration: mem=8192MB

pgsz            memory                           page
----- ------------------------- ------------------------------------
           siz      avm      fre    re    pi    po    fr     sr    cy
   4K   308832   228825    41133     0     0     0    13     42     0
  64K    60570    11370    51292     0     0    39    40    133     0
  16M      200       49      136     0     0     0     0      0     0
```

Example 4-20 shows that **svmon** with the flag **-G** is another command that can be used to verify memory utilization divided into different page sizes.

Example 4-20 svmon command to show memory utilization grouped by page sizes

```
# svmon -G
               size       inuse        free         pin     virtual   mmode
memory      2097152     1235568      861584     1129884      611529     Ded
pg space     655360       31314

               work        pers        clnt       other
pin          371788           0           0      135504
in use       578121           0       38951

PageSize   PoolSize       inuse        pgsp         pin     virtual
s     4 KB         -      267856        3858      176364      228905
m    64 KB         -        9282        1716        8395       11370
L    16 MB       200          49           0         200          49
```

In the three previous examples, the output shows 200 large pages configured in AIX and 49 in use.

4.3 I/O device tuning

When configuring AIX I/O devices for performance, there are many factors to take into consideration. It is important to understand the underlying disk subsystem, and how the AIX system is attached to it.

In this section we focus only on the tuning of disk devices and disk adapter devices in AIX. AIX LVM and file system performance tuning are discussed in detail in 4.4, “AIX LVM and file systems” on page 157.

4.3.1 I/O chain overview

Understanding I/O chain specifically regarding disks and disk adapters is important because it ensures that all devices in the stack have the appropriate tuning parameters defined.

We look at three types of disk attachments:

- Disk presented via dedicated physical adapters
- Virtualized disk using NPIV
- Virtualized disk using virtual SCSI

Refer to *IBM PowerVM Virtualization Introduction and Configuration,* SG24-7940-04, which describes in detail how to configure NPIV and Virtual SCSI. In this section we only discuss the concepts and how to tune parameters related to performance.

In 3.6.1, “Virtual SCSI” on page 75, 3.6.2, “Shared storage pools” on page 76, 3.6.3, “N_Port Virtualization” on page 79 we discuss in detail the use cases and potential performance implications of using NPIV and Virtual SCSI.

Dedicated physical adapters

When we referred to disk storage presented via direct physical adapters, this implies that the disk is attached to the AIX system without the use of Virtual I/O. This means that the AIX system has exclusive access to fiber channel adapters, which are used to send I/O to an external storage system.

Looking at Figure 4-7 from left to right, when a write or a read operation is issued to AIX, LVM uses one physical buffer (pbuf) for each request. The physical buffers are described in 4.3.3, “Pbuf on AIX disk devices” on page 148. The I/O is then queued to the physical volume (PV), then handed to the multipathing driver and queued to the disk adapter device. The I/O is then passed through one or more SAN fabric switches (unless the storage is direct-attached to the AIX system) and reaches the external storage. If the I/O can be written to or read from the storage system’s cache, it is, otherwise it goes to disk.

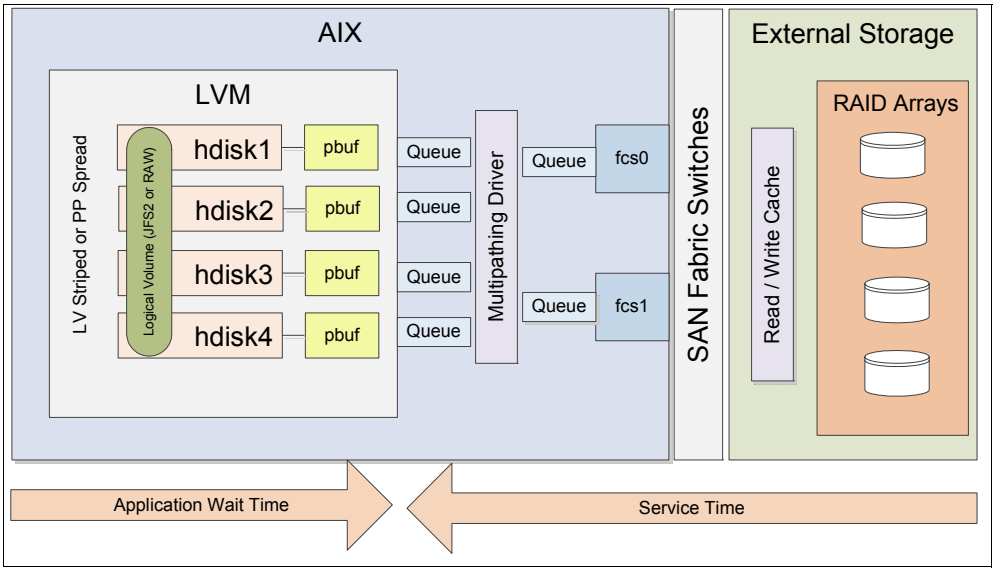


Figure 4-7 Dedicated adapters

NPIV

NPIV is a method where disk storage is implemented using PowerVM’s N_Port virtualization capability. In this instance, the Virtual I/O servers act as a passthrough enabling multiple AIX LPARs to access a single shared fiber channel (FC) port. A single FC adapter port on a Virtual I/O server is capable of virtualizing up to 64 worldwide port names (WWPN), meaning there are a maximum of 64 client logical partitions that can connect.

The I/O sequence is very similar to that of using dedicated physical adapters with the exception that there is an additional queue on each fiber channel adapter per Virtual I/O server, and there might be competing workloads on the fiber channel port from different logical partitions.

Figure 4-8 illustrates the I/O chain when NPIV is in use.

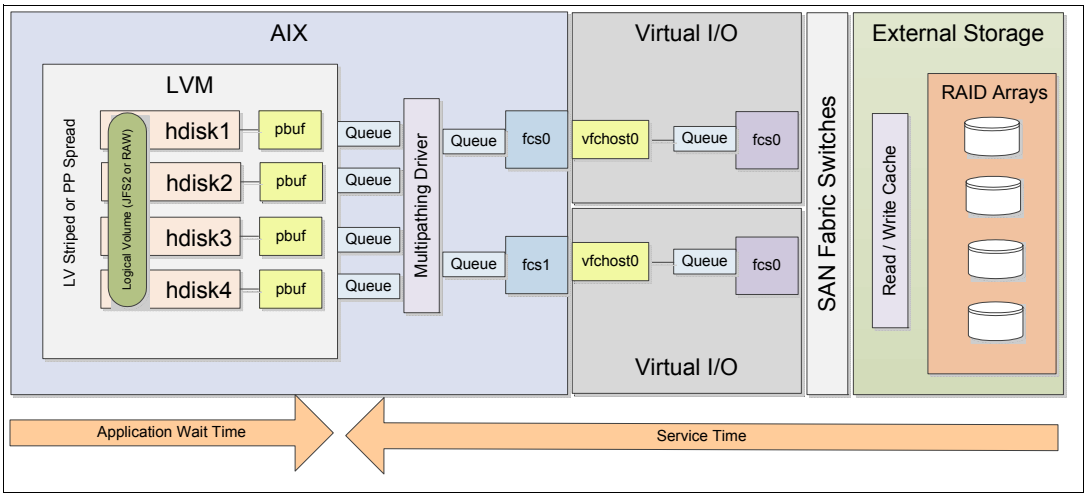


Figure 4-8 N_Port virtualization

Virtual SCSI

Virtual SCSI is a method of presenting a disk assigned to one or more Virtual I/O servers to a client logical partition. When an I/O is issued to the AIX LVM, the pbuf and hdisk queue is used exactly the same as in the dedicated physical adapter and NPIV scenarios. The difference is that there is a native AIX SCSI driver used and I/O requests are sent to a virtual SCSI adapter. The virtual SCSI adapter is a direct mapping to the Virtual I/O server's vhost adapter, which is allocated to the client logical partition.

The hdisk device exists on both the client logical partition and the virtual I/O server, so there is also a queue to the hdisk on the virtual I/O server. The multipathing driver installed on the virtual I/O server then queues the I/O to the physical fiber channel adapter assigned to the VIO server and the I/O is passed to the external storage subsystem as described in the dedicated physical adapter and NPIV scenarios. There may be some limitation with copy services from a storage system in the case that a device driver is required to be installed on the AIX LPAR for this type of functionality.

Figure 4-9 on page 143 illustrates the I/O chain when virtual SCSI is in use.

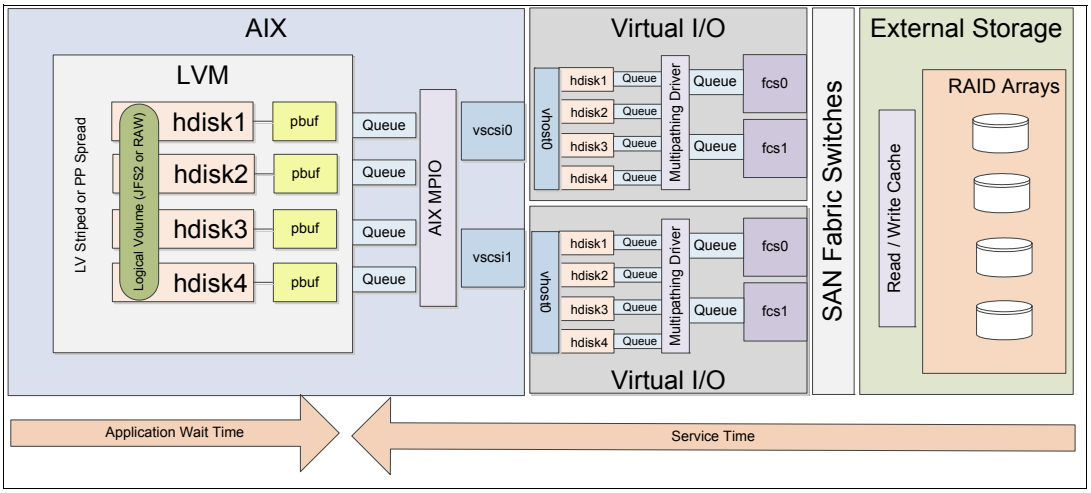


Figure 4-9 Virtual SCSI

Note: The same disk presentation method applies when presenting disks on a Virtual I/O server to a client logical partition as well as using shared storage pools.

4.3.2 Disk device tuning

The objective of this section is to discuss which AIX hdisk device settings can be tuned and what their purpose is. Most of the settings discussed here are dependant on the type of workload and the performance capabilities of the storage system.

Understanding your I/O workload is important when performing an analysis of which settings need to be tuned. Your workload may be an OLTP type workload processing small block random I/O or conversely a data warehouse type workload that processes large block sequential I/O. The tuning parameters here differ depending on the type of workload running on the system.

It is also important to understand that changing these values here may enable AIX to send more I/O and larger I/O to the storage system, but this adds additional load to the storage system and SAN fabric switches. We suggest that you work with your SAN and storage administrator to understand the effect that tuning the device will have on your storage system.

Table 4-5 provides a summary of the tuning parameters available on an hdisk device and their purpose.

Table 4-5 Tuning parameters on an AIX hdisk device

Setting	Description
algorithm	This determines the method by which AIX distributes I/O down the paths that are available. The typical values are failover, where only one path is used, and round_robin where I/O is distributed across all available paths. Some device drivers add additional options, such as SDDPCM, which adds load_balance, which is similar to round_robin except it has more intelligent queueing. We suggest that you consult your storage vendor to find out the optimal setting.

Setting	Description
hcheck_interval	This is the interval in seconds that AIX sends health check polls to a disk. If failed MPIO paths are found, the failed path will also be polled and re-enabled when it is found to be responding. It is suggested to confirm with the storage vendor what the recommended value to use here is.
max_transfer	This specifies the maximum amount of data that can be transmitted in a single I/O operation. If an application makes a large I/O request, the I/O is broken down into multiple I/Os the size of the max_transfer tunable. Typically, for applications transmitting small block I/O the default 256 KB is sufficient. However, in cases where there is large block streaming workload, the max_transfer size may be increased.
max_coalesce	This value sets the limit for the maximum size of an I/O that the disk driver will create by grouping together smaller adjacent requests. We suggest that the max_coalesce value match the max_transfer value.
queue_depth	The service queue depth of an hdisk device specifies the maximum number of I/O operations that can be in progress simultaneously on the disk. Any requests beyond this number are placed into another queue (wait queue) and remain in a pending state until an earlier request on the disk completes. Depending on how many concurrent I/O operations the backend disk storage can support, this value may be increased. However, this will place additional workload on the storage system.
reserve_policy	This parameter defines the reservation method used when a device is opened. The reservation policy is required to be set appropriately depending on what multipathing algorithm is in place. We suggest that you consult your storage vendor to understand what this should be set to based on the algorithm. Possible values include no_reserve, single_path, PR_exclusive, and PR_shared. This reservation policy is required to be set to no_reserve in a dual VIO server setup with virtual SCSI configuration, enabling both VIO servers to access the device.

As described in Table 4-5 on page 143, the max_transfer setting specifies the maximum amount of data that is transmitted in a single I/O operation. In Example 4-21, a simple I/O test is performed to demonstrate the use of the max_transfer setting. There is an AIX system processing a heavy workload of 1024 k block sequential I/Os with a read/write ratio of 80:20 to an hdisk (hdisk1) which has the max_transfer set to the default of 0x40000, which equates to 256 KB.

Typically, the default max_transfer value is suitable for most small block workloads. However, in a scenario with large block streaming workloads, it is suggested to consider tuning the max_transfer setting.

This is only an example test with a repeatable workload—the difference between achieving good performance and having performance issues is to properly understand your workload, establishing a baseline and tuning parameters individually and measuring the results.

Example 4-21 Simple test using 256KB max_transfer size

```
root@aix1:/ # lsattr -El hdisk1 -a max_transfer
max_transfer 0x40000 Maximum TRANSFER Size True
root@aix1:/ # iostat -D hdisk1 10 1

System configuration: lcpu=32 drives=3 paths=10 vdisks=2

hdisk1         xfer:  %tm_act      bps      tps      bread      bwrtn
                        100.0     2.0G   7446.8        1.6G     391.4M
```

```
               read:      rps  avgserv  minserv  maxserv   timeouts      fails
                        5953.9      8.2      0.6     30.0          0          0
              write:      wps  avgserv  minserv  maxserv   timeouts      fails
                        1492.9     10.1      1.2     40.1          0          0
              queue:  avgtime  mintime  maxtime  avgwqsz    avgsqsz     sqfull
                        20.0      0.0     35.2    145.0       62.0     7446.8
--------------------------------------------------------------------------------
root@aix1:/ #
```

The resulting output of **iostat -D** for a single 10-second interval looking at hdisk1 displays the following:

- Observed throughput is 2 GB per second. This is described as bytes per second (bps).
- This is made up of 7446.8 I/O operations per second. This is described as transfers per second (tps).
- The storage shows an average read service time of 8.2 milliseconds and an average write of 10.1 milliseconds. This is described as average service time (avgserv).
- The time that our application has to wait for the I/O to be processed in the queue is 20 milliseconds. This is described as the average time spent by a transfer request in the wait queue (avgtime). This is a result of our hdisk queue becoming full, which is shown as `sqfull`. The queue has filled up as a result of each I/O 1024 KB I/O request consisting of four 256 KB I/O operations. Handling the queue depth is described later in this section.
- The service queue for the disk was also full, due to the large number of I/O requests.

We knew that our I/O request size was 1024 KB, so we changed our max_transfer on hdisk1 to be 0x100000 which is 1 MB to match our I/O request size. This is shown in Example 4-22.

Example 4-22 Changing the max_transfer size to 1 MB

```
root@aix1:/ # chdev -l hdisk1 -a max_transfer=0x100000
hdisk1 changed
root@aix1:/
```

On completion of changing the max_transfer, we ran the same test again, as shown in Example 4-23, and observed the results.

Example 4-23 Simple test using 1MB max_transfer size

```
root@aix1:/ # lsattr -El hdisk1 -a max_transfer
max_transfer 0x100000 Maximum TRANSFER Size True
root@aix1:/ # iostat -D hdisk1 10 1

hdisk1         xfer:  %tm_act      bps      tps      bread      bwrtn
                        100.0      1.9G   1834.6        1.5G     384.8M
               read:      rps  avgserv  minserv  maxserv   timeouts      fails
                       1467.6     24.5     14.4    127.2          0          0
              write:      wps  avgserv  minserv  maxserv   timeouts      fails
                        367.0     28.6     16.2    110.7          0          0
              queue:  avgtime  mintime  maxtime  avgwqsz    avgsqsz     sqfull
                        0.0       0.0      0.3      0.0       61.0        0.0
--------------------------------------------------------------------------------
root@aix1:/ #
```

The output of **iostat -D** for a single 10-second interval looking at hdisk1 in the second test displayed the following:

- Observed throughput is 1.9 Gb per second. This is almost the same as the first test, shown in bps.
- This is made up of 1,834 I/O operations per second, which is shown in tps in the output in Example 4-23 on page 145. You can see that the number of I/O operations has been reduced by a factor of four, which is a result of moving from a max_transfer size of 256 KB to 1 MB. This means our 1024 KB I/O request is now processed in a single I/O operation.
- The storage shows an average read service time of `24.5` milliseconds and an average write service time of `28.6` milliseconds. This is shown as `avgserv`. Notice here that our service time from the storage system has gone up by a factor of 2.5, while our I/O size is four times larger. This demonstrates that we placed additional load on our storage system as our I/O size increased, while overall the time taken for the 1024 KB read I/O request to be processed was reduced as a result of the change.
- The time that our application had to wait for the I/O to be retrieved from the queue was 0.0 shown as `avgtime`. This was a result of the amount of I/O operations being reduced by a factor of four and their size increased by a factor of four. In the first test for a single read 1024 KB I/O request to be completed, this consisted of four 256 KB I/O operations with a 8.2 millisecond service time and a 20 millisecond wait queue time, giving an overall average response time to the I/O request of 52.8 milliseconds since a single 1024 KB I/O request consists of four 256 KB I/Os.
- In the second test after changing the max_transfer size to 1 MB, we completed the 1024 KB I/O request in a single I/O operation with an average service time of 24.5 milliseconds, giving an average of a 28.3 millisecond improvement per 1024 KB I/O request. This can be calculated by the formula avg IO time = avgtime + avgserv.

The conclusion of this test is that for our large block I/O workload, increasing the value of the max_transfer size to enable larger I/Os to be processed without filling up the disk's I/O queue provided a significant increase in performance.

Important: If you are using virtual SCSI and you change to max_transfer on an AIX hdisk device, it is critical that these settings are replicated on the Virtual I/O server to ensure that the changes take effect.

The next setting that is important to consider is queue_depth on an AIX hdisk device. This is described in Table 4-5 on page 143 as the maximum number of I/O operations that can be in progress simultaneously on a disk device.

To be able to tune this setting, it is important to understand whether your queue is filling up on the disk and what value to set queue_depth to. Increasing queue_depth also places additional load on the storage system, because a larger number of I/O requests are sent to the storage system before they are queued.

Example 4-24 shows how to display the current queue_depth and observe what the maximum queue_depth is that can be set on the disk device. In this case the range is between 1 and 256. Depending on what device driver is in use, the maximum queue_depth may vary. It is always good practice to obtain the optimal queue depth for the storage system and its configuration from your storage vendor.

Example 4-24 Display current queue depth and maximum supported queue depth

```
root@aix1:/ # lsattr -El hdisk1 -a queue_depth
queue_depth 20 Queue DEPTH True
root@paix1:/ # lsattr -Rl hdisk1 -a queue_depth
1...256 (+1)
```

```
root@aix1:/ #
```

Note: In the event that the required queue_depth value cannot be assigned to an individual disk, as a result of being beyond the recommendation by the storage vendor, it is suggested to spread the workload across more hdisk devices because each hdisk has its own queue.

In Example 4-25 a simple test is performed to demonstrate the use of the queue_depth setting. There is an AIX system processing a heavy workload of 8 k small block random I/Os with an 80:20 read write ratio to an hdisk (hdisk1) which has its queue_depth currently set to 20. The **iostat** command issued here shows hdisk1 for a single interval of 10 seconds while the load is active on the system.

Example 4-25 Test execution with a queue_depth of 20 on hdisk1

```
root@aix1:/ # iostat -D hdisk1 10 1

System configuration: lcpu=32 drives=3 paths=10 vdisks=2

hdisk1         xfer:  %tm_act      bps      tps      bread      bwrtn
                        99.9    296.5M   35745.1      237.2M      59.3M
               read:      rps  avgserv  minserv  maxserv   timeouts      fails
                       28534.2      0.2      0.1     48.3           0          0
              write:      wps  avgserv  minserv  maxserv   timeouts      fails
                       7210.9      0.4      0.2     50.7           0          0
              queue:  avgtime  mintime  maxtime  avgwqsz    avgsqsz     sqfull
                        1.1      0.0     16.8     36.0        7.0     33898.5
--------------------------------------------------------------------------------
```

Looking at the resulting output of **iostat -D** in Example 4-25, you can observe the following:

- Our sample workload is highly read intensive and performing 35,745 I/O requests per second (tps) with a throughput of 296 MB per second (bps).
- The average read service time from the storage system is 0.2 milliseconds (avgserv).
- The average wait time per I/O transaction for the queue is 1.1 milliseconds (avgtime) and the disk's queue in the 10-second period iostat was monitoring the disk workload filled up a total of 33,898 times (sqfull).
- The average amount of I/O requests waiting in the service wait queue was 36 (avgwqsz).

Based on this we could add our current queue depth (20) to the number of I/Os on average in the service wait queue (36), and have a queue_depth of 56 for the next test. This should stop the queue from filling up.

Example 4-26 shows changing the queue_depth on hdisk1 to our new queue_depth value. The queue_depth value is our target queue_depth of 56, plus some slight headroom bringing the total queue_depth to 64.

Example 4-26 Changing the queue_depth to 64 on hdisk1

```
root@aix1:/ # chdev -l hdisk1 -a queue_depth=64
hdisk1 changed
root@aix1:/ #
```

Example 4-27 on page 148 demonstrates the same test being executed again, but with the increased queue_depth of 64 on hdisk1.

Example 4-27 Test execution with a queue_depth of 64 on hdisk1

```
root@aix1:/ # iostat -D hdisk1 10 1

System configuration: lcpu=32 drives=3 paths=10 vdisks=2

hdisk1         xfer:  %tm_act      bps      tps      bread      bwrtn
                       100.0    410.4M   50096.9      328.3M      82.1M
               read:      rps  avgserv  minserv  maxserv   timeouts      fails
                      40078.9      0.4      0.1     47.3           0          0
              write:      wps  avgserv  minserv  maxserv   timeouts      fails
                      10018.0      0.7      0.2     51.6           0          0
              queue:  avgtime  mintime  maxtime  avgwqsz    avgsqsz     sqfull
                        0.0      0.0      0.3      0.0       23.0         0.0
--------------------------------------------------------------------------------
```

Looking at the resulting output of **iostat -D** in Example 4-27, you can observe the following:

- Our sample workload is highly read intensive and performing 50,096 I/O requests per second (tps) with a throughput of 410 MB per second (bps). This is significantly more than the previous test.
- The average read service time from the storage stem is 0.4 milliseconds (avgserv), which is slightly more than it was in the first test, because we are processing significantly more I/O operations.
- The average wait time per I/O transaction for the queue is 0 milliseconds (avgtime) and the disk's queue in the 10-second period iostat was monitoring the disk workload did not fill up at all. In contrast to the previous test, where the queue filled up 33,898 times and the wait time for each I/O request was 1.1 milliseconds.
- The average amount of I/O requests waiting in the wait queue was 0 (avgwqsz), meaning our queue was empty; however; additional load was put on the external storage system.

Based on this test, we can conclude that each I/O request had an additional 0.2 millisecond response time from the storage system, while the 1.1 millisecond service queue wait time has gone away, meaning that after making this change, our workload's response time went from 1.3 milliseconds to 0.4 milliseconds.

Important: If you are using virtual SCSI and you change to queue_depth on an AIX hdisk device, it is critical that these settings are replicated on the Virtual I/O server to ensure that the changes take effect.

Note: When you change the max_transfer or queue_depth setting on an hdisk device, it will be necessary that the disk is not being accessed and that the disk is not part of a volume group that is varied on. To change the setting either unmount any file systems and vary off the volume group, or change the queue_depth option with the **-P** flag of the **chdev** command to make the change active at the next reboot.

4.3.3 Pbuf on AIX disk devices

AIX Logical Volume Manager (LVM) uses a construct named pbuf to control a pending disk I/O. Pbufs are pinned memory buffers and one pbuf is always used for each individual I/O request**,** regardless of the amount of data that is supposed to be transferred. AIX creates extra pbufs when a new physical volume is added to a volume group.

Example 4-28 shows the AIX volume group `data_vg` with two physical volumes. We can see that the pv_pbuf_count is 512, which is the pbuf size for each physical volume in the volume group, and the total_vg_pbufs is 1024, which is because there are two physical volumes in the volume group, each with a pbuf size of 512.

Example 4-28 lvmo -av output

```
root@aix1:/ # lsvg -p data_vg
data_vg:
PV_NAME           PV STATE          TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk1            active            399         239         79..00..00..80..80
hdisk2            active            399         239         79..00..00..80..80
root@aix1:/ # lvmo -av data_vg
vgname = data_vg
pv_pbuf_count = 512
total_vg_pbufs = 1024
max_vg_pbufs = 524288
pervg_blocked_io_count = 3047
pv_min_pbuf = 512
max_vg_pbuf_count = 0
global_blocked_io_count = 3136
root@aix1:/ #
```

Also seen in Example 4-28, you can see that the pervg_blocked_io_count is 3047 and the global_blocked_io_count is 3136. This means that the data_vg volume group has 3047 I/O requests that have been blocked due to insufficient pinned memory buffers (pervg_blocked_io_count). Globally across all of the volume groups, 3136 I/O requests have been blocked due to insufficient pinned memory buffers.

In the case where the pervg_blocked_io_count is growing for a volume group, it may be necessary to increase the number of pbuf buffers. This can be changed globally by using ioo to set pv_min_pbuf to a greater number. However, it is suggested to handle this on a per volume group basis.

pv_pbuf_count is the number of pbufs that are added when a physical volume is added to the volume group.

Example 4-29 demonstrates increasing the pbuf buffers for the data_vg volume group from 512 per physical volume to 1024 per physical volume. Subsequently, the total number of pbuf buffers for the volume group is also increased.

Example 4-29 Increasing the pbuf for data_vg

```
root@aix1:/ # lvmo -v data_vg -o pv_pbuf_count=1024
root@aix1:/ # lvmo -av data_vg
vgname = data_vg
pv_pbuf_count = 1024
total_vg_pbufs = 2048
max_vg_pbufs = 524288
pervg_blocked_io_count = 3047
pv_min_pbuf = 512
max_vg_pbuf_count = 0
global_blocked_io_count = 3136
root@aix1:/ #
```

If you are unsure about changing these values, contact IBM Support for assistance.

Note: If at any point the volume group is exported and imported, the pbuf values will reset to their defaults. If you have modified these, ensure that you re-apply the changes in the event that you export and import the volume group.

4.3.4 Multipathing drivers

Drivers for IBM storage include SDDPCM for IBM DS8000, DS6000, SAN Volume Controller, and Storwize® V7000 as well as the XIV® Host Attachment kit for an XIV Storage System.

The best source of reference for which driver to use is the IBM System Storage Interoperation Center (SSIC), which provides details on drivers for IBM storage at:

`http://www-03.ibm.com/systems/support/storage/ssic/interoperability.wss`

Third-party drivers should be obtained from storage vendors and installed to deliver the best possible performance.

4.3.5 Adapter tuning

The objective of this section is to detail what AIX storage adapter device settings can be tuned to and their purpose. Three scenarios are covered here:

- Dedicated fiber channel adapters
- NPIV virtual fiber channel adapters
- Virtual SCSI

The most important thing to do when tuning the adapter settings is to understand the workload that the disks associated with the adapters are handling and what their configuration is. 4.3.2, “Disk device tuning” on page 143 details the configuration attributes that are applied to hdisk devices in AIX.

Dedicated fiber channel adapters

The scenario of dedicated fiber channel (FC) adapters entails an AIX system or logical partition (LPAR) with exclusive use of one or more FC adapters. There are two devices associated with an FC adapter:

fcs*N* This is the actual adapter itself, and there is one of these devices per port on a fiber channel card. For example, you may have a dual port fiber channel card. Its associated devices could be fcs0 and fcs1.

fscsi*N* This is a child device that the FC adapter has which acts as a SCSI software interface to handle SCSI commands related to disk access. If you have a dual port fiber channel card associated with devices fcs0 and fcs1, their respective child devices will be fscsi0 and fscsi1.

Table 4-6 on page 151 describes attributes of the fcs device which it is advised to consider tuning.

Table 4-6 fcs device attributes

Attribute	Description
lg_term_dma	The attribute lg_term_dma is the size in bytes of the DMA memory area used as a transfer buffer. The default value of 0x800000 in most cases is sufficient unless there is a very large number of fiber channel devices attached. This value typically should only be tuned under the direction of IBM Support.
max_xfer_size	The max_xfer_size attribute dictates the maximum transfer size of I/O requests. Depending on the block size of the workload, this value may be increased from the default 0x100000 (1 MB) to 0x200000 (2 MB) when there are large block workloads, and the hdisk devices are tuned for large transfer sizes. This attribute must be large enough to accommodate the transfer sizes used by any child devices, such as an hdisk device.
num_cmd_elems	The attribute num_cmd_elems is the queue depth for the adapter. The maximum value for a fiber channel adapter is 2048 and this should be increased to support the total amount of I/O requests that the attached devices are sending to the adapter.

When tuning the attributes described in Table 4-6, the **fcstat** command can be used to establish whether the adapter is experiencing any performance issues (Example 4-30).

Example 4-30 fcstat output

```
root@aix1:/ # fcstat fcs0

FIBRE CHANNEL STATISTICS REPORT: fcs0

Device Type: 8Gb PCI Express Dual Port FC Adapter (df1000f114108a03)
(adapter/pciex/df1000f114108a0)
Serial Number: 1C041083F7
Option ROM Version: 02781174
ZA: U2D1.11X4
World Wide Node Name: 0x20000000C9A8C4A6
World Wide Port Name: 0x10000000C9A8C4A6

FC-4 TYPES:
  Supported: 0x0000012000000000000000000000000000000000000000000000000000000000
  Active:    0x0000010000000000000000000000000000000000000000000000000000000000
Class of Service: 3
Port Speed (supported): 8 GBIT
Port Speed (running):   8 GBIT
Port FC ID: 0x010000
Port Type: Fabric

Seconds Since Last Reset: 270300

        Transmit Statistics     Receive Statistics
        -------------------     ------------------
Frames: 2503792149              704083655
Words:  104864195328            437384431872

LIP Count: 0
NOS Count: 0
```

```
Error Frames:  0
Dumped Frames: 0
Link Failure Count: 0
Loss of Sync Count: 8
Loss of Signal: 0
Primitive Seq Protocol Error Count: 0
Invalid Tx Word Count: 31
Invalid CRC Count: 0

IP over FC Adapter Driver Information
  No DMA Resource Count: 3207
  No Adapter Elements Count: 126345

FC SCSI Adapter Driver Information
  No DMA Resource Count: 3207
  No Adapter Elements Count: 126345
  No Command Resource Count: 133


IP over FC Traffic Statistics
  Input Requests:   0
  Output Requests:  0
  Control Requests: 0
  Input Bytes:  0
  Output Bytes: 0

FC SCSI Traffic Statistics
  Input Requests:   6777091279
  Output Requests:  2337796
  Control Requests: 116362
  Input Bytes:  57919837230920
  Output Bytes: 39340971008
#
```

Highlighted in bold in the **fcstat** output in Example 4-30 on page 151 are the items of interest. These counters are held since system boot and Table 4-7 describes the problem and the suggested action.

Table 4-7 Problems detected in fcstat output

Problem	Action
No DMA Resource Count increasing	Increase max_xfer_size
No Command Resource Count	Increase num_cmd_elems

In Example 4-30 on page 151 we noticed that all three conditions in Table 4-7 are met, so we increased num_cmd_elems and max_xfer_size on the adapter.

Example 4-31 shows how to change fcs0 to have a queue depth (num_cmd_elems) of 2048, and a maximum transfer size (max_xfer_size) of 0x200000 which is 2 MB. The **-P** option was used on the **chdev** command for the attributes to take effect on the next reboot of the system.

Example 4-31 Modify the AIX fcs device

```
root@aix1:/ # chdev -l fcs0 -a num_cmd_elems=2048 -a max_xfer_size=0x200000 -P
fcs0 changed
```

```
root@aix1:/ #
```

Note: It is important to ensure that all fcs devices on the system that are associated with the same devices are tuned with the same attributes. If you have two FC adapters, you need to apply the settings in Example 4-31 to both of them.

There are no performance related tunables that can be set on the fscsi devices. However, there are two tunables that are applied. These are described in Table 4-8.

Table 4-8 fscsi device attributes

Attribute	Description
dyntrk	Dynamic tracking (dyntrk) is a setting that enables devices to remain online during changes in the SAN that cause an N_Port ID to change. This could be moving a cable from one switch port to another, for example.
fc_err_recov	Fiber channel event error recovery (fc_err_recov) has two possible settings. These are delayed_fail and fast_fail. The recommended setting is fast_fail when multipathed devices are attached to the adapter.

Example 4-32 demonstrates how to enable dynamic tracking and set the fiber channel event error recovery to fast_fail. The **-P** option on **chdev** will set the change to take effect at the next reboot of the system.

Example 4-32 Modify the AIX fscsi device

```
root@aix1:/ # chdev -l fscsi0 -a dyntrk=yes -a fc_err_recov=fast_fail -P
fscsi0 changed
root@aix1:/ #
```

Note: It is important to ensure that all fscsi devices on the system that are associated with the same devices are tuned with the same attributes.

NPIV

The attributes applied to a virtual fiber channel adapter on an AIX logical partition are exactly the same as those of a physical fiber channel adapter, and should be configured exactly as described in dedicated fiber channel adapters in this section.

The difference with NPIV is that on the VIO server there is a fiber channel fcs device that is shared by up to 64 client logical partitions. As a result there are a few critical considerations when using NPIV:

- Does the queue depth (num_cmd_elems) attribute on the fcs device support all of the logical partitions connecting NPIV to the adapter? In the event that the **fcstat** command run on the Virtual I/O server provides evidence that the queue is filling up (no adapter elements count and no command resource count), the queue depth will need to be increased. If queue_depth has already been increased, the virtual fiber channel mappings may need to be spread across more physical fiber channel ports where oversubscribed ports are causing a performance degradation.
- Does the maximum transfer size (max_xfer_size) set on the physical adapter on the VIO server match the maximum transfer size on the client logical partitions accessing the port? It is imperative that the maximum transfer size set in AIX on the client logical partition

matches the maximum transfer size set on the VIO server's fcs device that is being accessed.

Example 4-33 demonstrates how to increase the queue depth and maximum transfer size on a physical FC adapter on a VIO server.

Note: In the event that an AIX LPAR has its fcs port's attribute max_xfer_size greater than that of the VIO server's fcs port attribute max_xfer_size, it may cause the AIX LPAR to hang on reboot.

Example 4-33 Modify the VIO fcs device

```
$ chdev -dev fcs0 -attr num_cmd_elems=2048 max_xfer_size=0x200000 -perm
fcs0 changed
$
```

The settings `dynamic tracking` and `FC error recovery` discussed in Table 4-8 on page 153 are enabled by default on a virtual FC adapter in AIX. They are not, however, enabled by default on the VIO server. Example 4-34 demonstrates how to enable dynamic tracking and set the FC error recovery to fast fail on a VIO server.

Example 4-34 Modify the VIO fscsi device

```
$ chdev -dev fscsi0 -attr dyntrk=yes fc_err_recov=fast_fail -perm
fscsi0 changed
$
```

Note: If the adapter is in use, you have to make the change permanent with the **-perm** flag of **chdev** while in restricted shell. However, this change will only take effect next time the VIOS is rebooted.

Virtual SCSI

There are no tunable values related to performance for virtual SCSI adapters. However, there are two tunables that should be changed from their defaults in an MPIO environment with dual VIO servers. The virtual SCSI description in this section applies to both shared storage pools and traditional virtual SCSI implementations.

These settings are outlined in Table 4-9.

Table 4-9 vscsi device attributes

Attribute	Description
vscsi_err_recov	The vscsi_err_recov is used to determine how the vscsi driver will handle failed I/O requests. Possible values are set to delayed_fail and fast_fail. In scenarios where there are dual VIO servers and disk devices are multipathed, the recommended value is fast_fail so that in the event that an I/O request cannot be serviced by a path, that path is immediately failed. The vscsi_err_recov attribute is set to delayed_fail by default. Note that there is no load balancing supported across devices multipathed on multiple vscsi adapters.

Attribute	Description
vscsi_path_to	This is disabled by being set to 0 by default. The vscsi_path_to attribute allows the vscsi adapter to determine the health of its associated VIO server and in the event of a path failure, it is a polling interval in seconds where the failed path is polled and once it is able to resume I/O operations the path is automatically re-enabled.

Example 4-35 demonstrates how to enable vscsi_err_recov to fast fail, and set the vscsi_path_to to be set to 30 seconds.

Example 4-35 Modify the AIX vscsi device

```
root@aix1:/ # chdev -l vscsi0 -a vscsi_path_to=30 -a vscsi_err_recov=fast_fail -P
vscsi0 changed
root@aix1:/ # chdev -l vscsi1 -a vscsi_path_to=30 -a vscsi_err_recov=fast_fail -P
vscsi1 changed
```

Note: If the adapter is in use, you have to make the change permanent with the **-P** flag. This change will take effect next time AIX is rebooted.

In a virtual SCSI MPIO configuration, there is a path to each disk per virtual SCSI adapter. For instance, in Example 4-36, we have three virtual SCSI disks. One is the root volume group, the other two are in a volume group called data_vg.

Example 4-36 AIX virtual SCSI paths

```
root@aix1:/ # lspv
hdisk0          00f6600e0e9ee184                    rootvg          active
hdisk1          00f6600e2bc5b741                    data_vg         active
hdisk2          00f6600e2bc5b7b2                    data_vg         active
root@aix1:/ # lspath
Enabled hdisk0 vscsi0
Enabled hdisk0 vscsi1
Enabled hdisk1 vscsi0
Enabled hdisk2 vscsi0
Enabled hdisk1 vscsi1
Enabled hdisk2 vscsi1
root@aix1:/ #
```

By default, all virtual SCSI disks presented to a client logical partition use the first path by default. Figure 4-10 illustrates a workload running on the two virtual SCSI disks multipathed across with all of the disk traffic being transferred through vscsi0, and no traffic being transferred through vscsi1.

```
Disk-Adapter-I/O
Name          %busy      read     write        xfers Disks Adapter-Type
vscsi0         13.5       0.0  11723.1 KB/s   249.5 4     Virtual SCSI Client A
vscsi1          0.0       0.0      0.0 KB/s     0.0 3     Virtual SCSI Client A
TOTALS  2 adapters        0.0  11723.1 KB/s   249.5 7     TOTAL(MB/s)=11.4
```

Figure 4-10 Unbalanced vscsi I/O

It is not possible to use a round robin or load balancing type policy on a disk device across two virtual SCSI adapters. A suggested way to get around this is to have your logical volume spread or striped across an even number of hdisk devices with half transferring its data through one vscsi adapter and the other half transferring its data through the other vscsi adapter. The logical volume and file system configuration is detailed in 4.4, “AIX LVM and file systems” on page 157.

Each path to an hdisk device has a path priority between 1 and 255. The path with the lowest priority is used as the primary path. To ensure that I/O traffic is transferred through one path primarily you can change the path priority of each path. In Example 4-37, hdisk2 has a path priority of 1 for each vscsi path. To balance the I/O, you can change vscsi1 to be the primary path by setting a higher path priority on the vscsi1 adapter.

Example 4-37 Modifying vscsi path priority

```
root@aix1:/ # lspath -AEl hdisk2 -p vscsi0
priority 1 Priority True
root@aix1:/ # lspath -AEl hdisk2 -p vscsi1
priority 1 Priority True
root@aix1:/ # chpath -l hdisk2 -a priority=2 -p vscsi0
path Changed
root@aix3:/ #
```

In Figure 4-11, the exact same test is performed again and we can see that I/O is evenly distributed between the two vscsi adapters.

```
Disk-Adapter-I/O
Name            %busy      read     write        xfers Disks Adapter-Type
vscsi0           10.0       0.0    7295.4 KB/s   158.0 4     Virtual SCSI Client A
vscsi1           10.5       0.0    7391.4 KB/s   159.0 3     Virtual SCSI Client A
TOTALS  2 adapters          0.0   14686.8 KB/s   317.0 7     TOTAL(MB/s)=14.3
```

Figure 4-11 Balanced vscsi I/O

Another performance implication is the default queue_depth of a VSCSI adapter of 512 per adapter. However, two command elements are reserved for the adapter itself and three command elements for each virtual disk.

The number of command elements (queue depth) of a virtual SCSI adapter cannot be changed, so it is important to work out how many virtual SCSI adapters you will need.

Initially, you need to understand two things to calculate how many virtual SCSI adapters are be required:

- How many virtual SCSI disks will be mapped to the LPAR?
- What will be the queue_depth of each virtual SCSI disk? Calculating the queue depth for a disk is covered in detail in “Disk device tuning” on page 143.

The formula for how many virtual drives can be mapped to a virtual SCSI adapter is:

```
virtual_drives = (512 - 2) / (queue_depth_per_disk + 3)
```

For example, each virtual SCSI disk has a queue_depth of 32. You can have a maximum of 14 virtual SCSI disks assigned to each virtual SCSI adapter:

```
(512 - 2) / (32 + 3) = 14.5
```

In the event that you require multiple virtual SCSI adapters, Figure 4-12 provides a diagram of how this can be done.

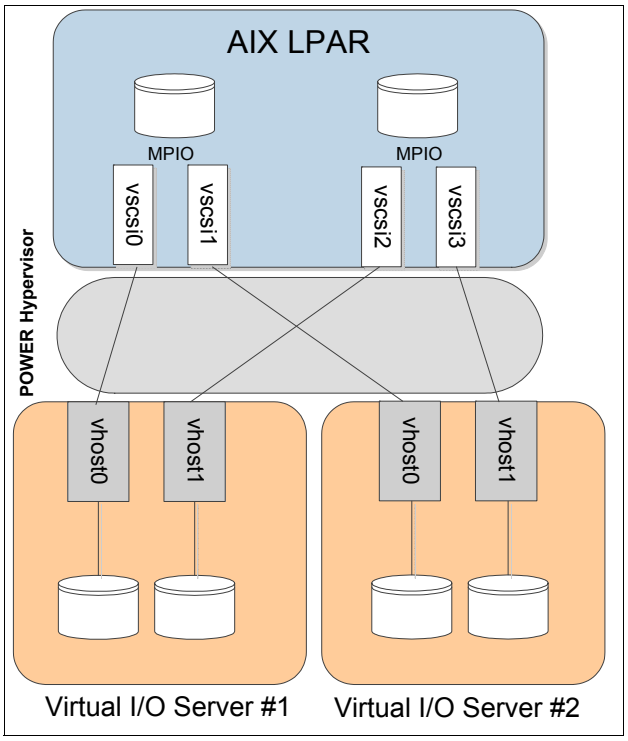


Figure 4-12 Example AIX LPAR with four vscsi adapters

Note: *IBM PowerVM Virtualization Introduction and Configuration*, SG24-7940-04, explains in detail how to configure virtual SCSI devices.

4.4 AIX LVM and file systems

In this chapter we focus on LVM and file systems performance, and best practices.

4.4.1 Data layout

Data layout is the most important part in I/O performance. The ultimate goal is to balance I/O across all paths including adapters, loops, disks, and to avoid I/O hotspot. Usually this contributes more to performance than any I/O tunables. In the following section, we introduce best practices on balancing I/O, and share some experiences on monitoring.

Random I/O best practice

For random I/O, the aim is to spread I/Os evenly across all physical disks. Here are some general guidelines:

- On disk subsystem, create arrays of equal size and type.
- Create VGs with one LUN per array.
- Spread all LVs across all PVs in the VG.

Sequential I/O best practice

For sequential I/O, the aim is to ensure full stripe write on the storage RAID. Here are some general guidelines:

1. Create RAID arrays with data spread across a power of two of disks.
   a. RAID 5 arrays of 4+1 or 8+1 disks
   b. RAID10 arrays of 4 or 8 disks
2. Create VGs with one LUN per array.
3. Create LVs that are spread across all PVs in the VG using a PP or LV strip size larger than or equal to the full stripe size on the RAID array
   a. The number of data disks times the segment size is equal to the array (full) stripe size.
   b. 8+1 RAID5 with a 256 KB segment size → 8 * 256 KB = 2048 KB stripe size
   c. 4+4 RAID10 with a 256 KB segment size → 4 * 256 KB = 1024 KB stripe size
4. Application I/Os equal to or a multiple of a full stripe on the RAID array.

Note: Ensure full stripe write is critical for RAID5 to avoid write penalties. Also we suggest that you check with your storage vendor for any specific best practice related to your storage system.

How to determine the nature of I/O

We can have an empirical judgement on whether the I/O type is sequential or random. For example, database files are usually random, and log files are usually sequential. There are tools to observe this. Example 4-38 shows a **filemon** approach to identify whether the current I/O is sequential or random. For more details on the **filemon** utility, refer to 4.4.4, “The filemon utility” on page 176.

Example 4-38 filemon usage

```
# filemon -T 1000000 -u -O all,detailed -o fmon.out
# sleep 3
# trcstop
```

Example 4-39 shows output of the **filemon** with the options specified in Example 4-38. The percent of seeks indicates the nature of the I/O. If seek is near zero, it means the I/O is sequential. If seek is near 100%, most I/Os are random.

Example 4-39 filemon ouput

```
------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------
...
VOLUME: /dev/sclvdst1  description: N/A
reads:   39 (0 errs)
  read sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):avg  30.485 min   3.057 max 136.050 sdev  31.600
  read sequences: 39
  read seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
writes:  22890(0 errs)
  write sizes (blks): avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):avg  15.943 min   0.498 max  86.673 sdev  10.456
  write sequences: 22890
```

```
  write seq. lengths:avg     8.0 min       8 max       8 sdev     0.0
seeks:   22929(100.0%)
  seek dist (blks):init 1097872,
        avg 693288.4 min      56 max 2088488 sdev 488635.9
time to next req(msec): avg   2.651 min   0.000 max 1551.894 sdev  53.977
throughput:1508.1 KB/sec
utilization:0.05
```

Note: Sequential I/O might degrade to random I/O if the data layout is not appropriate. If you get a **filemon** result that is contrary to empirical judgement, pay more attention to it. It might indicate a data layout problem.

RAID policy consideration

Table 4-10 explains the general performance comparison between RAID5 and RAID10. Here are some general guidelines:

- With enterprise class storage (large cache), RAID-5 performances are comparable to RAID-10 (for most customer workloads).
- Consider RAID-10 for workloads with a high percentage of random write activity (> 25%) and high I/O access densities (peak > 50%).

  RAID5 is not a good choice for such situations due to write penalty in random write access. One random write might result in two read operations and two write operations.

Table 4-10 RAID5 and RAID10 performance comparison

I/O characteristics	RAID-5	RAID-10
Sequential read	Excellent	Excellent
Sequential write	Excellent	Good
Random read	Excellent	Excellent
Random write	Fair	Excellent

4.4.2 LVM best practice

Here are some general guidelines:

- Use scalable VGs for AIX 5.3 and later releases as it has no LVCB in the head of the LV which ensure better I/O alignment. Also scalable VG has larger maximum PV/LV/PP numbers per VG.
- Use RAID in preference to LVM mirroring

  Using RAID reduces I/Os because there is no additional writes to ensure mirror write consistency (MWC) compared to LVM mirroring.
- LV striping best practice
  - Create a logical volume with the striping option.
    - **mklv -S <strip-size> ...**
    - Specify the stripe width with the **-C** or the **-u** option, or specify two or more physical volumes.
    - When using LV striping, the number of logical partitions should be a multiple of the stripe width. Example 4-40 on page 160 shows an example of creating logical volumes (LV) with 2 MB striping.

Example 4-40 create an LV using LV striping

```
#mklv -t jfs2 -C 4 -S2M -y lvdata01 datavg 32
```

- Valid LV strip sizes range from 4 KB to 128 MB in powers of 2 for striped LVs. The SMIT panels may not show all LV strip options, depending on your AIX version.
- Use an LV strip size larger than or equal to the stripe size on the storage side, to ensure full stripe write. Usually the LV strip size should be larger than 1 MB. Choose the strip size carefully, because you cannot change the strip size after you created the LV.
- Do not use LV striping for storage systems that already have the LUNs striped across multiple RAID/disk groups such as XIV, SVC, and V7000. We suggest PP striping for this kind of situation.

Note: We use the glossary strip here. The LV strip size multiplied by the LV stripe width (number of disks for the striping) equals the stripe size of the LV.

- PP striping best practice
  - Create LV with the maximum range of physical volumes option to spread PP on different hdisks in a round robin fashion:

    # **mklv –e x ...** as shown in Example 4-41.

    Example 4-41 create lv using PP striping

    ```
    #mklv -t jfs2 -e x -y lvdata02 datavg 32
    ```

  - Create a volume group with an 8 M,16 M or 32 M PP size. PP size is the strip size.

Note: LV striping can specify smaller strip sizes than PP striping, and this sometimes gets better performance in a random I/O scenario. However, it would be more difficult to add physical volumes to the LV and rebalance the I/O if using LV striping. We suggest to use PP striping unless you have a good reason for LV striping.

LVM commands

This section explains LVM commands.

1. **lsvg** can be used to view VG properties. As in Example 4-42, MAX PVs is equal to 1024, which means it is a scalable volume group.

   Example 4-42 lsvg output

```
#lsvg datavg
VOLUME GROUP:       datavg                  VG IDENTIFIER:
00f6601100004c000000013a32716c83
VG STATE:           active                  PP SIZE:        32 megabyte(s)
VG PERMISSION:      read/write              TOTAL PPs:      25594 (819008
megabytes)
MAX LVs:            256                     FREE PPs:       24571 (786272
megabytes)
LVs:                6                       USED PPs:       1023 (32736
megabytes)
OPEN LVs:           4                       QUORUM:         2 (Enabled)
TOTAL PVs:          2                       VG DESCRIPTORS: 3
```

```
STALE PVs:          0                       STALE PPs:      0
ACTIVE PVs:         2                       AUTO ON:        yes
MAX PPs per VG:     32768                   MAX PVs:        1024
LTG size (Dynamic): 1024 kilobyte(s)        AUTO SYNC:      no
HOT SPARE:          no                      BB POLICY:      relocatable
MIRROR POOL STRICT: off
PV RESTRICTION:     none                    INFINITE RETRY: no
```

- Use **lslv** to view the policy of an LV as shown in Example 4-43.
  - INTER-POLICY equal to “maximum” means the LV is using PP striping policy.
  - UPPER BOUND specifies the maximum number of PVs the LV can be created on. 1024 means the volume group is scalable VG.
  - DEVICESUBTYPE equals to DS_LVZ, which means there is no LVCB in the head of the LV.
  - IN BAND value shows the percentage of partitions that met the intra-policy criteria of the LV.

Example 4-43 lslv command output

```
#lslv testlv
LOGICAL VOLUME:     testlv                  VOLUME GROUP:   datavg
LV IDENTIFIER:      00f6601100004c000000013a32716c83.5 PERMISSION:
read/write
VG STATE:           active/complete         LV STATE:       closed/syncd
TYPE:               jfs                     WRITE VERIFY:   off
MAX LPs:            512                     PP SIZE:        32 megabyte(s)
COPIES:             1                       SCHED POLICY:   parallel
LPs:                20                      PPs:            20
STALE PPs:          0                       BB POLICY:      relocatable
INTER-POLICY:       maximum                 RELOCATABLE:    yes
INTRA-POLICY:       middle                  UPPER BOUND:    1024
MOUNT POINT:        N/A                     LABEL:          None
DEVICE UID:         0                       DEVICE GID:     0
DEVICE PERMISSIONS: 432
MIRROR WRITE CONSISTENCY: on/ACTIVE
EACH LP COPY ON A SEPARATE PV ?: yes
Serialize IO ?:     NO
INFINITE RETRY:     no
DEVICESUBTYPE:      DS_LVZ
COPY 1 MIRROR POOL: None
COPY 2 MIRROR POOL: None
COPY 3 MIRROR POOL: None

#lslv -l testlv
testlv:N/A
PV                  COPIES        IN BAND       DISTRIBUTION
hdisk2              010:000:000   100%          000:010:000:000:000
hdisk1              010:000:000   100%          000:010:000:000:000
```

- Use **lslv -p hdisk# lvname** to show the placement of LV on the specific hdisk, as shown in Example 4-44 on page 162. The state of the physical partition is as follows:
  - USED means the physical partition is used by other LVs than the one specified in the command.

– Decimal number means the logical partition number of the LV lies on the physical partition.
– FREE means the physical partition is not used by any LV.

Example 4-44 lslv -p output

```
# lslv -p hdisk2 informixlv
hdisk2:informixlv:/informix
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED       1-10
…
USED   USED   USED   USED                                        101-104

0001   0002   0003   0004   0005   0006   0007   0008   0009   0010     105-114
0011   0012   0013   0014   0015   0016   0017   0018   0019   0020     115-124
0021   0022   0023   0024   0025   0026   0027   0028   0029   0030     125-134
…
0228   0229   0230   0231   0232   0233   0234   0235   0236   0237     332-341
0238   0239   0240   USED   USED   USED   USED   USED   USED   USED     342-351
…

USED   USED   USED   FREE                                              516-519
```

- Use **lslv -m lvname** to show the mapping of the LV, as shown in Example 4-45.

Example 4-45 lslv -m lvname

```
#lslv -m testlv
testlv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  2881 hdisk2
0002  2882 hdisk1
0003  2882 hdisk2
0004  2883 hdisk1
0005  2883 hdisk2
0006  2884 hdisk1
0007  2884 hdisk2
0008  2885 hdisk1
0009  2885 hdisk2
0010  2886 hdisk1
0011  2886 hdisk2
0012  2887 hdisk1
0013  2887 hdisk2
0014  2888 hdisk1
0015  2888 hdisk2
0016  2889 hdisk1
0017  2889 hdisk2
0018  2890 hdisk1
0019  2890 hdisk2
0020  2891 hdisk1
```

- Use **lspv -p hdisk1** to get the distribution of LVs on the physical volume, as shown in Example 4-46.

Example 4-46 lspv -p hdisk#

```
#lspv -p hdisk1|pg
```

```
hdisk1:
PP RANGE  STATE   REGION        LV NAME             TYPE       MOUNT POINT
  1-2560  free    outer edge
2561-2561  used    outer middle  loglv00             jfs2log    N/A
2562-2721  used    outer middle  fslv00              jfs2       /iotest
2722-2881  used    outer middle  fslv02              jfs2       /ciotest512
2882-2891  used    outer middle  testlv              jfs        N/A
2892-5119  free    outer middle
5120-7678  free    center
7679-10237 free    inner middle
10238-12797 free    inner edge
```

Logical track group (LTG) size consideration

When the LVM layer receives an I/O request, it breaks the I/O down into multiple logical track group (LTG) sized I/O requests, and then submits them to the device driver of the underlying disks.

Thus LTG is actually the maximum transfer size of an LV, and it is common to all LVs in the same VG. LTG is similar to the MTU in network communications. Valid LTG sizes include 4 K, 8 K, 16 K, 32 K, 64 K, 128 K, 1 M, 2 M, 4 M, 8 M, 16 M, 32 M, and 128 M.

The LTG size should not be larger than the lowest maximum transfer size of the underlying disks in the same volume group. Table 4-11 shows the max transfer size attribute in different I/O layers.

Table 4-11 max transfer sizes in AIX I/O stack

LVM layer	logical track group (LTG)
Disk device drivers	max_transfer
Adapter device drivers	max_xfer_size

For performance considerations, the LTG size should match the I/O request size of the application. The default LTG value is set to the lowest maximum transfer size of all the underlying disks in the same VG. The default is good enough for most situations.

4.4.3 File system best practice

Journaled File System (JFS) is the default file system in AIX 5.2 and earlier AIX releases, while the Enhanced Journaled File System (JFS2) is the default file system for AIX 5.3 and later AIX releases. We can exploit JFS/JFS2 features according to application characteristics for better performance.

Conventional I/O

For read operations, the operating system needs to access the physical disk, read the data into file system cache, and then copy the cache data into the application buffer. The application is blocked until the cache data is copied into the application buffer.

For write operations, the operating system copies the data from the application buffer into file system cache, and flushes the cache to physical disk later at a proper time. The application returns after the data is copied to the file system cache, and thus there is no block of the physical disk write.

This kind of I/O is usually suitable for workloads that have a good file system cache hit ratio. Applications that can benefit from the read ahead and write behind mechanism are also good candidates for conventional I/O. The following section is a brief introduction of the read ahead and write behind mechanism.

- Read ahead mechanism

  JFS2 read ahead is controlled by two ioo options, j2_minPageReadAhead and j2_maxPageReadAhead, specifying the minimum page read ahead and maximum page read ahead, respectively. The j2_minPageReadAhead option is 2 by default, and it is also the threshold value to trigger an I/O read ahead. You can disable the sequential I/O read ahead by setting j2_minPageReadAhead to 0, if the I/O pattern is purely random.

  The corresponding options for JFS are minpgahead and maxpghead. The functionality is almost the same as the JFS2 options.

- Write behind mechanism

  There are two types of write behind mechanisms for JFS/JFS2, as follows:

  – Sequential write behind

    JFS2 sequential write behind is controlled by the j2_nPagesPerWriteBehindCluster option, which is 32 by default. This means that if there are 32 consecutive dirty pages in the file, a physical I/O will be scheduled. This option is good for smoothing the I/O rate when you have an occasional I/O burst.

    It is worthwhile to change j2_nPagesPerWriteBehindCluster to a larger value if you want to keep more pages in RAM before scheduling a physical I/O. However, this should be tried with caution because it might cause a heavy workload to syncd, which runs every 60 seconds by default.

    The corresponding ioo option for JFS is **`numclust`** in units of 16 K.

    Note: This is a significant difference of AIX JFS/JFS2 from other file sytems. If you are doing a small sized **`dd`** test less than the memory size, you will probably find the response time on AIX JFS2 to be much longer than on other operating systems. You can disable the sequential write behind by setting j2_nPagesPerWriteBehindCluster to 0 to get the same behavior. However, we suggest you keep the default value as it is, which is usually a better choice for most real workloads.

  – Random write behind

    JFS2 random write behind is used to control the number of random dirty pages to reduce the workload of syncd. This reduces the possible application pause when accessing files due to the inode locking when syncd is doing a flush. The random write behind is controlled by the j2_maxRandomWrite and j2_nRandomCluster ioo option, and is disabled by default on AIX.

    The corresponding ioo option for JFS is **`maxrandwrt`**.

As just mentioned, the JFS/JFS2 file system will cache the data in read and write accesses for future I/O operations. If you do not want to reuse the AIX file system cache, there are release behind mount options to disable it. Usually these features are good for doing an archive or recovering from an achive. Table 4-12 on page 165 gives an explanation of these mount options. Note that these options only apply when doing sequential I/O.

Table 4-12 Release behind options

Mount options	Explanation
rbr	Release behind when reading; it only applies when sequential I/O is detected.
rbw	Release behind when writing; it only applies when sequential I/O is detected.
rbrw	The combination of rbr and rbw.

Direct I/O

Compared to conventional I/O, direct I/O bypasses the file system cache layer (VMM), and exchanges data directly with the disk. An application that already has its own cache buffer is likely to benefit from direct I/O. To enable direct I/O, mount the file system with the **dio** option, as shown in Example 4-47.

Example 4-47 mount with DIO option

```
#mount -o dio <file system name>
```

To make the option persistent across a boot, use the **chfs** command as shown in Example 4-48 because it adds the mount options to the stanza of the related file system in `/etc/filesystems`.

Example 4-48 Use chfs to set the direct I/O option

```
#chfs -a options=dio /diotest
```

The application can also open the file with O_DIRECT to enable direct I/O. You can refer to the manual of the **open** subroutine on the AIX infocenter for more details at:

http://pic.dhe.ibm.com/infocenter/aix/v6r1/topic/com.ibm.aix.basetechref/doc/basetrf1/open.htm

Note: For DIO and CIO, the read/write requests should be aligned on the file block size boundaries. Both the offset and the length of the I/O request should be aligned. Otherwise, it might cause severe performance degradation due to I/O demotion.

For a file system with a smaller file block size than 4096, the file must be allocated first to avoid I/O demotion. Otherwise I/O demotions still occur during the file block allocations.

Table 4-13 explains the alignment requirements for DIO mounted file systems.

Table 4-13 Alignment requirements for DIO and CIO file systems

Available file block sizes at file system creation	I/O request offset	I/O request length
agblksize='512', '1024', 2048', '4096'	Multiple of agblksize	Multiple of agblksize

Example 4-49 on page 166 shows the trace output of a successful DIO write when complying with the alignment requirements. For details on tracing facilities, refer to "Trace tools and PerfPMR" on page 316.

Example 4-49 successful DIO operations

```
#trace -aj 59B
#sleep 5; #trcstop
#trcrpt > io.out
#more io.out
...
59B     9.232345185       0.008076                  JFS2 IO write: vp =
F1000A0242B95420, sid = 800FC0, offset = 0000000000000000, length = 0200
59B     9.232349035       0.003850                  JFS2 IO dio move: vp =
F1000A0242B95420, sid = 800FC0, offset = 0000000000000000, length = 0200
//comments: “JFS2 IO dio move” means dio is attempted.
59B     9.232373074       0.024039                  JFS2 IO dio devstrat: bplist
= F1000005B01C0228, vp = F1000A0242B95420, sid = 800FC0, lv blk = 290A, bcount =
0200
//comments: “JFS2 IO dio devstrat” will be displayed if the alignment requirements
are met. The offset is 0, and length is 0x200=512, whilst the DIO file system is
created with agblksize=512.
59B     9.232727375       0.354301                  JFS2 IO dio iodone: bp =
F1000005B01C0228, vp = F1000A0242B95420, sid = 800FC0
//comments: “JFS2 IO dio iodone” will be displayed if DIO is finished
successfully.
```

Example 4-50 shows an I/O demotion situation when failing to comply with the alignment requirements, and how to identify the root cause of the I/O demotion.

Example 4-50 DIO demotion

```
#trace -aj 59B
#sleep 5; trcstop
#trcrpt > io.out
#more io.out
...
59B     1.692596107       0.223762                  JFS2 IO write: vp =
F1000A0242B95420, sid = 800FC0, offset = 00000000000001FF, length = 01FF
59B     1.692596476       0.000369                  JFS2 IO dio move: vp =
F1000A0242B95420, sid = 800FC0, offset = 00000000000001FF, length = 01FF
//comments: a DIO attempt is made, however, the alignment requirements are not
met. The offset and length is both 0x1FF, which is 511. While the file system is
created with agblksize=512.
...
59B     1.692758767       0.018394                  JFS2 IO dio demoted: vp =
F1000A0242B95420, mode = 0001, bad = 0002, rc = 0000,
 rc2 = 0000
//comments: “JFS2 IO dio demoted” means there is I/O demotion.

To locate the file involved in the DIO demotion, we can use the svmon command. As
in the trcrpt output above, “sid = 800FC0” when the demoted I/O happens.
#svmon -S 800FC0 -O filename=on
Unit: page

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
  800fc0         - clnt /dev/fslv00:5                s      0     0    -       -
                        /iotest/testw
Then we know that DIO demotion happened on file “/iotest/testw”.
```

AIX trace can also be used to find the process or thread that caused the I/O demotion. Refer to “Trace tools and PerfPMR” on page 316. There is also an easy tool provided to identify I/O demotion issues.

Note: CIO is implemented based on DIO; thus the I/O demotion detection approaches also apply for CIO mounted file systems.

Concurrent I/O

POSIX standard requires file systems to impose inode locking when accessing files to avoid data corruption. It is a kind of read write lock that is shared between reads, and exclusive between writes.

In certain cases, applications might already have a finer granularity lock on their data files, such as database applications. Inode locking is not necessary in these situations. AIX provides concurrent I/O for such requirements. Concurrent I/O is based on direct I/O, but enforces the inode locking in shared mode for both read and write accesses. Multiple threads can read and write the same file simultaneously using the locking mechanism of the application.

However, the inode would still be locked in exclusive mode in case the contents of the inode need to be changed. Usually this happens when extending or truncating a file, because the allocation map of the file in inode needs to be changed. So it is good practice to use a fixed-size file in case of CIO.

Figure 4-13 on page 168 gives an example of the inode locking in a JFS2 file system. Thread0 and thread1 can read data from a shared file simultaneously because a read lock is in shared mode. However, thread0 cannot write data to the shared file until thread1 finishes reading the shared file. When the read lock is released, thread0 is able to get a write lock. Thread1 is blocked on the following read or write attemps because thread0 is holding an exclusive write lock.

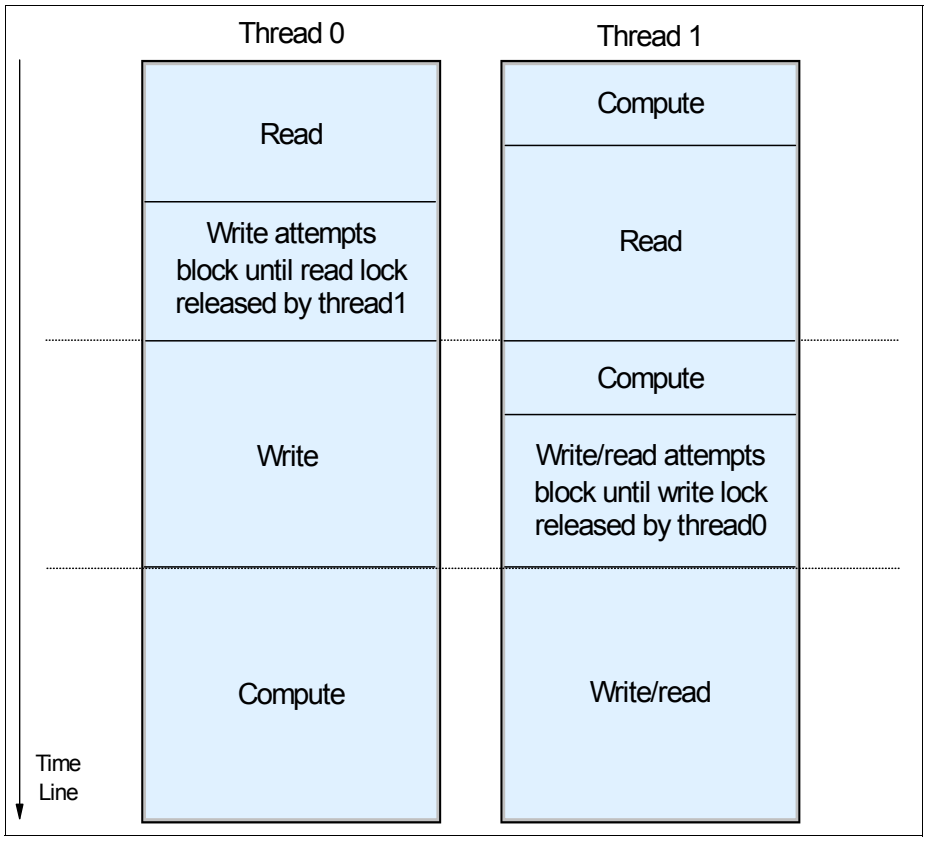


Figure 4-13 inode locking in a JFS2 file system

Figure 4-14 on page 169 gives an example of the inode locking in a CIO mounted JFS2 file system. Thread0 and thread1 can read and write the shared file simultaneously. However, when thread1 is extending or truncating the file, thread0 blocks read/write attempts. After the extending or truncating finishes, thread0 and thread1 can simultaneously access the shared file again.

.

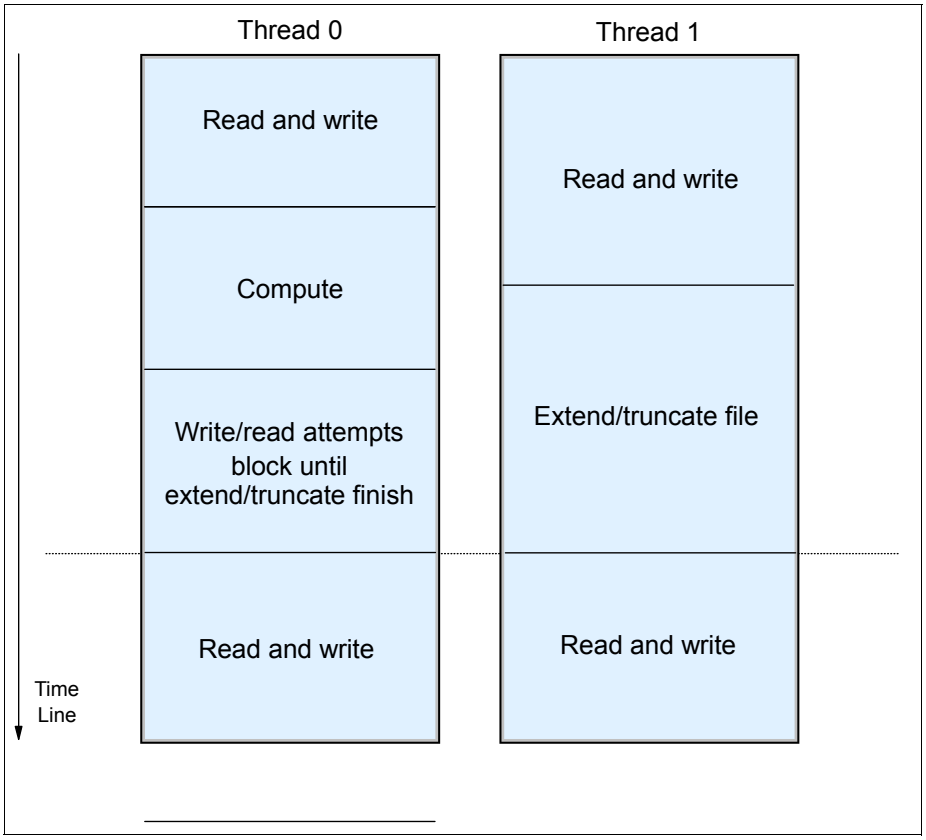


Figure 4-14 inode locking in CIO mounted JFS2 file system.

If the application does not have any kind of locking control for shared file access, it might result in data corruption. Thus CIO is usually only recommended for databases or applications that already have implemented fine level locking.

To enable concurrent I/O, mount the file system with the **cio** option as shown in Example 4-51.

Example 4-51 Mount with the cio option

```
#mount -o cio <file system name>
```

To make the option persistent across the boot, use the **chfs** command shown in Example 4-52.

Example 4-52 Use chfs to set the concurrent I/O option

```
#chfs -a options=cio /ciotest
```

The application can also open the file with O_CIO or O_CIOR to enable concurrent I/O. You can refer to the manual of the **open** subroutine on the AIX infocenter for more details.

Note: CIO inode locking still persists when extending or truncating files. So try to set a fixed size for files and reduce the chances of extending and truncating. Take an Oracle database as an example: set data files and redo log files to a fixed size and avoid using the auto extend feature.

Asynchronous I/O

If an application issues a synchronous I/O operation, it must wait until the I/O completes. Asynchronous I/O operations run in the background and will not block the application. This improves performance in certain cases, because you can overlap I/O processing and other computing tasks in the same thread.

AIO on raw logical volumes is handled by the kernel via the fast path, which will be queued into the LVM layer directly. Since AIX 5.3 TL5 and AIX 6.1, AIO on CIO mounted file systems can also submit I/O via the fast path, and AIX 6.1 enables this feature by default. In these cases, you do not need to tune the AIO subsystems. Example 4-53 shows how to enable the AIO fastpath for CIO mounted file systems on AIX 5.3, and also the related ioo options in AIX 6.1.

Example 4-53 AIO fastpath settings in AIX 5.3, AIX 6.1 and later releases

```
For AIX5.3, the fast path for CIO mounted file system is controlled by aioo option
fsfastpath. Note it is not a persistent setting, so we suggest adding it to the
inittab if you use it.
#aioo -o fsfastpath=1

For AIX6.1 and later release, the fast path for CIO mounted file system is on by
default.
#ioo -L aio_fastpath -L aio_fsfastpath -L posix_aio_fastpath -L
posix_aio_fsfastpath
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
------------------------------------------------------------------------------
aio_fastpath              1      1      1      0      1      boolean          D
------------------------------------------------------------------------------
aio_fsfastpath            1      1      1      0      1      boolean          D
------------------------------------------------------------------------------
posix_aio_fastpath        1      1      1      0      1      boolean          D
------------------------------------------------------------------------------
posix_aio_fsfastpath      1      1      1      0      1      boolean          D
------------------------------------------------------------------------------
```

For other kinds of AIO operations, the I/O requests are handled by AIO servers. You might need to tune the maximum number of AIO servers and the service queue size in such cases. In AIX 5.3, you can change the minservers, maxservers, and maxrequests with **smitty aio**. AIX 6.1 has more intelligent control over the AIO subsystem, and the aio tunables are provided with the **ioo** command. For legacy AIO, the tunables are aio_maxservers, aio_minservers, and aio_maxreqs. For POSIX AIO, the tunables are posix_aio_maxservers, posix_aio_minservers, and posix_aio_maxreqs.

For I/O requests that are handled by AIO servers, you can use **ps -kf|grep aio** to get the number of aioserver kernel processes. In AIX6.1, the number of aioservers can be dynamically adjusted according to the AIO workload. You can use this as an indicator for tuning the AIO subsystem. If the number of aioservers reaches the maximum, and there is still lots of free processor and unused I/O bandwidth, you can increase the maximum number of AIO servers.

Tip

Note: Note: AIO is compatible with all kinds of mount options, including DIO and CIO. Databases are likely to benefit from AIO.

Tipcan use the **iostat** command to retrieve AIO statistics. Table 4-14 shows the **iostat** options for AIO, and Example 4-54 gives an example of using **iostat** for AIO statistics. Note that at the time of writing this book, **iostat** statistics are not implemented for file system fastpath AIO requests used with the CIO option.

Table 4-14 iTipstat options for AIO statistics

Options	Explanation
-A	Display AIO statistics for AIX Legacy AIO.
-P	Display AIO statistics for POSIX AIO.
-Q	Displays a list of all the mounted file systems and the associated queue numbers with their request counts.
-q	Specifies AIO queues and their request counts.

Example 4-54 Tip statistics from iostat

```
ostat -PQ 1 100
System configuration: lcpu=8 maxserver=240
aio: avgc avfc maxgc maxfc maxreqs avg-cpu: % user % sys % idle % iowait
     845.0  0.0   897     0    131072             0.5   4.0   72.8     22.7

Queue#          Count          Filesystems
129             0              /
130             0              /usr
...
158             845            /iotest
```

The meanings of the metrics are shown in Table 4-15.

Table 4-15 iostat -A and iostat -P metrics

Column	Description
avgc	Average global AIO request count per second for the specified interval.
avfc	Average fastpath request count per second for the specified interval.
maxgc	Maximum global AIO request count since the last time this value was fetched.
maxfc	Maximum fastpath request count since the last time this value was fetched.
maxreqs	Specifies the maximum number of asynchronous I/O requests that can be outstanding at one time.

Note: If the AIO subsystem is not enabled on AIX 5.3, or has not been used on AIX 6.1, you get the error statement `Asynchronous I/O not configured on the system`.

Miscellaneous options

This section provides a few miscellaneous options.

noatime

According to the POSIX standard, every time you access a file, the operating system needs to update the “last access time” timestamp in the inode.

The noatime option is not necessary for most applications while it might deteriorate performance in case of heavy inode activities. To enable the noatime option, **mount** the file system with **noatime**:

```
mount -o noatime <file system name>
```

To make the option persistent, use the **chfs** command shown in Example 4-55.

Example 4-55 Use chfs to set the noatime option

```
#chfs -a options=noatime /ciotest

Use a comma to separate multiple options. To change the default mount options to
CIO and noatime:
#chfs -a options=cio,noatime /datafile

To change to default mount options to rbrw and noatime:
#chfs -a options=rbrw,noatime /archive
```

Creating an additional JFS/JFS2 log device

The JFS/JFS2 log works as follows:

- AIX uses a special logical volume called the log device as a circular journal for recording modifications to the file system metadata.
- File system metadata includes the superblock, inodes, indirect data pointers, and directories.
- When metadata is modified, a duplicate transaction is made to the JFS/JFS2 log.
- When a sync or fsync occurs, commit records are written to the JFS/JFS2 log to indicate that modified pages in memory have been committed to disk.

By default, all the file systems belong to the same VG and share the same log device. You can use the **lvmstat** or **filemon** commands to monitor the status of the log device as shown in Example 4-56. You need to enable the statistics for the logical volumes you observe, and disable the statistics after you finish observing. Note that the first line of **lvmstat** ouput is a cumulative value since the recording is enabled.

Example 4-56 Using lvmstat to monitor log device activities

```
#lvmstat -l loglv00 -e
#lvmstat -l loglv00 5
...
Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       1       1    2579         0     10316   2063.20
...
#lvmstat -l loglv00 -d
```

If the log device is busy, you can create a dedicated log device for critical file systems, as shown in Example 4-57 on page 173.

Example 4-57 Creating an additional JFS/JFS2 log device

```
Create new JFS or JFS2 log logical volume,
For JFS,
#mklv -t jfslog -y LVname VGname 1 PVname
For JFS2,
#mklv -t jfs2log -y LVname VGname 1 PVname

Unmount the filesystem and then format the log
#/usr/sbin/logform /dev/LVname

Modify /etc/filesystems and LVCB to use this log
#chfs -a log=/dev/LVname /filesystemname

mount filesystem
```

Using an INLINE log device

If the log device is the bottleneck, creating dedicated log devices is a viable solution. However, you might have large numbers of file systems that make the administration tedious. To circumvent this, AIX provides the INLINE log device for JFS2, and you can specify this option when creating the file system. Then each file system will have its own INLINE log device.

To create a file system with the INLINE log device:

```
#crfs -a logname=INLINE …
```

Or use **smitty crfs** and choose INLINE for the logical volume log. Note that JFS does not support INLINE log devices.

Note: We suggest using the INLINE log device with CIO mounted file systems.

Disabling JFS/JFS2 logging

JFS/JFS2 logging is critical for data integrity. However, there are some cases where you can disable it temporarily for performance. For example, if you are recovering the entire file system from backup, you can disable JFS/JFS2 logging for fast recovery. After the work is done, you can enable JFS/JFS2 logging again. Example 4-58 shows how to disable JFS/JFS2 logging.

Example 4-58 Disabling JFS/JFS2 logging

```
For JFS,
#mount -o nointegrity /jfs_fs

For JFS2(AIX6.1 and later releases),
#mount -o log=NULL /jfs2_fs
```

Another scenario for disabling a logging device is when using a RAM disk file system. Logging is not necessary because there is no persistent storage for RAM disk file systems. Example 4-59 shows how to create a RAM disk file system on AIX.

Example 4-59 Creating a RAM disk file system on AIX

```
# mkramdisk 1G
/dev/rramdisk0
# mkfs -V jfs2 /dev/ramdisk0
```

```
mkfs: destroy /dev/ramdisk0 (y)? y
File system created successfully.
1048340 kilobytes total disk space.
...
# mkdir /ramfs
# mount -V jfs2 -o log=NULL /dev/ramdisk0 /ramfs
# mount
  node       mounted        mounted over    vfs       date        options
-------- --------------  ---------------  ------ ------------ ---------------
...
         /dev/ramdisk0    /ramfs           jfs2   Oct 08 22:05 rw,log=NULL
```

Note: AIX 5.3 does not support disabling JFS2 logging because AIX 6.1 and later AIX releases do. Use JFS if you need to disable logging.

Disk I/O pacing

Disk-I/O pacing is intended to prevent programs with heavy I/O demands from saturing system I/O resources, and causing other programs with less I/O demand to hang for a long time. When a process tries to write to a file that already has high-water mark pending writes, the process is put to sleep until enough I/Os have completed to make the number of pending writes less than or equal to the low-water mark. This mechanism is somewhat similar to processor scheduling. Batch jobs that have consumed lots of resources tend to have lower priority, which ensures that the interactive jobs will run in time.

Disabling I/O pacing usually improves backup jobs and I/O throughput, while enabling I/O pacing ensures better response time for other kinds of jobs that have less I/O demand.

In AIX 5.3, I/O pacing is disabled by default. In AIX 6.1, the value is set to 8193 for the high-water mark and 4096 for the low-water mark, respectively. AIX 5.3 and later releases also support I/O pacing per file system via the mount command, for example:

```
mount -o minpout=4096 -o maxpout=8193 /filesystem
```

To make the option persistent across boot, use the **chfs** command shown in Example 4-60.

Example 4-60 Using chfs to set the I/O pacing option

```
#chfs -a options=minpout=4096,maxpout=8193 /iotest
```

File system defragmentation

You might create, extend, modify, or delete the LVs and files during daily maintenance. Also, the applications might do similar tasks. Due to the dynamic allocation nature of LVM and JFS/JFS2 file systems, logically contiguous LVs and files can be fragmented.

In such cases, file blocks might be scattered physically. If this happens, sequential access is no longer sequential and performance is likely to deteriorate. Random access tends to be affected too, because the seek distance could be longer and take more time. If the files are all in the memory and the cache hit ratio is high, the performance might be acceptable. However, if this is not the case, you are likely to experience performance problems.

Example 4-61 on page 175 shows how to determine the fragmentation using the **fileplace** command. This is an example with a severe fragmentation problem.

Example 4-61 Determine fragmentation using the fileplace command

```
#fileplace -pv m.txt
File: m.txt  Size: 33554432 bytes  Vol: /dev/hd3
Blk Size: 4096  Frag Size: 4096  Nfrags: 7920
Inode: 166  Mode: -rw-r--r--  Owner: root  Group: system

  Physical Addresses (mirror copy 1)                                    Logical Extent
  ----------------------------------                                    ----------------
  07351336-07351337  hdisk0            2 frags         8192 Bytes,   0.0%    00010760-00010761
  07351339           hdisk0            1 frags         4096 Bytes,   0.0%    00010763
  07351344           hdisk0            1 frags         4096 Bytes,   0.0%    00010768
…
 06989234           hdisk0           1 frags         4096 Bytes,   0.0%    00074642
  06989239           hdisk0            1 frags         4096 Bytes,   0.0%    00074647
  06989243           hdisk0            1 frags         4096 Bytes,   0.0%    00074651
  06989278           hdisk0            1 frags         4096 Bytes,   0.0%    00074686
  06989306           hdisk0            1 frags         4096 Bytes,   0.0%    00074714
  06989310           hdisk0            1 frags         4096 Bytes,   0.0%    00074718
        unallocated          272 frags      1114112 Bytes      0.0%

  7920 frags over space of 64051 frags:   space efficiency = 12.4%
  7919 extents out of 7920 possible:   sequentiality = 0.0%
```

A fast way to solve the problem is to back up the file, delete it, and then restore it as shown in Example 4-62.

Example 4-62 How to deal with file fragmentation

```
#cp m.txt m.txt.bak
#fileplace -pv m.txt.bak
File: m.txt.bak  Size: 33554432 bytes  Vol: /dev/hd3
Blk Size: 4096  Frag Size: 4096  Nfrags: 8192
Inode: 34  Mode: -rw-r--r--  Owner: root  Group: system

  Physical Addresses (mirror copy 1)                                    Logical Extent
  ----------------------------------                                    ----------------
  07218432-07226591  hdisk0         8160 frags     33423360 Bytes,  99.6%    00041696-00049855
  07228224-07228255  hdisk0           32 frags       131072 Bytes,   0.4%    00051488-00051519

  8192 frags over space of 9824 frags:   space efficiency = 83.4%
  2 extents out of 8192 possible:   sequentiality = 100.0%
#cp m.txt.bak m.txt
```

Example 4-63 shows an example of how to defragment the file system.

Example 4-63 Defragmenting the file system

```
#defragfs -r /tmp
Total allocation groups                                          : 64
Allocation groups skipped - entirely free                        : 52
Allocation groups skipped - too few free blocks                  : 3
Allocation groups that are candidates for defragmenting          : 9
Average number of free runs in candidate allocation groups       : 3

#defragfs /tmp
Defragmenting device /dev/hd3.  Please wait.

Total allocation groups                                          : 64
Allocation groups skipped - entirely free                        : 52
Allocation groups skipped - too few free blocks                  : 5
Allocation groups defragmented                                   : 7

defragfs completed successfully.
```

```
#defragfs -r /tmp
Total allocation groups                                          : 64
Allocation groups skipped - entirely free                        : 52
Allocation groups skipped - too few free blocks                  : 5
Allocation groups that are candidates for defragmenting          : 7
Average number of free runs in candidate allocation groups       : 4
```

4.4.4 The filemon utility

We now introduce the **filemon** utility.

Basic filemon utility

filemon is a tool based on the system trace facilities. You usually use **filemon** to find out the hotspot in the LVM and file system data layout. **filemon** can report the following major kinds of activities:

- Logical file system (lf)
- Virtual memory system (vm)
- Logical volumes (lv)
- Physical volumes (pv)
- All (short for lf, vm, lv, pv)

filemon runs in the background. Explicitly stop **filemon** at the end of data collection by executing **trcstop**. Example 4-64 shows the basic syntax of **filemon**. In the example, we start data collection for three seconds and then used **trcstop**. Also, we used the **-T** option to specify a larger trace buffer size (10 MB) than the default (64 KB per processor). The **filemon** report file is `fmon.out`.

Example 4-64 Basic filemon syntax

```
# filemon -T 10000000 -u -O lf,lv,pv,detailed -o fmon.out
# sleep 3
# trcstop
```

Note: Check for trace buffer wraparounds that may invalidate the **filemon** report. If you see **"**`xxx events were lost`**"**, run **filemon** with a smaller time interval or with a larger **-T** buffer value.

A larger trace buffer size results in pinned physical memory; refer to "Trace tools and PerfPMR" on page 316.

The **filemon** report contains two major parts, as follows. The report is generated using the command in Example 4-64.

Most active files, LVs, and PVs report

As shown in Example 4-65, this can be used to identify hotspots in the data layout.

Example 4-65 Most active LVs and PVs in filemon output

```
...
Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume                   description
------------------------------------------------------------------------
```

```
  1.00 181360 181392 90076.0  /dev/fslv02              /ciotest512b
  0.85  28768  31640 15000.1  /dev/fslv01              /diotest4k
  0.00      0    256   63.6  /dev/fslv00             /iotest512b
...

Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume                  description
------------------------------------------------------------------------
  1.00 181360 181640 90137.6  /dev/hdisk1              MPIO FC 2145
  0.80  28768  31640 15000.1  /dev/hdisk2              MPIO FC 2145
...
```

Detailed statistics data

After you pinpoint the hotspot files or LVs or PVs from the most active reports, you can get the detailed statistics of these files or LVs or PVs in the “detailed stats” section as shown in Example 4-66.

The number of reads, writes, and seeks in the monitoring interval is displayed. You can also see the average I/O size at LV and PV layers in 512-byte blocks, and the min/avg/max response time in milliseconds.

Example 4-66 Detailed statistics section in the filemon output

```
...
------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/fslv02  description: /ciotest512
reads:                  22670   (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg   0.145 min   0.083 max   7.896 sdev   0.145
  read sequences:       22670
  read seq. lengths:    avg     8.0 min       8 max       8 sdev     0.0
writes:                 22674   (0 errs)
  write sizes (blks):   avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):   avg   0.253 min   0.158 max  59.161 sdev   0.717
  write sequences:      22674
  write seq. lengths:   avg     8.0 min       8 max       8 sdev     0.0
seeks:                  45343   (100.0%) <=indicates random I/O
  seek dist (blks):     init 431352,
                        avg 697588.3 min      16 max 2083536 sdev 493801.4
time to next req(msec): avg   0.044 min   0.014 max  16.567 sdev   0.085
throughput:             90076.0 KB/sec
utilization:            1.00
...
```

Hot file detection enhancement

An enhancement to the **filemon** command was introduced in AIX 7.1, AIX 6.1 TL4, and AIX 5.3 TL11. A more detailed hot files, LVs, and PVs report is provided when using **-O hot** with the **filemon** command.

When **-O hot** is specified, the hotness of files, LVs, and PVs is sorted from diverse perspectives, including capacity accessed (CAP_ACC), number of I/O operations per unit of

data accessed (IOP/#), total number of read operations (#ROP), total number of write operations (#WOP), time taken per read operation (RTIME), and time taken per write operation (WTIME). The aim of the report is to guide the administrator in determining which files, LVs, and PVs are the ideal candidates for migration to SSDs.

filemon -O hot is only supported in offline mode. Example 4-67 shows the syntax of using **filemon** for the hot file report. The “fmon.out” hotness report is similar to basic **filemon** output, but has more content.

Example 4-67 Generating a hot file report in offline mode

```
#filemon -o fmon.out -O hot -r myfmon -A -x "sleep 2"
The filemon command store the trace data in “myfmon.trc” and store the symbol
information in “myfmon.syms”, as specified in the -r option. You can re-generate
the hot file report from the trace data file and symbol file whenever you want, as
follows:
#filemon -o fmon1.out -r myfmon -O hot
```

For more details about hot file detection, refer to *AIX 7.1 Difference Guide,* SG24-7910.

4.4.5 Scenario with SAP and DB2

Taking into practice the I/O device and file system tuning options discussed in this chapter, this section focuses on configuring storage for a DB2 database with SAP. This involves using a standard set of file systems and configuring them to deliver optimal performance.

The physical storage we were using was virtualized by an IBM SAN Volume Controller (SVC), making this scenario focused on a situation where the external storage is already striped, and how to configure AIX LVM appropriately.

To provide some background on the storage in this case, an SVC is a storage virtualization appliance where block storage can be presented to an SVC, and the SVC optimizes the external storage and manages the allocation to hosts.

The SVC has a concept of a managed disk group, which is LUNs from an external storage system from the same class of disks grouped together, forming a managed disk group.The SVC stripes the data across all of the mdisks in the managed disk group.

In our scenario, we have a managed disk group for our DB2 database and SAP binaries, and a managed disk group for SAP logs.

Figure 4-15 on page 179 provides a diagram of the environment used in this scenario.

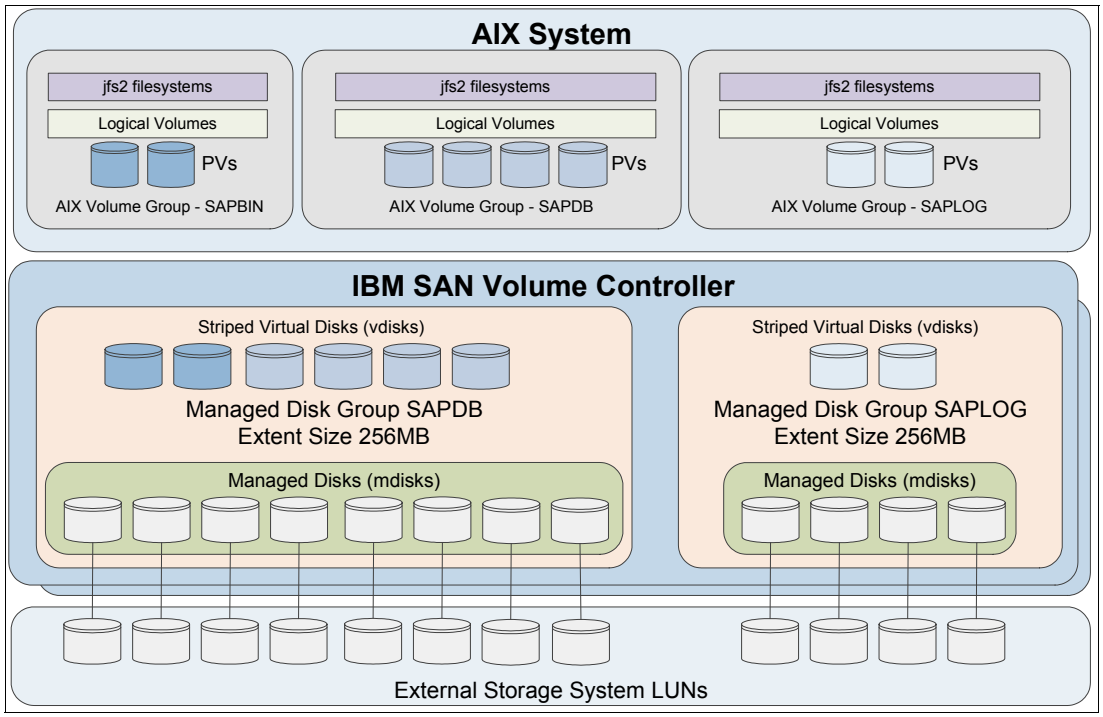


Figure 4-15 Storage overview

Table 4-16 provides a summary of the JFS2 file systems that are required for our SAP instance, their associated logical volumes, volume group, and mount options.

Table 4-16 File system summary for instance SID

Logical volume	Volume group	JFS2 file system	Mount options
usrsap_lv	sapbin_vg	/usr/sap	
sapmnt_lv	sapbin_vg	/sapmnt	
db2_lv	sapbin_vg	/db2	noatime
db2dump_lv	sapbin_vg	/db2/SID/db2dump	
logarch_lv	saplog_vg	/db2/SID/log_archive	rbrw
logret_lv	saplog_vg	/db2/SID/log_retrieve	
logdir_lv	saplog_vg	/db2/SID/log_dir	cio,noatime
db2sid_lv	sapdb_vg	/db2/SID/db2sid	
saptemp_lv	sapdb_vg	/db2/SID/saptemp1	cio,noatime
sapdata1_lv	sapdb_vg	/db2/SID/sapdata1	cio,noatime
sapdata2_lv	sapdb_vg	/db2/SID/sapdata2	cio,noatime
sapdata3_lv	sapdb_vg	/db2/SID/sapdata3	cio,noatime
sapdata4_lv	sapdb_vg	/db2/SID/sapdata4	cio,noatime

As discussed in 4.3.5, “Adapter tuning” on page 150, the first step performed in this example is to apply the required settings to our fiber channel devices to deliver the maximum throughput on our AIX system based on our workload (Example 4-68 on page 180).

Example 4-68 Set FC adapter attributes

```
root@aix1:/ # chdev -l fcs0 -a num_cmd_elems=2048 -a max_xfer_size=0x200000 -P
fcs0 changed
root@aix1:/ # chdev -l fcs1 -a num_cmd_elems=2048 -a max_xfer_size=0x200000 -P
fcs1 changed
root@aix1:/ # chdev -l fscsi0 -a fc_err_recov=fast_fail -a dyntrk=yes -P
fscsi0 changed
root@aix1:/ # chdev -l fscsi1 -a fc_err_recov=fast_fail -a dyntrk=yes -P
fscsi1 changed
root@aix1:/ # shutdown -Fr

..... AIX system will reboot .....
```

Since we were using storage front-ended by SVC, we needed to ensure that we had the SDDPCM driver installed. Example 4-69 shows that the latest driver at the time of writing is installed, and we have nine disks assigned to our system. We have hdisk0, which is the rootvg presented via virtual SCSI, and the remaining eight disks are presented directly from SVC to our LPAR using NPIV.

Example 4-69 Confirming that the required drivers are installed

```
root@aix1:/ # lslpp -l devices.sddpcm*
  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  devices.sddpcm.71.rte      2.6.3.2  COMMITTED  IBM SDD PCM for AIX V71

Path: /etc/objrepos
  devices.sddpcm.71.rte      2.6.3.2  COMMITTED  IBM SDD PCM for AIX V71
root@aix1:/ # lsdev -Cc disk
hdisk0 Available          Virtual SCSI Disk Drive
hdisk1 Available 02-T1-01 MPIO FC 2145
hdisk2 Available 02-T1-01 MPIO FC 2145
hdisk3 Available 02-T1-01 MPIO FC 2145
hdisk4 Available 02-T1-01 MPIO FC 2145
hdisk5 Available 02-T1-01 MPIO FC 2145
hdisk6 Available 02-T1-01 MPIO FC 2145
hdisk7 Available 02-T1-01 MPIO FC 2145
hdisk8 Available 02-T1-01 MPIO FC 2145
root@aix1:/ #
```

4.3.2, “Disk device tuning” on page 143 explains what attributes should be considered for an hdisk device. Based on what we knew about our environment from testing in other parts of the book, we understood that our storage had the capability to easily handle a queue_depth of 64 and a max_transfer size of 1 MB, which is 0x100000.

The device driver we were using was SDDPCM for IBM storage, the recommended algorithm was load_balance, so we set this attribute on our hdisks. This is also the default.

Example 4-70 demonstrates how to set the attributes on our hdisk devices, which were new LUNs from our SVC and were not assigned to a volume group.

Example 4-70 Setting hdisk attributes on devices used for SAP file systems

```
root@aix1:/ # for DISK in `lspv |egrep "None|none" |awk '{print $1}'`
> do
> chdev -l $DISK -a queue_depth=64 -a max_transfer=0x100000 -a algorithm=load_balance
> done
```

```
hdisk1 changed
hdisk2 changed
hdisk3 changed
hdisk4 changed
hdisk5 changed
hdisk6 changed
hdisk7 changed
hdisk8 changed
root@aix1:/ #
```

Example 4-71 demonstrates how to create our volume groups. In this case, we had three volume groups, one for SAP binaries, one for the database and one for the logs. We were using a PP size of 128 MB and creating a scalable type volume group.

Example 4-71 Volume group creation

```
root@aix1:/ # mkvg -S -y sapbin_vg -s 128 hdisk1 hdisk2
0516-1254 mkvg: Changing the PVID in the ODM.
0516-1254 mkvg: Changing the PVID in the ODM.
sapbin_vg
root@aix1:/ # mkvg -S -y sapdb_vg -s 128 hdisk3 hdisk4 hdisk5 hdisk6
0516-1254 mkvg: Changing the PVID in the ODM.
0516-1254 mkvg: Changing the PVID in the ODM.
0516-1254 mkvg: Changing the PVID in the ODM.
0516-1254 mkvg: Changing the PVID in the ODM.
sapdb_vg
root@aix1:/ # mkvg -S -y saplog_vg -s 128 hdisk7 hdisk8
0516-1254 mkvg: Changing the PVID in the ODM.
0516-1254 mkvg: Changing the PVID in the ODM.
saplog_vg
root@aix1:/ #
```

4.3.3, “Pbuf on AIX disk devices” on page 148 explains that each hdisk device in a volume group has a number of pbuf buffers associated with it. For the database and log volume groups that have most disk I/O activity, we increased the number of buffers from the default of 512 to 1024 buffers. A small amount of additional memory was required, while the status of the volume group’s blocked I/O count should be monitored with **`lvmo -av`**. This is shown in Example 4-72.

Example 4-72 Increasing the pv buffers on the busiest volume groups

```
root@aix1:/ # lvmo -v sapdb_vg -o pv_pbuf_count=1024
root@aix1:/ # lvmo -v saplog_vg -o pv_pbuf_count=1024
root@aix1:/ # lvmo -av sapdb_vg
vgname = sapdb_vg
pv_pbuf_count = 1024
total_vg_pbufs = 4096
max_vg_pbufs = 524288
pervg_blocked_io_count = 0
pv_min_pbuf = 512
max_vg_pbuf_count = 0
global_blocked_io_count = 1
root@aix1:/ # lvmo -av saplog_vg
vgname = saplog_vg
pv_pbuf_count = 512
total_vg_pbufs = 1024
max_vg_pbufs = 524288
pervg_blocked_io_count = 1
```

```
pv_min_pbuf = 512
max_vg_pbuf_count = 0
global_blocked_io_count = 1
root@aix1:/ #
```

When creating our logical volumes, we were using the 4.4.2, “LVM best practice” on page 159, and using the maximum range of physical volumes (**-e x**). This method of spreading the logical volumes over the four disks in the volume group has the following effect:

- 128 MB (the PP size) will be written to the first disk.
- 128 MB (the PP size) will be written to the second disk.
- 128 MB (the PP size) will be written to the third disk.
- 128 MB (the PP size) will be written to the fourth disk.
- Repeat.

To ensure that we did not have a situation where each of the disks in the volume group is busy one at a time, the order of disks specified on creation of the logical volume dictates the order of writes.

If you rotate the order of disks when each logical volume is created, you can balance the writes across all of the disks in the volume group.

Figure 4-16 demonstrates this concept for the four `sapdata` file systems, which are typically the most I/O intensive in an SAP system. Ensure that their write order is rotated.

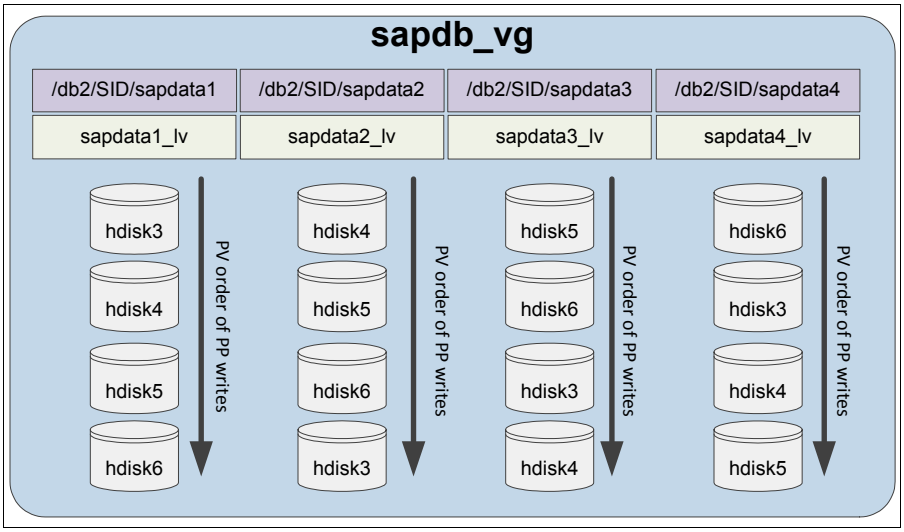


Figure 4-16 Rotating PV order per LV for sapdata file systems

Example 4-73 on page 183 shows our logical volume creation. The following options were set as part of the logical volume creation:

- The logical volume will be used for a file system type of JFS2 (**-t jfs2**).
- The logical volume has the range of physical volumes = maximum (**-e x**).
- The initial size of the file system is equal to the number of PVs in the VG.
- The order of hdisks that the logical volume is created on is rotated.

Example 4-73 Logical volume creation

```
root@aix1:/ # mklv -y usrsap_lv -t jfs2 -e x sapbin_vg 2 hdisk1 hdisk2
usrsap_lv
root@aix1:/ # mklv -y sapmnt_lv -t jfs2 -e x sapbin_vg 2 hdisk2 hdisk1
sapmnt_lv
root@aix1:/ # mklv -y db2_lv -t jfs2 -e x sapbin_vg 2 hdisk1 hdisk2
db2_lv
root@aix1:/ # mklv -y db2dump_lv -t jfs2 -e x sapbin_vg 2 hdisk2 hdisk1
db2dump_lv
root@aix1:/ # mklv -y logdir_lv -t jfs2 -e x saplog_vg 2 hdisk7 hdisk8
logdir_lv
root@aix1:/ # mklv -y logarch_lv -t jfs2 -e x saplog_vg 2 hdisk8 hdisk7
logarch_lv
root@aix1:/ # mklv -y logret_lv -t jfs2 -e x saplog_vg 2 hdisk7 hdisk8
logret_lv
root@aix1:/ # mklv -y sapdata1_lv -t jfs2 -e x sapdb_vg 4 hdisk3 hdisk4 hdisk5 hdisk6
sapdata1_lv
root@aix1:/ # mklv -y sapdata2_lv -t jfs2 -e x sapdb_vg 4 hdisk4 hdisk5 hdisk6 hdisk3
sapdata2_lv
root@aix1:/ # mklv -y sapdata3_lv -t jfs2 -e x sapdb_vg 4 hdisk5 hdisk6 hdisk3 hdisk4
sapdata3_lv
root@aix1:/ # mklv -y sapdata4_lv -t jfs2 -e x sapdb_vg 4 hdisk6 hdisk3 hdisk4 hdisk5
sapdata4_lv
root@aix1:/ # mklv -y db2sid_lv -t jfs2 -e x sapdb_vg 4 hdisk3 hdisk4 hdisk5 hdisk6
db2sid_lv
root@aix1:/ # mklv -y saptemp_lv -t jfs2 -e x sapdb_vg 4 hdisk4 hdisk5 hdisk6 hdisk3
saptemp_lv
root@aix1:/ #
```

4.4.3, “File system best practice” on page 163 explains the options available for JFS2 file systems. Example 4-74 shows our file system creation with the following options:

- The file systems are JFS2 (**-v jfs2**).
- The JFS2 log is inline rather than using a JFS2 log logical volume (**-a logname=INLINE**).
- The file systems will mount automatically on system reboot (**-A yes).**
- The file systems are enabled for JFS2 snapshots (**-isnapshot=yes**).

Example 4-74 File system creation

```
root@aix1:/ # crfs -v jfs2 -d usrsap_lv -m /usr/sap -a logname=INLINE -A yes -a
-isnapshot=yes
File system created successfully.
259884 kilobytes total disk space.
New File System size is 524288
root@aix1:/ # crfs -v jfs2 -d sapmnt_lv -m /sapmnt -a logname=INLINE -A yes -a
-isnapshot=yes
File system created successfully.
259884 kilobytes total disk space.
New File System size is 524288
root@aix1:/ # crfs -v jfs2 -d db2_lv -m /db2 -a logname=INLINE -A yes -a -isnapshot=yes
File system created successfully.
259884 kilobytes total disk space.
New File System size is 524288
root@aix1:/ # crfs -v jfs2 -d db2dump_lv -m /db2/SID/db2dump -a logname=INLINE -A yes -a
-isnapshot=yes
File system created successfully.
259884 kilobytes total disk space.
New File System size is 524288
```

```
root@aix1:/ # crfs -v jfs2 -d logarch_lv -m /db2/SID/log_archive -a logname=INLINE -A yes
-a -isnapshot=yes
File system created successfully.
259884 kilobytes total disk space.
New File System size is 524288
root@aix1:/ # crfs -v jfs2 -d logret_lv -m /db2/SID/log_retrieve -a logname=INLINE -A yes
-a -isnapshot=yes
File system created successfully.
259884 kilobytes total disk space.
New File System size is 524288
root@aix1:/ # crfs -v jfs2 -d logdir_lv -m /db2/SID/log_dir -a logname=INLINE -A yes -a
-isnapshot=yes -a options=cio,rw
File system created successfully.
259884 kilobytes total disk space.
New File System size is 524288
root@aix1:/ # crfs -v jfs2 -d db2sid_lv -m /db2/SID/db2sid -a logname=INLINE -A yes -a
-isnapshot=yes
File system created successfully.
519972 kilobytes total disk space.
New File System size is 1048576
root@aix1:/ # crfs -v jfs2 -d saptemp_lv -m /db2/SID/saptemp1 -a logname=INLINE -A yes -a
-isnapshot=yes -a options=cio,noatime,rw
File system created successfully.
519972 kilobytes total disk space.
New File System size is 1048576
root@aix1:/ # crfs -v jfs2 -d sapdata1_lv -m /db2/SID/sapdata1 -a logname=INLINE -A yes -a
-isnapshot=yes -a options=cio,noatime,rw
File system created successfully.
519972 kilobytes total disk space.
New File System size is 1048576
root@aix1:/ # crfs -v jfs2 -d sapdata2_lv -m /db2/SID/sapdata2 -a logname=INLINE -A yes -a
-isnapshot=yes -a options=cio,noatime,rw
File system created successfully.
519972 kilobytes total disk space.
New File System size is 1048576
root@aix1:/ # crfs -v jfs2 -d sapdata3_lv -m /db2/SID/sapdata3 -a logname=INLINE -A yes -a
-isnapshot=yes -a options=cio,noatime,rw
File system created successfully.
519972 kilobytes total disk space.
New File System size is 1048576
root@aix1:/ # crfs -v jfs2 -d sapdata4_lv -m /db2/SID/sapdata4 -a logname=INLINE -A yes -a
-isnapshot=yes -a options=cio,noatime,rw
File system created successfully.
519972 kilobytes total disk space.
New File System size is 1048576
root@aix1:/ #
```

The next step was to set the size of our file systems and mount them. Due to the order of mounting, we needed to create some directories for file systems mounted on top of `/db2`. It is also important to note that the sizes used here were purely for demonstration purposes only, and the inline log expands automatically as the file systems are extended. This is shown in Example 4-75.

Example 4-75 File system sizing and mounting

```
root@aix1:/ # chfs -a size=16G /usr/sap ; mount /usr/sap
Filesystem size changed to 33554432
Inlinelog size changed to 64 MB.
root@aix1:/ # chfs -a size=8G /sapmnt ; mount /sapmnt
Filesystem size changed to 16777216
```

```
Inlinelog size changed to 32 MB.
root@aix1:/ # chfs -a size=16G /db2 ; mount /db2
Filesystem size changed to 33554432
Inlinelog size changed to 64 MB.
root@aix1:/ # mkdir /db2/SID
root@aix1:/ # mkdir /db2/SID/db2dump
root@aix1:/ # mkdir /db2/SID/log_archive
root@aix1:/ # mkdir /db2/SID/log_retrieve
root@aix1:/ # mkdir /db2/SID/log_dir
root@aix1:/ # mkdir /db2/SID/db2sid
root@aix1:/ # mkdir /db2/SID/saptemp1
root@aix1:/ # mkdir /db2/SID/sapdata1
root@aix1:/ # mkdir /db2/SID/sapdata2
root@aix1:/ # mkdir /db2/SID/sapdata3
root@aix1:/ # mkdir /db2/SID/sapdata4
root@aix1:/ # chfs -a size=4G /db2/SID/db2dump ; mount /db2/SID/db2dump
Filesystem size changed to 8388608
Inlinelog size changed to 16 MB.
root@aix1:/ # chfs -a size=32G /db2/SID/log_archive ; mount /db2/SID/log_archive
Filesystem size changed to 67108864
Inlinelog size changed to 128 MB.
root@aix1:/ # chfs -a size=32G /db2/SID/log_retrieve ; mount /db2/SID/log_retrieve
Filesystem size changed to 67108864
Inlinelog size changed to 128 MB.
root@aix1:/ # chfs -a size=48G /db2/SID/log_dir ; mount /db2/SID/log_dir
Filesystem size changed to 100663296
Inlinelog size changed to 192 MB.
root@aix1:/ # chfs -a size=16G /db2/SID/db2sid ; mount /db2/SID/db2sid
Filesystem size changed to 33554432
Inlinelog size changed to 64 MB.
root@aix1:/ # chfs -a size=8G /db2/SID/saptemp1 ; mount /db2/SID/saptemp1
Filesystem size changed to 16777216
Inlinelog size changed to 32 MB.
root@aix1:/ # chfs -a size=60G /db2/SID/sapdata1 ; mount /db2/SID/sapdata1
Filesystem size changed to 125829120
Inlinelog size changed to 240 MB.
root@aix1:/ # chfs -a size=60G /db2/SID/sapdata2 ; mount /db2/SID/sapdata2
Filesystem size changed to 125829120
Inlinelog size changed to 240 MB.
root@aix1:/ # chfs -a size=60G /db2/SID/sapdata3 ; mount /db2/SID/sapdata3
Filesystem size changed to 125829120
Inlinelog size changed to 240 MB.
root@aix1:/ # chfs -a size=60G /db2/SID/sapdata4 ; mount /db2/SID/sapdata4
Filesystem size changed to 125829120
Inlinelog size changed to 240 MB.
root@aix1:/ #
```

To ensure that the file systems are mounted with the correct mount options, run the **mount** command. This is shown in Example 4-76.

Example 4-76 Verify that file systems are mounted correctly

```
root@aix1:/ # mount
  node       mounted        mounted over    vfs       date        options
-------- ---------------  ---------------  ------ ------------ ---------------
         /dev/hd4         /                jfs2   Oct 08 12:43 rw,log=/dev/hd8
         /dev/hd2         /usr             jfs2   Oct 08 12:43 rw,log=/dev/hd8
         /dev/hd9var      /var             jfs2   Oct 08 12:43 rw,log=/dev/hd8
         /dev/hd3         /tmp             jfs2   Oct 08 12:43 rw,log=/dev/hd8
         /dev/hd1         /home            jfs2   Oct 08 12:43 rw,log=/dev/hd8
```

```
        /dev/hd11admin   /admin           jfs2   Oct 08 12:43 rw,log=/dev/hd8
        /proc            /proc            procfs Oct 08 12:43 rw
        /dev/hd10opt     /opt             jfs2   Oct 08 12:43 rw,log=/dev/hd8
        /dev/livedump    /var/adm/ras/livedump jfs2   Oct 08 12:43 rw,log=/dev/hd8
        /dev/usrsap_lv   /usr/sap         jfs2   Oct 10 14:59 rw,log=INLINE
        /dev/sapmnt_lv   /sapmnt          jfs2   Oct 10 14:59 rw,log=INLINE
        /dev/db2_lv      /db2             jfs2   Oct 10 15:00 rw,log=INLINE
        /dev/db2dump_lv  /db2/SID/db2dump jfs2   Oct 10 15:00 rw,log=INLINE
        /dev/logarch_lv  /db2/SID/log_archive jfs2   Oct 10 15:01 rw,log=INLINE
        /dev/logret_lv   /db2/SID/log_retrieve jfs2   Oct 10 15:01 rw,log=INLINE
        /dev/logdir_lv   /db2/SID/log_dir jfs2   Oct 10 15:02 rw,cio,noatime,log=INLINE
        /dev/db2sid_lv   /db2/SID/db2sid  jfs2   Oct 10 15:03 rw,log=INLINE
        /dev/saptemp_lv  /db2/SID/saptemp1 jfs2   Oct 10 15:03 rw,cio,noatime,log=INLINE
        /dev/sapdata1_lv /db2/SID/sapdata1 jfs2   Oct 10 15:03 rw,cio,noatime,log=INLINE
        /dev/sapdata2_lv /db2/SID/sapdata2 jfs2   Oct 10 15:03 rw,cio,noatime,log=INLINE
        /dev/sapdata3_lv /db2/SID/sapdata3 jfs2   Oct 10 15:03 rw,cio,noatime,log=INLINE
        /dev/sapdata4_lv /db2/SID/sapdata4 jfs2   Oct 10 15:03 rw,cio,noatime,log=INLINE
root@aix1:/ #
```

Note: It is important to consult your storage administrator and SAP basis administrator during the configuration of storage for a new SAP system. This section simply demonstrates the concepts discussed in this chapter.

4.5 Network

When configuring an AIX system's networking devices, there are a number of performance options to consider in the AIX operating system to improve network performance.

This section focuses on these settings in the AIX operating system and the potential gains from tuning them. 3.7, "Optimal Shared Ethernet Adapter configuration" on page 82 provides details on PowerVM shared Ethernet tuning.

Important: Ensure that your LAN switch is configured appropriately to match how AIX is configured. Consult your network administrator to ensure that both AIX and the LAN switch configuration match.

4.5.1 Network tuning on 10 G-E

10-Gigabit Ethernet adapters provide a higher bandwidth and lower latency than 1-Gigabit Ethernet adapters. However, it is important to understand that additional processor resources are required for 10-Gigabit Ethernet, and there are some tuning steps that can be taken to get good throughput from the adapter.

For optimum performance ensure adapter placement according to Adapter Placement Guide and size partitions, and optionally VIOS, to fit the expected workload. From the 5803 Adapter Placement Guide:

- No more than one 10 Gigabit Ethernet adapter per I/O chip.
- No more than one 10 Gigabit Ethernet port per two processors in a system.
- If one 10 Gigabit Ethernet port is present per two processors in a system, no other 10 Gb or 1 Gb ports should be used.

Note: Refer to Adapter Placement Guides for further guidance, such as:

- IBM Power 780 Adapter Placement Guide:

  http://pic.dhe.ibm.com/infocenter/powersys/v3r1m5/topic/p7eab/p7eabprintthis77x78x.htm

- IBM Power 795 Adapter Placement:

  http://publib.boulder.ibm.com/infocenter/powersys/v3r1m5/index.jsp?topic=/areab/areabkickoff.htm

To ensure that the connected network switch is not overloaded by one or more 10 Gbit ports, verify that the switch ports have flow control enabled (which is the default for the adapter device driver).

If the 10 Gbit adapter is dedicated to a partition, enable Large Send offload (LS) and Large Receive Offload (LRO) for the adapter device driver. The LS will also have to be enabled on the network interface device level (enX) using the `mtu_bypass` attribute or by manually enabling every time after IPL (boot).

For streaming larger data packets over the physical network, consider enabling Jumbo Frames. However, it requires both endpoint and network switch support to work and will not have any throughput improvement for packets that can fit in a default MTU size of 1500 bytes.

The **entstat** command physical adapter (port) statistic No Resource Errors are the number of incoming packets dropped by the hardware due to lack of resources. This usually occurs because the receive buffers on the adapter were exhausted; to mitigate, increase the adapter size of the receive buffers, for example by adjusting “receive descriptor queue size” (rxdesc_que_sz) and “receive buffer pool size” (rxbuf_pool_sz), which, however, require deactivating and activating the adapter.

Consider doubling rxdesc_que_sz and set rxbuf_pool_sz to two (2) times the value of rxdesc_que_sz, with the **chdev** command, for example:

```
chdev -Pl ent# -a rxdesc_que_sz=4096 -a rxbuf_pool_sz=8192
```

Refer to:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/adapter_stats.htm

The **entstat** command physical 10 Gbit Ethernet adapter (port) statistic Lifetime Number of Transmit Packets/Bytes Overflowed occurs in the case that the adapter has a full transmit queue and the system is still sending data; the packets chain will be put to an overflow queue.

This overflow queue will be sent when the transmit queue has free entries again. This behaviour is reflected in the statistics above and these values do not indicate packet loss.

Frequently occurring overflows indicate that the adapter does not have enough resources allocated for transmit to handle the traffic load. In such a situation, it is suggested that the number of transmit elements be increased (transmit_q_elem), for example:

```
chdev -Pl ent# -a transmit_q_elem=2048
```

Etherchannel link aggregation spreads of outgoing packets are governed by the hash_mode attribute of the Etherchannel device, and how effective this algorithm is for the actual workload can be monitored by the **entstat** command or **netstat -v**.

In the following example, the 8023ad link aggregation Etherchannel consists of four adapter ports with the hash_mode load balancing option set to default, in which the adapter selection algorithm uses the last byte of the destination IP address (for TCP/IP traffic) or MAC address (for ARP and other non-IP traffic).

The **lsattr** command:

```
adapter_names   ent0,ent1,ent4,ent6 EtherChannel Adapters
hash_mode       default             Determines how outgoing adapter is chosen
mode            8023ad              EtherChannel mode of operation
```

Using the **entstat** command to display the statistics for ent0, ent1, ent4 and ent6, reveals that the current network workload is not spreading the outgoing traffic balanced over the adapters in the Etherchannel, as can be seen in Table 4-17. The majority of the outgoing traffic is over ent6, followed by ent4, but ent0 and ent1 have almost no outgoing traffic.

Changing the hash_mode from default to src_dst_port might improve the balance in this case, since the outgoing adapter is selected by an algorithm using the combined source and destination TCP or UDP port values.

Table 4-17 using entstat command to monitor Etherchannel hash_mode spread of outgoing traffic

Device	Transmit packets	% of total	Receive packets	% of total
ent0	811028335	3%	1239805118	12%
ent1	1127872165	4%	2184361773	21%
ent4	8604105240	28%	2203568387	21%
ent6	19992956659	65%	4671940746	45%
Total	30535962399	100%	10299676024	100%

Note: The receive traffic is dependent on load balancing and speading from the network and sending node, and the switch tables of MAC and IP addresses.

Refer to:

http://pic.dhe.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.commadmn/doc/commadmndita/etherchannel_loadbalance.htm

Table 4-18 provides details and some guidance relating to some of the attributes that can be tuned on the adapter to improve performance.

Table 4-18 10 gigabit adapter settings

Attribute	Description	Suggested Value
chksum_offload	This enables the adapter to compute the checksum on transmit and receive saving processor utilization in AIX because AIX does not have to compute the checksum. This is enabled by default in AIX.	Enabled
flow_ctrl	This specifies whether the adapter should enable transmit and receive flow control. This should be enabled in AIX and on the network switch. This is enabled in AIX by default.	Enabled

Attribute	Description	Suggested Value
jumbo_frames	This setting indicates that frames up to 9018 bytes can be transmitted with the adapter. In networks where jumbo frames are supported and enabled on the network switches, this should be enabled in AIX.	Enabled
large_receive	This enables AIX to coalesce receive packets into larger packets before passing them up the TCP stack.	Enabled
large_send	This option enables AIX to build a TCP message up to 64 KB long and send it in one call to the Ethernet device driver.	Enabled

Table 4-19 provides details and some guidance on the attributes that can be tuned on the interface to improve performance.

Table 4-19 Interface attributes

Attribute	Description	Suggested Value
mtu	The Media Transmission Unit (MTU) size is the maximum size of a frame that can be transmitted by the adapter.	9000 if using jumbo frames
mtu_bypass	This allows the interface to have *largesend* enabled.	On
rfc1323	This enables TCP window scaling. Enabling this may improve TCP streaming performance.	Set by “no” tunable to 1
tcp_recvspace	This parameter controls how much buffer space can be consumed by receive buffers, and to inform the sender how big its transmit window size can be.	16 k default, 64 k optional
tcp_sendspace	This attribute controls how much buffer space will be used to buffer the data that is transmitted by the adapter.	16 k default, 64 k optional
thread	Known as the dog threads feature, the driver will queue incoming packets to the thread.	On

4.5.2 Interrupt coalescing

Interrupt coalescing is introduced to avoid flooding the host with too many interrupts. Consider a typical situation for a 1-Gbps Ethernet: if the average package size is 1000 bytes, to achieve the full receiving bandwidth, there will be 1250 packets in each processor tick (10 ms). Thus, if there is no interrupt coalescing, there will be 1250 interrupts in each processor tick, wasting processor time with all the interrupts.

Interrupt coalescing is aimed at reducing the interrupt overhead with minimum latency. There are two typical types of interrupt coalescing in AIX network adapters.

Most 1-Gbps Ethernet adapters, except the HEA adapter, use the interrupt throttling rate method, which generates interrupts at fixed frequencies, allowing the bunching of packets based on time. Such adapters include FC5701, FC5717, FC5767, and so on. The default

interrupt rate is controlled by the **intr_rate** parameter, which is 10000 times per second. The **intr_rate** can be changed by the following command:

```
#chdev -l entX -a intr_rate=<value>
```

Before you change the value of intr_rate, you might want to check the range of possible values for it (Example 4-77).

Example 4-77 Value range of intr_rate

```
#lsattr -Rl entX -a intr_rate
0...65535 (+1)
```

For lower interrupt overhead and less processor consumption, you can set the interrupt rate to a lower value. For faster response time, you can set the interrupt rate to a larger value, or even disable it by setting the value to 0.

Most 10-Gb Ethernet adapters and HEA adapters use a more advanced interrupt coalescing feature. A timer starts when the first packet arrives, and then the interrupt is delayed for n microseconds or until m packets arrive.

Refer to Example 4-78 for the HEA adapter where the n value corresponds to `rx_clsc_usec`, which equals 95 microseconds by default. The m value corresponds to `rx_coalesce`, which equals 16 packets. You can change the n and m values, or disable the interrupt coalescing by setting `rx_clsc=none`.

Example 4-78 HEA attributes for interrupt coalescing

```
lsattr -El ent0
alt_addr      0x000000000000   Alternate Ethernet address                True
flow_ctrl     no               Request Transmit and Receive Flow Control True
jumbo_frames  no               Request Transmit and Receive Jumbo Frames True
large_receive yes              Enable receive TCP segment aggregation    True
large_send    yes              Enable hardware Transmit TCP segmentation True
media_speed   Auto_Negotiation Requested media speed                     True
multicore     yes              Enable Multi-Core Scaling                 True
rx_cksum      yes              Enable hardware Receive checksum          True
rx_cksum_errd yes              Discard RX packets with checksum errors   True
rx_clsc       1G               Enable Receive interrupt coalescing       True
rx_clsc_usec  95               Receive interrupt coalescing window       True
rx_coalesce   16               Receive packet coalescing                 True
rx_q1_num     8192             Number of Receive queue 1 WQEs            True
rx_q2_num     4096             Number of Receive queue 2 WQEs            True
rx_q3_num     2048             Number of Receive queue 3 WQEs            True
tx_cksum      yes              Enable hardware Transmit checksum         True
tx_isb        yes              Use Transmit Interface Specific Buffers   True
tx_q_num      512              Number of Transmit WQEs                   True
tx_que_sz     8192             Software transmit queue size              True
use_alt_addr  no               Enable alternate Ethernet address         True
```

Refer to Example 4-79 for the 10-Gb Ethernet adapter where the n value corresponds to `intr_coalesce`, which is 5 microseconds by default. The m value corresponds to `receive_chain`, which is 16 packets by default. Note the attribute name for earlier adapters might be different.

Example 4-79 10-Gb Ethernet adapter attributes for interrupt coalescing

```
# lsattr -El ent1
```

```
alt_addr        0x000000000000 Alternate ethernet address                  True
chksum_offload  yes            Enable transmit and receive checksum        True
delay_open      no             Enable delay of open until link state is known True
flow_ctrl       yes            Enable transmit and receive flow control    True
intr_coalesce   5              Receive interrupt delay in microseconds     True
jumbo_frames    no             Transmit/receive jumbo frames               True
large_receive   yes            Enable receive TCP segment aggregation      True
large_send      yes            Enable transmit TCP segmentation offload    True
rdma_enabled    no             Enable RDMA support                         True
receive_chain   16             Receive packet coalesce(chain) count        True
receive_q_elem  2048           Number of elements per receive queue        True
transmit_chain  8              Transmit packet coalesce(chain) count       True
transmit_q_elem 1024           Number of elements per transmit queue       True
tx_timeout      yes            N/A                                         True
use_alt_addr    no             Enable alternate ethernet address           True
```

You can see the effect of turning off interrupt coalescing in 4.5.5, “Network latency scenario” on page 196.

Note that interrupt coalescing only applies to network receiving interrupts. TCP/IP implementation in AIX eliminates the need for network transmit interrupts. The transmit status is only checked at the next transmit. You can get this from the network statistics (**`netstat -v`**), the interrupt for `transmit statistics` is always 0.

4.5.3 10-G adapter throughput scenario

Using some of the tunables discussed in 4.5.1, “Network tuning on 10 G-E” on page 186, we performed some throughput tests between two AIX systems with a dedicated 10-G Ethernet adapter assigned to each system, with a single network switch in between the two LPARs, each one in different POWER 750 frames.

A baseline test, and three subsequent tests with different values applied, were performed. These tests were aimed at maximizing the throughput between two AIX systems.

The baseline test was run with a throughput of 370 MBps.

The first set of changes was to modify the **`rfc1323`**, **`tcp_sendspace`** and **`tcp_recvspace`** options and to perform another test. Example 4-80 demonstrates how the tunables were changed on each of the AIX systems.

Example 4-80 Configuration changes for test 1

```
root@aix1:/ # no -p -o rfc1323=1
Setting rfc1323 to 1
Setting rfc1323 to 1 in nextboot file
Change to tunable rfc1323, will only be effective for future connections
root@aix1:/ # no -p -o tcp_sendspace=1048576
Setting tcp_sendspace to 1048576
Setting tcp_sendspace to 1048576 in nextboot file
Change to tunable tcp_sendspace, will only be effective for future connections
root@aix1:/ # no -p -o tcp_recvspace=1048576
Setting tcp_recvspace to 1048576
Setting tcp_recvspace to 1048576 in nextboot file
Change to tunable tcp_recvspace, will only be effective for future connections

root@aix2:/ # no -p -o rfc1323=1
```

```
Setting rfc1323 to 1
Setting rfc1323 to 1 in nextboot file
Change to tunable rfc1323, will only be effective for future connections
root@aix2:/ # no -p -o tcp_sendspace=1048576
Setting tcp_sendspace to 1048576
Setting tcp_sendspace to 1048576 in nextboot file
Change to tunable tcp_sendspace, will only be effective for future connections
root@aix2:/ # no -p -o tcp_recvspace=1048576
Setting tcp_recvspace to 1048576
Setting tcp_recvspace to 1048576 in nextboot file
Change to tunable tcp_recvspace, will only be effective for future connection
```

The result of the changes was a throughput of 450 MBps in Test 1.

The next test consisted of enabling jumbo frames in AIX, and ensuring that our switch was capable of jumbo frames, and had jumbo frame support enabled. Example 4-81 demonstrates how the changes were made. It is important to note that the interface had to be detached and attached for the change to be applied, so we ran the commands on the HMC from a console window to each LPAR.

Example 4-81 Configuration changes for Test 2

```
root@aix1:/ # chdev -l en0 -a state=detach
en0 changed
root@aix1:/ # chdev -l ent0 -a jumbo_frames=yes
ent0 changed
root@aix1:/ # chdev -l en0 -a state=up
en0 changed

root@aix2:/ # chdev -l en0 -a state=detach
en0 changed
root@aix2:/ # chdev -l ent0 -a jumbo_frames=yes
ent0 changed
root@aix2:/ # chdev -l en0 -a state=up
en0 changed
```

The result of the changes was a throughput of 965 MBps in Test 2.

The final test consisted of turning on the **mtu_bypass** and **thread** attributes. Example 4-82 shows how these attributes where set on each of the AIX systems.

Example 4-82 Configuration changes for test 3

```
root@aix1:/ # chdev -l en0 -a mtu_bypass=on
en0 changed
root@aix1:/ # chdev -l en0 -a thread=on
en0 changed

root@aix2:/ # chdev -l en0 -a mtu_bypass=on
en0 changed
root@aix2:/ # chdev -l en0 -a thread=on
en0 changed
```

The result of the changes in the final test throughput was 1020 MBps.

Table 4-20 provides a summary of the test results, and the processor consumption. The more packets and bandwidth were handled by the 10-G adapter, the more processing power was required.

Table 4-20 Throughput results summary

Test	Throughput	Processor usage
Baseline	370 MBps	1.8 POWER7 Processors
Test 1	450 MBps	2.1 POWER7 Processors
Test 2	965 MBps	1.6 POWER7 Processors
Test 3	1020 MBps	1.87 POWER7 Processors

4.5.4 Link aggregation

In the case where multiple adapters are allocated to an AIX LPAR, a link aggregation (also referred to as an EtherChannel device) should be configured to make best use of the two adapters. The link aggregation can provide redundancy if one adapter fails, and the combined throughput of the adapters can be made available as a single entity. There are also cases where due to a large number of packets per second, the latency increases. Having multiple adapters can counteract this problem.

When configuring any link aggregation configuration, it is important that the network infrastructure supports the configuration, and is configured appropriately. Unlike in the NIB mode, all the link aggregation ports should be on the same switch.

Table 4-21 provides a description of some of the attributes to consider, and some guidance on suggested values.

Table 4-21 Link aggregation attributes

Attribute	Description	Suggested value
mode	This attribute dictates the type of port channel that is configured. The mode 8023ad is available, which enables the adapter's EtherChannel to negotiate with a Link Aggregation Control Protocol (LCAP) enabled switch.	8023ad
hash_mode	If the EtherChannel is configured using standard or 8023ad mode, the hash_mode attribute determines how the outbound adapter for each packet is chosen. In src_dst_port both the source and destination TCP or UDP ports are used to determine the outgoing adapter.	src_dst_port
use_jumbo_frame	Setting this attribute to yes enables EtherChannel to use jumbo frames. This allows the Ethernet MTU to increase to 9000 bytes per frame instead of the default 1500 bytes.	yes

Example 4-83 on page 194 demonstrates how to configure a link aggregation of ports ent0 and ent1 with the attributes suggested in Table 4-21. This can also be performed using **`smitty addethch1`**.

Example 4-83 Configuring the EtherChannel device

```
root@aix1:/ # mkdev -c adapter -s pseudo -t ibm_ech -a adapter_names=ent0,ent1 -a
mode=8023ad -a hash_mode=src_dst_port -a use_jumbo_frame=yes
ent2 Available
```

When 8023.ad link aggregation is configured, you can use **entstat -d <etherchannel_adapter>** to check the negotiation status of the EtherChannel, as shown in Example 4-84.

The aggregation status of the EtherChannel adapter should be **Aggregated**. And all the related ports, including AIX side port (Actor) and switch port (Partner), should be in **IN_SYNC** status. All other values, such as **Negotiating** or **OUT_OF_SYNC**, means that link aggregation is not successfully established.

Example 4-84 Check the link aggregation status using entstat

```
#entstat -d ent2

------------------------------------------------------------
ETHERNET STATISTICS (ent2) :
Device Type: IEEE 802.3ad Link Aggregation
Hardware Address: 00:14:5e:99:52:c0
...
============================================================

Statistics for every adapter in the IEEE 802.3ad Link Aggregation:
------------------------------------------------------------------

Number of adapters: 2
Operating mode: Standard mode (IEEE 802.3ad)
IEEE 802.3ad Link Aggregation Statistics:
Aggregation status: Aggregated
        LACPDU Interval: Long   Received LACPDUs: 94
        Transmitted LACPDUs: 121
        Received marker PDUs: 0
        Transmitted marker PDUs: 0
        Received marker response PDUs: 0
        Transmitted marker response PDUs: 0
        Received unknown PDUs: 0
        Received illegal PDUs: 0
Hash mode: Source and destination TCP/UDP ports

------------------------------------------------------------

...


ETHERNET STATISTICS (ent0) :
...
IEEE 802.3ad Port Statistics:
-----------------------------
        Actor System Priority: 0x8000
        Actor System: 00-14-5E-99-52-C0
        Actor Operational Key: 0xBEEF
```

```
        Actor Port Priority: 0x0080
        Actor Port: 0x0001
        Actor State:
                LACP activity: Active
                LACP timeout: Long
                Aggregation: Aggregatable
                Synchronization: IN_SYNC
                Collecting: Enabled
                Distributing: Enabled
                Defaulted: False
                Expired: False

        Partner System Priority: 0x007F
        Partner System: 00-24-DC-8F-57-F0
        Partner Operational Key: 0x0002
        Partner Port Priority: 0x007F
        Partner Port: 0x0003
        Partner State:
                LACP activity: Active
                LACP timeout: Short
                Aggregation: Aggregatable
                Synchronization: IN_SYNC
                Collecting: Enabled
                Distributing: Enabled
                Defaulted: False
                Expired: False

        Received LACPDUs: 47
        Transmitted LACPDUs: 60
        Received marker PDUs: 0
        Transmitted marker PDUs: 0
        Received marker response PDUs: 0
        Transmitted marker response PDUs: 0
        Received unknown PDUs: 0
        Received illegal PDUs: 0

------------------------------------------------------------

...

ETHERNET STATISTICS (ent1) :
...

IEEE 802.3ad Port Statistics:
-----------------------------
        Actor System Priority: 0x8000
        Actor System: 00-14-5E-99-52-C0
        Actor Operational Key: 0xBEEF
        Actor Port Priority: 0x0080
        Actor Port: 0x0002
        Actor State:
                LACP activity: Active
                LACP timeout: Long
                Aggregation: Aggregatable
                Synchronization: IN_SYNC
```

```
        Collecting: Enabled
        Distributing: Enabled
        Defaulted: False
        Expired: False

Partner System Priority: 0x007F
Partner System: 00-24-DC-8F-57-F0
Partner Operational Key: 0x0002
Partner Port Priority: 0x007F
Partner Port: 0x0004
Partner State:
        LACP activity: Active
        LACP timeout: Short
        Aggregation: Aggregatable
        Synchronization: IN_SYNC
        Collecting: Enabled
        Distributing: Enabled
        Defaulted: False
        Expired: False

Received LACPDUs: 47
Transmitted LACPDUs: 61
Received marker PDUs: 0
Transmitted marker PDUs: 0
Received marker response PDUs: 0
Transmitted marker response PDUs: 0
Received unknown PDUs: 0
Received illegal PDUs: 0
```

4.5.5 Network latency scenario

In cases where applications are sensitive to latency in the network, there can be significant performance issues if the network latency is high. 3.7.9, “Measuring latency” on page 90 provides some details on how network latency can be measured.

Figure 4-17 on page 197 provides an overview of a test environment we prepared to measure the latency between different devices, and virtualization layers.

We had two POWER 750 systems, each with a VIO server sharing Ethernet, and two LPARs on each system. Each LPAR has a dedicated Ethernet adapter to test the hardware isolated to an AIX LPAR.

These tests were performed with no load on the POWER 750 systems to establish a baseline of the expected latency per device.

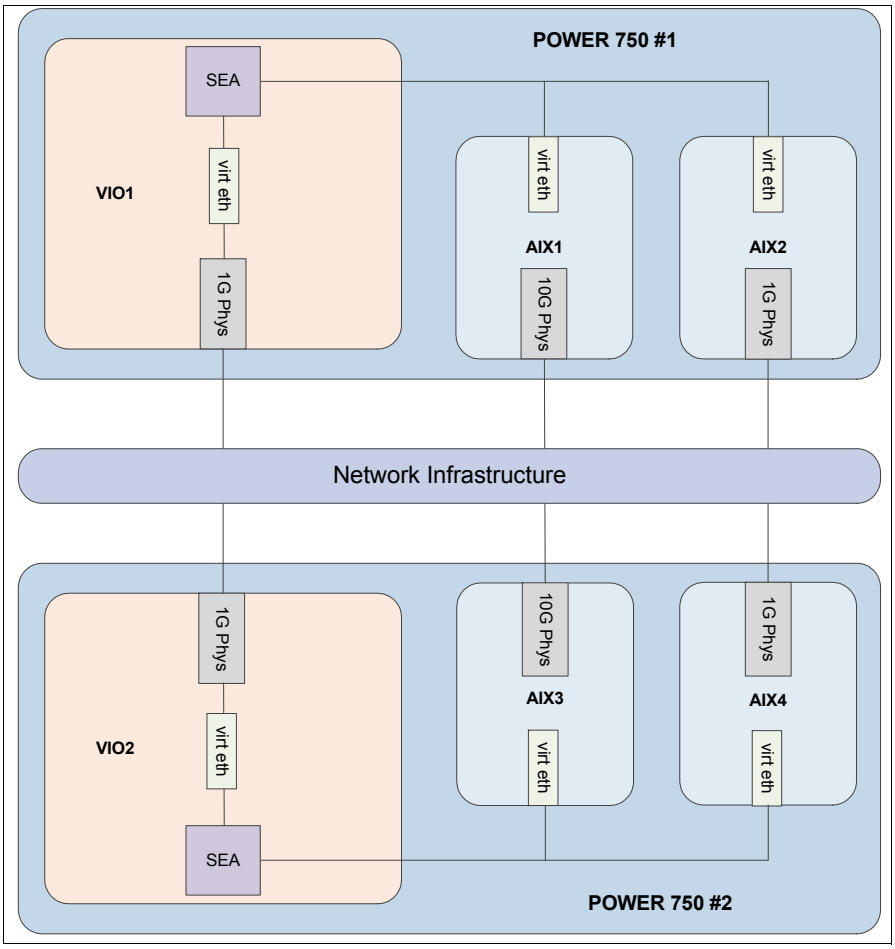


Figure 4-17 Sample scenario our network latency test results were based on

Table 4-22 provides a summary of our test results. The objective of the test was to compare the latency between the following components:

- Latency between two 10 G adapters
- Latency between two 1G adapters
- Latency between two virtual adapters on the same machine
- Latency between two LPARs on different machines communicating via shared Ethernet adapters

Table 4-22 Network latency test results

Source	Destination	Latency in milliseconds
AIX1 via 10 G Physical Ethernet	AIX3 via 10 G Physical Ethernet	0.062 ms
AIX1 via 10 G Physical Ethernet Interrupt Coalescing Disabled	AIX3 via 10 G Physical Ethernet Interrupt Coalescing Disabled	0.052 ms
AIX2 via 1 G Physical Ethernet	AIX4 via 1 G Physical Ethernet	0.144 ms
AIX2 via 1 G Physical Ethernet Interrupt Throttling Disabled	AIX4 via 1 G Physical Ethernet Interrupt Throttling Disabled	0.053 ms
AIX1 via Hyp Virtual Ethernet	AIX2 via Hyp Virtual Ethernet	0.038 ms

Source	Destination	Latency in milliseconds
AIX1 via Shared Ethernet	AIX3 via Shared Ethernet	0.274 ms

Conclusion

After the tests we found that the latency for 10 G Ethernet was significantly less than that of a 1 G adapter under the default setting, which was expected. What was also expected was that there is low latency across the hypervisor LAN, and some small latency added by using a shared Ethernet adapter rather than a dedicated adapter.

Also, some transaction type of workload might benefit from disabling interrupt coalescing, as the response time might be slightly improved. In our tests, you can see that in a 1-Gb Ethernet scenario, the latency is greatly improved after disabling interrupt coalescing. This was expected because the 1-Gb adapter waits 100 microseconds on average to generate an interrupt by default. However, change this value with caution, because it uses more processor time for faster response.

While this test was completed with no load on the network, AIX LPARs or VIO servers, it is important to recognize that as workload is added, latency may increase. The more packets that are being processed by the network, if there is a bottleneck, the more the latency will increase.

In the case that there are bottlenecks, there are some actions that might be considered:

- If the latency goes up, it is worthwhile measuring the latency between different components to try to identify a bottleneck.
- If there are a large number of LPARs accessing the same SEA, it may be worthwhile having multiple SEAs on different Vswitches and grouping a portion of the LPARs on one Vswitch/SEA and another portion of the LPARs on another Vswitch/SEA.

If there is a single LPAR producing the majority of the network traffic, it may be worthwhile to dedicate an adapter to that LPAR.

4.5.6 DNS and IPv4 settings

AIX name resolution by default tries to resolve both IPv4 and IPv6 addresses. The first attempt is to resolve the name locally and then request the DNS server. Some software that has IPv6 support, such as Oracle 11g, IBM Tivoli Directory Server, may suffer some delay. In the case of such software, even if hostname is resolved by IPv4, a second attempt takes place for IPv6 resolution. If the IPv6 resolution is unsuccessful by trying `/etc/hosts`, the request goes to the DNS server. If you use only IPv4 and your DNS server is not able to answer this request, your application waits until the DNS time-out occurs.

If you are not using IPv6, disable IPv6 lookups on AIX adding the following line to `/etc/netsvc.conf`:

```
hosts=local4,bind4
```

Note: If your NSORDER environment variable is set, it overrides the `/etc/netsvc.conf` file.

4.5.7 Performance impact due to DNS lookups

DNS lookups are often used by commercial applications as well as network daemons to resolve a given hostname to an IP address. Any delay in lookup due to a firewall, network congestion, or unreachable network can cause the host network to either retry the lookups or error out. This might impact some applications that are network sensitive. It is important that such delays are identified and addressed quickly to avoid any degradation in application performance.

Identify a lookup failure

In this section, we examine a DNS lookup response delay. The first DNS server is in close proximity to the requesting server, while the second DNS server is located at another geographical location (Example 4-85 and Example 4-86).

Example 4-85 DNS server lookup round trip time - Scenario 1: DNS lookup time 26 ms

```
   # startsrc -s iptrace -a "-a -b -s 9.184.192.240 /tmp/iptrace_local_dns"
   [4587574]
   0513-059 The iptrace Subsystem has been started. Subsystem PID is 4587574.
   # nslookup host.sample.com
   Server:         9.184.192.240
   Address:        9.184.192.240#53

Non-authoritative answer:
Name:   host.sample.com
Address: 9.182.76.38

# stopsrc -s iptrace
0513-044 The iptrace Subsystem was requested to stop.
iptrace: unload success!
```

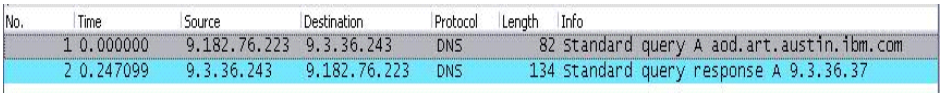
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	9.182.76.223	9.184.192.240	DNS	78	Standard query A ctb5p04.in.ibm.com
2	0.026574437	9.184.192.240	9.182.76.223	DNS	94	Standard query response A 9.182.76.38

Example 4-86 DNS server lookup round trip time - Scenario 2: DNS lookup time 247 ms

```
# startsrc -s iptrace -a "-a -b -s 9.3.36.243 /tmp/iptrace_remote_dns"
[4587576]
0513-059 The iptrace Subsystem has been started. Subsystem PID is 4587576.
# nslookup remote_host.sample.com 9.3.36.243
Server:         9.3.36.243
Address:        9.3.36.243#53

Name:   remote_host.sample.com
Address: 9.3.36.37

# stopsrc -s iptrace
0513-044 The iptrace Subsystem was requested to stop.
iptrace: unload success!
```



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9.182.76.223	9.3.36.243	DNS	82	Standard query A aod.art.austin.ibm.com
2	0.247099	9.3.36.243	9.182.76.223	DNS	134	Standard query response A 9.3.36.37

To overcome a delayed lookup, it is advised to configure the netcd daemon on the requesting host that would cache the response retrieved from resolvers.

4.5.8 TCP retransmissions

TCP retransmissions can occur due to a faulty network, the destination server not able to receive packets, the destination server not able to send the acknowledgement before the retransmission timer expires, or the acknowledgement getting lost somewhere in the middle, to name a few. This leads to the sender retransmitting the packets again, which could degrade the application performance. High retransmission rates between an application and database server, for example, need to be identified and corrected. We illustrate some details on TCP retransmission, the algorithm, and the timer wheel algorithm for retransmission in the following sections.

Identifying TCP retransmissions

The most common commands or tools to identify TCP retransmissions are **netstat** and **iptrace**. The first step is to use **iptrace** to identify whether there are TCP retransmissions in your environment, as shown in Example 4-87.

Example 4-87 Identifying TCP retransmission using iptrace

```
# startsrc -s iptrace -a "-a /tmp/iptrace_retransmission"
# stopsrc -s iptrace
# ipreport iptrace_retransmission > retransmission.out

# cat retransmission.out
====( 692 bytes transmitted on interface en0 )==== 22:25:50.661774216
ETHERNET packet : [ 3e:73:a0:00:80:02 -> 00:00:0c:07:ac:12 ]  type 800  (IP)
IP header breakdown:
        < SRC =     9.184.66.46 >  (stglbs9.in.ibm.com)
        < DST =    9.122.161.39 >  (aiwa.in.ibm.com)
        ip_v=4, ip_hl=20, ip_tos=0, ip_len=678, ip_id=25397, ip_off=0
        ip_ttl=60, ip_sum=2296, ip_p = 6 (TCP)
TCP header breakdown:
        <source port=23(telnet), destination port=32943 >
        th_seq=2129818250, th_ack=2766657268
        th_off=8, flags<PUSH | ACK>
        th_win=65322, th_sum=0, th_urp=0


====( 692 bytes transmitted on interface en0 )==== 22:25:51.719416953
ETHERNET packet : [ 3e:73:a0:00:80:02 -> 00:00:0c:07:ac:12 ]  type 800  (IP)
IP header breakdown:
        < SRC =     9.184.66.46 >  (stglbs9.in.ibm.com)
        < DST =    9.122.161.39 >  (aiwa.in.ibm.com)
        ip_v=4, ip_hl=20, ip_tos=0, ip_len=678, ip_id=25399, ip_off=0
        ip_ttl=60, ip_sum=2294, ip_p = 6 (TCP)
TCP header breakdown:
        <source port=23(telnet), destination port=32943 >
```

```
        th_seq=2129818250, th_ack=2766657268
        th_off=8, flags<PUSH | ACK>
        th_win=65322, th_sum=0, th_urp=0


====( 692 bytes transmitted on interface en0 )==== 22:25:54.719558660
ETHERNET packet : [ 3e:73:a0:00:80:02 -> 00:00:0c:07:ac:12 ]  type 800  (IP)
IP header breakdown:
        < SRC =     9.184.66.46 >  (stglbs9.in.ibm.com)
        < DST =    9.122.161.39 >  (aiwa.in.ibm.com)
        ip_v=4, ip_hl=20, ip_tos=0, ip_len=678, ip_id=25404, ip_off=0
        ip_ttl=60, ip_sum=228f, ip_p = 6 (TCP)
TCP header breakdown:
        <source port=23(telnet), destination port=32943 >
        th_seq=2129818250, th_ack=2766657268
        th_off=8, flags<PUSH | ACK>
        th_win=65322, th_sum=0, th_urp=0

====( 692 bytes transmitted on interface en0 )==== 22:26:00.719770238
ETHERNET packet : [ 3e:73:a0:00:80:02 -> 00:00:0c:07:ac:12 ]  type 800  (IP)
IP header breakdown:
        < SRC =     9.184.66.46 >  (stglbs9.in.ibm.com)
        < DST =    9.122.161.39 >  (aiwa.in.ibm.com)
        ip_v=4, ip_hl=20, ip_tos=0, ip_len=678, ip_id=25418, ip_off=0
        ip_ttl=60, ip_sum=2281, ip_p = 6 (TCP)
TCP header breakdown:
        <source port=23(telnet), destination port=32943 >
        th_seq=2129818250, th_ack=2766657268
        th_off=8, flags<PUSH | ACK>
        th_win=65322, th_sum=0, th_urp=0

====( 692 bytes transmitted on interface en0 )==== 22:26:12.720165401
ETHERNET packet : [ 3e:73:a0:00:80:02 -> 00:00:0c:07:ac:12 ]  type 800  (IP)
IP header breakdown:
        < SRC =     9.184.66.46 >  (stglbs9.in.ibm.com)
        < DST =    9.122.161.39 >  (aiwa.in.ibm.com)
        ip_v=4, ip_hl=20, ip_tos=0, ip_len=678, ip_id=25436, ip_off=0
        ip_ttl=60, ip_sum=226f, ip_p = 6 (TCP)
TCP header breakdown:
        <source port=23(telnet), destination port=32943 >
        th_seq=2129818250, th_ack=2766657268
        th_off=8, flags<PUSH | ACK>
        th_win=65322, th_sum=955e, th_urp=0
```

The sequence number (th_seq) uniquely identifies a packet, and if you observe multiple packets with the same sequence number in the ipreport output, then the particular packet is retransmitted. In the above output, the same packet with 692 bytes is retransmitted four times, which leads to a delay of 22 seconds.

Besides the **ipreport** command, you can use the Wireshark tool to analyze the iptrace output file. Wireshark is an open source network protocol analyzer. It has a GUI interface and can be used on your laptop. Wireshark can be downloaded at:

http://www.wireshark.org/

Figure 4-18 shows a TCP retransmission example using Wireshark. Note that data is collected when the timer wheel algorithm is enabled, which will be introduced later.

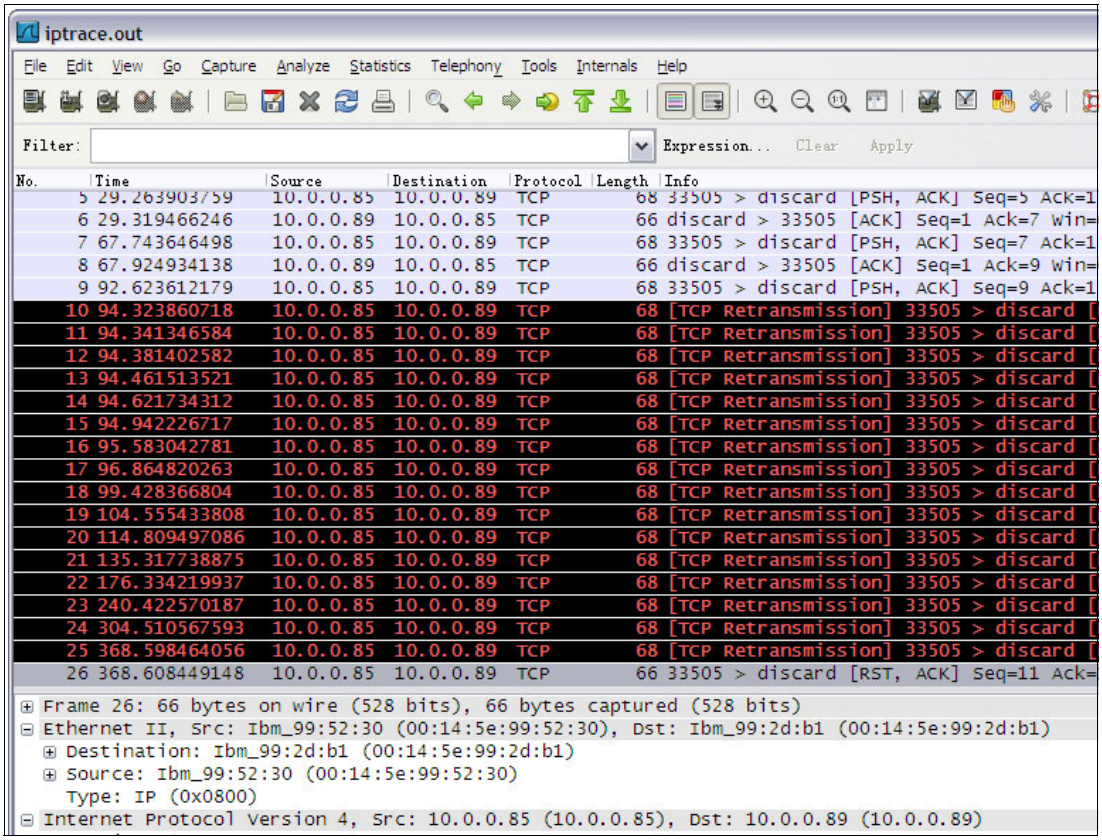


Figure 4-18 TCP retransmission example using the Wireshark tool

Conventional TCP retransmission

Conventional TCP retransmission happens in the following conditions:

- Retransmission timer (RTO) expires

  This depends on the RTO calculation. RTO is calculated by smoothed Round Trip Time (RTT). The initial value of RTO is 3 seconds. Because the RTO timer is implemented using the slower timer, the precision is 500 ms. Considering the variance introduced by the smoothed RTT algorithm, the normal RTO for intranet is 1.5 seconds or so.

- The connection goes into fast retransmit phase

  This is controlled by the `no` option `tcprexmtthresh`, which is 3 by default. This means when three consecutive duplicate ACKs are received, the TCP connection will go to fast retransmit phase, and the retransmit will happen right away.

If ACK is not received, it doubles the previous RTO for each consecutive retransmission of the same segment. The RTO will constantly be rto_high (64 seconds by default) if it exceeds rto_high. The maximum retransmission attempt is set by rto_length, which is 13 by default. This is called the exponential backoff algorithm.

Example 4-88 gives the tcpdump output in a typical TCP retransmission timeout scenario.

Example 4-88 tcpdump output in typical TCP retransmission scenario

```
09:14:50.731583 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32761 <nop,nop,timestamp 1351499394
1350655514>
```

```
09:14:52.046243 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32761 <nop,nop,timestamp 1351499396
1350655514> //this is the first retransmission, happens at 1.31 seconds (RTO = 1.5 seconds).
09:14:55.046567 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499402
1350655514> //2nd retransmission, RTO = 3 seconds, doubled.
09:15:01.047152 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499414
1350655514> //3rd retransmission, RTO = 6 seconds, doubled.
09:15:13.048261 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499438
1350655514> //4th retransmission, RTO = 12 seconds, doubled.
09:15:37.050750 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499486
1350655514> //5th retransmission, RTO = 24 seconds, doubled.
09:16:25.060729 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499582
1350655514> //6th retransmission, RTO = 48 seconds, doubled.
09:17:29.067259 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499710
1350655514> //7th retransmission, RTO = 64 seconds, which is equal to rto_high.
09:18:33.074418 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499838
1350655514> //8th retransmission, RTO = 64 seconds.
09:19:37.082240 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351499966
1350655514>
09:20:41.088737 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351500094
1350655514>
09:21:45.094912 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351500222
1350655514>
09:22:49.110835 IP p750s1aix1.38894 > 10.0.0.89.discard: P 9:11(2) ack 1 win 32844 <nop,nop,timestamp 1351500350
1350655514>
09:23:53.116833 IP p750s1aix1.38894 > 10.0.0.89.discard: R 11:11(0) ack 1 win 32844 <nop,nop,timestamp 1351500478
1350655514> //reach the maximum retransmission attempts, rto_length = 13, reset the connection.
```

Note: The tcpdump output in Example 4-88 and Example 4-89 on page 203 illustrates the cases when the maximum retransmission attempts are reached, and the connections are reset. In normal cases, if there is ACK to any of the retransmission packet, the TCP connection becomes normal again, as shown in Example 4-90 on page 204. When there are ACKs to the retransmitted packets, the retransmission ends.

Timer wheel algorithm for fine granularity retransmission

The timer wheel algorithm can be enabled by setting the **no** option `timer_wheel_tick=1` or larger value. When the timer wheel algorithm is enabled in AIX, TCP uses a fine granularity retransmission timer with precision equal to **`timer_wheel_tick`** * 10ms. When the timer wheel algorithm is in effect, the RTO is initially set by the **no** option **`tcp_low_rto`**, and is adjusted based on real RTT values.

Note: The timer wheel algorithm only takes effect after the connection experiences the first segment lost, that is, the two conditions mentioned in the “conventional TCP retransmission” section. Otherwise the conventional retransmission algorithm still prevails.

When the timer wheel algorithm is in effect, you can observe faster retransmission. Example 4-89 shows a TCP retransmission scenario when timer_wheel_tick=1 and tcp_low_rto=20. You can see that after the first conventional retransmission timeout (RTO=1.5 seconds), the RTO is set to 20 ms and the timer wheel algorithm is enabled, and the retransmission still uses the "exponential backoff" algorithm.

Example 4-89 tcpdump output for TCP retransmission when the timer wheel algorithm is in effect

```
10:16:58.014781 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32761
<nop,nop,timestamp 1350657966 1350657589>
10:16:59.543853 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32761
<nop,nop,timestamp 1350657969 1350657589>//1st retransmission timer expiry, RTO=1.5s, which
is using conventional algorithm
10:16:59.556742 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32761
<nop,nop,timestamp 1350657970 1350657589>//2nd retransmission, RTO = 13ms(~20ms), that is
the tcp_low_rto. timer wheel algorithm is in effect.
10:16:59.601225 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657970 1350657589>//3rd retransmission, RTO = 45ms(~40ms)
10:16:59.681372 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657970 1350657589>//4th retransmission, RTO = 80ms
```

```
10:16:59.841581 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657970 1350657589>//5th retransmission, RTO = 160ms
10:17:00.162023 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657971 1350657589>
10:17:00.802936 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657972 1350657589>
10:17:02.084883 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657975 1350657589>
10:17:04.648699 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657980 1350657589>
10:17:09.776109 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350657990 1350657589>
10:17:20.030824 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350658010 1350657589>
10:17:40.550530 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350658052 1350657589>
10:18:21.569311 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350658134 1350657589>
10:19:25.657746 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350658262 1350657589>
10:20:29.746815 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350658390 1350657589>
10:21:33.836267 IP p750s1aix1.32859 > 10.0.0.89.discard: P 18:20(2) ack 1 win 32844
<nop,nop,timestamp 1350658518 1350657589>
10:21:33.846253 IP p750s1aix1.32859 > 10.0.0.89.discard: R 20:20(0) ack 1 win 32844
<nop,nop,timestamp 1350658518 1350657589>//reach the maximum retransmission attempts,
TCP_LOW_RTO_LENGTH=15, reset the connection.
```

The tcp_low_rto is only used once for each TCP connection when the timer wheel algorithm starts to function. Afterward RTO is calculated based on RTT, and the value is dynamic, depending on the network conditions. Example 4-90 gives an example on future retransmission timeouts when the timer wheel algorithm has already been enabled.

Example 4-90 Following retransmission timeout when timer wheel algorithm is already enabled

```
10:52:07.343305 IP p750s1aix1.32907 > 10.0.0.89.discard: P 152:154(2) ack 1 win 32844
<nop,nop,timestamp 1350662185 1350661918>
10:52:07.482464 IP 10.0.0.89.discard > p750s1aix1.32907: . ack 154 win 65522
<nop,nop,timestamp 1350661918 1350662185>

10:52:22.351340 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662215 1350661918>
10:52:22.583407 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662215 1350661918> //This time the 1st retransmission happens at
230ms. This is based on the real RTO, not tcp_low_rto=20ms anymore.
10:52:23.064068 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662216 1350661918>
10:52:24.025950 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662218 1350661918>
10:52:25.948219 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662222 1350661918>
10:52:29.793564 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662230 1350661918>
10:52:37.484235 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662245 1350661918>
10:52:52.865914 IP p750s1aix1.32907 > 10.0.0.89.discard: P 154:156(2) ack 1 win 32844
<nop,nop,timestamp 1350662276 1350661918>
```

```
10:52:52.885960 IP 10.0.0.89.discard > p750s1aix1.32907: . ack 156 win 65522
<nop,nop,timestamp 1350662009 1350662276> //ACK received for the 7th retransmission, and
then the retransmission ends.
```

Note: For a high-speed network such as 10 Gb, if there is occasional data loss, it should help to enable the timer wheel algorithm by setting the `timer_wheel_tick` and `tcp_low_rto` no options. Then the TCP retransmission will be much faster than the default.

Due to the default *delayed acknowledgment* feature of AIX, the real RTT is usually larger than the value of the no option `fastimo`. Thus the `tcp_low_rto` should be larger than the value of `fastimo`, unless the no option `tcp_nodelayack` is set to 1.

We used the inet **`discard`** service to generate the data flow and **`tcpdump`** to dump information of the network packets for the samples in this section. You can duplicate the tests in your own environment.

For more details, refer to the “Implement lower timer granularity for retransmission of TCP” at:

`http://www.ibm.com/developerworks/aix/library/au-lowertime/index.html`

4.5.9 tcp_fastlo

Different applications that run on the same partition and communicate to each other via the loopback interface may gain some performance improvement by enabling the `tcp_fastlo` parameter. It will simplify the TCP stack loopback communication.

The `tcp_fastlo` parameter enables the *Fastpath Loopback* on AIX. With this option enabled, systems that make use of local communication can benefit from improved throughput and processor saving.

Traffic accounting for loopback traffic is not seen on the loopback interface when *Fastpath Loopback* is enabled. Instead that traffic is reported by specific counters. The TCP traffic though is still accounted as usual.

Example 4-91 illustrates the use of the **`netstat`** command to get the statistics when *Fastpath Loopback* is enabled.

Example 4-91 netstat output showing Fastpath LOOPBACK traffic

```
# netstat -p tcp | grep fastpath
        34 fastpath loopback connections
        14648280 fastpath loopback sent packets (14999698287 bytes)
        14648280 fastpath loopback received packets (14999698287 bytes
```

The parameter *tcp_fastlo_crosswpar* is also available to enable the same functionality for Workload Partition environments.

4.5.10 MTU size, jumbo frames, and performance

Maximum transmit unit (MTU) represents the maximum size that a frame being transmitted to the network can have. The MTU is a standard value that ideally must be set equally across all network devices. For Ethernet networks this size is 1500 bytes. Each piece of data transmitted among the hosts is done in small chunks of MTU size, representing some I/O system calls for each of them.

Jumbo frames enable systems for an MTU size of 9000 bytes, meaning that large transmission of data can be performed with fewer system calls.

Theoretically, since fewer system calls are required to transmit large data, some performance gain would be observed in some environments. We say theoretically, because to have jumbo frames enabled on a server, you must ensure that all the other network components, including other servers, are enabled as well. Devices that do not support jumbo frames would simply drop the frames and return some ICMP notification message to the sender, causing retransmission and implying some network performance problems.

Workloads such as web servers, which usually transmit small pieces of data, would usually not benefit from large MTU sizes.

Note: Before changing the MTU size of the server, ensure that your network supports that setting.

Important: Some environments block ICMP on firewalls to avoid network attacks. This means that an ICMP notification message may never reach the sender in these environments.

5

Testing the environment

In this chapter we provide information on how to establish test plans, test the different components of your environment at the operating system level, how to interpret information reported by the analysis tools, how to spot bottlenecks, and how to manage your workload. This information can be used by those who are either building a system from scratch, applying all the concepts from the previous chapters, or those who are looking for an improvement of a production environment.

We help you establish and implement tests of the environment, and give you some guidance to understand the results of the changes implemented.

The following topics are discussed in this chapter:

- Understand your environment
- Testing the environment
- Testing components
- Understanding processor utilization
- Memory utilization
- Disk storage bottleneck identification
- Network utilization
- Performance analysis at the CEC
- VIOS performance advisor tool and the part command
- Workload management

5.1 Understand your environment

Submitting an environment for performance analysis is often a complex task. It usually requires a good knowledge of the workloads running, system capacity, technologies available, and it involves a lot of tuning and tests.

To understand the limits of the different components of the environment is crucial to establish baselines and targets and set expectations.

5.1.1 Operating system consistency

While keeping AIX levels up to date is an obvious concern, keeping things consistent is often overlooked. One LPAR might have been updated with a given APAR or fix, but how to be sure it was also applied to other LPARs of the same levels? Not maintaining consistency is a way of introducing performance problems, as some LPARs can remain backlevel or unpatched. There are many ways to track levels of installed AIX across LPARs. One often overlooked option is provided by NIM.

The **niminv** command allows administrators to gather, conglomerate, compare and download fixes based on installation inventory of NIM objects. It provides an easy method to ensure systems are at an expected level.

niminv can use any NIM object that contains installation information. Examples include standalone client, SPOT, lpp_source and mksysb objects.

Using **niminv** has the following benefits:

- Hardware installation inventory is gathered alongside the software installation inventory.
- Data files are saved with a naming convention that is easily recognizable.

Example 5-1 illustrates using **niminv** to compare one NIM client (aix13) with another (aix19). For each NIM client there will be a column. The value will either be listed “same” if the level for the file set is the same for the target as the base, and “-” if missing and the actual level if existing but different (higher or lower).

Example 5-1 Using niminv with invcom to compare installed software levels on NIM clients

```
root@nim1: /usr/sbin/niminv -o invcmp -a targets='aix13,aix19' -a base='aix13' -a
location='/tmp/123'
Comparison of aix13 to aix13:aix19 saved to
/tmp/123/comparison.aix13.aix13:aix19.120426230401.
Return Status = SUCCESS

root@nim1: cat /tmp/123/comparison.aix13.aix13:aix19.120426230401
#                  name                        base         1           2
----------------------------------------   ----------  ----------  ----------
AIX-rpm-7.1.0.1-1                          7.1.0.1-1   same        same
...lines omitted...
bos.64bit                                  7.1.0.1     same        same
bos.acct                                   7.1.0.0     same        same
bos.adt.base                               7.1.0.0     same        same
bos.adt.include                            7.1.0.1     same        same
bos.adt.lib                                7.1.0.0     same        same
...lines omitted...
bos.rte                                    7.1.0.1     same        same
```

```
...lines omitted...

base = comparison base = aix13
1    = aix13
2    = aix19
'-'  = name not in system or resource
same = name at same level in system or resource
```

5.1.2 Operating system tunable consistency

In environments where tunables beyond the defaults are required, it is important to maintain an overview of what is applied across an environment, and to ensure that tunables are consistent and not removed. Also, to keep track of what is applied as a reminder in case some only need to be temporarily enabled.

System tunable consistency check can be done using the AIX Runtime Expert (ARTEX). The existing samples in the `/etc/security/artex/samples` directory can be used to create a new profile with the **`artexget`** command, which can be customized. The corresponding catalog in `/etc/security/artex/catalogs` is referred to for retrieving and setting values for that parameter.

Note: The **`artexget`** and **`artexset`** commands execute the <GET> and <SET> sections, respectively, in the cfgMethod of the Catalog which is defined for a particular parameter.

Example 5-2 shows a simple profile, which can be used with ARTEX tools.

Example 5-2 AIX Runtime Expert sample profile

```
root@nim1: cat /etc/security/artex/samples/aixpertProfile.xml
<?xml version="1.0" encoding="UTF-8"?>
<Profile origin="reference" readOnly="true" version="2.0.0">
 <Catalog id="aixpertParam" version="2.0">
  <Parameter name="securitysetting"/>
 </Catalog>
</Profile>
```

Example 5-3 shows a simple catalog that can be used with the ARTEX tools with a corresponding profile. Note the Get and Set stanzas and Command and Filter attributes, which can be modified and used to create customized catalogues to extend the capabilities of ARTEX.

Example 5-3 AIX Runtime Expert sample catalog

```
root@nim1: cat /etc/security/artex/catalogs/aixpertParam.xml
<?xml version="1.0" encoding="UTF-8"?>
<Catalog id="aixpertParam" version="2.0" priority="-1000">

 <ShortDescription><NLSCatalog catalog="artexcat.cat" setNum="2" msgNum="1">System
security level configuration.</NLSCatalog></ShortDescription>

 <Description><NLSCatalog catalog="artexcat.cat" setNum="2" msgNum="2">The aixpert
command sets a variety of system configuration settings to enable the desired
security level.</NLSCatalog></Description>
```

```
 <ParameterDef name="securitysetting" type="string">
  <Get type="current">
   <Command>/etc/security/aixpert/bin/chk_report</Command>
   <Filter>tr -d '\n'</Filter>
  </Get>
  <Get type="nextboot">
   <Command>/etc/security/aixpert/bin/chk_report</Command>
   <Filter>tr -d '\n'</Filter>
  </Get>
  <Set type="permanent">
   <Command>/usr/sbin/aixpert -l %a</Command>
   <Argument>`case %v1 in 'HLS') echo 'h';; 'MLS') echo 'm';; 'LLS') echo 'l';;
'DLS') echo 'd';; 'SCBPS') echo 's';; *) echo 'd';; esac`</Argument>
  </Set>
 </ParameterDef>
</Catalog>
```

One method to employ these capabilities is to use NIM to perform an ARTEX operation on a group of systems (Example 5-4); this would provide a centralized solution to GET, SET and compare (DIFF) the attribute values across the group.

Example 5-4 Using NIM script to run AIX Runtime Expert commands on NIM clients

```
root@nim1: cat /export/scripts/artex_diff

root@nim1: artexget -r -f txt /etc/security/artex/samples/viosdevattrProfile.xml

root@nim1: nim -o define -t script -a server=master -a
location=/export/scripts/artex_diff artex_diff

root@nim1: nim -o allocate -a script=artex_diff nimclient123

root@nim1: nim -o cust nimclient123
Component name     Parameter name       Parameter value    Additional Action
-----------------  -------------------  -----------------  ----------------------
viosdevattrParam   reserve_policy       no_reserve         NEXTBOOT
viosdevattrParam   queue_depth          3                  NEXTBOOT
...lines omitted...
```

5.1.3 Size that matters

The world is dynamic. Everything changes all the time and businesses react in the same way. When you do performance analysis on the environment, you will eventually find that the problem is not how your systems are configured, but instead how they are sized. The initial sizing for a specific workload may not fit your business needs after a while. You may find out that some of your infrastructure is undersized or even oversized for different workloads and you have to be prepared to change.

5.1.4 Application requirements

Different applications are built for different workloads. An application server built for a demand of ten thousand users per month may not be ready to serve one hundred thousand users. This is a typical scenario where no matter how you change your infrastructure environment,

you do not see real benefits of the changes you have made unless your application is also submitted to analysis.

5.1.5 Different workloads require different analysis

One of the most important factors when you analyze your systems is that you have a good understanding of the different types of workloads that you are running. Having that knowledge will lead you to more objective work and concise results.

5.1.6 Tests are valuable

Each individual infrastructure component has its own limitations, and understanding these different limits is never easy. For example, similar network adapters show different throughputs depending on other infrastructure components like the number of switches, routers, firewalls, and their different configurations. Storage components are not different, they behave differently depending on different factors.

- Individual tests

  A good way to understand the infrastructure limits is by testing the components individually so that you know what to expect from each of them.

- Integration tests

  Integration tests are good to get an idea about how the infrastructures interact and how that affects the overall throughput.

Note: Testing your network by transmitting packets between two ends separated by a complex infrastructure, for example, can tell you some data about your environment throughput but may not tell you much about your individual network components.

5.2 Testing the environment

This section offers some suggestions on how to proceed with testing your environment. By doing so systematically you should be able to determine whether the changes made based on the concepts presented throughout this book have beneficial results on your environment.

A good thing to keep in mind is that not every system or workload will benefit from the same tuning.

5.2.1 Planning the tests

When the environment is going to be tested, it is good practice to establish goals and build a test plan.

The following topics are important things to be considered when building a test plan:

- Infrastructure

  Knowledge about the type of machines, their capacity, how they are configured, partition sizing, resource allocation, and about other infrastructure components (network, storage) is important. Without this information it is just hard to establish baselines and goals, and to set expectations.

- Component tests

  Test one component at a time. Even though during the tests some results may suggest that other components should be tuned, testing multiple components may not be a good idea since it involves a lot of variables and may lead to confusing results.

- Correct workload

  The type of workload matters. Different workloads will have different impact on the tests, and thus it is good to tie the proper workload to the component being tested as much as possible.

- Impact and risk analysis

  Tests may stress several components at different levels. The impact analysis of the test plan should consider as many levels as possible to mitigate any major problems with the environment.

  In the past years, with all the advance of virtualized environments, shared resources have become a new concern when testing. Stressing a system during a processor test may result in undesired resource allocations. Stressing the disk subsystem might create bottlenecks for other production servers.

- Baselines and goals

  Establishing a baseline is not always easy. The current environment configuration has to be evaluated and monitored before going through tests and tuning. Without a baseline, you have nothing to compare with your results.

  Defining the goals you want to achieve depends on understanding of the environment. Before establishing a performance gain on network throughput of 20%, for instance, you must first know how the entire environment is configured.

  Once you have a good understanding of how your environment behaves and its limitations, try establishing goals and defining what is a good gain, or what is a satisfactory improvement.

- Setting the expectations

  Do not assume that a big boost in performance can always be obtained. Eventually you may realize that you are already getting the most out of your environment and further improvements can only be obtained with new hardware or with better-written applications.

  Be reasonable and set expectations of what is a good result for the tests.

  Expectations can be met, exceeded, or not met. In any case, tests should be considered an investment. They can give you a good picture of how the environment is sized, its ability to accommodate additional workload, estimation of future hardware needs, and the limits of the systems.

5.2.2 The testing cycle

A good approach to test the environment is to establish cycles of tests, broken into the following steps:

- Establish a plan

  Set the scope of your tests, which components will be tested, which workloads will be applied, whether they are real or simulation, when tests will be made, how often the system will be monitored, and so on.

- Make the changes

  Change the environment according to the plan, trying to stay as much as possible inside the scope of the defined plan.

- Monitor the components

  Establish a period to monitor the system and collect performance data for analysis. There is no best period of time for this, but usually a good idea is to monitor the behavior of the system for a few days at least and try to identify patterns.

- Compare the results

  Compare the performance data collected with the previous results. Analysis of the results can be used as an input to a new cycle of tests with a new baseline.

You can establish different plans, test each one in different cycles, measure and compare the results, always aiming for additional improvement. The cycle can be repeated as many times as necessary.

5.2.3 Start and end of tests

This section provides information on when to start and end the tests.

When to start testing the environment

A good time to start profiling the environment is now. Unless you have a completely static environment, well sized and stable, tests should be a constant exercise.

Workload demands tend to vary either by increasing or decreasing with time, and analyzing the environment is a good way to find the right moment to review the resource distribution.

Imagine a legacy system being migrated to a new environment. The natural behavior is for a new system to demand more resources with time, and the legacy system demanding less.

When to stop testing the environment

Testing the environment takes time, requires resources, and has costs. At some point, tests will be interrupted by such restrictions.

Despite these restrictions, assuming that a plan has been established at the beginning, the best moment to stop the tests is when the results achieve at least some of the established goals.

The reasons why an environment is submitted to tests can vary and no matter what the goals of the tests are, their results should be meaningful and in accordance with the goals defined, even if you cannot complete all the tests initially planned.

Systems have limits

Every environment has its limits but only tests will tell what your environment's are. Eventually you may find that even though everything has been done on the system side, the performance of the applications is still not good. You may then want to take a look at the application architecture.

5.3 Testing components

In this section we try to focus on simple tests of the components, and which tools you can use to monitor system behavior, to later demonstrate how to read and interpret the measured values.

Testing the system components is usually a simple task and can be accomplished by using native tools available on the operating system by writing a few scripts. For instance, you may not be able to simulate a multithread workload with the native tools, but you can spawn a few processor-intensive processes and have an idea of how your system behaves.

Basic network and storage tests are also easy to perform.

Note: It is not our intention to demonstrate or compare the behavior of processes and threads. The intention of this section is to put a load on the processor of our environment and use the tools to analyze the system behavior.

How can I know, for example, that the 100 MB file retrieval response time is reasonable? Its response time is composed of network transmission + disk reading + application overhead. I should be able to calculate that, in theory.

5.3.1 Testing the processor

Before testing the processing power of the system, it is important to understand the concepts explained in this book because there are a lot of factors that affect the processor utilization of the system.

To test the processor effectively, the ideal is to run a processor-intensive workload. Running complex systems that depend on components such as disk storage or networks might not result in an accurate test of the environment and can result in misleading data.

The process queue

The process queue is a combination of two different queues: the run queue and wait queue. Threads on the run queue represent either threads ready to run (awaiting for a processor time slice) or threads already running. The wait queue holds threads waiting for resources or I/O requests to complete.

Running workloads with a high number of processes is good for understanding the response of the system and to try to establish the point at which the system starts to become unresponsive.

In this section, the *nstress* suite has been used to put some load on the system. The tests are made running the **ncpu** command starting with 16 processes. On another window we monitored the process queue with the **vmstat** command, and a one-line script to add time information at the front of each line to check the output. Table 5-1 illustrates the results.

Table 5-1 Tests run on the system

Processes	System response
16	Normal
32	Normal
64	Minimal timing delays
96	Low response. Terminals not responding to input.
128	Loss of output from **vmstat**.

The system performed well until we put almost a hundred processes on the queue. Then the system started to show slow response and loss of output from **vmstat**, indicating that the system was stressed.

A different behavior is shown in Example 5-5. In this test, we started a couple of commands to create one big file and several smaller files. The system has only a few processes on the run queue, but this time it also has some on the wait queue, which means that the system is waiting for I/O requests to complete. Notice that the processor is not overloaded, but there are processes that will keep waiting on the queue until their I/O operations are completed.

Example 5-5 vmstat output illustrating processor wait time

```
vmstat 5

System configuration: lcpu=16 mem=8192MB ent=1.00

kthr         memory                 page                  faults             cpu
-----      -----------      ------------------------     ------------     -----------------------
  r  b       avm   fre      re  pi  po  fr   sr     cy   in   sy  cs      us sy id wa    pc    ec
  2  1    409300  5650       0   0   0 23800 23800   0  348  515 1071      0 21 75  4  0.33  33.1
  1  3    409300  5761       0   0   0 23152 75580   0  340  288 1030      0 24 67  9  0.37  36.9
  3  4    409300  5634       0   0   0 24076 24076   0  351  517 1054      0 21 66 12  0.34  33.8
  2  3    409300  5680       0   0   0 24866 27357   0  353  236 1050      0 22 67 11  0.35  34.8
  0  4    409300  5628       0   0   0 22613 22613   0  336  500 1036      0 21 67 12  0.33  33.3
  0  4    409300  5622       0   0   0 23091 23092   0  338  223 1030      0 21 67 12  0.33  33.3
```

5.3.2 Testing the memory

This topic addresses some tests that can be made at the operating system level to measure how much workload your current configuration can take before the system becomes unresponsive or kills processes.

The system we were using for the tests was a partition with 8 GB of RAM and 512 Mb of paging-space running AIX 7.1. To simulate the workload, we used the stress tool, publicly available under GPLv2 license at:

http://weather.ou.edu/~apw/projects/stress/

Packages ready for the AIX can be found at:

http://www.perzl.org/aix

The following tests were intended to test how much memory load our system could take before starting to swap, become unresponsive, and kill processes.

The first set of tests tried to establish how many processes we could dispatch using different memory sizes. Before starting the tests, it is important to have a good understanding of virtual memory concepts and how the AIX Virtual Memory Manager works.

There are a few tunables that will affect the behavior of our system during the tests.

The npswarn, npskill, and nokilluid tunables

When AIX detects that memory resource is running out, it might kill processes to release a number of paging space pages to continue running. AIX controls this behavior through the *npswarn*, *npskill* and *nokilluid* tunables.

- npswarn

  The npswarn tunable is a value that defines the minimum number of free paging space pages that must be available. When this threshold is exceeded, AIX will start sending the SIGDANGER signal to all processes except kernel processes.

  The default action for SIGDANGER is to ignore this signal. Most processes will ignore this signal. However, the init process does register a signal handler for the SIGDANGER signal, which will write the warning message `Paging space low` to the defined system console.

  The kernel processes can be shown using **`ps -k`**. Refer to the following website for more information about kernel processes (kprocs):

  `http://www-01.ibm.com/support/docview.wss?uid=isg3T1000104`

- npskill

  If consumption continues, this tunable is the next threshold to trigger; it defines the minimum number of free paging-space pages to be available *before the system starts killing processes*.

  At this point, AIX will send SIGKILL to eligible processes depending on the following factors:

  – Whether or not the process has a SIGDANGER handler

    By default, SIGKILL will only be sent to processes that do not have a handler for SIGDANGER. This default behavior is controlled by the **`vmo`** option **`low_ps_handling`**.

  – The value of the **`nokilluid`** setting, and the UID of the process, which is discussed in the following section.

  – The age of the process

    AIX will first send SIGKILL to the youngest eligible process. This helps to prevent long running processes against a low paging space condition caused by recently created processes. Now you understand why you cannot establish telnet or ssh connections to the system, but still ping it at this point?

    However, note that the long running processes could also be killed if the low paging space condition (below **`npskill`**) persists.

When a process is killed, the system logs a message with the label PGSP_KILL, as shown in Example 5-6.

Example 5-6 errpt output - Process killed by AIX due to lack of paging space

```
LABEL:          PGSP_KILL
IDENTIFIER:     C5C09FFA

Date/Time:       Thu Oct 25 12:49:32 2012
Sequence Number: 373
Machine Id:      00F660114C00
Node Id:         p750s1aix5
Class:           S
Type:            PERM
WPAR:            Global
Resource Name:   SYSVMM

Description
SOFTWARE PROGRAM ABNORMALLY TERMINATED

Probable Causes
```

```
SYSTEM RUNNING OUT OF PAGING SPACE

Failure Causes
INSUFFICIENT PAGING SPACE DEFINED FOR THE SYSTEM
PROGRAM USING EXCESSIVE AMOUNT OF PAGING SPACE

        Recommended Actions
        DEFINE ADDITIONAL PAGING SPACE
        REDUCE PAGING SPACE REQUIREMENTS OF PROGRAM(S)

Detail Data
PROGRAM
stress
USER'S PROCESS ID:
               5112028
PROGRAM'S PAGING SPACE USE IN 1KB BLOCKS
          8
```

The error message gives the usual information with timestamp, causes, recommended actions and details of the process.

In the example, the process **stress** has been killed. For the sake of our tests, it is indeed the guilty process for inducing shortages on the system. However, in a production environment the process killed is not always the one that is causing the problems. Whenever this type of situation is detected on the system, a careful analysis of all processes running on the system must be done during a longer period. The **nmon** tool is a good resource to assist with collecting data to identify the root causes.

In our tests, when the system was overloaded and short on resources, AIX would sometimes kill our SSH sessions and even the SSH daemon.

Tip: The default value for this tunable is calculated with the formula:

npskill = maximum(64, number_of_paging_space_pages/128)

- nokilluid

  This tunable accepts a UID as a value. All processes owned by UIDs below the defined value will be out of the killing list. Its default value is zero (0), which means that even processes owned by the root ID can be killed.

Now that we have some information about these tunables, it is time to proceed with the tests.

One major mistake that people make is to think that a system with certain amounts of memory can take a load matching that same size. This viewpoint is incorrect; if your system has 16 GB of memory, it does not mean that all the memory can be made available to your applications. There are several other processes and kernel structures that also need memory to work.

In Example 5-7, we illustrate the wrong assumption by adding a load of 64 processes, with 128 MB of size each to push the system to its limits (64 x 128 = 8192). The expected result is an overload of the virtual memory and a reaction from the operating system.

Example 5-7 stress - 64x 128MB

```
# date ; stress -m 64 --vm-bytes 128M -t 120 ; date
Thu Oct 25 15:22:15 EDT 2012
stress: info: [15466538] dispatching hogs: 0 cpu, 0 io, 64 vm, 0 hdd
```

```
stress: FAIL: [15466538] (415) <-- worker 4259916 got signal 9
stress: WARN: [15466538] (417) now reaping child worker processes
stress: FAIL: [15466538] (451) failed run completed in 46s
Thu Oct 25 15:23:01 EDT 2012
```

As seen in bold, the process receives a SIGKILL less than a minute after being started. The reason is that the resource consumption levels reached the limits defined by the *npswarn* and *npskill* parameters. This is illustrated in Example 5-8. At 15:22:52 (time is in the last column), the system is exhausted of free memory pages and showing some paging space activity. At the last line, the system had a sudden increase on the paging out and replacement, indicating that the operating system had to make some space by freeing some pages to accommodate the new allocation.

Example 5-8 vmstat output - 64 x 128 MB

```
# vmstat -t 1

System configuration: lcpu=16 mem=8192MB ent=1.00

kthr    memory              page                   faults              cpu             time
----- -----------     ------------------------   ------------ ----------------------- --------
 r  b   avm   fre     re  pi  po  fr     sr  cy   in   sy  cs  us sy id wa    pc    ec hr mi se
67  0 1128383 1012994  0   1   0   0      0   0    9  308  972 99  1  0  0  3.82 381.7 15:22:19
65  0 1215628 925753   0   0   0   0      0   0    2   50 1104 99  1  0  0  4.01 400.7 15:22:20
65  0 1300578 840779   0   0   0   0      0   0   10  254 1193 99  1  0  0  3.98 398.2 15:22:21
65  0 1370827 770545   0   0   0   0      0   0   11   54 1252 99  1  0  0  4.00 400.2 15:22:22
64  0 1437708 703670   0   0   0   0      0   0   20  253 1304 99  1  0  0  4.00 400.0 15:22:23
66  0 1484382 656996   0   0   0   0      0   0   11   50 1400 99  1  0  0  4.00 399.6 15:22:24
64  0 1554880 586495   0   0   0   0      0   0   12  279 1481 99  1  0  0  3.99 398.9 15:22:25
64  0 1617443 523931   0   0   0   0      0   0    4   47 1531 99  1  0  0  3.99 398.7 15:22:26
...
38 36 2209482  4526    0 383 770   0   54608  0  467  138 1995 85 15  0  0  3.99 398.7 15:22:52
37 36 2209482  4160    0 364   0   0   62175  0  317  322 1821 87 13  0  0  3.99 399.5 15:22:53
33 40 2209482  4160    0   0   0   0   64164  0    7  107 1409 88 12  0  0  4.00 399.7 15:22:54
34 42 2209544  4173    0  49 127   997 50978  0   91  328 1676 87 13  0  0  4.01 400.8 15:22:55
31 48 2211740  4508    0  52 2563 3403 27556  0  684  147 2332 87 13  0  0  3.98 398.5 15:22:56
Killed
```

This is normal behavior and indicates that the system is very low on resources (based on VMM tunable values). In sequence, the system would just kill the **vmstat** process along with other application processes in an attempt to free more resources.

Example 5-9 has the **svmon** output for a similar example (the header has been added manually to make it easier to identify the columns). This system has 512 MB of paging space, divided into 131072 x 4096 KB pages. The *npswarn* and *npskill* values are 4096 and 1024, respectively.

Example 5-9 svmon - system running out of paging space

```
# svmon -G -i 5 | egrep "^(s)"
PageSize   PoolSize       inuse        pgsp         pin     virtual
s    4 KB         -      188996       48955      179714      229005
s    4 KB         -      189057       48955      179725      229066
s    4 KB         -      442293       50306      181663      482303
s    4 KB         -      942678       51280      182637      982682
s    4 KB         -     1222664       51825      183184     1262663
s    4 KB         -     1445145       52253      183612     1485143
s    4 KB         -     1660541       52665      184032     1700504
```

```
s    4 KB         -      1789863       52916      184283     1829823
s    4 KB         -      1846800       53196      184395     1887575
s    4 KB         -      1846793       78330      184442     1912289
s    4 KB         -      1846766       85789      184462     1921204
s    4 KB         -      1846800       94455      184477     1929270
s    4 KB         -      1846800      110796      184513     1948082
s    4 KB         -      1846800      128755      184543     1963861
s    4 KB         -       185921       49540      179756      229097
s    4 KB         -       185938       49536      179756      229097
```

Subtracting the number of paging-space pages allocated from the total number of paging spaces, the number of free paging-space frames will be:

131072 - 128755 = 2317 (free paging-space frames)

The resulting value is between *npswarn* and *npskill*. Thus, at that specific moment, the system was about to start killing processes and the last two lines of Example 5-9 on page 218 show a sudden drop of the paging-space utilization indicating that some processes have terminated (in this case they were killed by AIX).

The last example illustrated the behavior of the system when we submitted a load of processes matching the size of the system memory. Now, let us see what happens when we use bigger processes (1024 MB each), but with a fewer number of processes (7).

The first thing to notice in Example 5-10 is that the main process got killed by AIX.

Example 5-10 stress output - 7x1024 MB processes

```
# stress -m 7 --vm-bytes 1024M -t 300
stress: info: [6553712] dispatching hogs: 0 cpu, 0 io, 7 vm, 0 hdd
Killed
```

Although our main process got killed, we still had six processes running, each 1024 MB in size, as shown in Example 5-11, which also illustrates the memory and paging space consumption.

Example 5-11 topas output - 7x1024 MB processes

```
Topas Monitor for host:p750s1aix5               EVENTS/QUEUES    FILE/TTY
Tue Oct 30 15:38:17 2012   Interval:2           Cswitch     226  Readch     1617
                                                Syscall     184  Writech    1825
CPU     User% Kern% Wait% Idle%   Physc  Entc% Reads         9  Rawin         0
Total    76.7   1.4   0.0  21.9    4.00 399.72 Writes       18  Ttyout      739
                                                Forks         0  Igets         0
Network    BPS  I-Pkts  O-Pkts    B-In   B-Out Execs         0  Namei         0
Total    2.07K   11.49    8.50   566.7   1.52K Runqueue   7.00  Dirblk        0
                                                Waitqueue   0.0
Disk    Busy%      BPS      TPS  B-Read  B-Writ                  MEMORY
Total     0.5    56.0K   13.99   56.0K       0 PAGING           Real,MB    8192
                                                Faults     5636  % Comp      94
FileSystem          BPS    TPS  B-Read  B-Writ Steals        0  % Noncomp    0
Total              1.58K   9.00   1.58K       0 PgspIn        13  % Client     0
                                                PgspOut       0
Name            PID  CPU%  PgSp Owner          PageIn        13  PAGING SPACE
stress     11927714 15.0 1.00G root             PageOut       0  Size,MB    512
stress     13893870 14.9 1.00G root             Sios          13  % Used      99
stress      5898362 12.5 1.00G root                              % Free      1
```

```
stress      9109570 12.2 1.00G root            NFS (calls/sec)
stress     11206792 11.1 1.00G root            SerV2         0  WPAR Activ    0
stress     12976324 10.9 1.00G root            CliV2         0  WPAR Total    2
svmon      13959288  0.4 1.13M root            SerV3         0  Press: "h"-help
sshd        4325548  0.2 1.05M root            CliV3         0         "q"-quit
```

In Example 5-12, the **svmon** output illustrates the virtual memory. Even though the system still shows some free pages, it is almost out of paging space. During this situation, dispatching a new command could result in a fork() error.

Example 5-12 svmon - 7x 1024MB processes

```
               size       inuse        free         pin      virtual   mmode
memory      2097152     1988288      108864      372047      2040133     Ded
pg space     131072      130056

              work        pers        clnt       other
pin           236543           0           0      135504
in use       1987398           0         890

PageSize   PoolSize       inuse        pgsp         pin      virtual
s     4 KB         -     1760688      130056      183295      1812533
m    64 KB         -       14225           0       11797        14225
```

Figure 5-1 illustrates a slow increase in memory pages consumption during the execution of six processes with 1024 MB each. We had almost a linear increase for a few seconds until the resources were exhausted and the operating system killed some processes.

The same tests running with memory sizes lower than 1024 MB would keep the system stable.

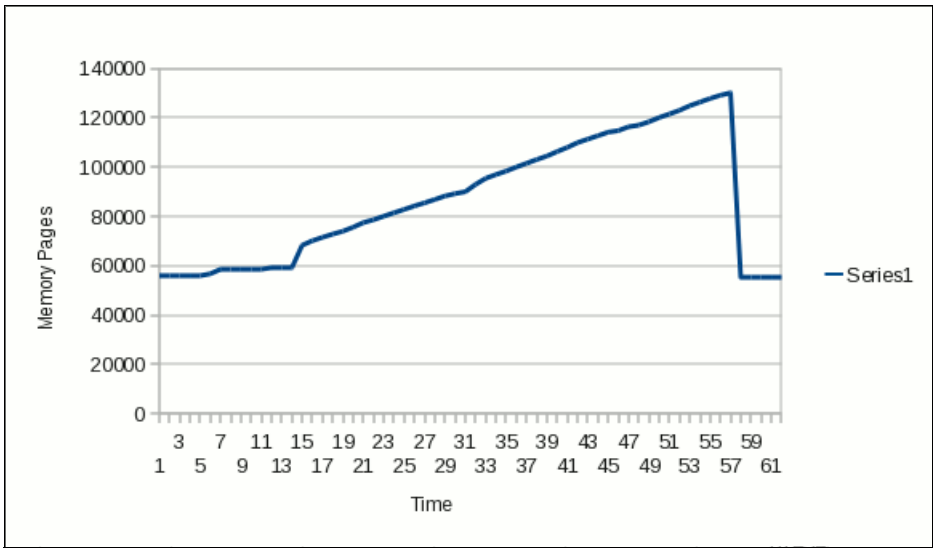


Figure 5-1 Memory pages slow increase

This very same test, running with 1048 MB processes for example, resulted in a stable system, with very low variation in memory page consumption.

These tests are all intended to understand how much load the server could take. Once the limits were understood, the application could be configured according to its requirements, behavior, and system limits.

5.3.3 Testing disk storage

When testing the attachment of an IBM Power System to an external disk storage system, and the actual disk subsystem itself, there are some important considerations before performing any meaningful testing.

Understanding your workload is a common theme throughout this book, and this is true when performing meaningful testing. The first thing is to understand the type of workload you want to simulate and how you are going to simulate it.

There are types of I/O workload characteristics that apply to different applications (Table 5-2).

Table 5-2 I/O workload types

I/O type	Description
Sequential	Sequential access to disk storage is where typically large I/O requests are sent from the server, where data is read in order, one block at a time one after the other. An example of this type of workload is performing a backup.
Random	Random access to disk storage is where data is read in random order from disk storage, typically in smaller blocks, and it is sensitive to latency.

An OLTP transaction processing-based workload typically will have a smaller random I/O request size between 4 k and 8 k. A data warehouse or batch type workload will typically have a larger sequential I/O request size of 16 k and larger. Again, a workload such as a backup server may have a sequential I/O block size of 64 k or greater.

Having a repeatable workload is key to be able to perform a test, make an analysis of the results, perform any attribute changes, and repeat the test. Ideally if you can perform an application-driven load test simulating the actual workload, this is going to be the most accurate method.

There are going to be instances where performing some kind of stress test without any application-driven load is going to be required. This can be performed with the ndisk64 utility, which requires minimal setup time and is available on IBM developerworks at:

`http://www.ibm.com/developerworks/wikis/display/WikiPtype/nstress`

Important: When running the ndisk64 utility against a raw device (such as an hdisk) or an existing file, the data on the device or file will be destroyed.

It is imperative to have an understanding of what the I/O requirement of the workload will be, and the performance capability of attached SAN and storage systems. Using SAP as an example, the requirement could be 35,000 SAPS, which equates to a maximum of 14,500 16 K random IOPS on a storage system with a 70:30 read/write ratio (these values are taken from the IBM Storage Sizing Recommendation for SAP V9).

Before running the ndisk64 tool, you need to understand the following:

- What type of workload are you trying to simulate? Random type I/O or sequential type I/O?

- What is the I/O request size you are trying to simulate?
- What is the read/write ratio of the workload?
- How long will you run the test? Will any production systems be affected during the running of the test?
- What is the capability of your SAN and storage system? Is it capable of handling the workload you are trying to simulate? We found that the ndisk64 tool was cache intensive on our storage system.

Example 5-13 demonstrates running the ndisk64 tool for a period of 5 minutes with our SAP workload characteristics on a test logical volume called ndisk_lv.

Example 5-13 Running the ndisk64 tool

```
root@aix1:/tmp # ./ndisk64 -R -t 300 -f /dev/ndisk_lv -M 20 -b 16KB -s 100G -r 70%
Command: ./ndisk64 -R -t 300 -f /dev/ndisk_lv -M 20 -b 16KB -s 100G -r 70%
        Synchronous Disk test (regular read/write)
        No. of processes = 20
        I/O type         = Random
        Block size       = 16384
        Read-WriteRatio: 70:30 = read mostly
        Sync type: none  = just close the file
        Number of files  = 1
        File size        = 107374182400 bytes = 104857600 KB = 102400 MB
        Run time         = 300 seconds
        Snooze %         = 0 percent
----> Running test with block Size=16384 (16KB) .....................
Proc - <-----Disk IO----> | <-----Throughput------> RunTime
 Num -      TOTAL   IO/sec |    MB/sec       KB/sec  Seconds
   1 -     136965    456.6 |      7.13      7304.84 300.00
   2 -     136380    454.6 |      7.10      7273.65 300.00
   3 -     136951    456.5 |      7.13      7304.08 300.00
   4 -     136753    455.8 |      7.12      7293.52 300.00
   5 -     136350    454.5 |      7.10      7272.05 300.00
   6 -     135849    452.8 |      7.08      7245.31 300.00
   7 -     135895    453.0 |      7.08      7247.49 300.01
   8 -     136671    455.6 |      7.12      7289.19 300.00
   9 -     135542    451.8 |      7.06      7228.26 300.03
  10 -     136863    456.2 |      7.13      7299.38 300.00
  11 -     137152    457.2 |      7.14      7314.78 300.00
  12 -     135873    452.9 |      7.08      7246.57 300.00
  13 -     135843    452.8 |      7.08      7244.94 300.00
  14 -     136860    456.2 |      7.13      7299.19 300.00
  15 -     136223    454.1 |      7.10      7265.29 300.00
  16 -     135869    452.9 |      7.08      7246.39 300.00
  17 -     136451    454.8 |      7.11      7277.23 300.01
  18 -     136747    455.8 |      7.12      7293.08 300.00
  19 -     136616    455.4 |      7.12      7286.20 300.00
  20 -     136844    456.2 |      7.13      7298.40 300.00
TOTALS    2728697   9095.6 |    142.12 Rand procs= 20 read= 70% bs= 16KB
root@aix1:/tmp #
```

Once the ndisk testing has been completed, if it is possible to check the storage system to compare the results, and knowing the workload you generated was similar to the workload on the storage, it is useful to validate the test results.

Figure 5-2 shows the statistics displayed on our storage system, which in this case is an IBM Storwize V7000 storage system.

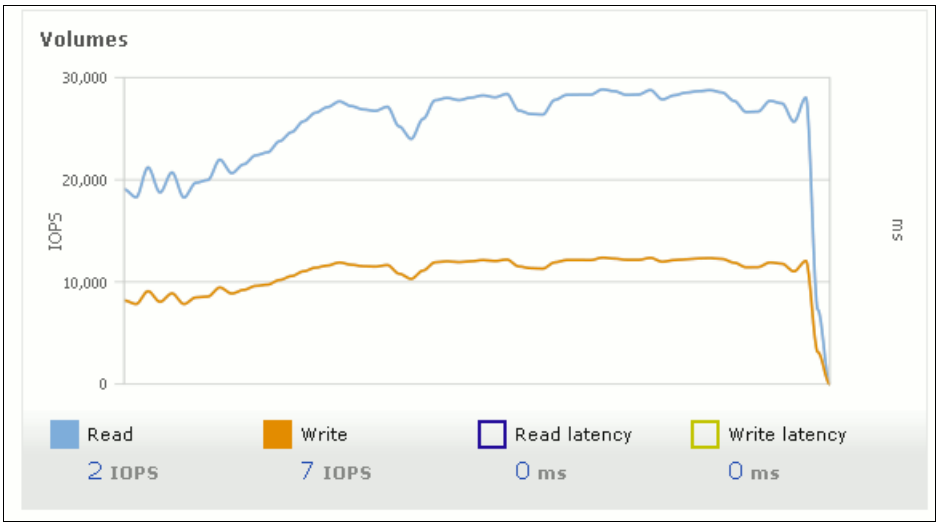


Figure 5-2 V7000 volume statistics

Note: 5.6, “Disk storage bottleneck identification” on page 251 describes how to interpret the performance data collected during testing activities.

It is also important to recognize that disk storage technology is evolving. With the introduction of solid state drives (SSD), new technologies have been adopted by most storage vendors, such as automated tiering. An example of this is the Easy Tier® technology used in IBM storage products such as IBM SAN Volume Controller, IBM DS8000 and IBM Storwize V7000.

Automated tiering monitors a workload over a period of time, and moves blocks of data in and out of SSD based on how frequently accessed they are. For example, if you run a test for 48 hours, and during that time the automated tiering starts moving blocks into SSD, the test results may vary. So it is important to consult your storage administrator on the storage system’s capabilities as part of the testing process.

5.3.4 Testing the network

Performing network tests on the environment is simpler than the other tests. From the operating system point of view, there is not much to be tested. Although some tuning can be performed on both AIX and Virtual I/O Server layers, for example, the information to be analyzed is more simple. However, when talking about networks, you should always consider all the infrastructure that may affect the final performance of the environment. Eventually you may find that the systems themselves are OK but some other network component, such as a switch, firewall, or router, is affecting the performance of the network.

Latency

Latency can be defined as the time taken to transmit a packet between two points. For the sake of tests, you can also define latency as the time taken for a packet to be transmitted and received between two points (round trip).

Testing the latency is quite simple. In the next examples, we used tools such as **tcpdump** and **ping** to test the latency of our infrastructure, and a shell script to filter data and calculate the mean latency (Example 5-14).

Example 5-14 latency.sh - script to calculate the mean network latency

```
#!/usr/bin/ksh

IFACE=en0
ADDR=10.52.78.9
FILE=/tmp/tcpdump.icmp.${IFACE}.tmp

# number of ICMP echo-request packets to send
PING_COUNT=10

# interval between each echo-request
PING_INTERVAL=10

# ICMP echo-request packet size
PING_SIZE=1

# do not change this. number of packets to be monitored by tcpdump before
# exitting. always PING_COUNT x 2
TCPDUMP_COUNT=$(expr "${PING_COUNT}*2")

tcpdump -l -i ${IFACE} -c ${TCPDUMP_COUNT} "host ${ADDR} and (icmp[icmptype] ==
icmp-echo or icmp[icmptype] == icmp-echoreply)" > ${FILE} 2>&1 &
ping -c ${PING_COUNT} -i ${PING_INTERVAL} -s ${PING_SIZE} ${ADDR} 2>&1

MEANTIME=$(cat ${FILE} | awk -F "[. ]" 'BEGIN { printf("scale=2;("); } { if(/ICMP
echo request/) { REQ=$2; getline; REP=$2; printf("(%d-%d)+", REP, REQ); } } END {
printf("0)/1000/10\n"); }' | bc)

echo "Latency is ${MEANTIME}ms"
```

The script in Example 5-14 has a few parameters that can be changed to test the latency. This script can be changed to accept some command line arguments instead of having to change it every time.

Basically the script monitors the ICMP echo-request and echo-reply traffic while performing some ping with small packet sizes, and calculate the mean round-trip time from a set of samples.

Example 5-15 latency.sh - script output

```
# ksh latency.sh
PING 10.52.78.9 (10.52.78.9): 4 data bytes
12 bytes from 10.52.78.9: icmp_seq=0 ttl=255
12 bytes from 10.52.78.9: icmp_seq=1 ttl=255
12 bytes from 10.52.78.9: icmp_seq=2 ttl=255
12 bytes from 10.52.78.9: icmp_seq=3 ttl=255
12 bytes from 10.52.78.9: icmp_seq=4 ttl=255
12 bytes from 10.52.78.9: icmp_seq=5 ttl=255
12 bytes from 10.52.78.9: icmp_seq=6 ttl=255
12 bytes from 10.52.78.9: icmp_seq=7 ttl=255
12 bytes from 10.52.78.9: icmp_seq=8 ttl=255
```

```
12 bytes from 10.52.78.9: icmp_seq=9 ttl=255

--- 10.52.78.9 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
Latency is .13ms
```

Example 5-15 on page 224 shows the output of the **latency.sh** script containing the mean latency time of 0.13 ms. This test has been run between two servers connected on the same subnet sharing the same Virtual I/O server.

In Example 5-16, we used the **tcpdump** output to calculate the latency. The script filters each pair of requests and reply packets, extracts the timing portion required to calculate each packet latency, and finally sums all latencies and divides by the number of packets transmitted to get the mean latency.

Example 5-16 latency.sh - tcpdump information

```
# cat tcpdump.icmp.en0.tmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type 1, capture size 96 bytes
15:18:13.994500 IP p750s1aix5 > nimres1: ICMP echo request, id 46, seq 1, length 12
15:18:13.994749 IP nimres1 > p750s1aix5: ICMP echo reply, id 46, seq 1, length 12
15:18:23.994590 IP p750s1aix5 > nimres1: ICMP echo request, id 46, seq 2, length 12
15:18:23.994896 IP nimres1 > p750s1aix5: ICMP echo reply, id 46, seq 2, length 12
15:18:33.994672 IP p750s1aix5 > nimres1: ICMP echo request, id 46, seq 3, length 12
15:18:33.994918 IP nimres1 > p750s1aix5: ICMP echo reply, id 46, seq 3, length 12
15:18:43.994763 IP p750s1aix5 > nimres1: ICMP echo request, id 46, seq 4, length 12
15:18:43.995063 IP nimres1 > p750s1aix5: ICMP echo reply, id 46, seq 4, length 12
15:18:53.994853 IP p750s1aix5 > nimres1: ICMP echo request, id 46, seq 5, length 12
15:18:53.995092 IP nimres1 > p750s1aix5: ICMP echo reply, id 46, seq 5, length 12
7508 packets received by filter
0 packets dropped by kernel
```

Latency times depend mostly on the network infrastructure complexity. This information can be useful if you are preparing the environment for applications that transmit a lot of small packets and demand low network latency.

Transmission tests - TCP_RR

Request and Response tests measure the number of transactions (basic connect and disconnect) that your servers and network infrastructure are able to handle. These tests were performed with the **netperf** tool (Example 5-17).

Example 5-17 netperf - TCP_RR test

```
# ./netperf -t TCP_RR -H 10.52.78.47
Netperf version 5.3.7.5 Jul 23 2009 16:57:35
TCP REQUEST/RESPONSE TEST: 10.52.78.47
        (+/-5.0% with 99% confidence) - Version: 5.3.7.5 Jul 23 2009 16:57:41
Local /Remote                                 ----------------
Socket Size   Request  Resp.   Elapsed        Response Time
Send   Recv   Size     Size    Time (iter)    ------- --------
bytes  Bytes  bytes    bytes   secs.          TRs/sec millisec*host

262088 262088    100    200     4.00(03)      3646.77     0.27
262088 262088
```

Transmission tests - TCP_STREAM

These tests attempt to send the most data possible from one side to another in a certain period and give the total throughput of the network. These tests were performed with the **netperf** tool (Example 5-18).

Example 5-18 netperf - TCP_STREAM test

```
# ./netperf -t TCP_STREAM -H 10.52.78.47
Netperf version 5.3.7.5 Jul 23 2009 16:57:35
TCP STREAM TEST: 10.52.78.47
        (+/-5.0% with 99% confidence) - Version: 5.3.7.5 Jul 23 2009 16:57:41
Recv   Send    Send                   ----------------------
Socket Socket  Message  Elapsed            Throughput
Size   Size    Size     Time (iter)   ----------------------
bytes  bytes   bytes    secs.         10^6bits/s   KBytes/s

262088 262088    100      4.20(03)      286.76    35005.39
```

Several tests other than TCP_STREAM and TCP_RR are available with the **netperf** tool that can be used to test the network. Remember that network traffic also consumes memory and processor time. The **netperf** tool can provide some processor utilization statistics as well, but we suggest that the native operating system tools be used instead.

Tip: The **netperf** tool can be obtained at:

http://www.netperf.org

5.4 Understanding processor utilization

This section provides details regarding processor utilization.

5.4.1 Processor utilization

In past readings, the processor utilization on old single-threaded systems used to be straight forward. Tools such as **topas**, **sar** and **vmstat** provided simple values that would let you know exactly how much processor utilization you had.

With the introduction of multiple technologies in the past years, especially the simultaneous multithreading on POWER5 systems, understanding processor utilization on systems became a much more complex task—first because of different new concepts such as Micro-Partitioning®, Virtual Processors and Entitled Capacity, and second because the inherent complexity of parallel processing on SMT.

Current technologies, for instance, allow a logical partition to go in a few seconds from a single idle logical processor to sixteen fully allocated processes to fulfill a workload demand, triggering several components on hypervisor and hardware levels and in less than a minute, go back to its stationary state.

The POWER7 technology brought important improvements of how processor utilization values are reported, offering more accurate data to systems administrators.

This section focuses on explaining some of the concepts involved in reading the processor utilization values on POWER7 environments and will go through a few well-known commands, explaining some important parameters and how to read them.

5.4.2 POWER7 processor utilization reporting

POWER7 introduces an improved algorithm to report processor utilization. This algorithm is based on calibrated Processor Utilization Resource Register (PURR) compared to PURR that is used for POWER5 and POWER6. The aim of this new algorithm is to provide a better view of how much capacity is used, and how much capacity is still available. Thus you can achieve linear PURR utilization and throughput (TPS) relationships. Clients would benefit from the new algorithm, which emphasizes more on PURR utilization metrics than other targets such as throughput and response time.

Figure 5-3 explains the difference between the POWER5, POWER6, and POWER7 PURR utilization algorithms. On POWER5 and POWER6 systems, when only one of the two SMT hardware threads is busy, the utilization of the processor core is reported 100%. While on POWER7, the utilization of the SMT2 processor core is around 80% in the same situation. Furthermore, when one of the SMT4 hardware thread is busy, the utilization of the SMT4 processor core is around 63%. Also note that the POWER7 utilization algorithm persists even if running in POWER6 mode.

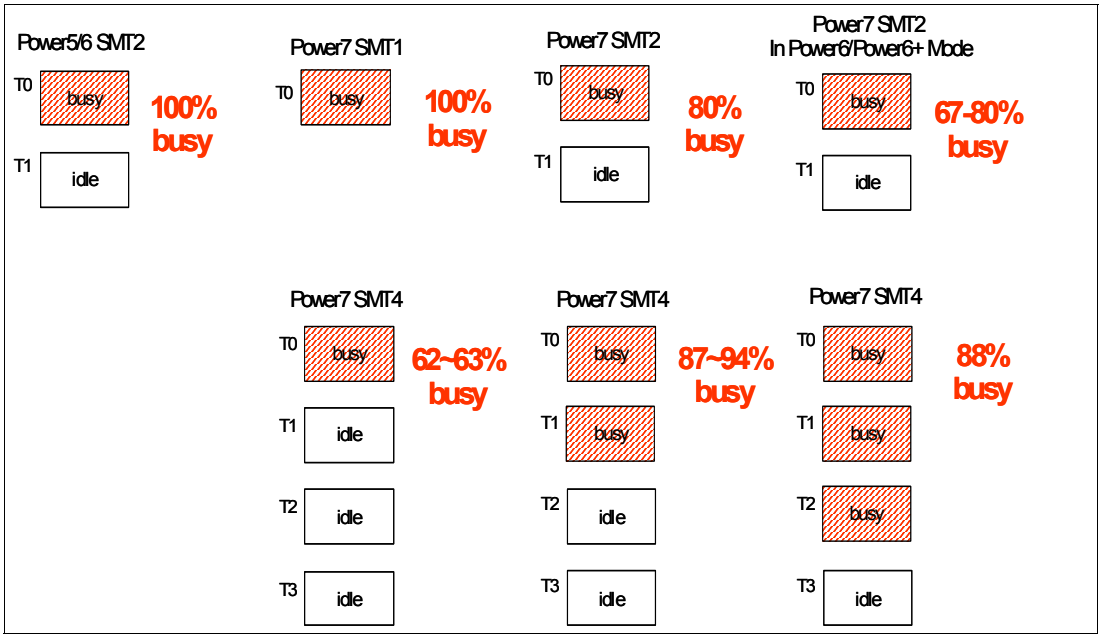


Figure 5-3 POWER7 processor *utilization reporting*

Note: The utilization reporting variance (87~94%) when two threads are busy in SMT4 is due to occasional load balancing to tertiary threads (T2/T3), which is controlled by a number of **`schedo`** options including **`tertiary_barrier_load`**.

The concept of the new improved PURR algorithm is not related to Scaled Process Utilization of Resource Register (SPURR). The latter is a conceptually different technology and is covered in 5.4.5, “Processor utilization reporting in power saving modes” on page 234.

POWER7 processor utilization example - dedicated LPAR

Example 5-19 demonstrates processor utilization when one hardware thread is busy in SMT4 mode. As in the example, the single thread application consumed an entire logical processor (CPU0), but not the entire capacity of the physical core, because there were still three idle hardware threads in the physical core. The physical consumed processor is about 0.62. Because there are two physical processors in the system, the overall processor utilization is 31%.

Example 5-19 Processor *utilization in SMT4 mode on a dedicated LPAR*

```
#sar -P ALL 1 100
AIX p750s1aix2 1 7 00F660114C00    10/02/12
System configuration: lcpu=8  mode=Capped
18:46:06  0      100       0       0       0    0.62
          1        0       0       0     100    0.13
          2        0       0       0     100    0.13
          3        0       0       0     100    0.13
          4        1       1       0      98    0.25
          5        0       0       0     100    0.25
          6        0       0       0     100    0.25
          7        0       0       0     100    0.25
          -       31       0       0      69    1.99
```

Example 5-20 demonstrates processor utilization when one thread is busy in SMT2 mode. In this case, the single thread application consumed more capacity of the physical core (0.80), because there was only one idle hardware thread in the physical core, compared to three idle hardware threads in SMT4 mode in Example 5-19. The overall processor utilization is 40% because there are two physical processors.

Example 5-20 Processor *utilization in SMT2 mode on a dedicated LPAR*

```
#sar -P ALL 1 100
AIX p750s1aix2 1 7 00F660114C00    10/02/12

System configuration: lcpu=4  mode=Capped

18:47:00 cpu    %usr    %sys    %wio   %idle   physc
18:47:01  0      100       0       0       0    0.80
          1        0       0       0     100    0.20
          4        0       1       0      99    0.50
          5        0       0       0     100    0.49
          -       40       0       0      60    1.99
```

Example 5-21 demonstrates processor utilization when one thread is busy in SMT1 mode. Now the single thread application consumed the whole capacity of the physical core, because there is no other idle hardware thread in ST mode. The overall processor utilization is 50% because there are two physical processors.

Example 5-21 Processor *utilization in SMT1 mode on a dedicated LPAR*

```
sar -P ALL 1 100

AIX p750s1aix2 1 7 00F660114C00    10/02/12
System configuration: lcpu=2  mode=Capped
18:47:43 cpu    %usr    %sys    %wio   %idle
18:47:44  0      100       0       0       0
```

```
          4        0        0        0      100
          -       50        0        0       50
```

POWER7 processor utilization example - shared LPAR

Example 5-22 demonstrates processor utilization when one thread is busy in SMT4 mode on a shared LPAR. As in the example, logical processor 4/5/6/7 consumed one physical processor core. Although logical processor 4 is 100% busy, the physical consumed processor (`physc`) is only 0.63. Which means the LPAR received a whole physical core, but is not fully driven by the single thread application. The overall system processor utilization is about 63%. For details aout system processor utilization reporting in a shared LPAR environment, refer to 5.4.6, “A common pitfall of shared LPAR processor utilization” on page 236.

Example 5-22 Processor *utilization in SMT4 mode on a shared LPAR*

```
#sar -P ALL 1 100
AIX p750s1aix2 1 7 00F660114C00    10/02/12
System configuration: lcpu=16 ent=1.00 mode=Uncapped
18:32:58 cpu    %usr    %sys    %wio   %idle   physc   %entc
18:32:59  0       24      61       0      15    0.01     0.8
          1        0       3       0      97    0.00     0.2
          2        0       2       0      98    0.00     0.2
          3        0       2       0      98    0.00     0.3
          4      100       0       0       0    0.63    62.6
          5        0       0       0     100    0.12    12.4
          6        0       0       0     100    0.12    12.4
          7        0       0       0     100    0.12    12.4
          8        0      52       0      48    0.00     0.0
          12       0      57       0      43    0.00     0.0
          -       62       1       0      38    1.01   101.5
```

Example 5-23 demonstrates processor utilization when one thread is busy in SMT2 mode on a shared LPAR. Logical processor 4/5 consumed one physical processor core. Although logical processor 4 is 100% busy, the physical consumed processor is only 0.80, which means the physical core is still not fully driven by the single thread application.

Example 5-23 Processor *utilization in SMT2 mode on a shared LPAR*

```
#sar -P ALL 1 100
AIX p750s1aix2 1 7 00F660114C00    10/02/12
System configuration: lcpu=8 ent=1.00 mode=Uncapped
18:35:13 cpu    %usr    %sys    %wio   %idle   physc   %entc
18:35:14  0       20      62       0      18    0.01     1.2
          1        0       2       0      98    0.00     0.5
          4      100       0       0       0    0.80    80.0
          5        0       0       0     100    0.20    19.9
          8        0      29       0      71    0.00     0.0
          9        0       7       0      93    0.00     0.0
          12       0      52       0      48    0.00     0.0
          13       0       0       0     100    0.00     0.0
          -       79       1       0      20    1.02   101.6
```

Example 5-24 on page 230 demonstrates processor utilization when one thread is busy in SMT1 mode on a shared LPAR. Logical processor 4 is 100% busy, and fully consumed one physical processor core. That is because there is only one hardware thread for each core, and thus there is no idle hardware thread available.

Example 5-24 Processor *utilization in SMT1 mode on a shared LPAR*

```
#sar -P ALL 1 100
AIX p750s1aix2 1 7 00F660114C00    10/02/12
System configuration: lcpu=4 ent=1.00 mode=Uncapped
18:36:10 cpu    %usr    %sys    %wio   %idle   physc   %entc
18:36:11  0       12      73       0      15    0.02     1.6
          4      100       0       0       0    1.00    99.9
          8       26      53       0      20    0.00     0.2
          12       0      50       0      50    0.00     0.1
          -       98       1       0       0    1.02   101.7
```

Note: The ratio is acquired using the **ncpu** tool. The result might vary slightly under different workloads.

5.4.3 Small workload example

To illustrate some of the various types of information, we created a simplistic example by putting a tiny workload on a free partition. The system is running a process called *cputest* that puts a very small workload, as shown in Figure 5-4.

```
Topas Monitor for host:p750s1aix5                EVENTS/QUEUES    FILE/TTY
Wed Oct  3 13:48:31 2012   Interval:2            Cswitch     200  Readch     2466
                                                 Syscall     206  Writech     759
CPU      User% Kern% Wait% Idle%   Physc  Entc%  Reads        22  Rawin         0
Total     3.2   0.2   0.0  96.5    0.06   5.70  Writes        0  Ttyout      367
                                                 Forks         0  Igets         0
Network    BPS  I-Pkts  O-Pkts    B-In   B-Out  Execs         0  Namei         7
Total    11.8K    9.50    1.50   9.99K   1.83K  Runqueue   1.00  Dirblk        0
                                                 Waitqueue   0.0
Disk    Busy%      BPS     TPS  B-Read  B-Writ                    MEMORY
Total     0.0        0       0       0       0  PAGING           Real,MB    8192
                                                 Faults        0  % Comp       18
FileSystem           BPS     TPS  B-Read  B-Writ  Steals        0  % Noncomp     2
Total              2.41K   22.50   2.41K       0  PgspIn        0  % Client      2
                                                 PgspOut       0
Name           PID  CPU%  PgSp Owner             PageIn        0  PAGING SPACE
cputest     7340162  3.2  136K root              PageOut       0  Size,MB     512
topas       8192010  0.1 1.98M root              Sios          0  % Used        2
xmtopasa    7864322  0.0  988K root                              % Free       98
```

Figure 5-4 single process - Topas simplified processor *statistics*

In the processor statistics, the graphic shows a total of 3.2% of utilization at the User% column. In the process table you can see that the *cputest* is consuming 3.2% of the processor on the machine as well, which seems accurate according to our previous read.

Note: The information displayed in the processor statistics is not intended to match any specific processes. The fact that it matches the utilization of *cputest* is just because the system does not have any other workload.

There are a few important details shown in Figure 5-4 on page 230:

- Columns User%, Kern%, and Wait%

  The column User% refers to the percentage of processor time spent running user-space processes. The Kern% refers to the time spent by the processor in kernel mode, and Wait% is the time spent by the processor waiting for some blocking event, like an I/O operation. This indicator is mostly used to identify storage subsystem problems.

  These three values together form your system utilization. Which one is larger or smaller will depend on the type of workload running on the system.

- Column Idle%

  Idle is the percent of time that the processor spends doing nothing. In production environments, having long periods of Idle% may indicate that the system is oversized and that it is not using all its resources. On the other hand, a system near to 0% idle all the time can be an alert that your system is undersized.

  There are no rules of thumb when defining what is a desired idle state. While some prefer to use as much of the system resources as possible, others prefer to have a lower resource utilization. It all depends on the users' requirements.

  For sizing purposes, the idle time is only meaningful when measured for long periods.

  Note: Predictable workload increases are easier to manage than unpredictable situations. For the first, a well-sized environment is usually fine while for the latter, some spare resources are usually the best idea.

- Column Physc

  This is the quantity of physical processors currently consumed. Figure 5-4 on page 230 shows Physc at 0.06 or 6% of physical processor utilization.

- Column Entc%

  This is the percentage of the entitled capacity consumed. This field should always be analyzed when dealing with processor utilization because it gives a good idea about the sizing of the partition.

A partition that shows the Entc% always too low or always too high (beyond the 100%) is an indication that its sizing must be reviewed. This topic is discussed in 3.1, "Optimal logical partition (LPAR) sizing" on page 42.

Figure 5-5 on page 232 shows detailed statistics for the processor. Notice that the reported values this time are a bit different.

```
Topas Monitor for host:p750s1aix5                EVENTS/QUEUES    FILE/TTY
Wed Oct  3 13:49:55 2012   Interval:2            Cswitch      235  Readch      2411
                                                 Syscall      209  Writech      648
CPU      User% Kern% Wait% Idle%   Physc        Reads         22  Rawin          0
0         90.9   6.8   0.0   2.4    0.04        Writes         1  Ttyout       312
2          0.0   0.7   0.0  99.3    0.01        Forks          0  Igets          0
3          0.0   0.7   0.0  99.3    0.01        Execs          0  Namei          8
4          0.0  44.7   0.0  55.3    0.00        Runqueue       0  Dirblk         0
5          0.0  33.7   0.0  66.3    0.00        Waitqueue    0.0
6          0.0  37.7   0.0  62.3    0.00                         MEMORY
                                                 PAGING          Real,MB     8192
Network     BPS  I-Pkts  O-Pkts    B-In   B-Out  Faults         0  % Comp       18
Total    11.9K   11.00    2.00   10.1K   1.80K  Steals         0  % Noncomp     2
                                                 PgspIn         0  % Client      2
Disk    Busy%      BPS      TPS  B-Read  B-Writ  PgspOut        0
Total     0.0        0        0       0       0  PageIn         0  PAGING SPACE
                                                 PageOut        0  Size,MB      512
FileSystem          BPS      TPS  B-Read  B-Writ  Sios           0  % Used       2
Total              2.35K   22.50   2.35K       0                    % Free      98
                                                 NFS (calls/sec)
Name            PID  CPU%  PgSp Owner           SerV2          0  WPAR Activ     0
cputest     7340162  3.2  136K root             CliV2          0  WPAR Total     0
topas       8192010  0.1 1.98M root             SerV3          0  Press: "h"-help
xmgc        1048608  0.1  448K root             CliV3          0         "q"-quit
```

Figure 5-5 Single process - Topas detailed processor *statistics*

Notice that **topas** reports CPU0 running at 90.9% in the User% column and only 2.4% in the Idle% column. Also, the Physc values are now spread across CPU0 (0.04), CPU2 (0.01), and CPU3 (0.01), but the sum of the three logical processors still matches the values of the simplified view.

In these examples, it is safe to say that **cputest** is consuming only 3.2% of the total entitled capacity of the machine.

In an SMT-enabled partition, the SMT distribution over the available cores can also be checked with the **mpstat -s** command, as shown in Figure 5-6.

```
System configuration: lcpu=16 ent=1.0 mode=Uncapped

               Proc0                Proc4                Proc8                Proc12
                5.55%                0.02%                0.00%                0.05%
cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 3.44%   0.70%   0.70%   0.70%   0.01%   0.00%   0.00%   0.00%   0.00%   0.00%   0.00%   0.00%   0.03%   0.01%   0.01%   0.01%
-------------------------------------------------------------------------------------------------------------------------------
               Proc0                Proc4                Proc8                Proc12
                5.86%                0.04%                0.01%                0.08%
cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 3.65%   0.74%   0.74%   0.74%   0.01%   0.01%   0.01%   0.01%   0.00%   0.00%   0.00%   0.00%   0.03%   0.02%   0.02%   0.02%
-------------------------------------------------------------------------------------------------------------------------------
               Proc0                Proc4                Proc8                Proc12
                5.49%                0.02%                0.00%                0.05%
cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 3.42%   0.69%   0.69%   0.69%   0.01%   0.00%   0.00%   0.00%   0.00%   0.00%   0.00%   0.00%   0.03%   0.01%   0.01%   0.01%
-------------------------------------------------------------------------------------------------------------------------------
               Proc0                Proc4                Proc8                Proc12
                5.88%                0.04%                0.01%                0.07%
cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 3.65%   0.74%   0.74%   0.74%   0.01%   0.01%   0.01%   0.01%   0.00%   0.00%   0.00%   0.00%   0.03%   0.02%   0.02%   0.02%
-------------------------------------------------------------------------------------------------------------------------------
               Proc0                Proc4                Proc8                Proc12
                5.55%                0.02%                0.00%                0.04%
cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 3.45%   0.70%   0.70%   0.70%   0.01%   0.00%   0.00%   0.00%   0.00%   0.00%   0.00%   0.00%   0.02%   0.01%   0.01%   0.01%
#
```

Figure 5-6 mpstat -s reporting a small load on cpu0 and using 5.55% of our entitled capacity

The **mpstat -s** command gives information about the physical processors (Proc0, Proc4, Proc8, and Proc12) and each of the logical processors (cpu0 through cpu15). Figure 5-6 on page 232 shows five different readings of our system processor while **cputest** was running.

Notes:

- The default behavior of **mpstat** is to present the results in 80 columns, thus wrapping the lines if you have a lot of processors. The flag **-w** can be used to display wide lines.
- The additional sections provide some information about SMT systems, focusing on the recent POWER7 SMT4 improvements.

5.4.4 Heavy workload example

With the basic processor utilization concepts illustrated, we now take a look at a heavier workload and see how the processor reports changed.

The next examples provide reports of an eight-processes workload with intensive processors.

In Figure 5-7 User% is now reporting almost 90% of processor utilization, but that information itself does not tell much. Physc and Entc% are now reporting much higher values, indicating that the partition is using more of its entitled capacity.

```
Topas Monitor for host:p750s1aix5                EVENTS/QUEUES    FILE/TTY
Thu Oct  4 16:38:27 2012   Interval:2            Cswitch     229  Readch     2392
                                                 Syscall     232  Writech     630
CPU     User% Kern% Wait% Idle%   Physc  Entc%  Reads        23  Rawin         0
Total    89.6   0.2   0.0  10.2    3.01 301.28  Writes        2  Ttyout      294
```

Figure 5-7 Topas simplified processor *statistics - Eight simultaneous processes running*

Looking at the detailed processor statistics (Figure 5-8), you can see that the physical processor utilization is still spread across the logical processors of the system, and the sum would approximately match the value seen in the simplified view in Figure 5-7.

```
Topas Monitor for host:p750s1aix5                EVENTS/QUEUES    FILE/TTY
Thu Oct  4 16:36:57 2012   Interval:2            Cswitch     236  Readch     2455
                                                 Syscall     239  Writech     749
CPU     User% Kern% Wait% Idle%   Physc         Reads        22  Rawin         0
13      100.0   0.0   0.0   0.0    0.47         Writes        1  Ttyout      357
10      100.0   0.0   0.0   0.0    0.33         Forks         0  Igets         0
6       100.0   0.0   0.0   0.0    0.33         Execs         0  Namei         9
5       100.0   0.0   0.0   0.0    0.28         Runqueue   9.00  Dirblk        0
12      100.0   0.0   0.0   0.0    0.47         Waitqueue  0.0
8       100.0   0.0   0.0   0.0    0.28                        MEMORY
9       100.0   0.0   0.0   0.0    0.28         PAGING         Real,MB    8192
4       100.0   0.0   0.0   0.0    0.28         Faults        0  % Comp      19
0        11.2  62.8   0.0  26.0    0.01         Steals        0  % Noncomp    2
3         2.9   6.9   0.0  90.2    0.00         PgspIn        0  % Client     2
1         1.6  13.8   0.0  84.7    0.00         PgspOut       0
2         1.2   2.8   0.0  96.0    0.00         PageIn        0  PAGING SPACE
14        0.0   0.1   0.0  99.9    0.03         PageOut       0  Size,MB      512
11        0.0   0.0   0.0 100.0    0.12         Sios          0  % Used       2
15        0.0   0.1   0.0  99.9    0.03                        % Free      98
7         0.0   0.0   0.0 100.0    0.12         NFS (calls/sec)
```

Figure 5-8 Topas detailed processor *statistics - Eight simultaneous processes running*

The thread distribution can be seen in Figure 5-9. This partition is an SMT4 partition, and therefore the system tries to distribute the processes as best as possible over the logical processors.

```
# mpstat -ws 5 3

System configuration: lcpu=16 ent=1.0 mode=Uncapped

            Proc0                   Proc4                   Proc8                   Proc12
            99.92%                  99.91%                  100.14%                   0.86%
 cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 27.69%  27.72%  12.02%  32.50%  24.99%  24.98%  24.97%  24.97%  63.02%  12.28%  12.50%  12.34%   0.39%   0.20%   0.13%   0.13%
--------------------------------------------------------------------------------------------------------------------------------
            Proc0                   Proc4                   Proc8                   Proc12
           100.05%                  99.94%                  100.15%                   0.22%
 cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 27.70%  27.72%  12.13%  32.50%  25.00%  24.99%  24.98%  24.98%  63.03%  12.49%  12.31%  12.33%   0.08%   0.06%   0.05%   0.04%
--------------------------------------------------------------------------------------------------------------------------------
            Proc0                   Proc4                   Proc8                   Proc12
            99.80%                  99.91%                  99.90%                    0.41%
 cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 27.68%  27.71%  11.91%  32.50%  24.99%  24.98%  24.97%  24.97%  62.98%  12.34%  12.27%  12.31%   0.20%   0.07%   0.07%   0.07%
```

Figure 5-9 mpstat threads view - Eight simultaneous processes running

For the sake of curiosity, Figure 5-10 shows a load of nine processes distributed across only three virtual processors. The interesting detail in this figure is that it illustrates the efforts of the system to make the best use of the SMT4 design by allocating all logical processors of Proc0, Proc4 and Proc2 while Proc6 is almost entirely free.

```
            Proc0                   Proc4                   Proc2                   Proc6
            99.92%                  99.92%                  99.92%                    0.61%
 cpu0    cpu1    cpu2    cpu3    cpu4    cpu5    cpu6    cpu7    cpu8    cpu9    cpu10   cpu11   cpu12   cpu13   cpu14   cpu15
 27.70%  27.71%  12.02%  32.49%  24.99%  24.98%  24.97%  24.97%  47.05%  47.04%   2.90%   2.93%   0.30%   0.13%   0.09%   0.09%
```

Figure 5-10 mpstat threads view - Nine simultaneous processes running

5.4.5 Processor utilization reporting in power saving modes

This section shows processor utilization reporting in power saving mode.

Concepts

Before POWER5, AIX calculated processor utilization based on decrementer sampling which is active every tick (10ms). The tick is charged to `user/sys/idle/wait` buckets, depending on the execution mode when the clock interrupt happens. It is a pure software approach based on the operating system, and not suitable when shared LPAR and SMT are introduced since the physical core is no longer dedicated to one hardware thread.

Since POWER5, IBM introduced Processor Utilization Resource Registers (PURR) for processor utilization accounting. Each processor has one PURR for each hardware thread, and the PURR is counted by hypervisor in fine-grained time slices at nanosecond magnitude. Thus it is more accurate than decrementer sampling, and successfully addresses the utilization reporting issue in SMT and shared LPAR environments.

Since POWER6, IBM introduced power saving features, that is, the processor frequency might vary according to different power saving policies. For example, in static power saving mode, the processor frequency will be at a fixed value lower than nominal; in dynamic power saving mode, the processor frequency can vary dynamically according to the workload, and can reach a value larger than nominal (over-clocking).

Because PURR increments independent of processor frequency, each PURR tick does not necessarily represent the same capacity if you set a specific power saving policy other than

the default. To address this problem, POWER6 and later chips introduced the Scaled PURR, which is always proportional to process frequency. When running at lower frequency, the SPURR ticks less than PURR, and when running at higher frequency, the SPURR ticks more than PURR. We can also use SPURR together with PURR to calculate the real operating frequency, as in the equation:

operating frequency = (SPURR/PURR) * nominal frequency

There are several monitoring tools based on SPURR, which can be used to get an accurate utilization of LPARs when in power saving mode. We introduce these tools in the following sections.

Monitor tools

Example 5-25 shows an approach to observe the current power saving policy. You can see that LPAR A is in static power saving mode while LPAR B is in dynamic power saving (favoring performance) mode.

Example 5-25 lparstat -i to observe the power saving policy of an LPAR

```
LPAR A:
#lparstat -i
…
Power Saving Mode                          : Static Power Saving

LPAR B:
#lparstat –i
…
Power Saving Mode                          : Dynamic Power Savings (Favor
Performance)
```

Example 5-26 shows how the processor operating frequency is shown in the **lparstat** output. There is an extra %nsp column indicating the current ratio compared to nominal processor speed, if the processor is not running at the nominal frequency.

Example 5-26 %nsp in lparstat

```
#lparstat 1
System configuration: type=Dedicated mode=Capped smt=4 lcpu=32 mem=32768MB

%user  %sys  %wait  %idle  %nsp
----- ----- ------ ------ -----
 76.7  14.5    5.6    3.1    69
 80.0  13.5    4.4    2.1    69
 76.7  14.3    5.8    3.2    69
 65.2  14.5   13.2    7.1    69
 62.6  15.1   14.1    8.1    69
 64.0  14.1   13.9    8.0    69
 65.0  15.2   12.6    7.2    69
```

Note: If %nsp takes a fixed value lower than 100%, it usually means you have enabled static power saving mode. This might not be what you want, because static power saving mode cannot fully utilize the processor resources despite the workload.

%nsp can also be larger than 100 if the processor is over-clocking in dynamic power saving modes.

Example 5-27 shows another **lparstat** option, **-E**, for observing the real processor utilization ratio in various power saving modes. As in the output, the actual metrics are based on PURR, while the normalized metrics are based on SPURR. The normalized metrics represent what the real capacity would be if all processors were running at nominal frequency. The sum of user/sys/wait/idle in normalized metrics can exceed the real capacity in case of over-clocking.

Example 5-27 lparstat -E

```
#lparstat -E 1 100

System configuration: type=Dedicated mode=Capped smt=4 lcpu=64 mem=65536MB
Power=Dynamic-Performance

Physical Processor Utilisation:

 --------Actual--------                ------Normalised------
 user   sys  wait  idle      freq     user   sys  wait  idle
 ----  ----  ----  ----   ---------  ----  ----  ----  ----
15.99 0.013 0.000 0.000 3.9GHz[102%] 16.24 0.014 0.000 0.000
15.99 0.013 0.000 0.000 3.9GHz[102%] 16.24 0.013 0.000 0.000
15.99 0.009 0.000 0.000 3.9GHz[102%] 16.24 0.009 0.000 0.000
```

Note: AIX introduces **lparstat** options **-E** and **-Ew** since AIX 5.3 TL9, AIX 6.1 TL2, and AIX 7.1

Refer to IBM EnergyScale for POWER7 Processor-Based Systems at:
http://public.dhe.ibm.com/common/ssi/ecm/en/pow03039usen/POW03039USEN.PDF

5.4.6 A common pitfall of shared LPAR processor utilization

For dedicated LPAR, the processor utilization reporting uses the same approach as in no virtualization environment. However, the situation is more complicated in shared LPAR situations. For shared LPAR, if the consumed processor is less than entitlement, the system processor utilization ratio uses the processor entitlement as the base.

As in Example 5-28, %user, %sys, %wait, and %idle are calculated based on the entitled processor, which is 1.00. Thus, 54% user percentage actually means that 0.54 physical processor is consumed in user mode, not 0.54 * 0.86 (physc).

Example 5-28 Processor utilization reporting when consumed processors is less than entitlement

```
#lparstat 5 3
System configuration: type=Shared mode=Uncapped smt=4 lcpu=16 mem=8192MB psize=16
ent=1.00
%user  %sys  %wait  %idle physc %entc  lbusy  vcsw phint
----- ----- ------ ------ ----- ----- ------ ----- -----
 54.1   0.4    0.0   45.5  0.86  86.0    7.3   338     0
 54.0   0.3    0.0   45.7  0.86  85.7    6.8   311     0
 54.0   0.3    0.0   45.7  0.86  85.6    7.2   295     0
```

If the consumed processor is larger than entitlement, the system processor utilization ratio uses the consumed processor as the base. Refer to Example 5-29 on page 237. In this case, %usr, %sys, %wait, and %idle are calculated based on the consumed processor. Thus 62.2% user percentage actually means that 2.01*0.622 processor is consumed in user mode.

Example 5-29 Processor utilization reporting when consumed processors exceeds entitlement

```
#lparstat 5 3
System configuration: type=Shared mode=Uncapped smt=4 lcpu=16 mem=8192MB psize=16
ent=1.00
%user  %sys  %wait  %idle physc %entc  lbusy  vcsw phint
----- ----- ------ ------ ----- ----- ------ ----- -----
 62.3   0.2    0.0   37.6  2.00 200.3   13.5   430     0
 62.2   0.2    0.0   37.6  2.01 200.8   12.7   569     0
 62.2   0.2    0.0   37.7  2.01 200.7   13.4   550     0
```

Note: The rule above applies to overall system processor utilization reporting. The specific logical processor utilization ratios in **`sar -P ALL`** and **`mpstat -a`** are always based on their physical consumed processors. However, the overall processor utilization reporting in these tools still complies with the rule.

5.5 Memory utilization

This section covers a suggested approach of looking at memory usage, how to read the metrics correctly and how to understand paging space utilization. It shows how to monitor memory in partitions with dedicated memory, active memory sharing, and active memory expansion. It also presents some information about memory leaks and memory size simulation.

5.5.1 How much memory is free (dedicated memory partitions)

In AIX, memory requests are managed by the Virtual Memory Manager (VMM). Virtual memory includes real physical memory (RAM) and memory stored on disk (paging space).

Virtual memory segments are partitioned into fixed-size units called pages. AIX supports four page sizes: 4 KB, 64 KB, 16 MB and 16 GB. The default page size is 4 KB. When free memory becomes low, VMM uses the Last Recently Used (LRU) algorithm to replace less frequently referenced memory pages to paging space. To optimize which pages are candidates for replacement, AIX classifies them into two types:

- Computational memory
- Non-computational memory

Computational memory, also known as computational pages, consists of the pages that belong to working-storage segments or program text (executable files) segments. Non-computational memory or file memory is usually pages from permanent data files in persistent storage.

AIX tends to use all of the physical memory available. Depending on how you look at your memory utilization, you may think you need more memory.

In Example 5-30, the `fre` column shows 8049 pages of 4 KB of free memory = 31 MB and the LPAR has 8192 MB. Apparently, the system has almost no free memory.

Example 5-30 vmstat shows there is almost no free memory

```
# vmstat

System configuration: lcpu=16 mem=8192MB ent=1.00
```

```
kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 1  1 399550  8049   0   0   0 434  468   0  68 3487 412  0  0 99  0  0.00   0.0
```

Using the command **dd if=/dev/zero of=/tmp/bigfile bs=1M count=8192**, we generated a file of the size of our RAM memory (8192 MB). The output of **vmstat** in Example 5-31 presents 6867 frames of 4 k free memory = 26 MB.

Example 5-31 vmstat still shows almost no free memory

```
kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 1  1 399538  6867   0   0   0 475  503   0  60 2484 386  0  0 99  0  0.000   0.0
```

Looking at the memory report of **topas**, Example 5-32, you see that the non-computational memory, represented by Noncomp, is 80%.

Example 5-32 Topas shows non-computational memory at 80%

```
Topas Monitor for host:p750s2aix4              EVENTS/QUEUES    FILE/TTY
Thu Oct 11 18:46:27 2012   Interval:2          Cswitch     271  Readch     3288
                                               Syscall     229  Writech     380
CPU     User% Kern% Wait% Idle%   Physc  Entc% Reads        38  Rawin         0
Total     0.1   0.3   0.0  99.6    0.01   0.88 Writes        0  Ttyout      178
                                               Forks         0  Igets         0
Network    BPS  I-Pkts  O-Pkts    B-In   B-Out  Execs         0  Namei        23
Total    1.01K    9.00    2.00   705.0   330.0  Runqueue   0.50  Dirblk        0
                                               Waitqueue   0.0
Disk    Busy%      BPS     TPS  B-Read  B-Writ                   MEMORY
Total     0.0        0       0       0       0  PAGING           Real,MB    8192
                                               Faults        0  % Comp       19
FileSystem          BPS    TPS  B-Read  B-Writ  Steals        0  % Noncomp    80
Total              3.21K  38.50   3.21K       0  PgspIn        0  % Client     80
                                               PgspOut       0
Name           PID  CPU%  PgSp Owner           PageIn        0  PAGING SPACE
topas       5701752  0.1 2.48M root            PageOut       0  Size,MB    2560
java        4456586  0.1 20.7M root            Sios          0  % Used        0
getty       5308512  0.0  640K root                             % Free      100
gil         2162754  0.0  960K root            NFS (calls/sec)
slp_srvr    4915352  0.0  472K root            SerV2         0  WPAR Activ    0
java        7536870  0.0 55.7M pconsole        CliV2         0  WPAR Total    1
pcmsrv      8323232  0.0 1.16M root            SerV3         0  Press: "h"-help
java        6095020  0.0 64.8M root            CliV3         0         "q"-quit
```

After using the command **rm /tmp/bigfile**, we saw that the **vmstat** output, shown in Example 5-33, shows 1690510 frames of 4 k free memory = 6603 MB.

Example 5-33 vmstat shows a lot of free memory

```
kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
r  b   avm   fre    re  pi   po  fr   sr  cy  in  sy  cs  us sy id wa    pc    ec
```

```
1  1 401118 1690510 0   0    0  263  279   0  35 1606 329   0   0 99  0  0.000    0.0
```

What happened with the memory after we issued the **rm** command? Remember non-computational memory? It is basically file system cache. Our **dd** filled the non-computational memory and **rm** wipes that cache from noncomp memory.

AIX VMM keeps a *free list*—real memory pages that are available to be allocated. When process requests memory and there are not sufficient pages in the free list, AIX first removes pages from non-computational memory.

Many monitoring tools present the utilized memory without discounting non-computational memory. This leads to potential misunderstanding of statistics and incorrect assumptions about how much of memory is actually free. In most cases it should be possible to make adjustments to give the right value.

In order to know the utilized memory, the correct column to look at, when using **vmstat**, is active virtual memory (*avm*). This value is also presented in 4 KB pages. In Example 5-30 on page 237, while the `fre` column of **vmstat** shows 8049 frames (31 MB), the `avm` is 399,550 pages (1560 MB). For 1560 MB used out of 8192 MB of the total memory of the LPAR, there are 6632 MB free. The `avm` value can be greater than the physical memory, because some pages might be in RAM and others in paging space. If that happens, it is an indication that your workload requires more than the physical memory available.

Let us play more with **dd** and this time analyze memory with **topas**. In Example 5-34, **topas** output shows 1% utilization of Noncomp (non-computational) memory.

Using the **dd** command again:

```
dd if=/dev/zero of=/tmp/bigfile bs=1M count=8192
```

The t**opas** output in Example 5-34 shows that the sum of computational and non-computational memory is 99%, so almost no memory is free. What will happen if you start a program that requests memory? To illustrate this, in Example 5-35, we used the **stress** tool from:

http://www.perzl.org/aix/index.php?n=Mains.Stress

Example 5-34 Topas shows Comp + Noncomp = 99% (parts stripped for better reading)

```
Disk    Busy%      BPS      TPS  B-Read  B-Writ                    MEMORY
Total     0.0        0        0       0       0  PAGING              Real,MB    8192
                                                  Faults         78  % Comp     23
FileSystem          BPS    TPS  B-Read  B-Writ  Steals          0  % Noncomp  76
Total              2.43K  28.50   2.43K       0  PgspIn          0  % Client   76
```

Example 5-35 Starting a program that requires 1024 MB of memory

```
# stress --vm 1 --vm-bytes 1024M --vm-hang 0
stress: info: [11600010] dispatching hogs: 0 cpu, 0 io, 1 vm, 0 hdd
```

In Example 5-36, non-computational memory dropped from 76% (Example 5-34) to 63% and the computational memory increased from 23% to 35%.

Example 5-36 Topas output while running a stress program

```
Disk    Busy%      BPS      TPS  B-Read  B-Writ                    MEMORY
Total     0.0        0        0       0       0  PAGING              Real,MB    8192
                                                  Faults        473  % Comp     35
```

```
FileSystem          BPS     TPS  B-Read  B-Writ  Steals        0  % Noncomp  63
```

After cancelling the stress program, Example 5-37 shows that non-computational memory remains at the same value and the computational returned to the previous mark. This shows that when a program requests computational memory, VMM allocates this memory as computational and releases pages from non-computational.

Example 5-37 Topas after cancelling the program

```
Disk    Busy%      BPS      TPS  B-Read  B-Writ                    MEMORY
Total     0.0        0        0       0       0  PAGING              Real,MB    8192
                                                 Faults       476  % Comp     23
FileSystem          BPS     TPS  B-Read  B-Writ  Steals        0  % Noncomp  63
```

Using **nmon**, in Example 5-38, the sum of the values `Process` and `System` is approximately the value of `Comp`. `Process` is memory utilized by the application process and System is memory utilized by the AIX kernel.

Example 5-38 Using nmon to analyze memory

```
??topas_nmon??b=Black&White??????Host=p750s2aix4?????Refresh=2 secs???18:49.27??
? Memory ??????????????????????????????????????????????????????????????????????
?           Physical  PageSpace |        pages/sec  In     Out | FileSystemCache?
?% Used        89.4%      2.4%  | to Paging Space   0.0    0.0 | (numperm) 64.5%?
?% Free        10.6%     97.6%  | to File System    0.0    0.0 | Process   15.7%?
?MB Used     7325.5MB    12.1MB | Page Scans        0.0        | System     9.2%?
?MB Free      866.5MB   499.9MB | Page Cycles       0.0        | Free      10.6%?
?Total(MB)   8192.0MB   512.0MB | Page Steals       0.0        |           -----?
?                              | Page Faults       0.0        | Total    100.0%?
?-------------------------------------------------------------- | numclient 64.5%?
?Min/Maxperm     229MB(  3%)  6858MB( 90%) <--% of RAM          | maxclient 90.0%?
?Min/Maxfree     960   1088       Total Virtual    8.5GB       | User      73.7%?
?Min/Maxpgahead    2      8    Accessed Virtual    2.0GB 23.2%| Pinned    16.2%?
?                                                              | lruable pages  ?
??????????????????????????????????????????????????????????????????????????????
```

svmon

Another useful tool to see how much memory is free is **svmon**. Since AIX 5.3 TL9 and AIX 6.1 TL2, **svmon** has a new metric called *available memory*, representing the free memory. Example 5-39 shows **svmon** output. The available memory is 5.77 GB.

Example 5-39 svmon output shows available memory

```
# svmon -O summary=basic,unit=auto
Unit: auto
---------------------------------------------------------------------------------------
               size       inuse        free         pin     virtual  available   mmode
memory       8.00G       7.15G     873.36M       1.29G       1.97G      5.77G     Ded
pg space   512.00M       12.0M

               work        pers        clnt       other
pin         796.61M          0K          0K     529.31M
in use        1.96G          0K       5.18G
```

One **svmon** usage is to show the top 10 processes in memory utilization, shown in Example 5-40 on page 241.

Example 5-40 svmon - top 10 memory consuming processes

```
# svmon -Pt10 -O unit=KB
Unit: KB

--------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
 5898490 java            222856    40080        0   194168
 7536820 java            214432    40180        0   179176
 6947038 cimserver       130012    39964        0   129940
 8126526 cimprovagt      112808    39836        0   112704
 8519700 cimlistener     109496    39836        0   109424
 6488292 rmcd            107540    39852        0   106876
 4063360 tier1slp        106912    39824        0   106876
 5636278 rpc.statd       102948    39836        0   102872
 6815958 topasrec        102696    39824        0   100856
 6357198 IBM.DRMd        102152    39912        0   102004
```

Example 5-41 illustrates the **svmon** command showing only **java** processes.

Example 5-41 svmon showing only Java processes

```
# svmon -C java -O unit=KB,process=on
Unit: KB
================================================================================
Command                                Inuse      Pin     Pgsp  Virtual
java                                  236568    39376   106200   312868

--------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual
 7274720 java            191728    38864     9124   200836
 6553820 java             38712      372    74956    88852
 4915426 java              6128      140    22120    23180
```

For additional information, refer to:

aix4admins.blogspot.com/2011/09/vmm-concepts-virtual-memory-segments.html

or

www.ibm.com/developerworks/aix/library/au-vmm/

Example 5-42 Output of the vmstat - v command

```
# vmstat -v
              2097152 memory pages
              1950736 lruable pages
               223445 free pages
                    2 memory pools
               339861 pinned pages
                 90.0 maxpin percentage
                  3.0 minperm percentage
                 90.0 maxperm percentage
                 69.3 numperm percentage
              1352456 file pages
                  0.0 compressed percentage
                    0 compressed pages
                 69.3 numclient percentage
```

```
   90.0 maxclient percentage
1352456 client pages
      0 remote pageouts scheduled
      0 pending disk I/Os blocked with no pbuf
 191413 paging space I/Os blocked with no psbuf
   2228 filesystem I/Os blocked with no fsbuf
      0 client filesystem I/Os blocked with no fsbuf
   2208 external pager filesystem I/Os blocked with no fsbuf
   24.9 percentage of memory used for computational pages
```

The **vmstat** command in Example 5-42 on page 241 shows: 1352456 client pages - non-computational, 1359411 - 1352456 = 6955.

Example 5-43 Output of the svmon command

```
# svmon -O summary=basic
Unit: page
--------------------------------------------------------------------------------
               size       inuse        free         pin     virtual  available   mmode
memory      2097152     1873794      223358      339861      515260    1513453     Ded
pg space     131072        3067

               work        pers        clnt       other
pin          204357           0           0      135504
in use       514383           0     1359411
```

The output of the **svmon** command in Example 5-43 shows: 1359411 client pages. Some of them are computational and the rest are non-computational.

5.5.2 Active memory sharing partition monitoring

This section shows how to monitor the memory in shared memory partitions. It can be done with the **vmstat**, **lparstat**, **topas commands**, and so on, and these commands with the **-h** option show hypervisor paging information.

The operating system sees a logical entity that is not always backed up with physical memory.

When using the **vmstat** command with the **-h** option, the hypervisor paging information will be displayed as shown in Example 5-44.

Example 5-44 vmstat with hypervisor

```
# vmstat -h 5 3

System configuration: lcpu=16 mem=8192MB ent=1.00 mmode=shared mpsz=8.00GB

kthr    memory              page              faults              cpu                   hypv-page
----- ----------- ------------------------ ------------ ----------------------- -------------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec   hpi  hpit   pmem   loan
 0  0 399386 1666515   0   0   0   0    0   0   3  124 120  0  0 99  0  0.00   0.4     0     0   8.00 0.00

In this case, It also shows the number of 4 KB pages in avm, fri and page columns
It shows 6G or more of free space (ie. 1666515 * 4k pages)
```

The fields highlighted in bold in Example 5-44 have been added for active memory sharing:

mmode Shows shared if the partition is running in shared memory mode. This field was not displayed on dedicated memory partitions.

mpsz Shows the size of the shared memory pool.

hpi Shows the number of hypervisor page-ins for the partition. A hypervisor page-in occurs if a page is being referenced that is not available in real memory because it has been paged out by the hypervisor previously. If no interval is specified when issuing the **vmstat** command, the value shown is counted from boot time.

hpit Shows the average time spent in milliseconds per hypervisor page-in. If no interval is specified when issuing the vmstat command, the value shown is counted from boot time.

pmem Shows the amount of physical memory backing the logical memory, in gigabytes.

loan Shows the amount of the logical memory in gigabytes that is loaned to the hypervisor. The amount of loaned memory can be influenced through the vmo ams_loan_policy tunable.

If the consumed memory is larger than the desired memory, the system utilization avm ratio is over desired memory capacity, as shown in Example 5-45.

Example 5-45 In case of larger than desired memory consumption (vmstat with hypervisor)

```
# vmstat -h 5 3

System configuration: lcpu=16 mem=8192MB ent=1.00 mmode=shared mpsz=8.00GB

kthr    memory              page                  faults             cpu                        hypv-page
----- -----------   ----------------------   ------------ ---------------------------  -------------------------
 r  b   avm   fre   re  pi  po     fr   sr      cy  in  sy    cs   us sy id wa   pc  ec     hpi  hpit   pmem   loan
 7  3 1657065  3840  0   0   0    115709 115712   0   3  16  622057 45 24 15 17  3.75 374.9    0     0   8.00   0.00
 7  3 2192911  4226  0   7 16747 102883 153553   0 1665 10  482975 41 26 18 15  3.19 318.6    0     0   8.00   0.00
 5  6 2501329  5954  0  48 54002 53855  99595    0 6166 11  36778  25 43 25  6  1.12 112.2    0     0   8.00   0.00
```

If loaning is enabled (ams_loan_policy is set to 1 or 2 in vmo), AIX loans pages when the hypervisor initiates a request. AIX removes free pages that are loaned to the hypervisor from the free list.

Example 5-44 on page 242 shows a partition that has a logical memory size of 8 GB. It has also assigned 8 GB of physical memory. Of this assigned 8 GB of physical memory, 6.3 GB (`1666515 4 k` pages) are free because there is no activity in the partition.

Example 5-46 shows the same partition a few minutes later. In the meantime, the hypervisor requested memory and the partition loaned 3.2 GB to the hypervisor. AIX has removed the free pages that it loaned from the free list.

The free list has therefore been reduced by 833215 4 KB pages, as shown in Example 5-46.

Example 5-46 vmstat command

```
# vmstat -h 5 3

System configuration: lcpu=16 mem=8192MB ent=1.00 mmode=shared mpsz=8.00GB

kthr    memory              page              faults              cpu                    hypv-page
----- ----------- ------------------------ ------------ ----------------------- -------------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec   hpi  hpit   pmem   loan
 0  0 399386 833215   0   0   0   0    0   0   3  124 120  0  0 99  0  0.00   0.4     0     0   4.76 3.24
```

AIX paging and hypervisor paging

When using active memory sharing, paging can occur on the AIX level or on the hypervisor level. When you see non-zero values in the pi or po column of the **vmstat** command, it means that AIX is performing paging activities.

In a shared memory partition, AIX paging occurs not only when the working set exceeds the size of the logical memory, as in a dedicated partition. This can happen even if the LPAR has less physical memory than logical memory. AIX is dependent on the amount of logical memory available.

Another reason is that AIX is freeing memory pages to loan them to the hypervisor. If the loaned pages are used pages, AIX has to save the content to its paging space before loaning them to the hypervisor.

This behavior will especially occur if you have selected an aggressive loaning policy (ams_loan_policy=2).

5.5.3 Active memory expansion partition monitoring

In 3.2, “Active Memory Expansion” on page 48, the concepts of active memory expansion (AME) were introduced. Now, a few examples of how to monitor AME behavior are shown.

Monitoring of AME can be done with the special tools **amepat**, **topas -L**.

The amepat command

The **amepat** command provides a summary of the active memory expansion configuration, and can be used for monitoring and fine-tuning the configuration.

The **amepat** command shows the current configuration and statistics of the system resource utilization over the monitoring period. It can be run for periods of time to collect metrics while a workload is running, as shown in Example 5-47.

Example 5-47 amepat with little memory consumption

```
# amepat 1 5

Command Invoked                : amepat 1 5

Date/Time of invocation        : Wed Oct 10 10:33:13 CDT 2012
Total Monitored time           : 5 mins 6 secs
Total Samples Collected        : 5

System Configuration:
---------------------
Partition Name                 : p750s1aix4
Processor Implementation Mode  : POWER7 Mode
Number Of Logical CPUs         : 16
Processor Entitled Capacity    : 1.00
Processor Max. Capacity        : 4.00
True Memory                    : 8.00 GB
SMT Threads                    : 4
Shared Processor Mode          : Enabled-Uncapped
Active Memory Sharing          : Disabled
Active Memory Expansion        : Disabled

System Resource Statistics:               Average             Min             Max
```

```
--------------------------              ----------      ----------      ----------
CPU Util (Phys. Processors)             0.00 [  0%]     0.00 [  0%]     0.00 [  0%]
Virtual Memory Size (MB)                1564 [ 19%]     1564 [ 19%]     1564 [ 19%]
True Memory In-Use (MB)                 1513 [ 18%]     1513 [ 18%]     1514 [ 18%]
Pinned Memory (MB)                      1443 [ 18%]     1443 [ 18%]     1443 [ 18%]
File Cache Size (MB)                      19 [  0%]       19 [  0%]       19 [  0%]
Available Memory (MB)                   6662 [ 81%]     6662 [ 81%]     6663 [ 81%]

Active Memory Expansion Modeled Statistics        :
-----------------------------------------
Modeled Expanded Memory Size :   8.00 GB
Achievable Compression ratio   :1.90

Expansion    Modeled True      Modeled             CPU Usage
Factor       Memory Size       Memory Gain         Estimate
---------    ------------    -----------------   -----------
     1.00          8.00 GB        0.00 KB [  0%]   0.00 [  0%]
     1.11          7.25 GB      768.00 MB [ 10%]   0.00 [  0%]
     1.19          6.75 GB        1.25 GB [ 19%]   0.00 [  0%]
     1.28          6.25 GB        1.75 GB [ 28%]   0.00 [  0%]
     1.34          6.00 GB        2.00 GB [ 33%]   0.00 [  0%]
     1.46          5.50 GB        2.50 GB [ 45%]   0.00 [  0%]
     1.53          5.25 GB        2.75 GB [ 52%]   0.00 [  0%]

Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 5.25 GB and to configure a memory expansion factor
of 1.53.  This will result in a memory gain of 52%. With this
configuration, the estimated CPU usage due to AME is approximately 0.00
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 0.00 physical processors.

NOTE: amepat's recommendations are based on the workload's utilization level
during the monitored period. If there is a change in the workload's utilization
level or a change in workload itself, amepat should be run again.

The modeled Active Memory Expansion CPU usage reported by amepat is just an
estimate.  The actual CPU usage used for Active Memory Expansion may be lower
or higher depending on the workload.
```

Example 5-48 shows **amepat** with heavy memory consumption. It shows a high memory compression ratio because the test program consumes garbage memory.

Example 5-48 The amepat command

```
# amepat 1 5

Command Invoked                : amepat 1 5

Date/Time of invocation        : Wed Oct 10 11:36:07 CDT 2012
Total Monitored time           : 6 mins 2 secs
Total Samples Collected        : 5

System Configuration:
---------------------
```

```
Partition Name                  : p750s1aix4
Processor Implementation Mode   : POWER7 Mode
Number Of Logical CPUs          : 16
Processor Entitled Capacity     : 1.00
Processor Max. Capacity         : 4.00
True Memory                     : 8.00 GB
SMT Threads                     : 4
Shared Processor Mode           : Enabled-Uncapped
Active Memory Sharing           : Disabled
Active Memory Expansion         : Disabled

System Resource Statistics:               Average             Min             Max
---------------------------               -----------       -----------     -----------
CPU Util (Phys. Processors)              0.13 [  3%]       0.00 [  0%]     0.31 [  8%]
Virtual Memory Size (MB)                 6317 [ 77%]       2592 [ 32%]     9773 [119%]
True Memory In-Use (MB)                  5922 [ 72%]       2546 [ 31%]     8178 [100%]
Pinned Memory (MB)                       1454 [ 18%]       1447 [ 18%]     1460 [ 18%]
File Cache Size (MB)                      296 [  4%]         23 [  0%]     1389 [ 17%]
Available Memory (MB)                    2487 [ 30%]          6 [  0%]     5630 [ 69%]

Active Memory Expansion Modeled Statistics        :
-----------------------------------------
Modeled Expanded Memory Size :   8.00 GB
Achievable Compression ratio   :6.63

Expansion    Modeled True      Modeled             CPU Usage
Factor       Memory Size       Memory Gain         Estimate
---------    -------------     -----------------   -----------
     1.04          7.75 GB       256.00 MB [  3%]   0.00 [  0%]
     1.34          6.00 GB         2.00 GB [ 33%]   0.00 [  0%]
     1.69          4.75 GB         3.25 GB [ 68%]   0.00 [  0%]
     2.00          4.00 GB         4.00 GB [100%]   0.00 [  0%]
     2.29          3.50 GB         4.50 GB [129%]   0.00 [  0%]
     2.67          3.00 GB         5.00 GB [167%]   0.00 [  0%]
     2.91          2.75 GB         5.25 GB [191%]   0.00 [  0%]

Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 2.75 GB and to configure a memory expansion factor
of 2.91.  This will result in a memory gain of 191%. With this
configuration, the estimated CPU usage due to AME is approximately 0.00
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 0.31 physical processors.

NOTE: amepat's recommendations are based on the workload's utilization level
during the monitored period. If there is a change in the workload's utilization
level or a change in workload itself, amepat should be run again.

The modeled Active Memory Expansion CPU usage reported by amepat is just an
estimate.  The actual CPU usage used for Active Memory Expansion may be lower
or higher depending on the workload.
```

The topas command

The **topas** command shows the following active memory expansion metrics on the default panel when started with no options:

- **TMEM, MB** - True memory size in megabytes.
- **CMEM, MB** - Compressed pool size in megabytes.
- **EF[T/A**] - Expansion factors: Target and Actual.
- **CI** - Compressed pool page-ins.
- **CO** - Compressed pool page-outs.

Example 5-49 shows an example of the **topas** panel.

Example 5-49 Monitoring active memory expansion with the topas command

```
Topas Monitor for host:p750s1aix4                EVENTS/QUEUES    FILE/TTY
Wed Oct 10 13:31:19 2012   Interval:FROZEN       Cswitch 4806.0M  Readch  3353.8G
                                                 Syscall 1578.4M  Writech 3248.8G
CPU     User% Kern% Wait% Idle%   Physc  Entc%  Reads     49.8M  Rawin    642.2K
Total    38.1  61.9   0.0   0.0    2.62 262.41  Writes   101.6M  Ttyout    25.5M
                                                 Forks   1404.8K  Igets      4730
Network    BPS  I-Pkts  O-Pkts    B-In   B-Out  Execs   1463.8K  Namei    110.2M
Total        0       0       0       0       0  Runqueue  3.00M  Dirblk    95749
                                                 Waitqueue 65384.6
Disk    Busy%      BPS     TPS  B-Read  B-Writ                    MEMORY
Total     0.0        0       0       0       0  PAGING           Real,MB    8192
                                                 Faults  2052.0M  % Comp     18
FileSystem          BPS    TPS  B-Read  B-Writ  Steals   802.8M  % Noncomp   0
Total             2.52M  2.52K   2.52M       0  PgspIn   547.4K  % Client    0
                                                 PgspOut  126.3M
Name           PID  CPU%  PgSp Owner            PageIn  4204.6K  PAGING SPACE
inetd      4718746  0.0  536K root             PageOut  936.4M  Size,MB    8192
lrud        262152  0.0  640K root             Sios     908.0M  % Used      1
hostmibd   4784176  0.0 1.12M root                             % Free     99
psmd        393228  0.0  640K root             AME
aixmibd    4849826  0.0 1.30M root             TMEM,MB 2815.2M  WPAR Activ    0
hrd        4915356  0.0  924K root             CMEM,MB 1315.7M  WPAR Total    0
reaffin     589842  0.0  640K root             EF[T/A]   2.91   Press: "h"-help
sendmail   4980888  0.0 1.05M root             CI:0.0KCO:0.1K           "q"-quit
lvmbb       720920  0.0  448K root
vtiol       786456  0.0 1.06M root
ksh        6226074  0.0  556K root
pilegc      917532  0.0  640K root
xmgc        983070  0.0  448K root
```

5.5.4 Paging space utilization

When a program requests some memory and that amount cannot be satisfied in RAM, the Virtual Memory Manager (VMM), through the last recently used (lru) algorithm, selects some pages to be moved to paging space, also called swap space. This is called a page-out. This allows the memory request to be fulfilled. When these pages in swap are needed again, they are read from hard disk and moved back into RAM. This is called page-in.

Excess of paging is bad for performance because access to paging devices (disks) is many times slower than access to RAM. Therefore, it is important to have a good paging setup, as shown in 4.2.2, “Paging space” on page 128, and to monitor the paging activity.

Beginners in AIX think that if they look at the paging space utilization and see a high number, that is bad. Looking at the output of **`lsps -a`** and having a paging space utilization greater than zero, does not mean that AIX is memory constraint at the moment.

Tip: It is safer to use the **`lsps -s`** command rather than the **`lsps -a`**.

Example 5-50 shows paging space utilization at 71%. However, this does not mean that AIX is paging or that the system has low free memory available. In Example 5-51, the output of **`svmon`** shows 6168.44 MB of available memory and 1810.98 MB of paging space used.

Example 5-50 Looking at paging space utilization

```
# lsps -a
Page Space      Physical Volume   Volume Group    Size %Used Active Auto  Type Chksum
hd6             hdisk0            rootvg        2560MB    71 yes   yes    lv     0
```

Example 5-51 Available memory and paging utilization

```
# svmon -O summary=basic,unit=MB
Unit: MB
--------------------------------------------------------------------------------
               size       inuse        free         pin     virtual   available   mmode
memory      8192.00     2008.56     6183.44     1208.25     3627.14     6168.44     Ded
pg space    2560.00     1810.98

               work        pers        clnt       other
pin          678.94           0           0      529.31
in use      1835.16           0      173.40
```

The percent paging space utilization means that at some moment AIX VMM required that amount of paging. After this peak of memory requirements, some process that had pages paged out did not require a page-in of such pages, or if the page-in was required, it was for read access and not for modifying. Paging space garbage collection, by default, only operates when a page-in happens. If the page is brought back into memory to read-only operations, it is not freed up from paging space. This provides better performance because if the page remains unmodified and is stolen from RAM by the LRU daemon, it is not necessary to perform the repage-out function.

One important metric regarding paging is *paging in* and *paging out*. In Example 5-52 using **`topas`** we see AIX during a low paging activity. `PgspIn` is the number of 4 K pages read from paging space per second over the monitoring interval. `PgspOut` is the number of 4 K pages written to paging space per second over the monitoring interval.

Example 5-52 topas showing small paging activity

```
Topas Monitor for host:p750s2aix4                EVENTS/QUEUES    FILE/TTY
Mon Oct  8 18:52:33 2012   Interval:2            Cswitch     400  Readch     2541
                                                 Syscall     227  Writech     512
CPU     User% Kern% Wait% Idle%   Physc  Entc%  Reads        28  Rawin         0
Total     0.2   0.5   0.0  99.3    0.01   1.36  Writes        1  Ttyout      246
                                                 Forks         0  Igets         0
Network    BPS  I-Pkts  O-Pkts    B-In   B-Out  Execs         0  Namei        24
Total    677.0    4.00    2.00   246.0   431.0  Runqueue   1.00  Dirblk        0
```

```
                                                  Waitqueue   0.0
Disk    Busy%      BPS     TPS  B-Read  B-Writ                    MEMORY
Total     0.0     458K   81.50    298K    160K  PAGING           Real,MB    8192
                                                 Faults       88  % Comp     98
FileSystem          BPS    TPS  B-Read  B-Writ  Steals       40  % Noncomp   1
Total              2.48K  28.50   2.48K       0  PgspIn       74  % Client    1
                                                 PgspOut      40
Name            PID  CPU%  PgSp Owner           PageIn       74  PAGING SPACE
java        8388796  0.5 65.2M root            PageOut      40  Size,MB    2560
syncd        786530  0.2  596K root            Sios         87  % Used     89
java        6553710  0.1 21.0M root                            % Free     11
topas       7995432  0.1 2.04M root            NFS (calls/sec)
lrud         262152  0.0  640K root            SerV2         0  WPAR Activ    0
getty       6815754  0.0  640K root            CliV2         0  WPAR Total    1
vtiol        851994  0.0 1.06M root            SerV3         0  Press: "h"-help
gil         2162754  0.0  960K root            CliV3         0         "q"-quit
```

In Example 5-53, using **vmstat**, you see AIX during a high paging activity.

Example 5-53 vmstat showing considerable paging activity

```
# vmstat 5

System configuration: lcpu=16 mem=8192MB ent=1.00

kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 2  0 2241947  5555   0 3468 3448 3448 3448   0 3673  216 7489  9  4 84  3  0.26  26.0
 2  0 2241948  5501   0 5230 5219 5219 5222   0 5521  373 11150 14  6 71  9  0.39  38.6
 2  1 2241948  5444   0 5156 5145 5145 5145   0 5439   83 10972 14  6 76  4  0.40  40.1
 1  0 2241959  5441   0 5270 5272 5272 5272   0 5564  435 11206 14  6 70  9  0.39  38.9
 1  1 2241959  5589   0 5248 5278 5278 5278   0 5546   82 11218 14  6 76  4  0.41  40.9
```

If your system is consistently presenting high page-in or page-out rates, your performance is probably being affected due to memory constraints.

5.5.5 Memory size simulation with rmss

It is possible to simulate reduced sizes of memory without performing a *dlpar* operation and without stopping the partition. The **rmss** command—reduced memory system simulator—can be used to test application and system behavior with different memory scenarios.

The main use for the **rmss** command is as a capacity planning tool to determine how much memory a workload needs.

To determine whether the **rmss** command is installed and available, run the following command:

```
# lslpp -lI bos.perf.tools
```

You can use the **rmss** command in two modes:

- To change the system memory size.
- To execute a specified application multiple times over a range of memory sizes and display important statistics that describe the application's performance at each memory size.

Example 5-54 shows **rmss** changing the memory to 4 GB, the first mode.

Example 5-54 Using rmss to simulate a system with 4 GB memory

```
# rmss -c 4096
Simulated memory size changed to 4096 Mb.
Warning: This operation might impact the system environment.
Please refer vmo documentation to resize the appropriate parameters.
```

The simulated memory can be verified with the **-p** flag; to reset to physical real memory, use the **-r** flag.

5.5.6 Memory leaks

Memory leak is a software error in which the program allocates memory and never releases it *after use*. In a long-running program, memory leak is a serious problem because it can exhaust the system real memory and paging space, leading to a program or system crash.

Memory leaks are not to be confused with caching or any other application behavior. Processes showing an increase in memory consumption may not be leaking memory. Instead, that can actually be the expected behavior, depending on what the program is intended to do.

Before continuing, it must be clear that memory leaks can only be confirmed with source code analysis. However, some system analysis may help in identifying possible programs with problems.

A memory leak can be detected with the **ps** command, using the ***v*** flag. This flag displays a SIZE column, which shows the virtual size of the data section of the process.

Note: The SIZE column does not represent the same as the SZ column produced by the ***-l*** flag. Although sometimes they show the same value, they can be different if some pages of the process are paged out.

Using **ps**, the information of a process with pid 6291686 is collected at 30-second intervals, as seen in Example 5-55.

Example 5-55 Using ps to collect memory information

```
# while true ; do ps v 6291686 >> /tmp/ps.out ; sleep 30 ; done
```

Example 5-56 shows the increase in the SIZE.

Example 5-56 Increase in memory utilization

```
# grep PID /tmp/ps.out  | head -n 1 ; grep 6291686 /tmp/ps.out
     PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 6291686  pts/2 A     0:00    0   156   164    xx     1     8  0.0  0.0 ./test_
 6291686  pts/2 A     0:00    0   156   164    xx     1     8  0.0  0.0 ./test_
 6291686  pts/2 A     0:00    0   160   168    xx     1     8  0.0  0.0 ./test_
 6291686  pts/2 A     0:00    0   164   172    xx     1     8  0.0  0.0 ./test_
 6291686  pts/2 A     0:00    0   164   172    xx     1     8  0.0  0.0 ./test_
 6291686  pts/2 A     0:00    0   168   176    xx     1     8  0.0  0.0 ./test_
 6291686  pts/2 A     0:00    0   172   180    xx     1     8  0.0  0.0 ./test_
```

Another command that can be used is **svmon**, which looks for processes whose working segment continually grows. To determine whether a segment is growing, use **svmon** with the *-i* <interval> option to look at a process or a group of processes and see whether any segment continues to grow.

Example 5-57 shows how to start collecting data with **svmon**. Example 5-58 shows the increase in size for segment 2 (Esid 2 - process private).

Example 5-57 Using svmon to collect memory information

```
# svmon -P  6291686 -i 30  > /tmp/svmon.out
```

Example 5-58 Output of svmon showing increase in memory utilization

```
# grep Esid /tmp/svmon.out | head -n 1 ; grep " 2 work" /tmp/svmon.out
    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
  9a0dba         2 work process private             sm     20     4    0      20
  9a0dba         2 work process private             sm     20     4    0      20
  9a0dba         2 work process private             sm     21     4    0      21
  9a0dba         2 work process private             sm     22     4    0      22
  9a0dba         2 work process private             sm     22     4    0      22
  9a0dba         2 work process private             sm     23     4    0      23
  9a0dba         2 work process private             sm     24     4    0      24
  9a0dba         2 work process private             sm     24     4    0      24
```

Important: Never assume that a program is leaking memory only by monitoring the operating system. Source code analysis must always be conducted to confirm the problem.

5.6 Disk storage bottleneck identification

When finding that there is a performance bottleneck related to external disk storage, it can be challenging to find the source of the problem. It can be in any component of the server, SAN or storage infrastructures. This section explains some of the performance metrics to look at when diagnosing a performance problem, and where to look in the event that you have an I/O performance problem.

5.6.1 Performance metrics

There are different metrics to consider when looking at the disk utilization of an AIX system. To identify a bottleneck, you first need to understand the metrics involved to be able to identify a problem.

Table 5-3 gives a summary of key performance metrics to understand when investigating an I/O performance problem on an AIX system.

Table 5-3 Key performance metrics

Metric	Description
IOPS	IOPS represents the amount of read or write I/O operations performed in a 1-second time interval.
Throughput	Throughput is the amount of data that can be transferred between the server and the storage measured in megabytes per second.

Metric	Description
Transfer size	The transfer size typically measured in kilobytes is the size of an I/O request.
Wait time	Wait time is the amount of time measured in milliseconds that the server's processor has to wait for a pending I/O to complete. The pending I/O could be in the queue for the I/O device, increasing the wait time for an I/O request.
Service time	Service time is the time taken for the storage system to service an I/O transfer request in milliseconds.

Depending on the type of I/O that is being performed, the service times may differ. An I/O operation with a small transfer size would be expected to have a significantly smaller service time than an I/O with a large transfer size because the larger I/O operation is bigger and more data has to be processed to service it. Larger I/O operations are also typically limited to throughput. For instance, the service time of a 32 k I/O will be significantly larger than an 8 k I/O because the 32 k I/O is four times the size of the 8 k I/O.

When trying to identify a performance bottleneck it is necessary to understand whether the part of your workload that may not be performing adequately (for example, a batch job) is using small block random I/O or large block sequential I/O.

5.6.2 Additional workload and performance implications

A storage system, dependant on its configuration, will be able to sustain a certain amount of workload until at some point one or more components become saturated and the service time, also known as response time, increases exponentially.

It is important to understand the capability of the storage system that the AIX system is using, and what its upper boundary is in terms of performance.

In Figure 5-11 a storage system shows to have the capability to service I/O requests up to 50,000 IOPS of a certain transfer size under 10 milliseconds, which is considered to be acceptable in most cases. You can see that once the storage system is performing beyond 50,000 I/O operations, it reaches a breaking point where response time rises significantly.

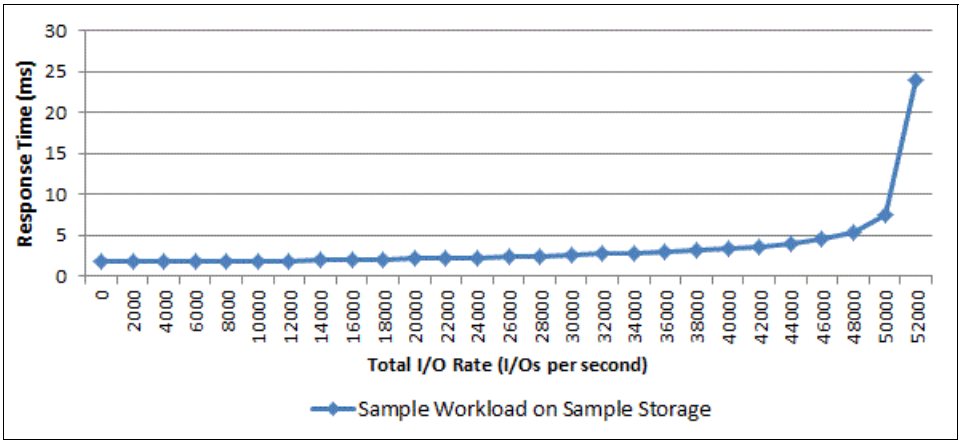


Figure 5-11 Effect of I/O rate on response time

When a workload increases, or new workloads are added to an existing storage system, we suggest that you talk to your storage vendor to understand what the capability of the current storage system is before it is saturated. Either adding more disks (spindles) to the storage or

looking at intelligent automated tiering technologies with solid state drives might be necessary to boost the performance of the storage system.

5.6.3 Operating system - AIX

When looking at the AIX operating system to find the source of an I/O performance bottleneck, it needs to be established whether there is a configuration problem causing the bottleneck, or whether the I/O bottleneck exists outside of the AIX operating system.

The initial place to look in AIX is the error report to check whether there are any problems that have been detected by AIX, because an event may have occurred for the problem to arise. Example 5-59 demonstrates how to check the AIX error report.

Example 5-59 Checking errpt in AIX

```
root@aix1:/ # errpt
DE3B8540   1001105612 P H hdisk0         PATH HAS FAILED
DE3B8540   1001105612 P H hdisk2         PATH HAS FAILED
DE3B8540   1001105612 P H hdisk3         PATH HAS FAILED
DE3B8540   1001105612 P H hdisk1         PATH HAS FAILED
4B436A3D   1001105612 T H fscsi0        LINK ERROR
root@aix1:/ #
```

If any errors are present on the system, such as failed paths, they need to be corrected. In the event that there are no physical problems, another place to look is at the disk service time by using the **iostat** and **sar** commands. Using **iostat** is covered in 4.3.2, “Disk device tuning” on page 143**.** The **sar** command is shown in Example 5-60.

Example 5-60 Disk analysis with sar with a single 10-second interval

```
root@aix1:/ # sar -d 10 1

AIX aix1 1 7 00F6600E4C00    10/08/12

System configuration: lcpu=32 drives=4 ent=3.00 mode=Uncapped

06:41:48     device    %busy    avque    r+w/s    Kbs/s   avwait   avserv

06:41:58     hdisk3    100       4.3     1465  1500979     96.5      5.5
             hdisk1      0       0.0        0        0      0.0      0.0
             hdisk0      0       0.0        0        0      0.0      0.0
             hdisk2    100      18.0      915   234316    652.7      8.8

root@aix1:/ #
```

The output of Example 5-60 shows the following indicators of a performance bottleneck:

- Disks hdisk2 and hdisk3 are busy, while hdisk0 and hdisk 1 are idle. This is shown by % busy, which is the percentage of time that the disks have been servicing I/O requests.
- There are a number of requests outstanding in the queue for hdisk2 and hdisk3. This is shown in `avque` and is an indicator that there is a performance problem.
- The average number of transactions waiting for service on hdisk2 and hdisk3 is also indicating a performance issue, shown by `avwait`.
- The average service time from the physical disk storage is less than 10 milliseconds on both hdisk2 and hdisk3, which is acceptable in most cases. This is shown in `avserv`.

The output of **sar** shows us that we have a queuing issue on hdisk2 and hdisk3, so it is necessary to follow the steps covered in 4.3.2, “Disk device tuning” on page 143 to resolve this problem.

Note: If you are using Virtual SCSI disks, be sure that any tuning attributes on the hdisk in AIX match the associated hdisk on the VIO servers. If you make a change, the attributes must be changed on the AIX device and on the VIO server backing device.

When looking at fiber channel adapter statistics, it is important to look at the output of the **fcstat** command in both AIX and the VIO servers. The output demonstrates whether there are issues with the fiber channel adapters. Example 5-61 shows the items of interest from the output of **fcstat**. 4.3.5, “Adapter tuning” on page 150 describes how to interpret and resolve issues with queuing on fiber channel adapters.

Example 5-61 Items of interest in the fcstat output

```
FC SCSI Adapter Driver Information
  No DMA Resource Count: 0
  No Adapter Elements Count: 0
  No Command Resource Count: 0
```

A large amount of information is presented by commands such as **iostat**, **sar**, and **fcstat**, which typically provide real time monitoring. To look at historical statistics, **nmon** is included with AIX 6.1 and later, and can be configured to record performance data. With the correct options applied, **nmon** recording can store all the information.

It is suggested to use **nmon** to collect statistics that can be opened with the nmon analyzer and converted into Microsoft Excel graphs.

The nmon analyzer can be obtained from:

http://www.ibm.com/developerworks/wikis/display/Wikiptype/nmonanalyser

This link contains some further information about the nmon analyzer tool:

http://www.ibm.com/developerworks/aix/library/au-nmon_analyser/index.html

Example 5-62 demonstrates how to create a 5 GB file system to store the nmon recordings. Depending on how long you want to store the nmon recordings and how many devices are attached to your system, you may need a larger file system.

Example 5-62 Creating a jfs2 file system for NMON recordings

```
root@aix1:/ # mklv -y nmon_lv -t jfs2 rootvg 1 hdisk0
nmon_lv
root@aix1:/ # crfs -v jfs2 -d nmon_lv -m /nmon -a logname=INLINE -A yes
File system created successfully.
64304 kilobytes total disk space.
New File System size is 131072
root@aix1:/ # chfs -a size=5G /nmon
Filesystem size changed to 10485760
Inlinelog size changed to 20 MB.
root@aix1:/ # mount /nmon
root@aix1:/ # df -g /nmon
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
/dev/nmon_lv       5.00      4.98    1%        4     1% /nmon
```

```
root@aix1:/ #
```

Once the file system is created, the next step is to edit the root crontab. Example 5-63 demonstrates how to do this.

Example 5-63 How to edit the root crontab

```
root@aix1:/# crontab -e
```

Example 5-64 shows two sample crontab entries. One entry is to record daily **nmon** statistics, while the other entry is to remove the **nmon** recordings after 60 days. Depending on how long you require **nmon** recordings to be stored for, you may need to have a crontab entry to remove them after a different period of time. You manually need to insert entries into your root crontab.

Example 5-64 Sample crontab to capture nmon recordings and remove them after 60 days

```
# Start NMON Recording
00 00 * * * /usr/bin/nmon -dfPt -^ -m /nmon
# Remove NMON Recordings older than 60 Days
01 00 * * * /usr/bin/find /nmon -name "*.nmon" -type f -mtime +60 ! -name
"*hardened*" |xargs -n1 /bin/rm -f
```

5.6.4 Virtual I/O Server

When looking at one or more VIOSs to find the source of an I/O performance bottleneck, it needs to be established whether there is a configuration problem causing the bottleneck, or whether the I/O bottleneck exists outside of the VIOS.

The initial place to look in VIOS is the error log to check whether there are any problems that have been detected by VIOS, because there may be an event that has occurred for the problem to arise. Example 5-65 demonstrates how to check the VIOS error log.

Example 5-65 Checking the VIOS error log

```
$ errlog
DF63A4FE   0928084612 T S vhost8         Virtual SCSI Host Adapter detected an er
DF63A4FE   0928084512 T S vhost8         Virtual SCSI Host Adapter detected an er
$
```

If any errors are present, they need to be resolved to ensure that there are no configuration issues causing a problem. It is also important to check the fiber channel adapters assigned to the VIOS to ensure that they are not experiencing a problem.

4.3.5, “Adapter tuning” on page 150 describes how to interpret and resolve issues with queuing on fiber channel adapters. You can check **fcstat** in exactly the same way you would in AIX, and the items of interest are the same. This is shown in Example 5-66.

Example 5-66 Items of interest in the fcstat output

```
FC SCSI Adapter Driver Information
  No DMA Resource Count: 0
  No Adapter Elements Count: 0
  No Command Resource Count: 0
```

Another consideration when using NPIV is to make an analysis of how many virtual fiber channel adapters are mapped to the physical fiber channel ports on the VIOS.

If there is a case where there are some fiber channel ports on the VIOS that have more virtual fiber channel adapters mapped to them than others, this could cause some ports to be exposed to a performance degradation and others to be underutilized.

Example 5-67 shows the **lsnports** command, which can be used to display how many mappings are present on each physical fiber channel port.

Example 5-67 The lsnports command

```
$ lsnports
name             physloc                        fabric tports aports swwpns  awwpns
fcs0             U78A0.001.DNWK4AS-P1-C2-T1          1     64     51   2048    2015
fcs1             U78A0.001.DNWK4AS-P1-C2-T2          1     64     50   2048    2016
fcs2             U78A0.001.DNWK4AS-P1-C4-T1          1     64     51   2048    2015
fcs3             U78A0.001.DNWK4AS-P1-C4-T2          1     64     50   2048    2016
$
```

The **lsnports** command displays the information summarized in Table 5-4.

Table 5-4 lsnports

Field	Description
name	Physical port name
physloc	Physical location code
fabric	Fabric support
tports	Total number of NPIV ports
aports	Number of available NPIV ports
swwpns	Total number of worldwide port names supported by the adapter
awwpns	Number of world-wide port names available for use

The output of **lsnports** in Example 5-67 shows the following:

- Our Virtual I/O Server has two dual-port fiber channel adapters.
- Each port is capable of having 64 virtual fiber channel adapters mapped to it.
- The ports fcs0 and fcs2 have 13 client virtual fiber channel adapters mapped to them and fcs1 and fcs3 have14 virtual fiber channel adapters mapped to them. This demonstrates a balanced configuration were load is evenly distributed across multiple virtual fiber channel adapters.

Note: When looking at a VIOS, some statistics are also shown in the VIOS Performance Advisor, which is covered in 5.9, “VIOS performance advisor tool and the part command” on page 271, which can provide some insight into the health of the VIOS.

5.6.5 SAN switch

In the event that the AIX system and VIOS have the optimal configuration, and an I/O performance issue still exists, the next thing to be checked in the I/O chain is the SAN fabric.

If you are using an 8 G fiber channel card, we suggest that you use a matching 8 G small form-factor pluggable (SFP) transceiver in the fabric switch.

It is worthwhile to check the status of the ports that the POWER system is using to ensure that there are no errors on the port. Example 5-68 demonstrates how to check the status of port 0 on an IBM B type fabric switch.

Example 5-68 Use of the portshow command

```
pw_2002_SANSW1:admin> portshow 0
portIndex:   0
portName:
portHealth: HEALTHY

Authentication: None
portDisableReason: None
portCFlags: 0x1
portFlags: 0x1024b03     PRESENT ACTIVE F_PORT G_PORT U_PORT NPIV LOGICAL_ONLINE
LOGIN NOELP LED ACCEPT FLOGI
LocalSwcFlags: 0x0
portType:  17.0
POD Port: Port is licensed
portState: 1    Online
Protocol: FC
portPhys:  6    In_Sync         portScn:   32   F_Port
port generation number:    320
state transition count:    47

portId:    010000
portIfId:    4302000f
portWwn:   20:00:00:05:33:68:84:ae
portWwn of device(s) connected:
        c0:50:76:03:85:0e:00:00
        c0:50:76:03:85:0c:00:1d
        c0:50:76:03:85:0e:00:08
        c0:50:76:03:85:0e:00:04
        c0:50:76:03:85:0c:00:14
        c0:50:76:03:85:0c:00:08
        c0:50:76:03:85:0c:00:10
        c0:50:76:03:85:0e:00:0c
        10:00:00:00:c9:a8:c4:a6
Distance:  normal
portSpeed: N8Gbps

LE domain: 0
FC Fastwrite: OFF
Interrupts:        0          Link_failure: 0          Frjt:         0
Unknown:           38         Loss_of_sync: 19         Fbsy:         0
Lli:               152        Loss_of_sig:  20
Proc_rqrd:         7427       Protocol_err: 0
Timed_out:         0          Invalid_word: 0
Rx_flushed:        0          Invalid_crc:  0
Tx_unavail:        0          Delim_err:    0
Free_buffer:       0          Address_err:  0
Overrun:           0          Lr_in:        19
Suspended:         0          Lr_out:       0
```

```
Parity_err:        0          Ols_in:       0
2_parity_err:      0          Ols_out:      19
CMI_bus_err:       0

Port part of other ADs: No
```

When looking at the output of Example 5-68 on page 257, it is important to determine the WWNs of the connected clients. In this example, there are eight NPIV clients attached to the port. It is also important to check the `overrun` counter to see whether the switch port has had its buffer exhausted.

The switch port configuration can be modified. However, this may affect other ports and devices attached to the fabric switch. In the case that a SAN switch port is becoming saturated, it is suggested that you balance your NPIV workload over more ports. This can be analyzed by using the **`lsnports`** command on the VIOS as described in 5.6.4, “Virtual I/O Server” on page 255.

5.6.6 External storage

The final link in the I/O chain is the physical storage attached to the POWER system. When AIX, the VIOS, and the SAN fabric have been checked and are operating correctly, the final item to check is the physical storage system.

Depending on the storage system you are using, the storage vendor typically has a number of tools available to view the storage system utilization. It is suggested to consult your storage administrator to look at the performance of the volumes presented to the POWER system. An example tool that can be used is IBM Tivoli Storage Productivity Center to perform an analysis of IBM disk storage products.

The items of interest determine whether there is a configuration problem on the storage side, including but not limited to some of the items in Table 5-5.

Table 5-5 External storage items of interest

Item	Description
Read Response Time	When a read I/O request is issued by an attached host, this is the amount of time taken by the storage to service the request. The response time is dependant on the size of the I/O request, and the utilization of the storage. Typically, the read response time for small block I/O should be 10 milliseconds or less, while for large block I/O the response time should be 20 milliseconds or less.
Write Response Time	When a host performs a write to the storage system, the write response time is the amount of time in milliseconds it takes for the storage system to accept the write and send an acknowledgment back to the host. The response time for writes should ideally be less than 5 milliseconds because the write will be cached in the storage controller’s write cache. In the event that the response time for writes is large, then this suggests that the writes are missing the storage controller’s cache.
Read Cache Hit %	When a host issues a read request, this is the percentage of the I/O requests that the read is able to read from the storage controller’s cache, rather than having to read the data from disk. When a workload is considered cache friendly, this describes a workload that will have its read predominantly serviced from cache.

Item	Description
Write Cache Hit %	When a host performs a write, the percentage of the writes that are able to be cached in the storage controller's write cache. In the event that the write cache hit % is low, this may indicate a problem with the storage controller's write cache being saturated.
Volume Placement	Volume placement within a storage system is important when considering AIX LVM spreading or striping. When a logical volume is spread over multiple hdisk devices, there are some considerations for the storage system volumes that are what AIX sees as hdisks. It is important that all storage system volumes associated with an AIX logical volume exist on the same disk performance class.
Port Saturation	It is important to check that the storage ports that are zoned to the AIX systems are not saturated or overloaded. It is important for the storage administrator to consider the utilization of the storage ports.
RAID Array Utilization	The utilization of a single RAID array is becoming less of an issue on many storage systems that have a wide striping capability. This is where multiple RAID arrays are pooled together and when a volume is created it is striped across all of the RAID arrays in the pool. This ensures that the volume is able to take advantage of all of the aggregate performance of all of the disks in the pool. In the event that a single RAID array is performing poorly, examine the workload on that array, and that any pooling of RAID arrays has the volumes evenly striped.
Automated Tiering	It is becoming more common that storage systems have an automated tiering feature. This provides the capability to have different classes of disks inside the storage system (SATA, SAS, and SSD), and the storage system will examine how frequently the blocks of data inside host volumes are accessed and place them inside the appropriate storage class. In the event that the amount of fast disk in the storage is full, and some workloads are not having their busy blocks promoted to a faster disk class, it may be necessary to review the amount of fast vs. slow disk inside the storage system.

5.7 Network utilization

Usually, network utilization is quite easy to understand because it does not have as many factors changing the way it behaves as the processors have. However, network topology is built of several layers that can individually affect the network performance on the environment.

Think of a complex environment with WPARs running on top of one or more LPARs using virtual network adapters provided by VIOS with a Shared Ethernet Adapter configured over Etherchannel interfaces, which in turn connect to network switches, routers, and firewalls. In such scenarios, there would be many components and configurations that could simply slow down the network throughput.

Measuring network statistics in that environment would be quite a complex task. However, from an operating system point of view, there are some things that we can do to monitor a smaller set of network components.

5.7.1 Network statistics

Network statistics tell how the network is behaving. Several counters and other information is available to alert about the system running out of resources, possible hardware faults, some workload behavior, and other problems on the network infrastructure itself.

On AIX, network statistics can be gathered with the commands **entstat**, **netstat**, and **netpmon**.

The entstat command

Example 5-69 illustrates the output of the **entstat** command used to gather statistics from the Ethernet adapters of the system. Some of these are as follows:

- Transmit errors and receive errors

  These two counters indicate whether any communication errors have occurred due to problems on the hardware or the network. Ideally these fields should report a value of zero, but specific events on the network may cause these fields to report some positive value. However, if any of these fields present a non-zero value, it is suggested that the system be monitored for some time because this may also indicate a local hardware fault.

- Packets dropped

  Packets dropped appear on both transmit and receive sides and are an indication of problems. They are not tied to specific events, but if packets are dropped for any reason this counter will increase.

- Bad packets

  Bad packets can be caused by several different problems on the network. The importance of this counter is that bad packets cause retransmission and overhead.

- Max packets on S/W transmit queue

  This value indicates the maximum size of the transmit queue supported by the hardware when it does not support software queues. If this limit is reached, the system reports that information on the *S/W transmit queue overflow* counter.

- No mbuf errors

  These errors should appear if the system runs out of mbuf structures to allocate data for transmit or receive operations. In this case, the *packets dropped* counter will also increase.

Example 5-69 entstat - output from interface ent0

```
# entstat ent0
-------------------------------------------------------------
ETHERNET STATISTICS (en0) :
Device Type: Virtual I/O Ethernet Adapter (l-lan)
Hardware Address: 52:e8:76:4f:6a:0a
Elapsed Time: 1 days 22 hours 16 minutes 21 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 12726002                             Packets: 48554705
Bytes: 7846157805                             Bytes: 69529055717
Interrupts: 0                                 Interrupts: 10164766
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
                                              Bad Packets: 0

Max Packets on S/W Transmit Queue: 0
```

```
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 755                      Broadcast Packets: 288561
Multicast Packets: 755                      Multicast Packets: 23006
No Carrier Sense: 0                         CRC Errors: 0
DMA Underrun: 0                             DMA Overrun: 0
Lost CTS Errors: 0                          Alignment Errors: 0
Max Collision Errors: 0                     No Resource Errors: 0
Late Collision Errors: 0                    Receive Collision Errors: 0
Deferred: 0                                 Packet Too Short Errors: 0
SQE Test: 0                                 Packet Too Long Errors: 0
Timeout Errors: 0                           Packets Discarded by Adapter: 0
Single Collision Count: 0                   Receiver Start Count: 0
Multiple Collision Count: 0
Current HW Transmit Queue Length: 0

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 20000
Driver Flags: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        DataRateSet
```

Note: **entstat** does not report information on loopback adapters or other encapsulated adapters. For example, if you create an Etherchannel ent3 on a VIOS with two interfaces, ent0 and ent1, encapsulate it in a Shared Ethernet Adapter ent5 using a control-channel adapter ent4. **entstat** will only report statistics if ent5 is specified as the argument, but it will include full statistics for all the underlying adapters. Trying to run **entstat** on the other interfaces will result in errors.

netstat

The **netstat** command is another tool that gathers useful information about the network. It does not provide detailed statistics about the adapters themselves but offers a lot of information about protocols and buffers. Example 5-72 on page 264 illustrates the use of **netstat** to check the network buffers.

netpmon

This tool traces the network subsystem and reports statistics collected. This tool is used to provide some information about processes using the network. Example 5-70 shows a sample taken from an **scp** session copying a thousand files with four megabytes. The *TCP Socket Call Statistics* reports **sshd** as the top process using the network. This is because the command was writing a lot of output to the terminal as the files were transferred.

Example 5-70 netpmon - sample output with Internet Socket Call i/O options (netpmon -O so)

```
# cat netpmon_multi.out
Wed Oct 10 15:31:01 2012
System: AIX 7.1 Node: p750s1aix5 Machine: 00F660114C00


========================================================================
```

```
TCP Socket Call Statistics (by Process):
----------------------------------------
                                  ------ Read -----   ----- Write -----
Process (top 20)             PID   Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
ssh                      7012600    119.66    960032    117.07      6653
sshd:                    7405786      0.37      6057     93.16     14559
------------------------------------------------------------------------
Total (all processes)               120.03    966089    210.23     21212

========================================================================

Detailed TCP Socket Call Statistics (by Process):
-------------------------------------------------

PROCESS: /usr//bin/ssh   PID: 7012600
reads:                   971
  read sizes (bytes):    avg 8023.3  min 1       max 8192    sdev 1163.4
  read times (msec):     avg 0.013   min 0.002   max 7.802   sdev 0.250
writes:                  950
  write sizes (bytes):   avg 56.8    min 16      max 792     sdev 25.4
  write times (msec):    avg 0.016   min 0.004   max 0.081   sdev 0.005

PROCESS: sshd:   PID: 7405786
reads:                   3
  read sizes (bytes):    avg 16384.0 min 16384   max 16384   sdev 0.0
  read times (msec):     avg 0.010   min 0.006   max 0.012   sdev 0.003
writes:                  756
  write sizes (bytes):   avg 156.3   min 48      max 224     sdev 85.6
  write times (msec):    avg 0.008   min 0.003   max 0.051   sdev 0.007

PROTOCOL: TCP (All Processes)
reads:                   974
  read sizes (bytes):    avg 8049.0  min 1       max 16384   sdev 1250.6
  read times (msec):     avg 0.013   min 0.002   max 7.802   sdev 0.250
writes:                  1706
  write sizes (bytes):   avg 100.9   min 16      max 792     sdev 77.8
  write times (msec):    avg 0.013   min 0.003   max 0.081   sdev 0.007
```

When **scp** is started with the **-q** flag to suppress the output, the reports are different. As shown in Example 5-71, the sshd daemon this time reports zero read calls and only a few write calls. As a result ssh experienced a gain from almost 30% on the read and write calls per second. This is an example of how the application behavior may change depending on how it is used.

Example 5-71 netpmon - sample output with Internet Socket Call i/O options (netpmon -O so)

```
# cat netpmon_multi.out
Wed Oct 10 15:38:08 2012
System: AIX 7.1 Node: p750s1aix5 Machine: 00F660114C00


========================================================================
```

```
TCP Socket Call Statistics (by Process):
----------------------------------------
                                  ------ Read -----   ----- Write -----
Process (top 20)             PID   Calls/s   Bytes/s   Calls/s   Bytes/s
------------------------------------------------------------------------
ssh                      7078142    155.33   1246306    152.14      8640
sshd:                    7405786      0.00         0      0.16        10
------------------------------------------------------------------------
Total (all processes)               155.33   1246306    152.30      8650

========================================================================

Detailed TCP Socket Call Statistics (by Process):
-------------------------------------------------

PROCESS: /usr//bin/ssh   PID: 7078142
reads:                   974
  read sizes (bytes):    avg 8023.8  min 1       max 8192    sdev 1161.6
  read times (msec):     avg 0.010   min 0.002   max 6.449   sdev 0.206
writes:                  954
  write sizes (bytes):   avg 56.8    min 16      max 792     sdev 25.4
  write times (msec):    avg 0.014   min 0.004   max 0.047   sdev 0.002

PROCESS: sshd:   PID: 7405786
writes:                  1
  write sizes (bytes):   avg 64.0    min 64      max 64      sdev 0.0
  write times (msec):    avg 0.041   min 0.041   max 0.041   sdev 0.000

PROTOCOL: TCP (All Processes)
reads:                   974
  read sizes (bytes):    avg 8023.8  min 1       max 8192    sdev 1161.6
  read times (msec):     avg 0.010   min 0.002   max 6.449   sdev 0.206
writes:                  955
  write sizes (bytes):   avg 56.8    min 16      max 792     sdev 25.3
  write times (msec):    avg 0.014   min 0.004   max 0.047   sdev 0.003
```

Tip: Additional information on tracing and **`netpmon`** can be found in Appendix A, “Performance monitoring tools and what they are telling us” on page 315.

5.7.2 Network buffers

Network memory in AIX is controlled by the *mbuf management facility*, which manages buckets of different buffer sizes ranging from 32 bytes to 16 kilobytes. The buckets are constrained on each processor forming a small subset of the entire mbuf pool.

The Virtual Memory Manager (VMM) allocates real memory to the pools. Therefore, the network buffers are pinned into the real memory and cannot be paged out. This behavior is good for network performance but also means that network-intensive workloads will consume more physical memory.

AIX automatically controls the *mbuf* allocation. The maximum pool size is represented by the `thewall` parameter, which in turn represents half the physical memory of the machine limited to a maximum of 65 GB. This parameter can be overridden by setting the maxmbuf tunable to a non-zero value (0 = disabled).

Important: We suggest to let the operating system manage the network buffers as much possible. Attempting to limit the maximum size of memory available for the network buffers can cause performance issues.

Example 5-72 illustrates the distribution of the mbuf pool along the processors CPU0, CPU3 and CPU15. Notice that CPU 0 has the highest number of buckets with different sizes (first column) and with some use. CPU 3 has a lower number with very low utilization and CPU 15 has only four, none of them used.

Example 5-72 netstat output - network memory buffers

```
# netstat -m | egrep -p "CPU (0|3|15)"
******* CPU 0 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
64                  663     86677      0        13     297    5240       0
128                 497     77045      0        14     271    2620       0
256                1482    228148      0        99     742    5240       0
512                2080  14032002      0       311    1232    6550       0
1024                279     11081      0       121     269    2620       0
2048                549      9103      0       284      53    3930       0
4096                 38       829      0        17       2    1310       0
8192                  6       119      0        12       1     327       0
16384               128       272      0        25      19     163       0
32768                29       347      0        23      22      81       0
65536                59       162      0        40       9      81       0
131072                3        41      0         0      43      80       0

******* CPU 3 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
64                    0      4402      0         0      64    5240       0
128                   1         9      0         0      31    2620       0
256                   2        20      0         0      14    5240       0
512                   2    519181      0         0     102    6550       0
2048                  2        21      0         0      10    3930       0
4096                  0        66      0         0      10    1310       0
131072                0         0      0         0      16      32       0

******* CPU 15 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
64                    0        23      0         0      64    5240       0
512                   0     31573      0         0      88    6550       0
4096                  0         0      0         0      20    1310       0
131072                0         0      0         0      16      32       0
```

5.7.3 Virtual I/O Server networking monitoring

In order to configure a network connection in a virtual environment using the VIO Server, first the link aggregation device should be created in terms of service continuity. The link aggregation device (ent6) is created using the following command (alternatively, you can use smitty Etherchannel from the root shell):

```
$ mkvdev -lnagg ent0,ent2 -attr mode=8023ad hash_mode=src_dsc_port
ent6 Available
```

After the link aggregation device has been created, the Shared Ethernet Adapter (SEA) can be configured. To create a SEA, use the following command:

```
$ mkvdev -sea ent6 -vadapter ent4 -default ent4 -defaultid 1
ent8 Available
```

Next, configure the IP address on the SEA with the following command:

```
$ mktcpip -hostname 'VIO_Server1' -inetaddr '10.10.10.15' –netmask '255.0.0.0'
interface 'en8
```

Before starting the transfer tests, however, reset all the statistics for all adapters on the Virtual I/O Server:

```
$ entstat -reset ent8 [ent0, ent2, ent6, ent4]
```

The **entstat -all** command can be used to provide all the information related to ent8 and all the adapters integrated to it, as shown in Example 5-73. All the values should be low because they have just been reset.

Example 5-73 entstat -all command after reset of Ethernet adapters

```
$ entstat -all ent8 |grep -E "Packets:|ETHERNET"
ETHERNET STATISTICS (ent8) :
Packets: 121                                  Packets: 111
                                              Bad Packets: 0
Broadcast Packets: 10                         Broadcast Packets: 10
Multicast Packets: 113                        Multicast Packets: 108
ETHERNET STATISTICS (ent6) :
Packets: 15                                   Packets: 97
                                              Bad Packets: 0
Broadcast Packets: 7                          Broadcast Packets: 0
Multicast Packets: 9                          Multicast Packets: 109
ETHERNET STATISTICS (ent0) :
Packets: 5                                    Packets: 87
                                              Bad Packets: 0
Broadcast Packets: 0                          Broadcast Packets: 0
Multicast Packets: 5                          Multicast Packets: 87
ETHERNET STATISTICS (ent2) :
Packets: 13                                   Packets: 6
                                              Bad Packets: 0
Broadcast Packets: 8                          Broadcast Packets: 0
Multicast Packets: 5                          Multicast Packets: 6
ETHERNET STATISTICS (ent4) :
Packets: 92                                   Packets: 9
                                              Bad Packets: 0
Broadcast Packets: 0                          Broadcast Packets: 8
Multicast Packets: 93                         Multicast Packets: 0
Invalid VLAN ID Packets: 0
Switch ID: ETHERNET0
```

You can see the statistics of the Shared Ethernet Adapter (ent8), the link aggregation device (ent6), the physical devices (ent0 and ent2), and the virtual Ethernet adapter (ent4) by executing the following commands:

```
ftp> put "| dd if=/dev/zero bs=1M count=100" /dev/zero
local: | dd if=/dev/zero bs=1M count=100 remote: /dev/zero
229 Entering Extended Passive Mode (|||32851|)
```

```
150 Opening data connection for /dev/zero.
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 8.85929 seconds, 11.8 MB/s
226 Transfer complete.
104857600 bytes sent in 00:08 (11.28 MB/s)
```

You can check which adapter was used to transfer the file. Execute the **entstat** command and note the number of packets, as shown in Example 5-74.

Example 5-74 entstat - all command after opening one ftp session

```
$ entstat -all ent8 |grep -E "Packets:|ETHERNET"
ETHERNET STATISTICS (ent8) :
Packets: 41336                                 Packets: 87376
                                               Bad Packets: 0
Broadcast Packets: 11                          Broadcast Packets: 11
Multicast Packets: 38                          Multicast Packets: 34
ETHERNET STATISTICS (ent6) :
Packets: 41241                                 Packets: 87521
                                               Bad Packets: 0
Broadcast Packets: 11                          Broadcast Packets: 0
Multicast Packets: 4                           Multicast Packets: 34
ETHERNET STATISTICS (ent0) :
Packets: 41235                                 Packets: 87561
                                               Bad Packets: 0
Broadcast Packets: 0                           Broadcast Packets: 0
Multicast Packets: 2                           Multicast Packets: 32
ETHERNET STATISTICS (ent2) :
Packets: 21                                    Packets: 2
                                               Bad Packets: 0
Broadcast Packets: 11                          Broadcast Packets: 0
Multicast Packets: 2                           Multicast Packets: 2
ETHERNET STATISTICS (ent4) :
Packets: 34                                    Packets: 11
                                               Bad Packets: 0
Broadcast Packets: 0                           Broadcast Packets: 11
Multicast Packets: 34                          Multicast Packets: 0
Invalid VLAN ID Packets: 0
Switch ID: ETHERNET0
```

Compared to the number of packets shown in Example 5-74, see that the number increased after the first file transfer.

To verify network stability, you can also use **entstat** (Example 5-75). Confirm all errors, for example, transmit errors, receive errors, CRC errors, and so on.

Example 5-75 entstat shows various items to verify errors

```
$ entstat ent8
-------------------------------------------------------------
ETHERNET STATISTICS (ent8) :
Device Type: Shared Ethernet Adapter
Hardware Address: 00:21:5e:aa:af:60
Elapsed Time: 12 days 4 hours 25 minutes 27 seconds

Transmit Statistics:                           Receive Statistics:
```

```
-------------------                             ------------------
Packets: 64673155                               Packets: 63386479
Bytes: 65390421293                              Bytes: 56873233319
Interrupts: 0                                   Interrupts: 12030801
Transmit Errors: 0                              Receive Errors: 0
Packets Dropped: 0                              Packets Dropped: 0
                                                Bad Packets: 0

Max Packets on S/W Transmit Queue: 56
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 23

Broadcast Packets: 5398                         Broadcast Packets: 1204907
Multicast Packets: 3591626                      Multicast Packets: 11338764
No Carrier Sense: 0                             CRC Errors: 0
DMA Underrun: 0                                 DMA Overrun: 0
Lost CTS Errors: 0                              Alignment Errors: 0
Max Collision Errors: 0                         No Resource Errors: 0
Late Collision Errors: 0                        Receive Collision Errors: 0
Deferred: 0                                     Packet Too Short Errors: 0
SQE Test: 0                                     Packet Too Long Errors: 0
Timeout Errors: 0                               Packets Discarded by Adapter: 0
Single Collision Count: 0                       Receiver Start Count: 0
Multiple Collision Count: 0
Current HW Transmit Queue Length: 23

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 2000
Driver Flags: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        LargeSend DataRateSet
```

Advanced SEA monitoring

To use the SEA monitoring tool (**seastat**), first enable the tool as follows:

```
$ chdev -dev ent8 -attr accounting=enabled
ent8 changed
```

Example 5-76 shows SEA statistics without any search criterion. Therefore, it displays statistics for all clients that this Virtual I/O Server is serving.

Example 5-76 Sample seastat statistics

```
$ seastat -d ent8
================================================================================
Advanced Statistics for SEA
Device Name: ent8
================================================================================
MAC: 6A:88:82:AA:9B:02
----------------------
VLAN: None
VLAN Priority: None
Transmit Statistics:                Receive Statistics:
--------------------                -------------------
```

```
Packets: 7                          Packets: 2752
Bytes: 420                          Bytes: 185869
=============================================================================
MAC: 6A:88:82:AA:9B:02
---------------------
VLAN: None
VLAN Priority: None
IP: 9.3.5.115
Transmit Statistics:                Receive Statistics:
--------------------                -------------------
Packets: 125                        Packets: 3260
Bytes: 117242                       Bytes: 228575
=============================================================================

This command will show an entry for each pair of VLAN, VLAN priority, IP-address,
and MAC address. So, you will notice in Example there are two entries for several
MAC addresses. One entry is for MAC address and the other one is for the IP
address configured over that MAC
```

5.7.4 AIX client network monitoring

On the AIX virtual I/O client, you can use the **entstat** command to monitor a virtual Ethernet adapter, as shown in the preceding examples. It can also be used to monitor a physical Ethernet adapter.

5.8 Performance analysis at the CEC

This section gives an overview of monitoring a Power system at the Central Electronics Complex (CEC) level. The Hardware Management Console (HMC) helps to connect with multiple Power servers and to perform administrative tasks both locally and remotely. Using the LPAR2RRD tool you can monitor all Power servers connected to the HMC and their respective LPARs. Install LPAR2RRD on an LPAR and configure it in such a way that it communicates with the HMC using password-less authentication.

Tip: LPAR2RRD and the detailed installation and configuration of the tool are available at:

http://lpar2rrd.com/

Figure 5-12 on page 269 shows the LPAR2RRD monitoring features list and history details.

Figure 5-12 lpar2rrd - monitoring features

Figure 5-13 shows the processor pool graph of one of the servers connected with the HMC that is being monitored by the LPAR2RRD tool.

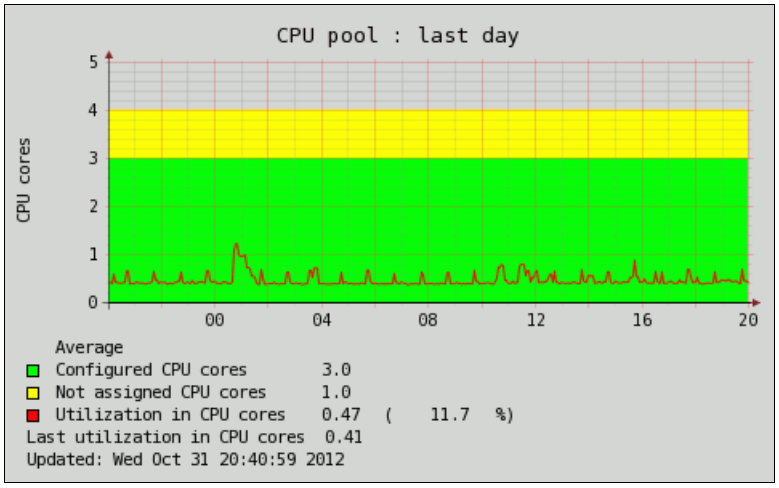


Figure 5-13 lpar2rrd - Processor pool graph

Figure 5-14 on page 270 shows the LPAR's aggregated graph for the server. Figure 5-15 on page 270 shows an LPAR-specific processor usage graph, which shows only the last day graphs, but the tool provides the last week, the last four weeks and the last year graphs as well. The historical reports option provides a historical graph of certain time periods. Figure 5-16 on page 270 shows the historical reports for the memory usage of the last two days.

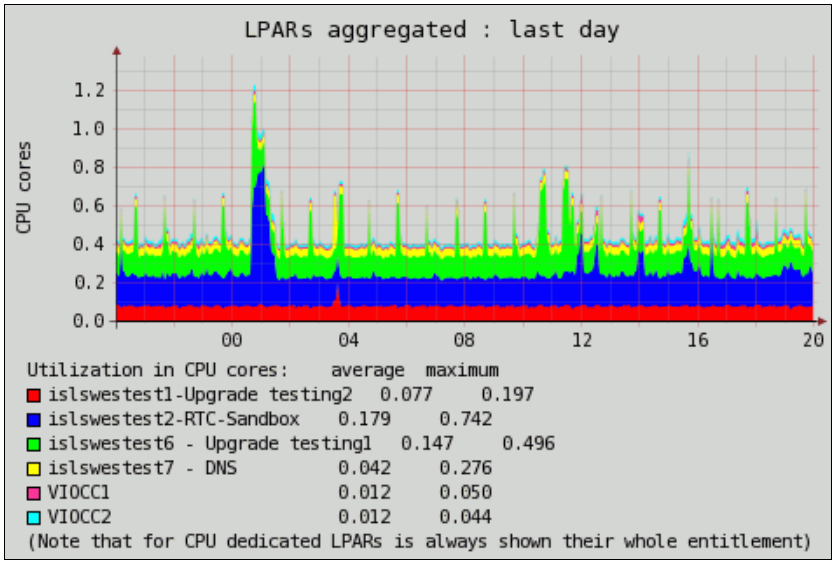


Figure 5-14 lpar2rrd - multiple partitions graph

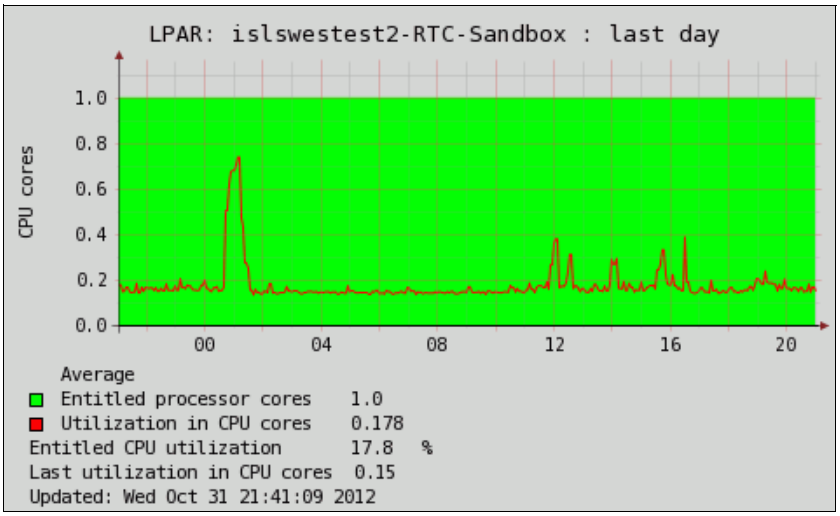


Figure 5-15 lpar2rrd - single partition graph

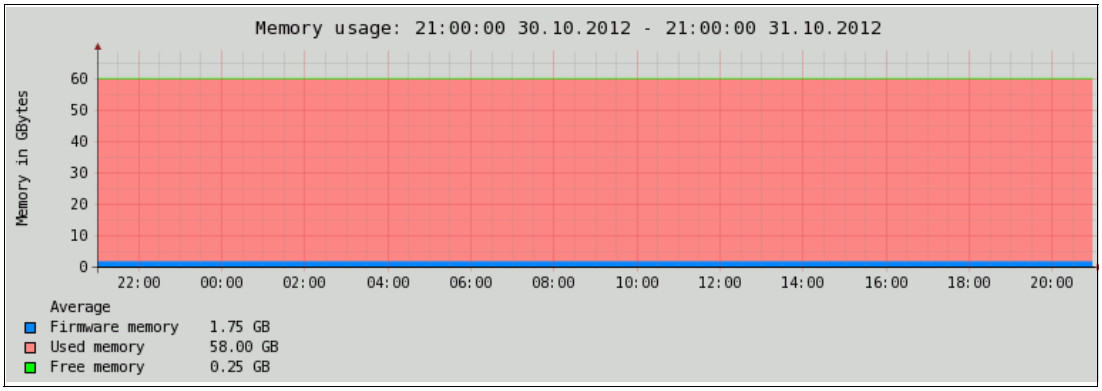


Figure 5-16 lpar2rrd - memory statistics of the past two days

The **lpar2rrd** tool uses the native HMC tool **lslparutil** to capture data for analysis. As an alternate, the command can also be used from the HMC to list the utilization data. But to visualize the utilization results in graphic form, LPAR2RRD would be a preferred method.

5.9 VIOS performance advisor tool and the part command

In VIOS 2.2.2.0 and later, the VIOS performance advisor tool has been imbedded into the VIOS code. The VIOS performance advisor tool summarizes the health of a given VIOS; even where a pair exists, they are handled individually. The advisor can identify bottlenecks and provide recommendations by polling key performance metrics and providing a report in an XML format.

The performance analysis and reporting tool (**part**) is included in the VIOS restricted shell, and can be executed in two different modes:

- Monitoring mode - The **part** tool is executed for a period of time between 10 and 60 minutes. This collects the data for the period of time it is run for, at the point of time that you run it.
- Post processing mode - The **part** tool is executed against a previously run nmon recording.

The final report, inclusive of all required files to view the report, is combined into a .tar file that can be downloaded and extracted into your PC.

The processor overhead of running the tool on a VIOS is the same as that of collecting nmon data, and the memory footprint is kept to a minimum.

5.9.1 Running the VIOS performance advisor in monitoring mode

Example 5-77 demonstrates running the VIOS performance advisor in monitoring mode for a period of 10 minutes.

Example 5-77 Running the VIOS performance advisor in monitoring mode

```
$ part -i 10
part: Reports are successfully generated in p24n27_120928_13_34_38.tar
$ pwd
/home/padmin
$ ls /home/padmin/p24n27_120928_13_34_38.tar
/home/padmin/p24n27_120928_13_34_38.tar
$
```

The tar file `p24n27_120928_13_34_38.tar` is now ready to be copied to your PC, extracted, and viewed.

5.9.2 Running the VIOS performance advisor in post processing mode

Running the VIOS performance advisor in post processing mode requires that the VIOS is already collecting nmon recordings. To configure the VIOS to capture nmon recordings, first create a logical volume and file system, as shown in Example 5-78 on page 272.

Depending on how long you want to store the nmon recordings and how many devices are attached to your VIOS, you may need a larger file system (Example 5-78 on page 272).

Example 5-78 Create a jfs2 file system for nmon recordings

```
$ oem_setup_env
# mklv -y nmon_lv -t jfs2 rootvg 1 hdisk0
nmon_lv
# crfs -v jfs2 -d nmon_lv -m /home/padmin/nmon -a logname=INLINE -A yes
File system created successfully.
64304 kilobytes total disk space.
New File System size is 131072
# chfs -a size=5G /home/padmin/nmon
Filesystem size changed to 10485760
Inlinelog size changed to 20 MB.
# mount /home/padmin/nmon
# df -g /home/padmin/nmon
Filesystem    GB blocks      Free %Used    Iused %Iused Mounted on
/dev/nmon_lv       5.00      4.98    1%        4     1% /home/padmin/nmon
# exit
$
```

Once the file system is created, the next step is to edit the root crontab. Example 5-79 demonstrates how to do this.

Example 5-79 How to edit the root crontab on a Virtual I/O server

```
$ oem_setup_env
# crontab -e
```

Example 5-80 shows two sample crontab entries. One entry is to record daily nmon statistics, while the other entry is to remove the nmon recordings after 60 days. Depending on how long you require nmon recordings to be stored, you may need to have a crontab entry to remove them after a different period of time. You need to manually insert entries into your root crontab.

Example 5-80 Sample crontab to capture nmon recordings and remove them after 60 days

```
# Start NMON Recording
00 00 * * * /usr/bin/nmon -dfOPt -^ -m /home/padmin/nmon
# Remove NMON Recordings older than 60 Days
01 00 * * * /usr/bin/find /home/padmin/nmon -name "*.nmon" -type f -mtime +60 !
-name "*hardened*" |xargs -n1 /bin/rm -f
```

Example 5-81 demonstrates how to process an existing nmon recording using the part tool. This consists of locating an nmon recording in `/home/padmin/nmon`, where you are storing them, and running the part tool against it. The resulting tar file can be copied to your PC, extracted, and opened with a web browser.

Example 5-81 Processing an existing nmon recording

```
$ part -f /home/padmin/nmon/p24n27_120930_0000.nmon
part: Reports are successfully generated in p24n27_120930_0000.tar
$
```

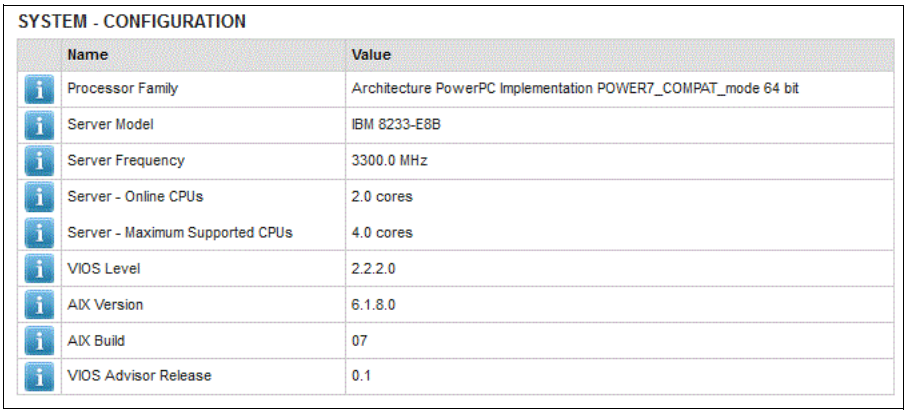
The tar file is now ready to be copied to your PC, extracted and viewed.

5.9.3 Viewing the report

Once you have the tar file copied to your PC, extract the contents and open the file `vios_advisor_report.xml` to view the report.

Once it is open, you see a number of sections, including a summary of the system configuration, processor configuration and usage, memory configuration and usage, as well as I/O device configuration and usage.

Figure 5-17 shows the system configuration section of the VIOS performance advisor.

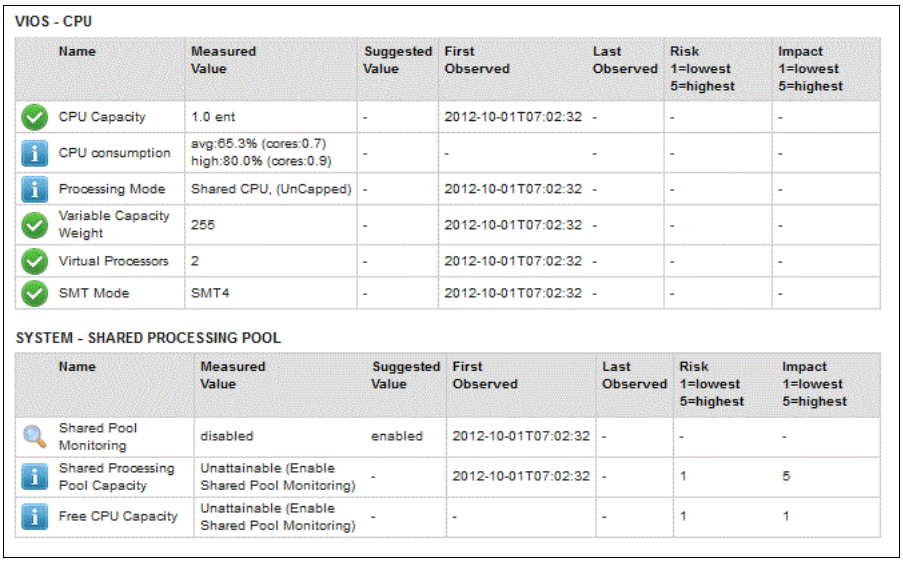


SYSTEM - CONFIGURATION

Name	Value
Processor Family	Architecture PowerPC Implementation POWER7_COMPAT_mode 64 bit
Server Model	IBM 8233-E8B
Server Frequency	3300.0 MHz
Server - Online CPUs	2.0 cores
Server - Maximum Supported CPUs	4.0 cores
VIOS Level	2.2.2.0
AIX Version	6.1.8.0
AIX Build	07
VIOS Advisor Release	0.1

Figure 5-17 System configuration summary

Figure 5-18 shows the processor summary from the report. You can click any of the sections to retrieve an explanation of what the VIOS advisor is telling you, why it is important, and how to modify if there are problems detected.



VIOS - CPU

Name	Measured Value	Suggested Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
CPU Capacity	1.0 ent	-	2012-10-01T07:02:32	-	-	-
CPU consumption	avg:65.3% (cores:0.7) high:80.0% (cores:0.9)	-	-	-	-	-
Processing Mode	Shared CPU, (UnCapped)	-	2012-10-01T07:02:32	-	-	-
Variable Capacity Weight	255	-	2012-10-01T07:02:32	-	-	-
Virtual Processors	2	-	2012-10-01T07:02:32	-	-	-
SMT Mode	SMT4	-	2012-10-01T07:02:32	-	-	-

SYSTEM - SHARED PROCESSING POOL

Name	Measured Value	Suggested Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
Shared Pool Monitoring	disabled	enabled	2012-10-01T07:02:32	-	-	-
Shared Processing Pool Capacity	Unattainable (Enable Shared Pool Monitoring)	-	2012-10-01T07:02:32	-	1	5
Free CPU Capacity	Unattainable (Enable Shared Pool Monitoring)	-	-	-	1	1

Figure 5-18 Processor summary from the VIOS performance advisor

Figure 5-19 on page 274 shows the memory component of the VIOS Advisor report. If the VIOS performance advisor detects that more memory is to be added to the VIOS partition, it suggests the optimal amount of memory.

VIOS - MEMORY

Name	Measured Value	Suggested Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
Real Memory	4.000 GB	-	2012-10-01T07:02:32	-	-	-
Available Memory	2.569 GB	-	-	-	-	-
Paging Rate	10.0 MB/s Paging Rate	No Paging	2012-10-01T07:02:28	2012-10-01T07:02:43	-	-
Paging Space Size	1.500 GB	-	2012-10-01T07:02:32	-	-	-
Free Paging Space	1.492 GBfree	-	-	-	-	-
Pinned Memory	0.696 GB pinned	-	-	-	-	-

Figure 5-19 Memory summary

Figure 5-20 shows the disk and I/O summary. This shows the average amount of I/O and the block size being processed by the VIOS. It shows the amount of FC Adapters and their utilization.

Note: If the FC port speed is not optimal, it is possible that the FC adapter is attached to a SAN fabric switch that is either not capable of the speed of the FC adapter, or the switch ports are not configured correctly.

VIOS - I/O ACTIVITY

Name	Value
Disk I/O Activity	avg: 19007 iops @ 8.47 KB peak: 29072 iops @ 8KB
Network I/O Activity	[avgSend: 0 iops 0.0 MBps , avgRcv: 0 iops 0.0MBps] [peakSend: 0 iops 0.0MBps , peakRcv: 0 iops 0.0MBps]

VIOS - DISK ADAPTERS

Name	Measured Value	Suggested Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
FC Adapter Count	2	-	2012-10-01T07:02:32	-	-	-
FC Avg IOps	avg: 19007 iops @ 7KB	-	2012-10-01T07:02:32	2012-10-01T07:22:29	-	-
FC Idle Port : (fcs1)		-	2012-10-01T07:02:32	2012-10-01T07:22:29	4	4
FC Adapter Utilization	optimal	-	-	-	-	-
FC Port Speeds	running at speed	-	-	-	-	-

VIOS - DISK DRIVES

Name	Measured Value	Suggested Value	First Observed	Last Observed	Risk 1=lowest 5=highest	Impact 1=lowest 5=highest
Physical Drive Count	6	-	2012-10-01T07:02:32	-	-	-
I/Os Blocked	pass	-	-	-	-	-
Long I/O Latency	pass	-	-	-	-	-

Figure 5-20 I/O and disk summary

If you click on the icon to the right of any item observed by the VIOS performance advisor, it provides a window, as shown in Figure 5-21 on page 275. This gives a more detailed description of what observation the VIOS performance advisor has made. The example shown in the figure shows an FC adapter that is unused, and the suggestion is to ensure that I/O is balanced across the available adapters. In an NPIV scenario, it could be that there are no LPARs mapped yet to this particular port.

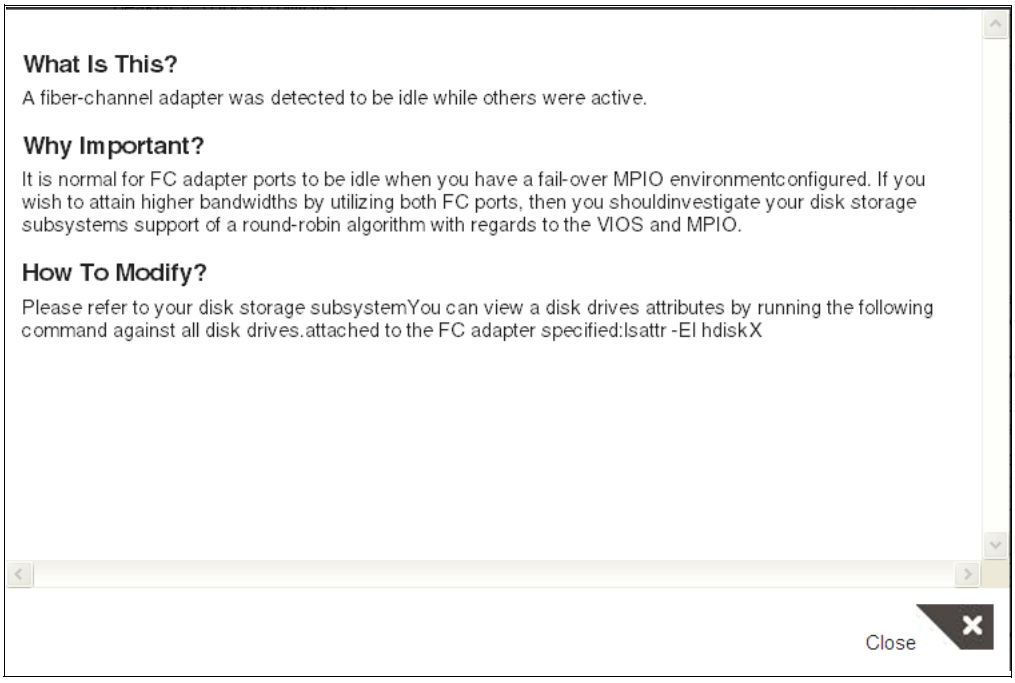


Figure 5-21 Example of VIOS performance advisor recommendation

5.10 Workload management

One of the reasons why many systems have performance problems is because of poor workload distribution.

All planned activities can be better managed by following certain workload management techniques. This also helps to avoid bottleneck situations. There is work that can wait for a while. For instance, a report that needs to be generated for the next morning can be started at 5 p.m. or at 5 a.m. The difference is that during night the processor is probably idle. The critical data backup can also be initiated during the night to better manage the resources.

This type of workload management is provided by many different third-party software vendors, but the operating system itself has tools that may help before investing in such tools.

The cron daemon can be used to organize all planned workloads by running at different times. Use the **at** command to take advantage of the capability or set up a `crontab` file.

Using **job queue** is another way of workload management where the programs or procedures can be executed sequentially.

Submitting an environment for performance analysis is most of the time a complex task. It usually requires a good knowledge of the workloads running, system capacity, technologies available, and involves a lot of tuning and testing.

To understand the limits of the components of the environment is crucial for establishing targets and setting expectations.

Example 5-82 - qdaemon - configuration example

```
* BATCH queue for running shell scripts
bsh:
       device = bshdev
```

```
        discipline = fcfs

bshdev:
        backend = /usr/bin/bsh
```

In Example 5-82 on page 275, we define a **bsh** queue that uses `/usr/bin/bsh` as backend. The backend is the program that is called by qdaemon.

The queue can be reduced by putting the jobs in the queue during the day and starting the queue up during the night using the commands shown in Example 5-83.

Example 5-83 - qdaemon - usage example

```
To bring the Queue down

# qadm -D bsh

To put the jobs in queue

# qprt -P bsh script1
# qprt -P bsh script2
# qprt -P bsh script3

To start the queue during night

# qadm -U bsh
```

When starting the queue during the night, the jobs will be executed sequentially.

Example 5-84 illustrates the use of queues to run a simple script with different behavior depending on the status of its control file. At first, ensure that the queue is down and some jobs are added to the queue. Next, the **qchk** output shows that our queue is down and has four jobs queued. When the queue is brought up, the jobs will run, all in sequence, sending output data to a log file. At last, with the queue still up, the job is submitted two more times. Check the timestamps of the log output.

Example 5-84 qdaemon - using the queue daemon to manage jobs

```
# qadm -D bsh
# qprt -P bsh /tests/job.sh
# qprt -P bsh /tests/job.sh
# qprt -P bsh /tests/job.sh
# qprt -P bsh /tests/job.sh
# qchk -P bsh
Queue   Dev   Status    Job Files              User         PP %   Blks  Cp Rnk
------- ----- --------- --- ------------------ ---------- ---- -- ----- --- ---
bsh     bshde DOWN
              QUEUED     20 /tests/job.sh      root                1   1   1
              QUEUED     21 /tests/job.sh      root                1   1   2
              QUEUED     22 /tests/job.sh      root                1   1   3
              QUEUED     23 /tests/job.sh      root                1   1   4
# qadm -U bsh
# qchk -P bsh
Queue   Dev   Status    Job Files              User         PP %   Blks  Cp Rnk
```

```
------- ----- --------- --- ----------------- ---------- ---- -- ----- --- ---
bsh     bshde READY
# cat /tmp/jobctl.log
[23/Oct/2012] - Phase: [prepare]
[23/Oct/2012] - Phase: [start]
[23/Oct/2012-18:24:49] - Phase: [finish]
Error
[23/Oct/2012-18:24:49] - Phase: []
[23/Oct/2012-18:24:49] - Phase: [prepare]
[23/Oct/2012-18:24:49] - Phase: [start]
[23/Oct/2012-18:27:38] - Creating reports.
[23/Oct/2012-18:27:38] - Error
[23/Oct/2012-18:27:38] - Preparing data.
[23/Oct/2012-18:27:38] - Processing data.
# qchk -P bsh
Queue   Dev   Status    Job Files              User         PP %   Blks  Cp Rnk
------- ----- --------- --- ----------------- ---------- ---- -- ----- --- ---
bsh     bshde READY
# qprt -P bsh /tests/job.sh
# qprt -P bsh /tests/job.sh
# tail -3 /tmp/jobctl.log
# tail -3 /tmp/jobctl.log
[23/Oct/2012-18:27:38] - Processing data.
[23/Oct/2012-18:28:03] - Creating reports.
[23/Oct/2012-18:33:38] - Error
```

Using this queueing technique to manage the workload can be useful to prevent some tasks running in parallel. For instance, it may be desired that the backups start only after all nightly reports are created. So instead of scheduling the reports and backup jobs with the cron daemon, you can use the queue approach and schedule only the queue startup within the crontab.

6

Application optimization

In this chapter we discuss application optimization, including the following topics:

- Optimizing applications with AIX features
- Application side tuning
- IBM Java Support Assistant

6.1 Optimizing applications with AIX features

Even if you have followed all the advice in 2.2.4, “Optimizing the LPAR resource placement” on page 18, you can still have an LPAR composed by some local and remote memory for several reasons:

- Your LPAR placement is optimized, but your LPAR is too big to be contained within a single chip or node.
- Your LPAR is not critical (or the last created) and has to run on the free resources available but scattered in the system.

Fortunately, AIX can help to limit the effect of the NUMA architecture (2.2.1, “Power Systems and NUMA effect” on page 10).

This section provides some explanations about:

- The default behavior of the AIX scheduler with POWER7 architecture.
- How AIX can help to localize a process to a specific set of resources (RSET).
- How AIX can help to automatically tune your system for an application (Dynamic System Optimizer).

6.1.1 Improving application memory affinity with AIX RSETs

The AIX operating system has been optimized to run on the POWER7 architecture. AIX 6.1 TL5 and AIX 7.1 are aware of the system topology at boot time, and dynamically aware since AIX 6.1 TL8 or AIX 7.1 TL2. This topology information can be visualized with the **lssrad** command (Example 6-1).

Example 6-1 lssrad output example

```
{D-PW2k2-lpar1:root}/ # lssrad -av
REF1   SRAD        MEM CPU
0
          0   15662.56      0-15
1
          1   15857.19      16-31
```

The AIX scheduler uses the Scheduler Resource Allocation Domain Identifier (SRADID) to try to redispatch the thread on a core as close as possible to its previous location in this order:

1. Redispatch the thread on the same core to keep L2, L3, and memory data.
2. Redispatch the thread on another core but in the same POWER7 chip. Some data can be retrieved through a remote L3 cache, and memory affinity is preserved.
3. Redispatch the thread on another core, in another POWER7 chip but in the same node. In this case, memory affinity is lost and data has to be retrieved with a *remote* (near) memory access.
4. Redispatch the thread on another core, in another POWER7 chip and in another node (for Power 770 or above only). In this case, memory affinity is lost and data has to be retrieved with a *distant* (far) memory access.

In AIX 6.1 TL5 and later and AIX 7.1, the behavior is the same as in a virtualized Shared Processor Logical Partition (SPLPAR) if the restricted **vmo** parameter enhanced_memory_affinity is set to 1 (default value). Refer to Example 6-2.

Example 6-2 Checking vmo parameter enhanced_memory_affinity

```
{D-PW2k2-lpar1:root}/ # vmo -FL enhanced_memory_affinity
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
------------------------------------------------------------------------------
enhanced_memory_affinity  1      1      1      0      1      boolean          B
------------------------------------------------------------------------------
```

Memory affinity

Beside the enhanced_memory_affinity **vmo** parameter, AIX 6.1 TL5 and AIX 7.1 bring another restricted parameter called enhanced_affinity_private. Refer to Example 6-3.

Example 6-3 Checking the vmo parameter enhanced_affinity_private

```
{D-PW2k2-lpar1:root}/ # vmo -FL enhanced_affinity_private
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
------------------------------------------------------------------------------
enhanced_affinity_private 40     40     40     0      100    numeric          D
------------------------------------------------------------------------------
```

This parameter helps to get more memory local to the chip where the application is running. It is a percentage. By default it is set to 40 (for AIX 6.1 TL6 and beyond, only 20 for AIX 6.1 TL5) which means 40% of the memory allocated by a process is localized and the other 60% is equally spread across *all* the memory.

The configuration described in Example 6-1 has two SRADID (two POWER7 chips with memory). If a process starts in a core of chip1 and allocates some memory, 40% will be allocated next to chip1, and the 60% left spread equally across SRADID 0 and 1. So the final allocation is 40% + 30% next to chip1 and 30% next to chip2.

Note: A few lines of advice:

- **enhanced_affinity_private** is a restricted tunable parameter and must not be changed unless recommended by IBM Support.
- This is an advisory option, not compulsory. For large memory allocation, the system might need to balance the memory allocation between different vmpools. In such cases, the locality percentage cannot be ensured. However, you are still able to set **enhanced_affinity_vmpool_limit**=-1 to disable the balancing.
- Shared memory is not impacted by the **enhanced_affinity_private** parameter. This kind of memory allocation is controlled by **memplace_shm_named** and **memplace_shm_anonymous**.

You can force a process to allocate all its memory next to the chip where it is running without changing the vmo tunable **enhanced_affinity_private.** This can be done by exporting the variable MEMORY_AFFINITY with a value set to MCM before starting your process (Example 6-4).

Example 6-4 Forcing the start_program memory to be allocated locally

```
{D-PW2k2-lpar1:root}/ #export MEMORY_AFFINITY=MCM
{D-PW2k2-lpar1:root}/ #./start_program.ksh
```

But, even if the AIX scheduler tries to redispatch a thread on the same core, there is no guarantee, and some threads can migrate to another core or chip during its runtime (Figure 6-1).

You can monitor the threads' migration with an AIX command such as **topas -M** (Figure 6-2 on page 283).

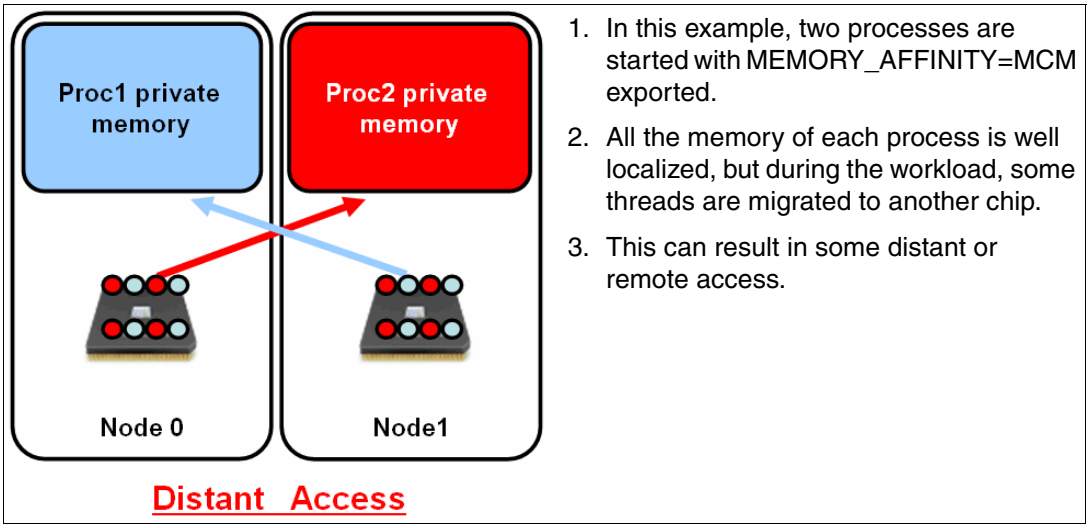


Figure 6-1 Distant/remote access with MEMORY_AFFINITY=MCM

topas -M (Figure 6-2 on page 283) gives you the following output divided into two parts:

- The first part gives you the processor and memory topology (**lssrad**). It also gives the amount of memory usage for each memory domain.
- The second part gives you the amount of threads dispatched per logical processor with a repartition Local, Near, and Far.
  - **LocalDisp%** is the percentage of threads redispatched from the same chip.
  - **NearDisp%** is the percentage of threads redispatched from another chip in the same node.
  - **FarDisp%** is the percentage of threads redispatched from another chip from another node.

```
Topas Monitor for host: D-PW2k2-lpar1  Interval: 2      Mon Oct 15 23:31:42 2012
================================================================================
REF1     SRAD  TOTALMEM  INUSE    FREE    FILECACHE  HOMETHRDS  CPUS
--------------------------------------------------------------------------------
0           0     24.0G  17.2G   6.89G        99.7M      519.0   0-31
1           1     24.2G  17.4G   6.79G        97.6M      459.0   32-63
================================================================================
CPU      SRAD  TOTALDISP    LOCALDISP%  NEARDISP%   FARDISP%
------------------------------------------------------------
0           0      178.0       100.0        0.0        0.0
2           0      142.0       100.0        0.0        0.0
1           0      138.0       100.0        0.0        0.0
33          1      129.0        99.2        0.0        0.8
34          1      129.0        99.2        0.0        0.8
35          1      127.0        99.2        0.0        0.8
3           0      70.00       100.0        0.0        0.0
22          0      68.00       100.0        0.0        0.0
32          1      49.00        98.0        0.0        2.0
56          1      38.00        97.4        0.0        2.6
8           0      36.00       100.0        0.0        0.0
11          0      33.00       100.0        0.0        0.0
9           0      29.00       100.0        0.0        0.0
10          0      29.00       100.0        0.0        0.0
28          0      12.00       100.0        0.0        0.0
40          1      10.00        90.0        0.0       10.0
36          1       6.00        83.3        0.0       16.7
44          1       6.00        83.3        0.0       16.7
4           0       5.00       100.0        0.0        0.0
20          0       4.00       100.0        0.0        0.0
37          1       3.00        66.7        0.0       33.3
16          0       3.00       100.0        0.0        0.0
42          1       3.00        66.7        0.0       33.3
23          0       2.00       100.0        0.0        0.0
Page: 1/3   Use 'PgDn/PgUp' to move to next/previous page
```

Figure 6-2 topas -M output

So, exporting the variable MEMORY_AFFINITY= MCM is not enough to maintain a good processor and memory affinity. You need to be sure that threads will stay in the domain where their memory is located and avoid migration to another domain. To do this, you can use the AIX Resource Set (RSET).

The AIX RSET enables system administrators to define and name a group of resources such as a logical processor. This service comes with its own set of commands, as described in Table 6-1.

Table 6-1 RSET set of AIX commands

mkrset[a]	Create and name an RSET.
rmrset[a]	Delete the RSET.
lsrset[a]	List available RSETs.
execrset[a]	Execute a process inside an RSET.
attachrset[a]	Place a process inside an RSET after its execution.
detachrset[a]	Detach a process from an RSET.

a. Refer to *AIX 7.1 Difference Guide*, SG24-7910 for more details.

Note: To be able to use the RSET commands, a non-root user needs to have the following capabilities: CAP_NUMA_ATTACH,CAP_PROPAGATE (Example 6-5).

```
chuser capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE <username>
```

Example 6-5 Adding and checking NUMA control capabilities of a user

```
{D-PW2k2-lpar1:root}/ # chuser capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE bruce
{D-PW2k2-lpar1:root}/ # lsuser -a capabilities bruce
```

```
bruce capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE
```

A root or a non-root user with NUMA capabilities can create an RSET, and execute a program within this RSET with the **mkrest** and **execrset** commands, as shown in Example 6-6 to Example 6-8.

Example 6-6 Creating RSET named test/0 with processor 0 to 15

```
{D-PW2k2-lpar1:root}/ #mkrset -c 0-15 test/0
1480-353 rset test/0 created
```

- Execute a program in the RSET test/0 (Example 6-7)

```
{D-PW2k2-lpar1:root}/ #execrset test/0 -e ./start_program.ksh
```

Example 6-7 Checking RSET defined in a system

```
D-PW2k2-lpar1:root}/ #lsrset -av
T  Name              Owner     Group     Mode    CPU  Memory
...(lines omitted)...
a  test/0            root      system    rwr-r-   16       0
     CPU: 0-15
     MEM: <empty>
```

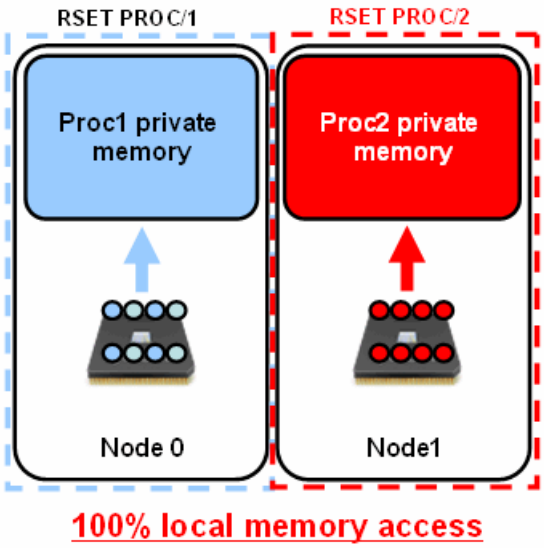
Example 6-8 Executing “start_progrm.ksh” within test/0 RSET

```
{D-PW2k2-lpar1:root}/ #execrset test/0 -e ./start_program.ksh
```

When a process is attached to an RSET, it can only run on the logical processor that composed this RSET. It is like the **bindprocessor** command, but RSET has the advantage to bind a process and its children to a group of logical processors that allow us to:

- Bind a process to a core and let AIX manage the SMT thread as usual. This creates an RSET with the four logical processors (if SMT4) and runs the process inside the RSET with **execrset**.
- Bind a process to all the cores of a node for a very large LPAR. This can limit the number of distant memory accesses.

Let us continue the example illustrated by Figure 6-1 on page 282. We have a system with two chips and two processes. We create two RSETs based on the information given by the **lssrad -av** command.



With this RSET configuration, each process and its threads are bound to one chip.

Because on the exported variable MEMORY_AFFINITY=MCM, all the memory will be allocated near to the chip where the correspondent threads are running.

This RSET configuration helps to keep the threads running on their HOME chip, which maintains the memory affinity near 100%.

Figure 6-3 RESETs + MEMORY_AFFINITY=MCM configuration

Note: Unlike `enhanced_affinity_private`, setting MEMORY_AFFINITY=MCM will cause the allocation of shared memory to be local too.

However, the vmo parameter `enhanced_affinity_vmpool_limit` still applies for MEMORY_AFFINITY=MCM. Thus some memory might be allocated in other vmpools in case of large memory allocation, if local allocation causes vmpool imbalance that exceeds the threshold set by `enhanced_affinity_vmpool_limit`.

If you have multiple RSETs accessing the same shared memory regions, setting MEMORY_AFFINITY=MCM@SHM=RR should be a better choice than MEMORY_AFFINITY=MCM.

To really take advantage of RSET + MEMORY_AFFINITY tuning, an application must be *"Multi-instances and share-nothing."* This means that the application can be divided into several independent instances with their own "private" memory.

Good candidates: Multi-instances Java applications, DB2 DPF, Informix® XDS.

Bad candidates: DB2 UDB, Informix IDS, all big application single instance multiprocesses and large shared memory.

Testing RSET and MEMORY_AFFINITY improvement on a memory-sensitive application

To run this test, we wrote a small C program that searched a specific pattern into memory. This program is named latency; a full description can be found in ""latency" test for RSET, ASO and DSO demo program illustration" on page 347.

The system used for this test was a POWER 780 MHB, four drawers, 64 cores, firmware AM730_095 with one LPAR, 16 cores and 50 GB of memory, AIX 7.1 TL1 SP4. Refer to Example 6-9.

Example 6-9 Affinity test LPAR: lssrad output

```
{D-PW2k2-lpar1:root}/ # lssrad -av
```

```
REF1    SRAD        MEM      CPU
0
           0    24624.25      0-31
1
           1    24787.69      32-63
```

On this LPAR, we ran two tests:

- For the first test (test A), we started two processes searching patterns in their own private 16 GB of memory. Each process has 30 threads. We let these processes run for two hours without any memory affinity tuning (Figure 6-4). Processor usage of the LPAR was around 100% during the test. Refer to Example 6-10

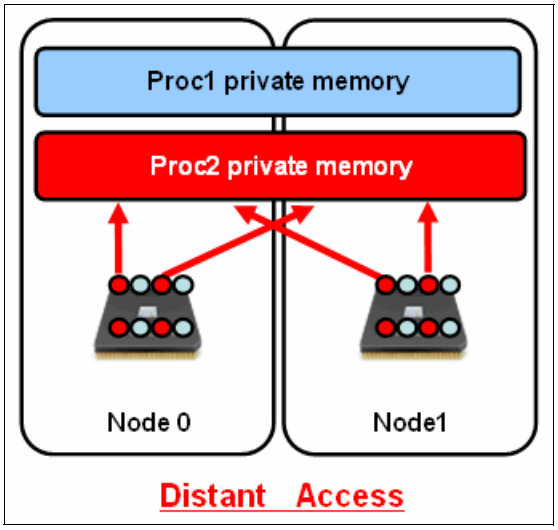


Figure 6-4 Two processes working without affinity tuning

Example 6-10 Starting test A without RSET and MEMORY_AFFINITY

```
{D-PW2k2-lpar1:root}/ # ./proc1/latency4 16384 1024 30 7200 &
[3] 18153958
{D-PW2k2-lpar1:root}/ # ./proc2/latency4 16384 1024 30 7200 &
[4] 5701778
```

- For the second test (test B), we restarted two *latency* programs, but with MEMORY_AFFINITY=MCM and RSET configuration as shown in Example 6-13.
  - Before creating the RSETs, we checked the processor and memory topology of our LPAR with **lssrad** (Example 6-9).
  - The **lssrad** output shows two chips with eight cores each (SMT4). We created two RSETs, one called *proc/1* with logical processors 0-31 and the other *proc/2* with 32-63 as described in Example 6-11.

    Example 6-11 Creating AIX RSET for test B

    ```
    {D-PW2k2-lpar1:root}/ #mkrset -c 0-31 proc/1
    1480-353 rset proc/1 created
    {D-PW2k2-lpar1:root}/ #mkrset -c 31-63 proc/2
    1480-353 rset proc/2 created
    ```

    Before switching to the next step, we checked our RSET configuration with the **lsrset** command, as shown in Example 6-12.

Example 6-12 Checking AIX RSET configuration

```
{D-PW2k2-lpar1:root}/ # lsrset -av
T  Name              Owner     Group     Mode    CPU  Memory
...(lines omitted)...
a  proc/1            root      system    rwr-r-   32       0
     CPU: 0-31
     MEM: <empty>

a  proc/2            root      system    rwr-r-   32       0
     CPU: 32-63
     MEM: <empty>
```

– Now, our AIX RSET configuration was done and we could export MEMORY_AFFINITY=MCM and start our two processes in the created RSETs (Example 6-13).

Example 6-13 Starting two latency programs in RSETs with MEMORY_AFFINITY=MCM

```
{D-PW2k2-lpar1:root}/ #export MEMORY_AFFINITY=MCM
{D-PW2k2-lpar1:root}/ #execrset proc/1 -e ./proc1/latency4 16384 1024 30 7200&
[1] 14614922
{D-PW2k2-lpar1:root}/ #execrset proc/2 -e ./proc2/latency4 16384 1024 30 7200&
[2] 14483742
```

– We checked that the two latency processes were well bound to the desired RSET with the **lsrset** command, as shown in Example 6-14.

Example 6-14 Checking process RSET binding with lsrset

```
{D-PW2k2-lpar1:root}/ # lsrset -vp 14614922
Effective rset:  32 CPUs, 0 Memory
     CPU: 0-31
     MEM: <empty>

{D-PW2k2-lpar1:root}/ # lsrset -vp 14483742
Effective rset:  32 CPUs, 0 Memory
     CPU: 32-63
     MEM: <empty>
```

Test results

At the end of the test, we added the number of transactions generated by each process. Table 6-2 shows that test B with RSET and MEMORY_AFFIMITY=MCM generates two times more transactions than test A.

Table 6-2 RSETs vs. “no tuning” test results

Test name	Transactions	Memory locality[a]	Latency
Test A (no tuning)	65 402	40%	350 ns
Test B (RSET)	182 291	100%	170 ns

a. Measure with hpmstat “The hpmstat and hpmcount utilities” on page 334.

Note: The test we used to illustrate this RSET section is only waiting for memory, not for disk or network I/O. This is why we could achieve a 2x improvement by tuning the memory affinity.

Do not expect such results in your production environment. Most commercial applications wait for memory, but also for disk and network, which are much slower than memory. However, some improvement between 20% to 40% can usually be achieved.

Tuning memory affinity can really improve performance for memory-sensitive applications. But it can sometimes be difficult to implement. You also need to know how your application is working, and how your system is designed to be able to size your RSET. You also need to have an application that can be divided into multiple instances with no shared memory between the instances. And when everything is done, you need to continuously monitor your system to adapt your RSET sizing.

To conclude, “manual RSET” can be difficult to set up and maintain for some workloads. If you are in AIX 6.1 TL8, AIX 7.1 TL1 or later, Dynamic System Optimizer can help you do this job for you.

6.1.2 IBM AIX Dynamic System Optimizer

To accommodate today’s complex workloads, AIX introduced a new technology called Dynamic System Optimizer (DSO). DSO can help to address system optimization in an autonomous fashion and tune the system dynamically and automatically.

It is a framework that currently consists of the Active System Optimizer (ASO) daemon and four types of optimization strategies.

Active System Optimizer daemon

ASO is a user-level daemon that keeps monitoring system resource utilization and tunes the allocation of system resources based on the optimization strategies. It was originally introduced in AIX 7.1 TL1 and has since been added to AIX 6.1 TL08.

Optimization strategies

This section describes the optimization strategies:

- Cache Affinity Optimization

  ASO analyzes the cache access patterns based on information from the kernel and the PMU to identify potential improvements in Cache Affinity by moving threads of workloads closer together.

- Memory Affinity Optimization

  If ASO finds that the workload benefits from moving processes' private memory closer to the current affinity domain, then hot pages are identified and migrated into local.

- Large Page Optimization

  ASO promotes heavily used regions of memory to 16 MB pages to reduce the number of TLB/ERAT for such workloads that use large chunks of data.

- Data Stream Prefetch Optimization

  According to information collected from the kernel and the PMU, ASO dynamically configures the Data Stream Control Register (DSCR) setting to speed up hardware data stream fetching.

Note: By default, ASO provides Cache and Memory Affinity Optimization. To enable Large Page and Data Stream Prefetch Optimization, acquire and install the dso.aso package, which prereqs AIX7.1 TL2 SP1 or AIX6.1 TL8 SP1.

DSO (5765-PWO) can be ordered as a stand-alone program or as part of the AIX Enterprise Edition 7.1 (5765-G99) and AIX Enterprise Edition 6.1 (5765-AEZ) bundled offerings.

Clients that currently are licensed for either of these offerings and have a current SWMA license for those products are entitled to download DSO from the Entitled Software Support site:

https://www.ibm.com/servers/eserver/ess/OpenServlet.wss

Once activated, ASO runs without interaction in the background. Complex, multiple-thread, long-running applications with stable processor utilization will be eligible workloads for ASO tuning. For more details about eligible workloads and more detailed descriptions of the four optimizations, refer to *POWER7 Optimization and Tuning Guide,* SG248079.

How to start the ASO daemon

Before starting ASO, check to make sure ASO can be activated on your system. To verify the system level, refer to Example 6-15.

Example 6-15 Command to verify the AIX level

```
# oslevel -s
7100-02-00-0000
```

Example 6-16 shows how to verify the ASO and optional DSO fileset.

Example 6-16 Command to verify fileset

```
#lslpp -l|grep aso
  bos.aso                   7.1.2.0  COMMITTED  Active System Optimizer
  dso.aso                   1.1.0.0  COMMITTED  IBM AIX Dynamic System
```

Example 6-17 shows how to verify that your LPAR is running in POWER7 mode, because only LPARs hosted on POWER7 or later hardware are supported.

Example 6-17 Command to verify processor mode

```
#lsconf | grep ^Processor
Processor Type: PowerPC_POWER7
```

```
Processor Implementation Mode: POWER 7
Processor Version: PV_7_Compat
Processor Clock Speed: 3300 MHz
```

Use the command shown in Example 6-18 to start ASO.

Example 6-18 Command to start ASO

```
#asoo -o aso_active=1
Setting aso_active to 1
#startsrc -s aso
0513-059 The aso Subsystem has been started. Subsystem PID is 3080470.
#
```

How ASO impacts your application

Once ASO starts, it keeps monitoring the system utilization every minute. When a workload is running, ASO detects the process and applies each kind of optimization strategy in turn to the workload to determine the best tuning approach. If the performance results are not as expected, ASO reverses its actions immediately.

Now let us try to restart the same workload and environment as in 6.1.1, “Improving application memory affinity with AIX RSETs” on page 280. After starting the two “latency” processes without any specific tuning (Figure 6-4 on page 286), ASO starts to optimize it. At the end of the test, we compared results achieved with ASO and the manual RSET tuning.

Test results

The result is described in Figure 6-5. The results achieved by ASO are closed to manual RSET (+8% for manual RSET only).

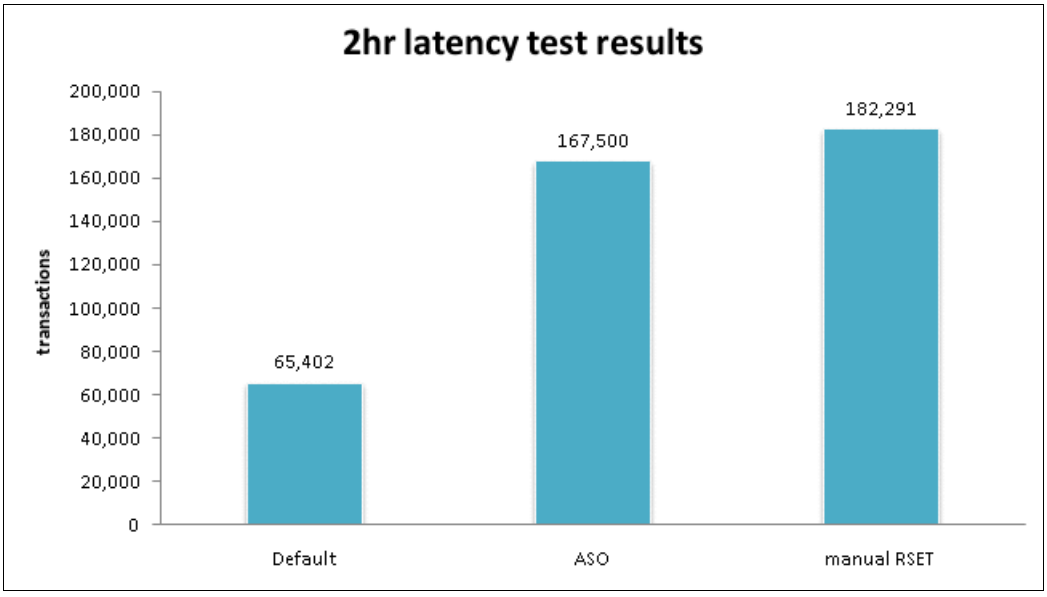


Figure 6-5 Manual RSET vs. ASO 2-hour “latency” test result

Let us have a closer look at the test by analyzing the transaction rate all along the test and the ASO logs available in the `/var/log/aso/` directory (Figure 6-6 on page 291).

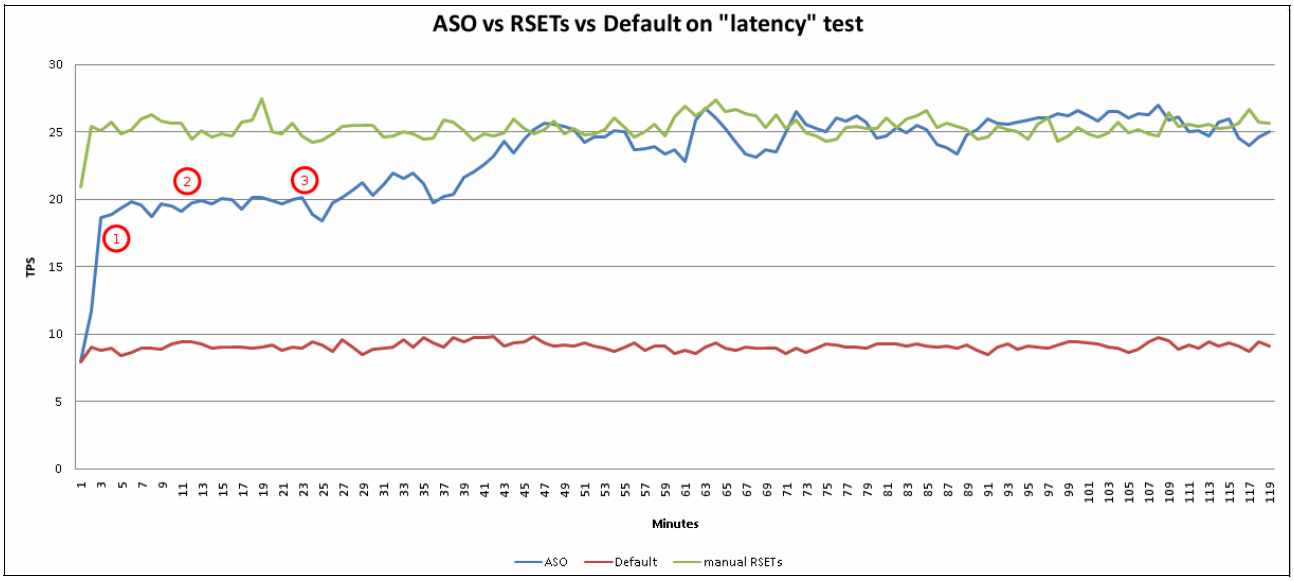


Figure 6-6 TPS evolution graph during 2-hour "latency" test

- Cache Affinity Optimization - After two minutes analyzing the loads of our two processes, ASO decided to create two RSETs with one chip per RSET, and bind each of the processes to one different RSET (Example 6-19). By always using the same chip, each thread of a process benefits from a better L3 cache affinity. This allows the TPS to go from 8 to 18 TPS.

Example 6-19 Cache Affinity Optimization extracted from /var/log.aso/aso_process.log

```
...(lines omitted)...
aso:info aso[4784168]: [SC][6029526] Considering for optimisation (cmd='latency4',
utilisation=7.21, pref=0; attaching StabilityMonitorBasic)
aso:info aso[4784168]: [SC][5177674] Considering for optimisation (cmd='latency4',
utilisation=5.89, pref=0; attaching StabilityMonitorBasic)
aso:info aso[4784168]: [perf_info] system utilisation 15.28; total process load
59.82
aso:info aso[4784168]: attached(   6029526): cores=8, firstCpu= 32, srads={1}
aso:info aso[4784168]: [WP][6029526] Placing non-FP (norm load 8.00) on 8.00 node
aso:info aso[4784168]: attached(   5177674): cores=8, firstCpu=  0, srads={0}
```

- After 5 minutes, ASO tries to evaluate more aggressive Cache Optimization strategies. But no improvement was found for our workload (Example 6-20).

Example 6-20 Extract from /var/log.aso/aso_process.log

```
...(lines omitted)...

aso:info aso[4784168]: [SC][6029526] Considering for optimisation (cmd='latency4',
utilisation=8.63, pref=0; attaching StabilityMonitorAdvanced)
aso:infoaso[4784168]:[EF][6029526]attachingstrategyStabilityMonitorAdvanced
aso:info aso[4784168]: [SC][5177674] Considering for optimisation (cmd='latency4',
utilisation=6.83, pref=0; attaching StabilityMonitorAdvanced)
aso:infoaso[4784168]:[EF][5177674]attachingstrategyStabilityMonitorAdvanced
aso:info aso[4784168]: [perf_info] system utilisation 15.29; total process load
59.86
aso:info aso[4784168]: [SC][5177674] Considering for optimisation (cmd='latency4',
utilisation=8.25, pref=0; attaching PredictorStrategy)
```

```
aso:info aso[4784168]: [EF][5177674] attaching strategy PredictorStrategy
aso:info aso[4784168]: [SC][5177674] Considering for optimisation (cmd='latency4',
utilisation=8.25, pref=0; attaching ExperimenterStrategy)
aso:info aso[4784168]: [EF][5177674] attaching strategy ExperimenterStrategy
```

- Memory Optimization - 20 minutes after the beginning of the test, ASO tried to optimize the memory affinity. It detected, for each process, 30% of non-local memory and decided to migrate 10% next to the RSET (Example 6-21). This optimization is made every five minutes. 45 minutes after the beginning of the run, most of the memory access was local, and TPS was comparable to manual RSET.

Note:

- Memory optimization needs to have enough free memory to perform the page migration.
- For test purposes and to compare with RESTs manual tuning, the value of `enhanced_affinity_private` was also set to 100 (instead of the default value of 40) one minute after the beginning of the run. This forced ASO to migrate pages to reach a memory affinity target near 100%.

Example 6-21 Memory Affinity Optimization sample in aso_process.log

```
...(lines omitted)...
aso:info aso[4784168]: [SC][5177674] Considering for optimisation (cmd='latency4',
utilisation=6.65, pref=0; attaching MemoryAffinityStrategy)
aso:info aso[4784168]: [perf_info] system utilisation 14.64; total process load
59.87
aso:info aso[4784168]: [MEM][5177674] 70.36% local, striped local 0.00%
aso:info aso[4784168]: [MEM][5177674] 100% max affinitised, 100.00% max local
aso:info aso[4784168]: [MEM][5177674] Accumulated remote accesses: 10177107.448366
aso:info aso[4784168]: [MEM][5177674] Recommending MIGRATE to 1(10%)
20354223.081760

Comment: Attached Memory strategy to the workloads
...(lines omitted)...

aso:info aso[4784168]: [MEM][5177674] Current migration request completed
aso:info aso[4784168]: [MEM][5177674] 419876 pages moved, target 419473 (progress
100.10%)
aso:info aso[4784168]: [MEM][5177674] Sufficient progress detected, detaching
monitor.
Comment: Completed current memory migration
...(lines omitted)...
```

6.2 Application side tuning

We focused on applications written in C/C++ and Java this time.

6.2.1 C/C++ applications

For C/C++ applications, you can utilize the XL C/C++ compiler to optimize code during compilation. Also, there are some other options and environment variables that can help

maximum performance based on the application characteristics. Most of the environment variables, including the PTHREAD tunables and MALLOCOPTIONS, can also be used to optimize the performance for applications other than C/C++.

Compiler version

To exploit the new POWER7 features, the IBM XL C/C++ for AIX V11.1 or later version is a must. Earlier versions of XL C, XL C/C++ only provide POWER7 tolerance. For additional information, refer to POWER7 tolerance for IBM XL compilers at:

http://www-01.ibm.com/support/docview.wss?uid=swg21427761

At the time of writing this book, there was no specific PTF for POWER7+ exploitation in XL C/C++ including V11.1.0.12 and V12.1.0.2. We suggest to use the POWER7 options.

Compiler options

There are numerous optimization compiler options. Here we just list the most common. For detailed information, refer to XL C/C++ InfoCenter at:

For IBM AIX Compilers - XL C/C++ for AIX, V11.1, XL Fortran for AIX, V13.1:

http://publib.boulder.ibm.com/infocenter/comphelp/v111v131/index.jsp

For IBM AIX Compilers - XL C/C++ for AIX, V12.1, XL Fortran for AIX, V14.1

http://publib.boulder.ibm.com/infocenter/comphelp/v121v141/index.jsp

Optimization levels

This section describes the optimization levels supported:

- `-0`

  The XL C/C++ compiler supports several levels of optimization: 0, 2, 3, 4, and 5. The higher level of optimization is built on lower levels with more aggressive tuning options. For example, on top of `-02` optimization, `-03` includes extra floating point optimization, minimum high-order loop analysis and transformations (HOT), and relaxes the compilation resource limits to compensate for better runtime performance. `-04` enables higher level HOT, interprocedural analysis (IPA), as well as machine-dependent optimization. `-05` enables higher level IPA on top of `-04`. Refer to Table 6-3 for more detailed information.

Table 6-3 Optimization levels in xlC/C++

Optimization level	Extra compiler options implied besides the options at lower levels	Comments
-00	N/A	Ensure your application runs well in the default level. That is the basis for further optimization.
-02/-0	**-qmaxmem=8192**	This option is suggested for most commercial applications.
-03	**-qnostrict** **-qmaxmem=-1** **-qhot=level=0**	Certain semantics of the program might be altered slightly, especially floating point operations. Specify **-qstrict** to avoid this.

Optimization level	Extra compiler options implied besides the options at lower levels	Comments
-O4	**-qhot=level=1** **-qipa** **-qarch=auto** **-qtune=auto** **-qcache=auto**	IPA (**-qipa)** is included in **-O4**, which might increase compilation time significantly, especially at the link step. Use **make -j[Jobs]** to start multiple compiling jobs to circumvent such issues.
-O5	**-qipa=level=2**	This option enables a higher level of IPA on top of **-O4** and requires more compilation time.

However, note that there are tradeoffs between increased compile time, debugging capability and the performance improvement gained by setting higher optimization levels. Also, a higher level of optimization does not necessarily mean better performance. It depends on the application characteristics. For example, `-O3` might not outperform `-O2` if the workload is neither numerical nor compute intensive.

Note: For most commercial applications, we suggest a minimum level of 2 for better performance, acceptable compiler time, and debugging capability.

Machine-dependent optimization

Machine-dependent optimization options can instruct the compiler to generate optimal code for a given processor or an architecture family.

- `-qarch`

  Specifies the processor architecture for which the code (instructions) should be generated. The default is `ppc` for 32-bit compilation mode or `ppc64` for 64-bit compilation mode, which means that the object code will run on any of the PowerPC® platforms that support 32-bit or 64-bit applications, respectively.

  This option could instruct the compiler to take advantage of specific Power chip architecture and instruction set architecture (ISA). For example, to take advantage of POWER6 and POWER7 hardware decimal floating point support (`-qdfp`), we should compile with `-qarch=pwr6` or above explicitly; refer to Example 6-22.

Example 6-22 Enable decimal floating point support

```
#xlc -qdfp -qarch=pwr6 <source_file>
```

  Note that you should specify the oldest platform that your application will run on if you explicitly specify this option. For example, if the oldest platform for your application is POWER5, you should specify `-qarch=pwr5`.

- `-qtune`

  This option tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run best on a specific hardware architecture. `-qtune` is usually used together with `-qarch`.

  You should specify the platform which you application is most likely to run on. Then the compiler will generate instruction sequences with the best performance for that platform. For example, if the oldest platform for you application is POWER5, and the most common

platform is POWER7, you should specify "`-qarch=pwr5 -qtune=pwr7`" and the compiler will target the POWER7 platform for best performance.

Tip: `-qarch=auto` and `-qtune=auto` are implied using high-level optimization `-O4` and `-O5`. These options assume the execution environment is the same with the compiling machine, and optimize your application based on that assumption. You should specify correct values of `-qarch` and `-qtune` explicitly to override the default, if the processor architecture of your compiling machine is not identical to the execution environment.

Other compiler options

Here are some other options that are frequently considered in compiler optimization.

- `-qalias=ansi|noansi`

  When `ansi` is in effect, type-based aliasing is used during optimization, which restricts the lvalues that can be safely used to access a data object. The optimizer assumes that pointers can only point to an object of the same type or a compatible type. This suboption has no effect unless you also specify an optimization option.

  The "compatible type" rules are slightly different between C and C++, but in both some small type differences are allowed. The C and C++ language standards provide more detail. Here is the C99 wording in Section 6.5 Expressions:

  An object shall have its stored value accessed only by an lvalue expression that has one of the following types:

  – A type compatible with the effective type of the object.
  – A qualified version of a type compatible with the effective type of the object.
  – A type that is the signed or unsigned type corresponding to the effective type of the object.
  – A type that is the signed or unsigned type corresponding to a qualified version of the effective type of the object.
  – An aggregate or union type that includes one of the aforementioned types among its members (including, recursively, a member of a subaggregate or contained union).
  – A character type.

The C++ rules are similar but expanded to deal with base and derived classes. Refer to "3.10 Lvalues and rvalues" in C++ 2003 standard (ISO/IEC 14882).

When `noansi` is in effect, the optimizer makes worst-case aliasing assumptions. It assumes that a pointer of a given type can point to an external object or any object whose address is already taken, regardless of type.

Example 6-23 gives a common source code example that violates the ANSI aliasing rule, and encounters problems under a higher level of optimization. The example also gives an approach on how to identify the aliasing problem using `-qinfo=als`, and how to work around it using `-qalias=noansi`.

Example 6-23 Dealing with the ANSI aliasing rule in compiler optimization

```
# cat test.c
#include <stdio.h>
struct refhead {
int refid;
};
```

```
struct pagehead {
char pageid;
char pageindex;
char paddings[2];
};

int main(int argc, char* argv[])
{
  int myid ;
  struct pagehead* optr;
  struct refhead* nptr;
  nptr = (struct refhead *) &myid; /*violate ansi aliasing rule*/
  nptr->refid = 0;

  optr = (struct pagehead*) &myid; /*violate ansi aliasing rule*/
  optr->pageid = 0x12;
  optr->pageindex = 0x34;

  printf("nptr->id = %x\n", nptr->refid);
  printf("optr->id = %x %x\n", optr->pageid, optr->pageindex);
  return 0;
}
# xlc test.c
# ./a.out
nptr->id = 12340000
optr->id = 12 34
# xlc test.c -O4
# ./a.out
nptr->id = 0 <= incorrect result, should be 12340000.
optr->id = 12 34
# xlc test.c -qinfo=als
"test.c", line 18.7: 1506-1393 (I) Dereference may not conform to the current
aliasing rules.
"test.c", line 18.7: 1506-1394 (I) The dereferenced expression has type "struct
refhead". "nptr" may point to "myid" which has incompatible type "int".
"test.c", line 18.7: 1506-1395 (I) Check pointer assignment at line 17 column 8 of
test.c.
"test.c", line 21.7: 1506-1393 (I) Dereference may not conform to the current
aliasing rules.
"test.c", line 21.7: 1506-1394 (I) The dereferenced expression has type "struct
pagehead". "optr" may point to "myid" which has incompatible type "int".
"test.c", line 21.7: 1506-1395 (I) Check pointer assignment at line 20 column 8 of
test.c.
# xlc test.c -O4 -qalias=noansi
# ./a.out
nptr->id = 12340000 <= correct!
optr->id = 12 34
```

Tip: If you encounter aliasing problems with -O2 or higher optimization levels, consider the following approaches rather than turning off optimization:

- Use -qalias=noansi, which is a fast workaround.

  Compiling with -qalias=noansi will limit the amount of optimization the compiler can do to the code. However, -O2 -qalias=noansi will be better than disabling optimization altogether (-O0).

- Use -qinfo=als without optimization options to list the problem lines of code, and correct them to comply with the ANSI aliasing rule. This might help if optimal performance is desired for a particular area of code.

- -qinline

  This option, instead of generating calls to functions, attempts to inline these functions at compilation time to reduce function call overhead. We suggest you use -qinline together with a minimum optimization level of -O2.

  The inlining is usually important for C++ applications. In XL C/C++ V11.1 and V12.1, you can use -qinline=level=<number> to control the aggressiveness of inlining. The number must be a positive integer between 0 and 10 inclusive; the default is 5. Larger value implies more aggressive inlining. For example, -qinline=level=10 means the most aggressive inlining.

- -s

  This option strips the symbol table, line number information, and relocation information from the output file, which has the same effect as the **strip** command. Usually, large applications can get some performance benefit from this option. Note that it should not be used with debugging options such as -g.

  Note: You can either use the **strip** command or the -s option. You should not use both because you will get an error message stating that The file was already stripped as specified.

- -qlistfmt

  This feature is provided since XL C/C++ V11.1 and enhanced in XL C/C++ V12.1. It is aimed to assist user finding optimization opportunities.

  With -qlistfmt, XL compilers provide compiler transformation report information about the optimizations that the compiler is able to perform and also which optimization opportunities were missed. The compiler reports are available in the xml and html formats. Example 6-24 shows an example of using the -qlistfmt option.

Example 6-24 -qlistfmt example

```
//comments: the sample program used in this example
# cat testarg.c
#include <stdio.h>
#include <stdarg.h>
#define MACRO_TEST(start,...) myvatest(start, __VA_ARGS__);

void myvatest(int start, ...)
{
    va_list parg;
    int value;
    int cnt=0;
```

```
    va_start(parg, start);
    printf("addr(start) = %p, parg = %p\n", &start, parg);
    while( (value = va_arg(parg, int)) != -1)
    {
        cnt++;
        printf("the %dth argument: %d, current pointer = %p\n", cnt, value, parg);
    }

    va_end(parg);
}

inline void mytest(int arg1, int arg2, int arg3)
{
   printf("arg1 = %d, addr(arg1) = %p\n", arg1, &arg1);
   printf("arg2 = %d, addr(arg2) = %p\n", arg2, &arg2);
   printf("arg3 = %d, addr(arg3) = %p\n", arg3, &arg3);
}


int main(int argc, char* argv[])
{
    mytest(1,2,3);
    myvatest(1,2,3,4,5,6,7,8,-1);
    MACRO_TEST(1,2,3,4,5,6,7,8,9,-1);
    return 0;
}

//comments: generate the inlining reports in html format, by default a.html will
be generated during the IPA link phase.
#xlc_r -qlistfmt=html=all -O5 -qinline ./testarg.c -o testarg
```

From the “Inline Optimization Table” section of the `-qlistfmt` report, a.htm is shown in Figure 6-7, and you can see that some of the inlining opportunities failed.

Inline Optimization Table

Seq #	Type	Phase	Caller Region #	Callee Region #	Callsite File #	Callsite Line #	Callsite Column #	Description
1	FunctionTooBig (fail)	High Level Optimizer	2	1	1	32	not available	The function was not inlined because it is too big to be inlined.
2	SuccessfulInline (success)	Low Level Optimizer	2	3	1	32	not available	The function was successfully inlined.
3	MiscellaneousLimitation (fail)	Low Level Optimizer	not available	not available	not available	33	not available	The function was not inlined due to a limitation.
4	MiscellaneousLimitation (fail)	Low Level Optimizer	not available	not available	not available	34	not available	The function was not inlined due to a limitation.

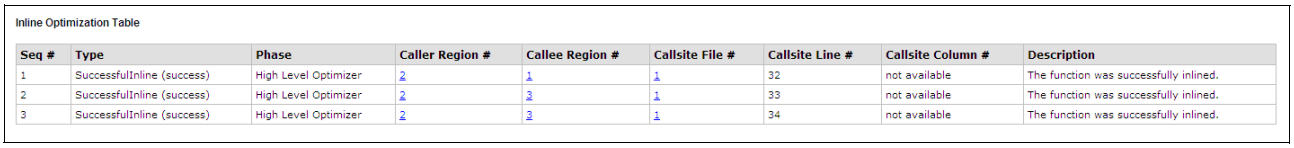
Figure 6-7 Inline Optimization Table in -qlistfmt report (missing opportunities)

Then we tried a more aggressive inlining level, as shown in Example 6-25.

Example 6-25 -qlistfmt example (more aggressive inlining)

```
#xlc_r -qlistfmt=html=all -O5 -qinline=level=10 ./testarg.c -o testarg
```

Part of the new `-qlistfmt` report is shown in Figure 6-8 on page 299. All the inlining attempts succeeded in the new report.



Inline Optimization Table

Seq #	Type	Phase	Caller Region #	Callee Region #	Callsite File #	Callsite Line #	Callsite Column #	Description
1	SuccessfulInline (success)	High Level Optimizer	2	1	1	32	not available	The function was successfully inlined.
2	SuccessfulInline (success)	High Level Optimizer	2	3	1	33	not available	The function was successfully inlined.
3	SuccessfulInline (success)	High Level Optimizer	2	3	1	34	not available	The function was successfully inlined.

Figure 6-8 Inline Optimization Table in -qlistfmt report (all inlining succeeded)

Debugging support in optimized programs

Higher level optimization is more sensitive to application programming errors, and is usually difficult to debug. As a result, there are related compiler options to address the debugging requirement in optimized programs. However, you should use them with caution because these debugging options tend to have a negative performance impact.

- `-qkeepparm`

  This option with `-O2` or higher optimization level specifies whether procedure parameters are stored on the stack.

  A function usually stores its incoming parameters on the stack at the entry point. However, when you compile code with optimization options enabled, the compiler might remove these parameters from the stack if it sees an optimizing advantage in doing so. When `-qkeepparm` is in effect, parameters are stored on the stack even when optimization is enabled. This provides better debugging support, but it might negatively affect application performance.

- `-qoptdebug`

  In XL C/C++ V9.0 and later releases, for `-O3` or higher optimization level, -g -qoptdebug is provided for easy debugging purposes. Pseudo source files are generated under the same directory of output object files with suffix `.optdbg`, and can be used with dbx for debugging. Refer to Example 6-26 for more details.

Example 6-26 Pseudo code generation for high-level optimization

```
# cat test.c
#include <stdio.h>
void test(void)
{
    printf("This is a test.\n");
}

void main(int argc, char* argv[])
{
  int c; /*not used, will be deleted by compile optimization*/
  test();/*will be inlined by compile optimization*/
}

# /usr/vac/bin/xlc -O5 -qoptdebug -g test.c -o test
# ls -lrt test*
-rw-r--r--    1 root     system          225 Oct 30 21:55 test.c
-rw-r--r--    1 root     system          147 Oct 30 21:55 test.optdbg
-rwxr-xr-x    1 root     system         6185 Oct 30 21:55 test
# cat test.optdbg
     7 |  void main(long argc, char * argv)
     8 |  {
     4 |    printf("This is a test./n");
    11 |    return;
```

```
        } /* function */
```

- `-g<level>`

  `-g` is extended to improve the debugging of optimized programs in XL C/C++ V12.1. Since XL C/C++ V12.1, the debugging capability is completely supported when the `-O2` optimization level is in effect, which means `-O2 -g` is officially accepted and enhanced. However, when an optimization level higher than `-O2` is in effect, the debugging capability is still limited.

  You can use different `-g` levels to balance between debugging capability and compiler optimization. The level must be a positive integer between 0 and 9 inclusive, and the default is 2 when `-O2 -g` is specified. Higher `-g` levels provide more complete debugging support at the cost of runtime performance.

Profile directed feedback

This section provides details about the profile directed feedback (PDF) option.

`-qpdf1`, `-qpdf2` - PDF is a two-stage compilation process that allows the developer to provide the compiler with data characteristic of typical program behavior. At the first stage of compilation, `-qpdf1` is specified and special instrumentation code is inserted into the executable to gather information about the program's execution pattern. Then the application is run under typical scenarios to collect feedback information. At the second stage of compilation, the `-qpdf2` is specified instead of `-qpdf1`. Other options are unchanged. The compiler takes feedback information and generates an optimized executable. Example 6-27 gives an example of using the PDF options.

Example 6-27 Using profile directed feedback options

```
//comments: instrument with -qpdf1
#/usr/vac/bin/xlc -O5 -qpdf1 test.c -o test

//comments: run "test" with typical data set. "._pdf" file will be generated by
default.
#./test

//comments: the profiling information is recorded in "._pdf".
#ls -l ._pdf
-rwxr-Sr--    1 root     system          712 Oct 31 03:46 ._pdf

//comments: re-compile using -qpdf2 to generate the optimized binary.
#/usr/vac/bin/xlc -O5 -qpdf2 test.c -o test
```

Profile directed feedback is usually intended for the last few steps before application release, after all debugging and other tunings are done.

Feedback directed program restructuring (FDPR)

FDPR® can optimize the application binary based on runtime profiling data, and the optimization does not need the source code of the application. Here we only give a brief introduction to the usage of FDPR. More details of the FDPR tool can be found in *POWER7 and POWER7+ Optimization and Tuning Guide*, SG24-8079.

The FDPR optimization process is quite similar to the PDF optimization process, and can be used together with the PDF optimization. The PDPR optimization process also consists of three steps, as follows:

1. Instrument the application executable. A new instrumented executable with suffix .instr will be generated.
2. Run the instrumented executable and collect profiling data.
3. Optimize the executable using the profiled information. A new optimized executable with suffix .fdpr will be generated.

Example 6-28 gives an example using the FDPR tool.

Example 6-28 Using FDPR tool

```
//comments: instrument the binary "test" using fdpr command.
# fdpr -a instr test
FDPR 5.6.1.0: The fdpr tool has the potential to alter the expected
behavior of a program.  The resulting program will not be
supported by IBM.  The user should refer to the fdpr document
for additional information.

fdprpro (FDPR) Version 5.6.1.0 for AIX/POWER
/usr/lib/perf/fdprpro -a instr -p test --profile-file /home/bruce/test.nprof
--output-file /home/bruce/test.instr
> reading_exe ...
> adjusting_exe ...
> analyzing ...
> building_program_infrastructure ...
> building_profiling_cfg ...
> instrumentation ...
>> adding_universal_stubs ...
>> running_markers_and_instrumenters ...
>> mapping ...
>>> glue_and_alignment ...
>> writing_profile_template -> /home/bruce/test.nprof ...
> symbol_fixer ...
> updating_executable ...
> writing_executable -> /home/bruce/test.instr ...
bye.

//comments: run the instrumented executable "test.instr" under typical scenarios.
# ./test.instr
This is a test.

//comments: the profiling data is recorded in <executable_name>.nprof.
# ls -lrt test*
-rwxr-xr-x    1 root     system        24064 Oct 31 03:42 test
-rwxr-xr-x    1 root     system        65190 Oct 31 04:11 test.instr
-rw-rw-rw-    1 root     system        40960 Oct 31 04:12 test.nprof

//comments: Optimize the executable using the profiled information.
# fdpr -a opt -O3 test
FDPR 5.6.1.0: The fdpr tool has the potential to alter the expected
behavior of a program.  The resulting program will not be
supported by IBM.  The user should refer to the fdpr document
for additional information.
```

```
fdprpro (FDPR) Version 5.6.1.0 for AIX/POWER
/usr/lib/perf/fdprpro -a opt -p test --profile-file /home/bruce/test.nprof
--branch-folding --branch-prediction --branch-table-csect-anchor-removal
--hco-reschedule --inline-small-funcs 12 --killed-registers --load-after-store
--link-register-optimization --loop-unroll 9 --nop-removal --derat-optimization
--ptrgl-optimization --reorder-code --reorder-data --reduce-toc 0 -see 1
--selective-inline --stack-optimization --tocload-optimization
--volatile-registers-optimization --output-file /home/bruce/test.fdpr
> reading_exe ...
> adjusting_exe ...
> analyzing ...
> building_program_infrastructure ...
> building_profiling_cfg ...
> add_profiling ...
>> reading_profile ...
>> building_control_flow_transfer_profiling ...
> pre_reorder_optimizations ...
>> derat_optimization ...
>> nop_optimization ...
>> load_after_store_optimization ...
>> loop_unrolling ...
>> branch_folding_optimization ...
>> pre_cloning_high_level_optimizations ...
>> inline_optimization_phase_1 ...
>> dfa_optimizations ...
>>> flattening ...
>>> calculating_input_parameters_area_sizes ...
>> high_level_analysis ...
>> pre_inline_high_level_optimizations ...
>> inline_optimization_phase_2 ...
>> lr_optimization ...
>> bt_csect_anchor_removal_optimization ...
>> ptrglr11_optimization ...
>> high_level_optimizations ...
>> branch_folding_optimization ...
>> removing_traceback_tables ...
> reorder_and_reorder_optimizations ...
>> getting_order_and_running_fixers ...
>>> tocload_data_reordering ...
>> code_reorder_optimization ...
>> tocload_optimization ...
>> memory_access_fixer ...
>> glue_and_alignment ...
>> symbol_fixer ...
>> branch_prediction_optimization ...
> updating_executable ...
> writing_executable -> /home/bruce/test.fdpr ...
bye.

//comments: the “test.fdpr” will be generated.
# ls -lrt test*
-rwxr-xr-x    1 root     system        24064 Oct 31 03:42 test
-rwxr-xr-x    1 root     system        65190 Oct 31 04:11 test.instr
-rw-rw-rw-    1 root     system        40960 Oct 31 04:12 test.nprof
```

```
-rwxr-xr-x    1 root     system        24167 Oct 31 04:12 test.fdpr
```

PTHREAD environment variables

The following is a list of the environment variables for POSIX threads that might impact application performance.

- AIXTHREAD_SCOPE

  Controls the contention scope. A value of P signifies process-based contention scope (M:N). A value of S signifies system-based contention scope (1:1), which is the default in AIX 6.1 and later releases.

  If a user thread is created with the system-based contention scope, it is bound to a dedicated kernel thread and scheduled directly by the kernel. For most cases, setting AIXTHREAD_SCOPE=S should benefit. However, there are rare situations where setting AIXTHREAD_SCOPE=P helps. Usually, in such cases, threads within a specific process have lots of complicated interactions with each other.

  AIXTHREAD_MUTEX_DEBUG, AIXTHREAD_COND_DEBUG, and AIXTHREAD_RWLOCK_DEBUG

  These are the debug options for mutexes, conditional variables, and read-write locks, respectively. You can turn off the options explicitly by setting those variables to OFF for better performance.

- AIXTHREAD_MUTEX_FAST

  If the program experiences performance degradation due to heavy mutex contention, then setting this variable to ON will force the pthread library to use an optimized mutex locking mechanism that works only on process-private mutexes, which must be initialized using the pthread_mutex_init routine and must be destroyed using the pthread_mutex_destroy routine. Leaving the variable set to OFF forces the pthread library to use the default mutex locking mechanism.

- SPINLOOPTIME=n

  This variable controls the number of times the system tries to get a busy mutex or spin lock without taking a secondary action such as calling the kernel to yield the process. This control is intended for multiprocessor systems, where it is hoped that the lock being held by another actively running pthread will be released. The parameter works only within libpthreads (user threads). If locks are usually available within a short amount of time, you may want to increase the spin time by setting this environment variable. The number of times to retry a busy lock before yielding to another pthread is n. The default is 40 and n must be a positive value.

- YIELDLOOPTIME=n

  This variable controls the number of times the system yields the processor when trying to acquire a busy mutex or spin lock before actually going to sleep on the lock. The processor is yielded to another kernel thread, assuming there is another executable with sufficient priority. This variable has been shown to be effective in complex applications, where multiple locks are in use. The number of times to yield the processor before blocking on a busy lock is n. The default is 0 and n must be a positive value.

- AIXTHREAD_AFFINITY

  Controls the placement of pthread structures, stacks, and thread-local storage on an enhanced affinity-enabled system.

  Setting AIXTHREAD_AFFINITY=first-touch should benefit some memory-sensitive applications.

MALLOCTYPE and MALLOCOPTIONS tunables

We can use MALLOCTYPE to specify the allocation policy used by the malloc subsystem. The most frequently used policies are the default and the watson allocation policy.

- The default allocation policy (MALLOCTYPE not set)

  The default allocation policy maintains the free space in the heap as nodes in one cartesian binary search tree in which the nodes are ordered left-to-right by address (increasing address to the right) and top-to-bottom by length (such that no child is larger than its parent). Tree reorganization techniques optimize access times for node location, insertion, and deletion, and also protect against fragmentation.

4. The watson allocation policy (MALLOCTYPE=watson)

   The watson allocation policy maintains the free space in the heap as nodes in two separate red-black trees, one sorted by address, the other by size. Red-black trees provide simpler and more efficient tree operations than the cartesian tree of the default allocator, thus the watson allocation policy is often faster than the default.

We can use MALLOCOPTIONS to specify different options to the chosen allocation policy: `multiheap` and `pool` are the most frequently used options for best performance, and both are compatible with the default allocator and watson allocator.

- multiheap

  By default, the system uses only one heap for all threads in the process when allocating memory, and this tends to be a performance bottleneck for multithread applications. By setting MALLOCOPTIONS to `multiheap` allows multiple heaps to be created. By default, if you do not specify a value for `multiheap`, it uses 32 heaps. However, we strongly suggest that you specify an *n* value, especially for 32-bit applications, to better utilize the process address space and avoid fragmentations. The n value should depend on the concurrent need of the application and also the hardware configuration. For example, if you have only four worker threads, you can set the number to 4. The n value should also be proportional to the number of cores.

- pool

  Pool allocation policy is a high performance front end of the malloc subsystem for managing objects <= 512 bytes. It can be used together with the `multiheap` option.

  By specifying the pool allocation policy, the malloc subsystem creates a linked list of fixed sized blocks. The first linked list contains objects of pointer size (4 bytes for 32-bit applications, and 8 bytes for 64-bit applications); the successive linked list contains objects pointer-size larger than the previous linked list.

  The implementation of the pool allocation policy is similar to the buckets allocation policy, but more efficient in most real scenarios.

- no_mallinfo

  If you specify MALLOCOPTIONS=no_mallinfo, the information about the heap managed by the malloc subsystem is not logged. This option is usually specified for performance gains.

Checking the environment variables in effect

Example 6-29 shows an approach to determine which environment variables have been set for the specific process by using the **`ps ewww`** command.

Example 6-29 The ps ewww command

```
# ps ewww 6946928
    PID    TTY STAT  TIME COMMAND
```

```
 6946928  pts/0 A     0:15 ./test _=./test LANG=C LOGIN=root YIELDLOOPTIME=20
MALLOCOPTIONS=pool,multiheap:4,no_mallinfo
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/java5/jre/bin:/usr/j
ava5/bin:/usr/vac/bin:/usr/vacpp/bin AIXTHREAD_MUTEX_DEBUG=OFF
AIXTHREAD_RWLOCK_DEBUG=OFF LC__FASTMSG=true AIXTHREAD_COND_DEBUG=OFF LOGNAME=root
MAIL=/usr/spool/mail/root LOCPATH=/usr/lib/nls/loc
LDR_CNTRL=TEXTPSIZE=64K@STACKPSIZE=64K@DATAPSIZE=64K@SHMPSIZE=64K USER=root
AUTHSTATE=compat SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos AIXTHREAD_MUTEX_FAST=ON
HOME=/ SPINLOOPTIME=4000 TERM=vt100 MAILMSG=[YOU HAVE NEW MAIL]
PWD=/home/bruce/samples TZ=BEIST-8 AIXTHREAD_AFFINITY=first-touch
AIXTHREAD_SCOPE=S A__z=! LOGNAME
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
```

Large and medium page size support

Applications tend to benefit from large page or medium page size to help improve virtual memory addressing efficiency. For details about this, refer to 4.2.4, “Multiple page size support” on page 138.

6.2.2 Java applications

IBM Java has already been tuned to work on AIX. However, you might also need to adjust some of the options to maximize the performance gain. You can refer to the following links for Java tuning best practices on IBM Power Systems.

For Java application tuning on Power Systems, refer to the following documentation:

Java Performance on POWER7:

http://www-03.ibm.com/systems/power/hardware/whitepapers/java_perf.html

Java Tuning for Performance and IBM POWER6 Support:

http://www-03.ibm.com/systems/power/hardware/whitepapers/java_tuning.html

6.2.3 Java Performance Advisor

Based on Java tuning best practices, the Java Performance Advisor tool can provide recommendations to improve the performance of Java applications running on AIX. Three factors are used to determine the recommendations:

- Relative importance of the Java application
- Machine usage (test or production)
- User’s expertise level

Note: Currently this tool only runs on AIX version 6.1 and 7.1 with root privileges.

Usage

The Java Performance Advisor tool can be started by executing **jpa.pl**, which is located in the directory from which you extracted the tool package.

Syntax

jpa.pl [[-e Beginner|Intermediate|Expert] [-u Test|Production] [-i Primary|Secondary] [-o OutputFile] pid]

Table 6-4 shows the Java Performance Advisor flags and arguments.

Table 6-4 Java Performance Advisor flags and arguments

Flag	Arguments	Description
-e	Beginner	Either the person running this tool is unfamiliar with this environment and the workloads running on it or is just starting to perform AIX administration. The recommendations that the tool will make will be conservative and will only include the lowest risk options, while flagging the more aggressive options as possibilities.
	Intermediate	The person using this tool is knowledgeable about the environment and being an AIX administrator. Recommendations will be more aggressive than the administrators in the Beginner category.
	Expert	The person running the tool is very knowledgeable about the environment and being an AIX administrator. All recommendations that the tool will make will be verified by the administrator before the setting is changed. Thus, the tool will be the most aggressive on making recommendations and the administrator will make judgment calls about each and every recommendation.
-u	Test	This partition is primarily a test partition, thus the performance recommendations can be more aggressive without affecting a production environment.
	Production	This partition is being used for production use, thus down-time is not an option. The recommendations will be less aggressive in a production environment.
-i	Primary	The job has paramount importance compared to the other jobs on the system. Recommendations can be made that could affect other jobs running on the system, if it improves the speed.
	Secondary	Although this job is important, there are other jobs that have a higher priority. Any recommendations made should have a small chance of affecting other jobs on the system.
-d		Enables debug information. This option enables JPA to run in debug mode, which generates enough information for troubleshooting issues with JPA.
-o	OutputFile	The output result file.
	pid	PID of the process that needs to be tuned. if no PID is supplied, all JVMs will be shown along with their PIDs.
-h		Print help message.
-v		Print version.

Viewing the result

The result can be viewed by opening the output file from the browser. By default, the JPA writes output to jpa_output.xml under the present working directory. The output filename can be changed with the **-o** option in the command. The sample is shown in Figure 6-9 on page 307.

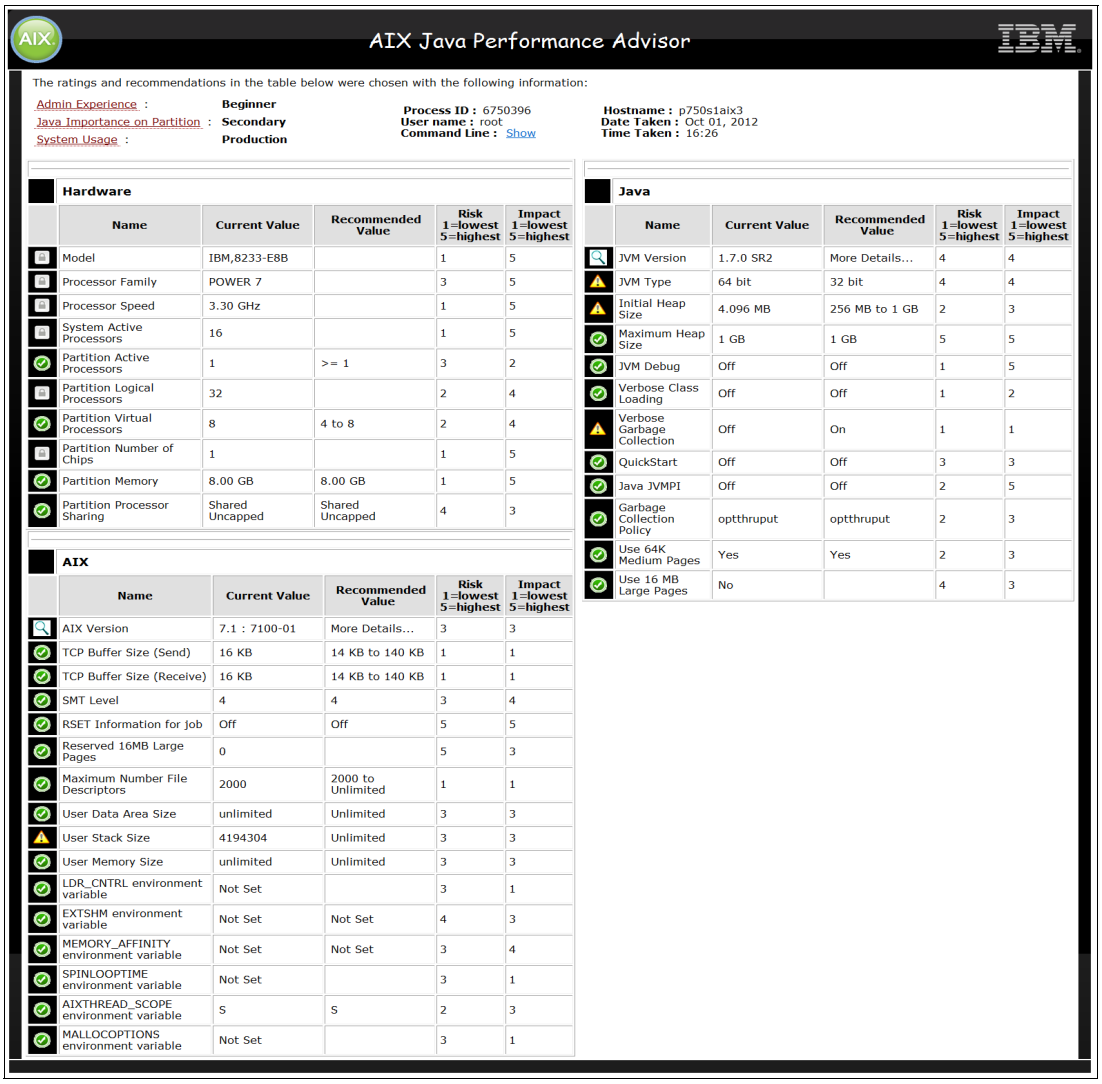


Figure 6-9 Java Performance Advisor analysis result

Apply the tuning recommendations

You can go through the recommended setting list generated by the Java Performance Advisor tool. For details, click the row that you are interesting in. A pop-up window shows you how to apply the settings, as shown in Figure 6-10 on page 308.

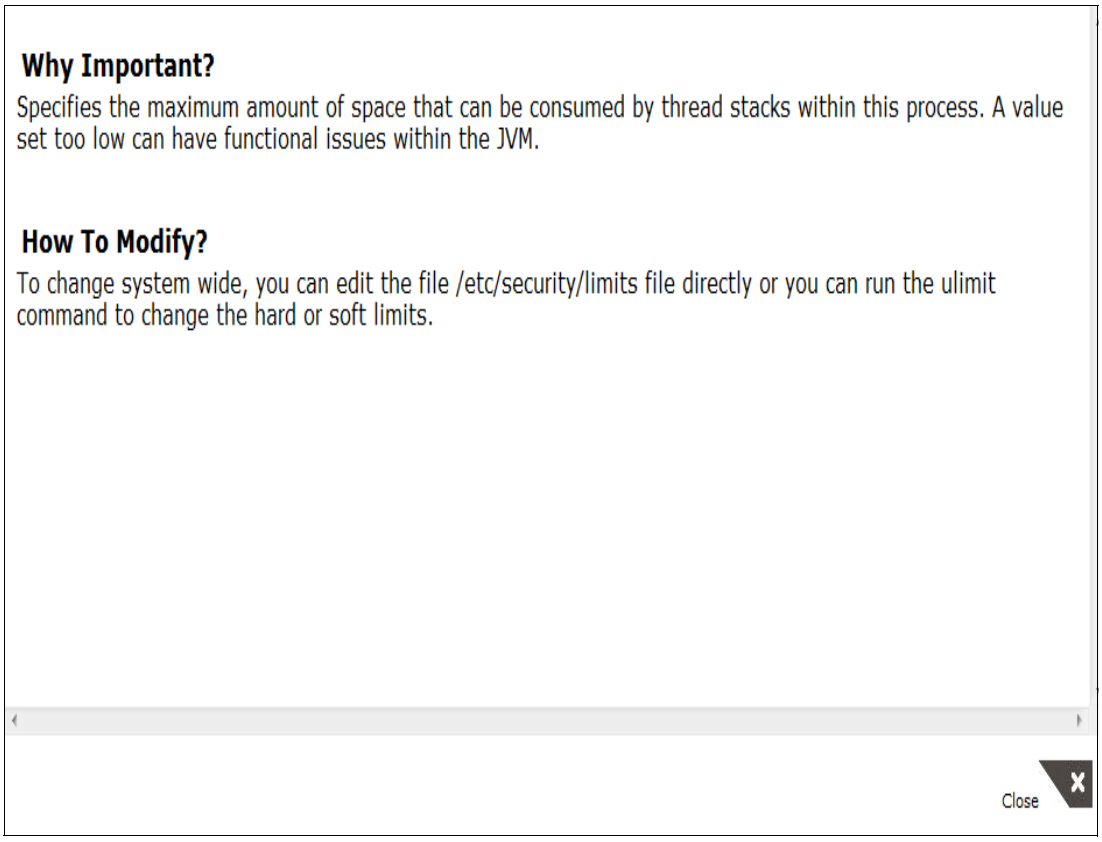


Figure 6-10 Hints from the Java Performance Advisor

6.3 IBM Java Support Assistant

IBM Java Support Assistant (ISA) software is a one stop solution for performance monitoring of IBM Java virtual machines, and also for obtaining advisory tuning suggestions.

The ISA is in fact a collection of performance monitoring utilities for various aspects such as JVM thread analysis, memory analysis, and so on. IBM provides tooling and documentation to assist in the understanding, monitoring, and problem diagnosis of applications and deployments running IBM Runtime Environments for Java.

The following sections provide details about ISA utilities.

6.3.1 IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer

The IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer is a Java heap analysis tool based on the Eclipse Memory Analyzer. The IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer brings the diagnostic capabilities of the Eclipse Memory Analyzer Tool (MAT) to the IBM virtual machines for Java.

Memory Analyzer extends Eclipse MAT version 1.1 using the Diagnostic Tool Framework for Java (DTFJ), which enables Java heap analysis using operating system level dumps and IBM Portable Heap Dumps (PHD).

Using Memory Analyzer enables you to:

- Diagnose and resolve memory leaks involving the Java heap.
- Derive architectural understanding of your Java application through footprint analysis.
- Improve application performance by tuning memory footprint and optimizing Java collections and Java cache usage.
- Produce analysis plug-ins with capabilities specific to your application.

Memory Analyzer is a powerful and flexible tool for analyzing Java heap memory using system dump or heap dump snapshots of a Java process. Memory Analyzer provides both high-level understanding and analysis summaries using a number of standard reports. It allows you to carry out in-depth analyses through browsing and querying the Java objects present on the Java heap.

The following features combine to make it possible to:

- Diagnose memory leaks
  - Leak Suspects Report - Memory Analyzer provides a standard report that uses its in-built capabilities to look for probable leak suspects: large objects or collections of objects that contribute significantly to the Java heap usage. It displays information about those suspects: memory utilization, number of instances, total memory usage, and owning class.
  - Leak identification analysis - Memory Analyzer also provides a number of in-depth leak identification capabilities that look for collections with large numbers of entries, single large objects, or groups of objects of the same class.
- Analyze application footprint
  - Heapdump Overview Report - In order to provide a general understanding of the Java application being analyzed, Memory Analyzer provides an overview report that provides information on the Java heap usage, system property settings, threads present, and a class histogram of the memory usage.
  - Top Consumers Report - Memory Analyzer also produces the Top Consumers Report that gives a breakdown of the Java heap usage by largest objects, and also which class loaders and classes are responsible for utilizing the most memory. This provides a high-level insight into which J2EE application and/or code is contributing most to the overall memory footprint.
  - Component Report - Memory Analyzer can create reports outlining the top memory consumers and providing information about potential memory inefficiencies in any selected component.
  - Object tree browsing - In addition to the report capabilities, Memory Analyzer provides the ability to browse the Java heap using a reference tree-based view. This makes it possible to understand the relationships between Java objects and to develop a greater understanding of the Java object interactions and the memory requirements of the application code.
- Analyze Java collection usage
  - Array and collection fill ratio - Memory Analyzer allows you to understand the efficiency of object arrays and collections by informing you of the fill ratio, that is, the ratio of used elements to the array or collection size. This shows how efficiently the collections are being used.
  - Map collision ratio - For map-like collections that are keyed on object hash codes, Memory Analyzer provides an understanding of the collision ratio for those collections. It will also show the key value pairs for these collections.

- Automate custom analysis
  - Object Query Language (OQL) - Memory Analyzer contains an SQL-like query for running Java object and field level analysis of the Java heap, using classes as tables, objects as rows, and field or attributes as columns. This makes it possible to generate reusable queries to locate certain objects or object collections of interest.
  - Create MAT plug-ins that use the MAT snapshot and the IBM DTFJ APIs - Memory Analyzer provides Eclipse extension points to produce custom reports against a Java API that provides representations of the Java heap, the data on the Java heap, and the relationships between Java objects. In addition, Memory Analyzer provides access to the IBM DTFJ API, giving access to all of the data available in the operating system dump.

Memory Analyzer is delivered in the IBM Support Assistant (ISA) Workbench. ISA is a free software offering that provides a single point of access for the IBM Monitoring and Diagnostic Tools for Java. When new versions of the tools become available, ISA notifies you and helps you retrieve the latest version. Using ISA helps you to troubleshoot and fix problems in your Java application.

You can expand the capabilities of Memory Analyzer using the IBM Extensions for Memory Analyzer from alphaWorks®[1]. The IBM extensions provide the ability to easily analyze the state of IBM software products, including the WebSphere Application Server.

Figure 6-11 on page 311 shows a Memory Analyzer session running in the IBM Support Assistant Workbench.

[1] http://alphaworks.ibm.com/tech/iema

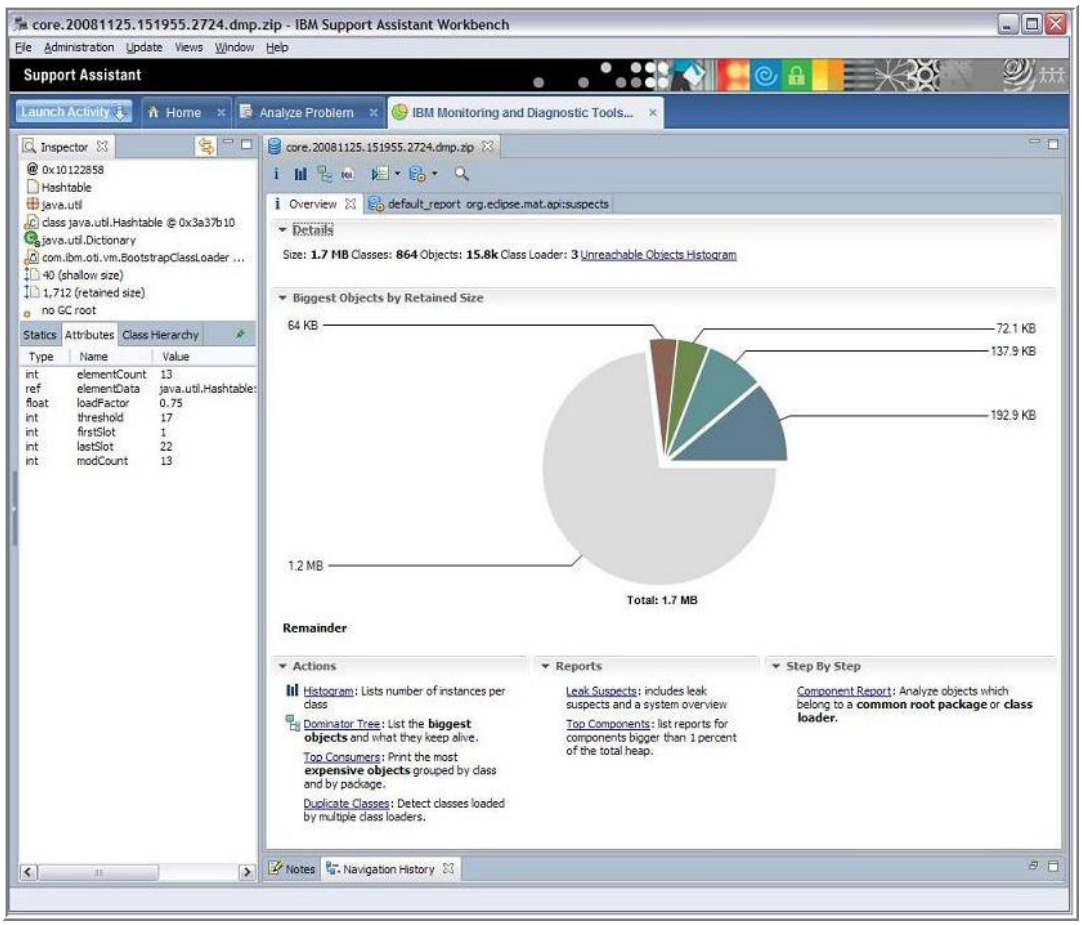


Figure 6-11 ISA JVM Memory Analyzer

6.3.2 Other useful performance advisors and analyzers

In this section we discuss additional performance advisors and analyzers.

IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer (GCMV)

What it is used for: Analyzing and visualizing verbose GC logs to help you:

- Monitor and fine tune Java heap size and garbage collection performance.
- Flag possible memory leaks.
- Size the Java heap correctly.
- Select the best garbage collection policy.

Description: The IBM Monitoring and Diagnostic Tools for Java - Garbage Collection and Memory Visualizer (GCMV) provides analysis and views of your application's verbose GC output. GCMV displays the data in both graphical and tabulated form. It provides a clear summary and interprets the information to produce a series of tuning recommendations, and it can save reports to HTML, JPEG or .csv files (for export to spreadsheets). GCMV parses and plots various log types including:

- Verbose GC logs
- -Xtgc output
- Native memory logs (output from ps, svmon and perfmon)

IBM Monitoring and Diagnostic Tools for Java - Dump Analyzer

What it is used for: Analyzing system dumps produced by IBM JVMs to diagnose typical problems such as:

- Out of memory
- Deadlocks
- Java Virtual Machine (JVM) crashes
- Java Native Interface (JNI) crashes

Description: The IBM Monitoring and Diagnostic Tools for Java - Dump Analyzer performs automated analysis of dump files produced by the IBM Java VM. Starting with the name of the dump to be analyzed the analysis attempts to localize the problem and if successful produces a diagnosis of the error together with sufficient information to fix it or suggestions on how to continue the analysis using other tools (for example, Memory Analyzer to diagnose out-of-memory situations). If localization fails, then the tool will default to producing summary information from the dump intended to aid further diagnosis.

IBM Monitoring and Diagnostic Tools for Java - Health Center

What it is used for: Interactive analysis of JVM problems using post mortem artifacts such as core files or javacores.

Description: The IBM Monitoring and Diagnostic Tools for Java - Interactive Diagnostic Data Explorer (IDDE) is a lightweight tool that helps you quickly get information from the artifact you are investigating and where you are not sure what the problem is and you want to avoid launching resource-intensive analysis. It supports the following features and more:

- System cores, IBM javacores and PHD files
- Full content assist for available commands
- Syntax highlighting
- Investigation log, which mixes free text with live session data
- Multiple JVM support

IBM Thread and Monitor Dump Analyzer for Java (TMDA)

What it is used for: Analyzing Java core files to help you identify threading problems such as:

- Hangs
- Deadlocks
- Resource contention
- Bottlenecks

Description: The IBM Thread and Monitor Dump Analyzer for Java (TMDA) analyzes javacores and diagnoses monitor locks and thread activities to identify the root cause of hangs, deadlocks, and resource contention or bottlenecks. It compares each javacore and provides process ID information for threads, garbage collection frequency, allocation failure frequency, and a list of hang suspects.

IBM Monitoring and Diagnostic Tools for Java - Interactive Diagnostic Data Explorer

What it is used for: Interactive analysis of JVM problems using post mortem artifacts such as core files or javacores.

Description: The IBM Monitoring and Diagnostic Tools for Java - Interactive Diagnostic Data Explorer (IDDE) is a lightweight tool that helps you quickly get information from the artifact you are investigating, where you are not sure what the problem is and you want to avoid launching resource-intensive analysis. It supports the following features and more:

- System cores, IBM javacores and PHD files
- Full content assist for available commands
- Syntax highlighting
- Investigation log, which mixes free text with live session data
- Multiple JVM support

IBM Pattern Modeling and Analysis Tool for Java Garbage Collector

What it is used for: Analyzing verbose GC logs to help you:

- Fine tune the Java heap
- Visualize garbage collection behavior
- Determine whether memory might be leaking

Description: IBM Pattern Modeling and Analysis Tool for Java Garbage Collector (PMAT) parses verbose GC logs to show how heap use changes over time as a result of garbage collection activity. Its graphical and tabular reports help you tell if there is excessive memory usage, if the heap is becoming fragmented, and if memory might be leaking.

While carrying out analysis of JVM behavior, it is essential to understand the basic structure and hierarchy of classes, and interaction between JVM, middleware, and applications. Figure 6-12 shows this interaction.

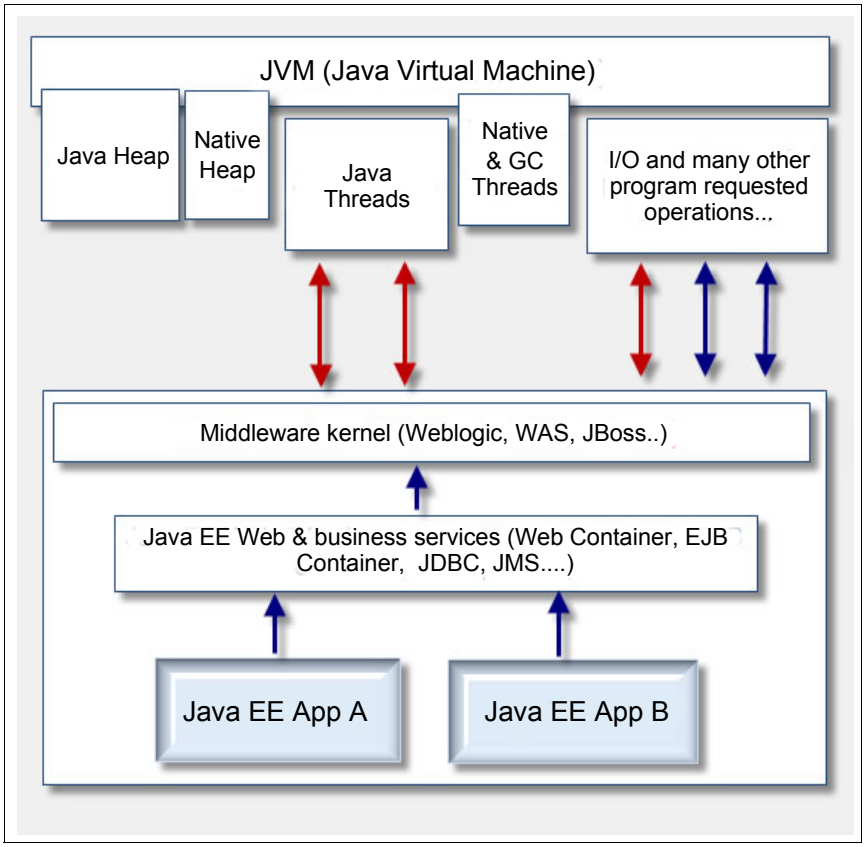


Figure 6-12 JVM interaction with middleware and applications

A

Performance monitoring tools and what they are telling us

This appendix discusses performance monitoring tools and explains what the tools are telling us while trying to tune your IBM POWER Systems environment.

The following topics are discussed:

- NMON
- lpar2rrd
- Trace tools and PerfPMR

NMON

Any AIX administrator who has worked on any performance issue should already be very familiar with and fond of the **nmon** tool. For those who have not, here is a short history lesson:

- **nmon** (Nigel's performance MONitor) was created in 1997 by Nigel Griffiths, an employee of IBM UK Ltd. He saw a need for a simple, small and safe tool to monitor AIX in real-time, and additionally capture numerous elements of AIX usage including processor, processes, memory, network and storage.
- Originally written for himself, news of its existence quickly spread.
- The first releases supported AIX 4 (which was the current major release at the time) and Nigel continued to improve and enhance **nmon** to support subsequent AIX 5 and 6 releases.
- Included and installed by default since 2008 with AIX from 5.3 TL09, 6.1 TL02, VIOS 2.1 and beyond. The version included with AIX is maintained and supported by the AIX Performance Tools team.
- **nmon** has since been ported to Linux and is included in various distributions for PowerPC (PPC), x86 and even Raspberry Pi.

Note: Because **nmon** is included with all current releases of AIX, there is no need to install a legacy **nmon** release (known as classic **nmon**). Classic **nmon** was written for versions of AIX no longer supported and that do not contain support for newer hardware or software generations. Neither IBM nor Nigel support or recommend the use of classic **nmon** on any current AIX releases.

lpar2rrd

lpar2rrd is a tool based on perl that connects to HMC (or IVM), via SSH, and collects performance data. It is agentless and collects processor utilization for each LPAR and global processor utilization of the frames managed by the HMCs. Data is stored in an RRD database. Graphs are automatically generated and presented by a web server. Information is available for each LPAR or grouped in shared processor pools and entire frames.

The tool is free (GPL) and there is an option to contract support to get enhanced features. lpar2rrd can be downloaded at:

```
www.lpar2rrd.com
```

It can be installed in any UNIX. The version used in this book is 3.20.

Trace tools and PerfPMR

The AIX system trace is designed for tracing of activities inside both the kernel and kernel extension. When trace is activated, the selected trace events are recorded in an internal buffer. The data in the buffer is written into the trace log file later. The raw trace data is in binary format, and AIX provides further utilities to process it and generate readable report from different perspectives. Such utilities include **trcrpt**, **tprof**, **curt**, **splat**, **pprof**, **filemon**, and **netpmon**.

The tracing utilities are very useful for identifying difficult performance problems. In the following section we discuss some details on how to use the tools and read the reports.

Note: Although trace facilities are designed to have little impact on system performance, it is not introduced for auditing purposes. You should not use it as a batch job in production systems unless you are required to by the IBM support personnel.

AIX system trace basics

The system trace is activated using the **trace** command. Here we introduce the trace command and some of the trace options, and offer examples on how to use it.

A quick start example

Example A-1 shows how to get a **filemon** report from **trace**. This is useful when you need to postprocess the trace log file from a remote machine, or when the system load is high and trace events are being missed by **filemon**.

Example A-1 Quick start example of using the trace utilities

```
#trace -andfp -J filemon -C all -r PURR -T 30000000 -L 30000000 -o trace.raw
#trcon
#sleep 5
#trcstop
#gensyms -F > gensyms.out
#/usr/bin/trcrpt -C all -r trace.raw >  trace.bin

#filemon -i trace.bin -n gensyms.out -O all,detailed -o filemon.out
```

System trace mode

The system trace can be used in asynchronous mode or in subcommand mode. Choose asynchronous mode by specifying the **-a** option with the **trace** command. The trace daemon runs in the background in this mode, and you can use **trcon**, **trcoff**, and **trcstop** to start tracing, stop tracing, and exit a tracing session, respectively (Example A-2).

Example A-2 Start, stop system trace in asynchronous mode

```
To start trace daemon and trace data collection:
#trace -a
To stop:
#trcstop

To start trace daemon and delay the data collection
#trace -a -d

To start data collection:
#trcon

To stop data collection:
#trcoff

To exit tracing session:
#trcstop
```

The system **trace** is in subcommand mode if **-a** is not specified. You get a dialog to interact with the tracing facilities in this mode, which is seldom used and is not discussed here.

Default trace setting

Get the default trace setting with the command **trcctl** as shown in Example A-3. You can see that the default trace log file is located in `/var/adm/ras` with the name `trcfile` and size `2621440`.

Example A-3 Get the default trace settings using trcctl

```
# trcctl -l
Default Buffer Size: 262144
Default Log File Size: 2621440
Default Log File: /var/adm/ras/trcfile
Non-Root User Buffer Size Maximum: 1048576
Default Components Directory: /var/adm/ras/trc_ct
Default LMT Log Dir: /var/adm/ras/mtrcdir
```

We suggest that you keep the default value of **trcctl** and use trace options to set values other than default, which are discussed later. If you change it by mistake, you can use **trcctl -r** to restore the default settings, as shown in Example A-4.

Example A-4 Restore the default trace settings using trcctl

```
# trcctl -r
```

System trace buffer size and buffer mode options

The system trace utilities have three buffer modes. Table A-1 shows the available buffer modes and the corresponding options in parentheses.

Table A-1 Available trace buffer modes

Buffer mode	Description
Alternative (default)	Two trace buffers are allocated. All trace events written into the buffers are captured in the trace log file. The data in one buffer will be written to the trace log file while trace events start recording into the alternative buffer. Trace buffer wraparound will occur if the buffer switching happens and the trace events are not written into the trace log file in a timely manner. Also, the trace log wraparound will occur when it fills up and the trace daemon continues to collect trace data. However, this can be avoided by specifying the **-s** option with trace, to stop tracing when the trace log fills.
Circular (**-l**)	The trace events wrap within the trace buffers and are not captured in the trace log file until the trace data collection is stopped.
Single (**-f**)	The collection of trace events stops when the trace buffer fills up and the contents of the buffer are captured in the trace log file. This option is frequently used to analyze performance issues. Most of the examples used in Appendix A adopt this option, such as Example A-1, Example A-10, Example A-11, and so on.

Use **-T** to specify a trace buffer size other than the default. For alternative and circular mode, the buffer size ranges from 8192 bytes to 268,435,184 bytes. For single buffer mode, the buffer size ranges from 16,392 bytes to 536,870,368 bytes.

If you specify a trace buffer size using **-T**, you should also need to specify the trace log file size using **-L**. The following criteria must be met:

- In the circular and alternate modes, the trace buffer size must be one-half or less the size of the trace log file.
- In the single mode, the trace log file must be at least the size of the buffer.

By default, all logical processors share the same trace buffer. However, the system trace subsystem can use separate trace buffers for each logical processor, too. This is useful to avoid the situation that one logical processor has much more activities and overflows the trace events of other logical processors. This is achieved by specifying the **-C all** option.

Note: The trace buffer is pinned in memory, and so it consumes more physical memory if you specify a larger value.

Trace hooks and event groups

Because there are large numbers of trace events, AIX uses trace event identifiers and trace event groups to categorize and manage them. The trace event identifier is also called trace hook identifier.

Example A-5 shows how to get all the trace hook identifiers using **trcrpt**. The four digits at the beginning of each line are the trace event identifier, for example, 0010 is the trace event identifier for the TRACE ON event.

Example A-5 Using trcrpt -j to get all the track hook identifiers

```
#trcrpt -j|pg
0010 TRACE ON
0020 TRACE OFF
0030 TRACE HEADER
0040 TRACEID IS ZERO
0050 LOGFILE WRAPAROUND
0060 TRACEBUFFER WRAPAROUND
0070 UNDEFINED TRACE ID
0080 DEFAULT TEMPLATE
0090 trace internal events
00a0 TRACE_UTIL
1000 FLIH
1001 OFED Trace
1002 STNFS GENERAL
1003 STNFS VNOPS
1004 STNFS VNOPS HOLD/RELE
1005 STNFS ID
1006 STNFS IO
1007 STNFS OTW
1008 STNFS TREE
1009 STNFS VFSOPS
100b TRACE_UNMANAGED
1010 SYSTEM CALL
101d DISPATCH SRAD
101e DISPATCH Affinity
1020 SLIH
```

```
1022 MSTOR_ERR
1025 SFWCOM
1027 THRD_REAFF
1028 THRD_MREAFF
1029 SFW
...
```

Trace event groups are defined sets of trace events. Example A-6 shows how to get the trace event group using **trcrpt**. The name at the beginning of each stanza is the name of the trace group. For example, `tidhk` is the name of the group “Hooks needed to display thread name”.

Example A-6 Using trcrpt to get all trace event groups

```
#trcrpt -G|pg
tidhk - Hooks needed to display thread name (reserved)
        106,10C,134,139,465
...
tprof - Hooks for TPROF performance tool (reserved)
        134,139,210,234,38F,465,5A2,5A5,5D8
pprof - Hooks for PPROF performance tool (reserved)
        106,10C,134,135,139,419,465,467,4B0,5D8
filemon - Hooks for FILEMON performance tool (reserved)

101,102,104,106,107,10B,10C,10D,12E,130,137,139,154,15B,163,19C,1BA,1BE,1BC,1C9,22
1,222,228,232,2A1,2A2,3D3,419,45B,4B0,5D8,AB2
netpmon - Hooks for NETPMON performance tool (reserved)

100,101,102,103,104,106,107,10C,134,135,139,163,19C,200,210,211,212,213,215,216,25
2,255,256,262,26A,26B,2A4,2A5,2A7,2A8,2C3,2C4,2DA,2DB,2E6,2E7,2EA,2EB,30A,30B,320,
321,32D,32E,330,331,334,335,351,352,38F,419,465,467,46A,470,471,473,474,488,48A,48
D,4A1,4A2,4B0,4C5,4C6,598,599,5D8
curt - Hooks for CURT performance tool (reserved)

100,101,101D,102,103,104,106,10C,119,134,135,139,200,210,215,38F,419,465,47F,488,4
89,48A,48D,492,4B0,5D8,600,603,605,606,607,608,609
splat - Hooks for SPLAT performance tool (reserved)
        106,10C,10E,112,113,134,139,200,419,465,46D,46E,492,5D8,606,607,608,609
perftools - Hooks for all six performance tools (reserved)

100,101,101D,102,103,104,106,107,10B,10C,10D,10E,112,113,119,12E,130,134,135,137,1
39,154,15B,163,19C,1BA,1BC,1BE,1BC,1C9,200,210,211,212,213,215,216,221,222,228,232
,234,252,255,256,262,26A,26B,2A1,2A2,2A4,2A5,2A7,2A8,2C3,2C4,2DA,2DB,2E6,2E7,2EA,2
EB,30A,30B,320,321,32D,32E,330,331,334,335,351,352,38F,3D3,419,45B,465,467,46A,46D
,46E,470,471,473,474,47F,488,489,48A,48D,492,4A1,4A2,4B0,4C5,4C6,598,599,5A2,5A5,5
D8,600,603,605,606,607,608,609,AB2
tcpip - TCP/IP PROTOCOLS AND NETWORK MEMORY (reserved)
        252,535,536,537,538,539,254,25A,340
...
```

Trace event groups are useful because you usually need to trace a collection of trace events to identify a specific functional or performance problem. As an example, the most frequently used trace-based tools **tprof**, **pprof**, **filemon**, **netpmon**, **curt**, and **splat** all have their corresponding specific trace event groups that contain all necessary trace event identifiers for generating reports.

You can also use **trcevgrp -l** to display details about specific event groups, as shown in Example A-7.

Example A-7 Using trcevgrp to display specific event groups

```
#trcevgrp -l filemon
filemon - Hooks for FILEMON performance tool (reserved)

101,102,104,106,107,10B,10C,10D,12E,130,137,139,154,15B,163,19C,1BA,1BE,1BC,1C9,22
1,222,228,232,2A1,2A2,3D3,419,45B,4B0,5D8,AB2
```

Choosing trace hooks and event groups

You can choose the trace hooks and event groups using specific **trace** options, shown in Table A-2.

Table A-2 Trace options and descriptions

Trace options	Description
-j <event identifier>	Specifies the user-defined events to collect trace data[a]. Specifies a two-digit hook ID in the form hh as in hh00, hh10,...,hhF0. Specifies a three-digit hook ID in the form hhh as in hhh0. Specifies a four-digit hook ID in the form hhhh as in hhhh.
-J <event group>	Specifies the event groups to be included[b].
-k <event identifier>	Specifies the user-defined events to exclude trace data[a].
-K <event group>	Specifies the event groups to be excluded[b].

a. Multiple events can be separated by commas.
b. Multiple groups can be separated by commas.

Note: Usually, you need to include the `tidhk` event group to get detailed process and thread information, and exclude `vmm` events if it is irrelevant to the problem you are analyzing, because usually there are lots of `vmm` events.

Merging trace log files and reordering

In alternative mode, the trace daemon normally wraps around the trace log when it fills up and continues to collect trace data, unless the **-s** option is specified. In circular mode, the trace buffer wraparound might occur. Due to the wraparound events, the trace log file might not be in chronological order. You need the trace log file in chronological order to do the post processing with **curt**, **tprof**, **filemon**, and so on.

You can force the reordering with the **-r** command, as shown in Example A-8.

Example A-8 Reordering the trace log file

```
The trace.raw is the original trace file, and we can reoder it chronologically to
trace.r.
#trcrpt -o trace.r -r trace.raw
```

In systems with multiple logical processors and trace with the **-C all** option, you need to specify the **-C all** option with the **-r** option to merge, reorder and sort the trace log files into one raw data file in chronological order. We suggest that you perform this step before every post processing. Refer to Example A-9 on page 322.

Example A-9 Merging, reordering, and sorting the trace log file in chronological order

```
#trcrpt -o trace.r -r -C all trace.raw
```

Generate the trace report

We demonstrated using **trcrpt** to generate a report in ASCII format in 4.4.3, “File system best practice” on page 163. Specifying the **-O** option can generate more detailed reports, shown in Table A-3. Multiple options can be separated by commas. **smitty** and **trcrpt** are also available for use.

Table A-3 trcrpt -O options

Frequently used trcrpt -O options	Explanation
pid=on	Displays the process IDs in the trace report.
tid=on	Displays the thread ID in the trace report.
exec=on	Displays the exec path names in the trace report.
svc=on	Displays the value of the system call in the trace report.
timestamp=[0|1|2|3|4]	The most frequently used values are 0 and 1. 0 - Time elapsed since the trace was started and delta time from the previous event. The elapsed time is in seconds and the delta time is in milliseconds. This is the default. 1 - Short elapsed time. Reports only the elapsed time (in seconds) from the start of the trace.
cpuid=on	Displays the physical processor number in the trace report.

Example A-10 shows how to start the trace data collection and generate a trace report.

Example A-10 General a trace report

```
//comments: start trace immediately.
#trace -a -f -T 10000000 -L 10000000 -o trace.raw

//comments: monitor for 5 seconds; then stop tracing.
#sleep 5; trcstop

//comments: generate trace report “trace.int” from the trace log file “trace.raw”.
#trcrpt -Opid=on,tid=on,exec=on,svc=on,cpuid=on,timestamp=1 -o trace.int
./trace.raw

//comments: now you can read the trace report.
#more trace.int
...
```

Note: This example does not specify specific trace hooks, or exclude any trace hooks. Thus all trace events are logged. The trace buffer might get filled up before any useful trace events are written.

You can now get a more detailed report using the **-0** option for similar DIO demotion cases; refer to Example A-11. From this report, you can see that “dd” is causing the IO demotion with unaligned IO length (0x1001).

Example A-11 Using trcrpt with -O options

```
//comments: include hook id “59b” for DIO activities, and “tidhk” for process name
reporting.
#trace -J tidhk -j 59b -a -f -T 10000000 -L 10000000 -o trace.raw

//comments: monitor for 5 seconds, and then stop tracing.
#sleep 5; trcstop

//comments: filter the trace report to display only hook id “59b” events.
#trcrpt -d 59b -Opid=on,tid=on,exec=on,svc=on,cpuid=on,timestamp=1 -o trace.int
./trace.raw
#more trace.int
...
ID   PROCESS NAME   CPU PID      TID      I SYSTEM CALL     ELAPSED   APPL
SYSCALL KERNEL  INTERRUPT
...
59B  dd             0   2162750  30933019                 0.000113
JFS2 IO write: vp = F1000A0232DF8420, sid = DE115E, offset = 00000000052FB2F6,
length = 1001
59B  dd             0   2162750  30933019                 0.000113
JFS2 IO dio move: vp = F1000A0232DF8420, sid = DE115E, offset = 00000000052FB2F6,
length = 1001
59B  dd             0   2162750  30933019                 0.000126
JFS2 IO devstrat (pager strategy): bplist = F1000A00E067C9A0, vp =
F1000A0232DF8420, sid = DE115E, lv blk = A60650, bcount = 0400
59B  dd             4   2162750  30933019                 0.000492
JFS2 IO gather: bp = F1000A00E067D2E0, vp = F1000A0232DF8420, sid = DE115E, file
blk = 297D8, bcount = 2000
59B  dd             4   2162750  30933019                 0.000495
JFS2 IO devstrat (pager strategy): bplist = F1000A00E067D2E0, vp =
F1000A0232DF8420, sid = DE115E, lv blk = A60650, bcount = 1400
59B  dd             4   2162750  30933019                 0.000514
JFS2 IO dio demoted: vp = F1000A0232DF8420, mode = 0001, bad = 0002, rc = 0000,
rc2 = 0000
```

If using trace data from another machine, you need the trcnm output and the `/etc/trcfmt` file to generate the ASCII report, otherwise the output will be incorrect. Refer to Example A-12.

Example A-12 Generate a trace report for a remote machine

```
On remote machine,
#trcnm > trcnm.out
#cp /etc/trcfmt trace.fmt

On reporting machine,
Download the “trcnm.out” and “trace.fmt” together with the trace raw data.
#trcrpt -n trcnm.out -t trace.fmt <other options>
```

Generate curt, tprof, filemon, splat, and netpmon reports from trace logs

You can generate **curt**, **tprof**, **pprof**, **splat**, **filemon**, and **netpmon** reports from the trace log if the related trace events are included in the trace log. Example A-13 shows an example of using the trace log to generate a **curt** report.

Example A-13 Generate a curt report using trace log

```
//comments: start trace using single buffer mode to collect trace event group
“curt”, and delay starting of the trace data collection.
#/usr/bin/trace -andfp -C all -J curt -r PURR -T 20000000  -L 20000000  -o
trace_bin

//comments: start trace data collection and monitor for 10 seconds, then stop
tracing.
#trcon; sleep 10; trcstop

//comments: gather the symbol information necessary to run the “curt” command.
#gensyms > gensyms.out

//comments: reorder the trace logs to a single trace raw file “trace_curt”.
#trcrpt -C all -r trace_bin > trace_curt

//comments: generate curt report
#/usr/bin/curt -i trace_curt -n  gensyms.out -ptes > curt.out
```

Example A-14 shows an example of using the trace log to generate a **tprof** report. Note that to run the **tprof** command in manual offline mode, the following files must be available:

- The symbolic information file `rootstring.syms`
- The trace log file `rootstring.trc [-cpuid]`

Example A-14 Generate a tprof report using trace log

```
//comments: start trace using single buffer mode to collect trace event group
“tprof”, and delay starting of the trace data collection.
#/usr/bin/trace -andfp -C all -J tprof -r PURR -T 20000000  -L 20000000  -o
myprof.trc

//comments: start trace data collection and monitor for 10 seconds, then stop
tracing.
#trcon; sleep 10; trcstop

//comments: gather the symbol information necessary to run the “tprof” command.
#gensyms > myprof.syms

//comments: generate tprof report; rootstring equals to “myprof” here.
#tprof -uskejzlt  -r myprof
# more myprof.prof
```

For a detailed explanation of the `tprof` and `curt` reports, refer to *POWER7 and POWER7+ Optimization and Tuning Guide*, SG24-8079.

Example A-15 on page 325 shows an example of using the trace log to generate a **filemon** report. It is good practice to avoid “trace events lost” by using **-f** to select the single buffer mode, as in this example.

Example A-15 Generate a filemon report using the trace log

```
//comments: start trace using single buffer mode to collect trace event group
“filemon”, and delay starting of the trace data collection.
#trace -andfp -J filemon -C all -r PURR -T 20000000 -L 20000000 -o trace.raw

//comments: start trace data collection and monitor for 10 seconds, then stop
tracing.
#trcon; sleep 10; trcstop

//comments: gather the symbol information necessary to run the “filemon” command.
#gensyms -F > gensyms.out

//comments: reorder the trace logs to a single trace raw file “trace.fmon”.
#/usr/bin/trcrpt -C all -r trace.raw >  trace.fmon

//comments: generate filemon report “filemon.out”
#filemon -u -i trace.fmon -n gensyms.out -O all,detailed -o filemon.out
```

For a detailed explanation of the **filemon** report, refer to 4.4.4, “The filemon utility” on page 176.

Using the truss command

The **truss** command is useful when identifying application problems together with the system tracing utilities such as **tprof**, **curt**, and so on.

When we identifying the problematic process with tprof, curt, filemon, and so on, use the **truss** command to trace the process. Some frequently used options are in Table A-4.

Table A-4 truss options

truss options	Explanation
-p <pid>	Specifies the processes for tracing the execution.
-c	Counts traced system calls, faults, and signals instead of displaying trace results line by line. Produces a summary report after the traced command terminates or when **truss** is interrupted. Use this option alone or with **-f**.
-d	Includes a timestamp with each line of output.
-f	Follows all child processes.
-l	Displays the thread ID in the output.
-t[!]syscall	Includes or excludes system calls from the trace process.
-u [!] [LibraryName [...]::[!]FunctionName [...]]	Traces dynamically loaded user level function calls from user libraries.

Example A-16 gives an example of using **truss** to debug application issues.

Example A-16 Using truss to identify application issues

```
To get a summary report of the process, truss -c -p <pid> for an interval and then
Ctrl+C:
```

```
#truss -c -p 6619376
^Csyscall               seconds   calls  errors
kwrite                      .00       6
munmap                    12.06      81
msync                     24.36      81
mmap                        .00      79
                                   ----     ---    ---
sys totals:                 .00     247      0
usr time:                   .00
elapsed:                    .00

truss the execution of the process for an interval and Ctrl+C to stop truss:
#truss -d -o truss.log -l -p 6619376
^C
With -l, the thread id is shown at each line:
#more truss.log
Mon Oct 15 16:54:36 2012
29294829: 0.0000:        mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE,
MAP_FILE|MAP_VARIABLE|MAP_SHARED, 10, 29360128) = 0x3040000
0
30933087: kwrite(1, " m s y n c   i n t e r v"..., 45) = 45
30933087: 0.0007:        munmap(0x30000000, 4194304) = 0
30933087: 0.0014:        mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE,
MAP_FILE|MAP_VARIABLE|MAP_SHARED, 8, 20971520) = 0x30000000
29294829: 0.0062:        msync(0x30400000, 4194304, 32) = 0
30933087: 0.2532:        msync(0x30000000, 4194304, 32) = 0
29294829: 0.6684:        munmap(0x30400000, 4194304) = 0
30933087: 0.6693:        munmap(0x30000000, 4194304) = 0
29294829: 0.6699:        mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE,
MAP_FILE|MAP_VARIABLE|MAP_SHARED, 10, 29360128) = 0x3000000
0
30671061: 0.6702:        mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE,
MAP_FILE|MAP_VARIABLE|MAP_SHARED, 9, 25165824) = 0x30400000
29294829: 0.6751:        msync(0x30000000, 4194304, 32) = 0
30671061: 0.7616:        msync(0x30400000, 4194304, 32) = 0

Use “-t” flag to trace system calls, as follows:
#truss -tmmap,msync,munmap -p 6619376
munmap(0x30400000, 4194304)                     = 0
munmap(0x31400000, 4194304)                     = 0
mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE, MAP_FILE|MAP_VARIABLE|MAP_SHARED,
10, 29360128) = 0x30400000
munmap(0x30800000, 4194304)                     = 0
mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE, MAP_FILE|MAP_VARIABLE|MAP_SHARED,
8, 20971520) = 0x30800000
munmap(0x30C00000, 4194304)                     = 0
mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE, MAP_FILE|MAP_VARIABLE|MAP_SHARED,
6, 12582912) = 0x30C00000
msync(0x30800000, 4194304, 32)                  = 0

Use “-u” to trace dynamically loaded user level function calls from user
libraries, such as libc.a.
#truss -u libc.a::* -p 6619376
kwrite(1, " m s y n c   i n t e r v"..., 46)     = 46
->libc.a:gettimeofday(0x200dba98, 0x0)
```

```
->libc.a:gettimeofday(0x200f9a90, 0x0)
<-libc.a:gettimeofday() = 0             0.000000
munmap(0x31400000, 4194304)                     = 0
<-libc.a:gettimeofday() = 0             0.000000
->libc.a:gettimeofday(0x200f9a98, 0x0)
<-libc.a:gettimeofday() = 0             0.000000
mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE, MAP_FILE|MAP_VARIABLE|MAP_SHARED,
10, 29360128) = 0x30000000
mmap(0x00000000, 4194304, PROT_READ|PROT_WRITE, MAP_FILE|MAP_VARIABLE|MAP_SHARED,
9, 25165824) = 0x30400000
->libc.a:gettimeofday(0x200bda98, 0x0)
->libc.a:gettimeofday(0x200f9a90, 0x0)
```

Tip: You can implement most of the **truss** features using probevue, too. Refer to the probevue user guide at:

http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.genprogc/doc/genprogc/probevue_userguide.htm

Real case studies using tracing facilities

Here we introduce a real case scenario to explain how to use the trace tools to identify a performance problem, based on the approaches mentioned.

Note: To avoid disclosure of sensitive data, we used a sample program here to duplicate the client problem.

Problem description

The client complains that I/O is quite slow when writing to a mmaped data file using a multithreaded application. From the application log, the application occasionally hangs many seconds waiting for I/O to complete.

Also, the client believes the I/O pattern should be sequential but the bandwidth is limited, only 70 MBps on V7000 storage.

The sample code used to duplicate the client problem is in Example A-17. It opens a file, truncates it to size so that there is 4 MB space for each thread. Then all the threads will **mmap** their own 4 MB space, modify it, **msync** it, and then **munmap** it. We simply duplicated the client problem by running ./testmmap 8.

Example A-17 Sample code for real case studies

```
/*
 * The following [enclosed] code is sample code created by IBM
 * Corporation.  This sample code is not part of any standard IBM product
 * and is provided to you solely for the purpose of demonstration.
 * The code is provided 'AS IS',
 * without warranty of any kind.  IBM shall not be liable for any damages
 * arising out of your use of the sample code, even if they have been
 * advised of the possibility of such damages.
 */
/*
To compile: xlC_r testmmap.cpp -o testmmap
```

```
Problem report: chenchih@cn.ibm.com
*/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <sys/select.h>

#define MAXTHREADNUM 128
#define INTERVAL 10000
#define BUF_SIZE 4*1024*1024


int testmmap(int arg)
{
    int fd = 0, n = 0, i = 0;
    timeval start, end;
    void *mp = NULL;
    struct stat sbuf;
    char idx = 0;
    long interval;
    int mplen = BUF_SIZE;

    if((fd = open("m.txt", O_RDWR, 0644)) >= 0)
    {
       fstat(fd, &sbuf);
       printf("file size = %d\n", sbuf.st_size);
    }
    else
    {
       printf("open failed\n");
       return 0;
    }

    while(1)
    {
      idx++;
      if((mp = mmap(NULL, mplen, PROT_READ|PROT_WRITE, MAP_SHARED, fd,
arg*BUF_SIZE)) != (void*)-1 )
      {
            memset(mp, idx, mplen);

            msync(mp, mplen, MS_SYNC);

            munmap(mp, mplen);
      }
    } //while

    close(fd);
```

```
    return 0;
}

extern "C" void* testfunc(void* arg)
{
    unsigned int i=0;
    int ret = 0;
    testmmap((int)arg);
    return NULL;
}


int main(int argc, char* argv[])
{
    pthread_t tid[MAXTHREADNUM] = {0};
    int ret;
    struct stat sbuf;

    if(argc < 2)
    {
        printf("usage: %s <num>\n", argv[0]);
        return 0;
    }

    int count = atoi(argv[1]);
    if(count > MAXTHREADNUM)
        count = MAXTHREADNUM;

    int fd, bufsz;
    bufsz = count * BUF_SIZE;
    if((fd = open("m.txt", O_RDWR|O_CREAT, 0644)) > 0)
    {
     fstat(fd, &sbuf);
     printf("file size = %d\n", sbuf.st_size);

      ftruncate(fd, bufsz);  //file is truncated, this might cause fragmentation
if the file is not allocated.
      fstat(fd, &sbuf);
      printf("file size after truncate = %d\n", sbuf.st_size);
    }
    close(fd);


    for(int i =0; i < count; i++)
    {
        ret = pthread_create(&tid[i], NULL, testfunc, (void*)i);
        if(ret != 0)
        {
            printf("pthread_create error, err=%d\n", ret);
            return -1;
        }
    }

    void* status;
    for(int i =0; i < count; i++)
```

```
    {
        pthread_join(tid[i], &status);
        printf("thread %d exit status = %d\n", tid[i], (int)status);
    }
    return 0;
}
```

Problem analysis

We generated a **curt** report, shown in Example A-18, using the approach illustrated in Example A-13 on page 324. We found that a “msync” system call was very slow, with 261.1781 ms avg elapsed time, and 610.8883 ms max elapsed time, while the average processor time consumed was only 1.3897 ms. The behavior seemed quite abnormal. Looking into the process detail part, we could see that the msync subroutine is called by the testmmap process. The PID of testmmap was 17629396.

Example A-18 Curt report

```
#more curt.out
...
System Calls Summary
                    --------------------
   Count   Total Time   % sys  Avg Time  Min Time  Max Time  Tot ETime  Avg ETime
Min ETime  Max ETime  SVC (Address)
              (msec)    time    (msec)    (msec)    (msec)     (msec)     (msec)
(msec)     (msec)
========  ===========  ======  ========  ========  ========   ========  =========
=========  =========  =================
      48     66.7060   0.32%    1.3897    0.0279    7.5759  12536.5502   261.1781
56.6501   610.8883  msync(32a63e8)
      47     21.8647   0.10%    0.4652    0.4065    0.5593  7108.8461   151.2520
3.1755   558.8734  munmap(32a63d0)
      55      0.8696   0.00%    0.0158    0.0071    0.0274   509.6747     9.2668
2.4847    20.3143  mmap(32a6400)
...

Process Details for Pid: 17629396
    Process Name: testmmap
    8 Tids for this Pid: 38731907 37945371 32964661 29622293 29425739 26476695
    9961589 9502927

 Total Application Time (ms):  70.766818
 Total System Call Time (ms):  89.458826
 Total Hypervisor Call Time (ms):  8.406510

                           Process System Call Summary
                           ---------------------------
   Count   Total Time   % sys  Avg Time  Min Time  Max Time  Tot ETime  Avg ETime
Min ETime  Max ETime  SVC (Address)
              (msec)    time    (msec)    (msec)    (msec)     (msec)     (msec)
(msec)     (msec)
========  ===========  ======  ========  ========  ========   ========  =========
=========  =========  =================
      48     66.7060   0.32%    1.3897    0.0279    7.5759  12536.5502   261.1781
56.6501   610.8883  msync(32a63e8)
```

```
     47     21.8647   0.10%    0.4652    0.4065    0.5593  7108.8461   151.2520
3.1755   558.8734  munmap(32a63d0)
     55      0.8696   0.00%    0.0158    0.0071    0.0274   509.6747     9.2668
2.4847    20.3143  mmap(32a6400)

                    Pending System Calls Summary
                    ----------------------------
Accumulated   Accumulated   SVC (Address)                 Tid
Time (msec)   ETime (msec)
============  ============  ========================  ================
      0.0054       7.7179   msync(32a63e8)             38731907
      0.0050       1.5159   msync(32a63e8)             9502927
      0.0042       6.4927   msync(32a63e8)             37945371
      0.0022       5.3160   msync(32a63e8)             9961589
      0.0019      13.7364   mmap(32a6400)              32964661
```

Then we could trace the msync system call via the **trace** or **truss** command. Because we already had the process ID here, it was simpler to use **truss** directly, as in Example A-19. We saw that the msync I/O size was always 4194304 (4 MB). According to the application logic, the 4 MB is purely dirty pages, so it is a large block of sequential I O, and should be very fast.

Example A-19 trace msync subroutine

```
#truss -d -fael -tmsync,munmap -p 17629396
17629396:       psargs: ./testmmap 8
Tue Oct 16 12:58:09 2012
17629396:       32964661: 0.0000:        munmap(0x30400000, 4194304) = 0
17629396:       9961589: 0.0008:        munmap(0x30C00000, 4194304) = 0
17629396:       9502927: 0.0014:        munmap(0x30800000, 4194304) = 0
17629396:       26476695: 0.0022:        munmap(0x31800000, 4194304) = 0
17629396:       9961589: 0.0074:        msync(0x30000000, 4194304, 32) = 0
17629396:       29622293: 0.0409:        munmap(0x31400000, 4194304) = 0
17629396:       9502927: 0.0419:        msync(0x30400000, 4194304, 32) = 0
17629396:       26476695: 0.6403:        msync(0x30C00000, 4194304, 32) = 0
17629396:       37945371: 0.6723:        munmap(0x31000000, 4194304) = 0
17629396:       29622293: 0.6779:        msync(0x31800000, 4194304, 32) = 0
17629396:       38731907: 0.7882:        msync(0x30800000, 4194304, 32) = 0
17629396:       29622293: 0.9042:        munmap(0x31800000, 4194304) = 0
17629396:       9502927: 0.9049:        munmap(0x30400000, 4194304) = 0
17629396:       32964661: 0.9104:        msync(0x31C00000, 4194304, 32) = 0
17629396:       9502927: 0.9105:        msync(0x30400000, 4194304, 32) = 0
17629396:       9961589: 0.9801:        munmap(0x30000000, 4194304) = 0
17629396:       38731907: 0.9809:        munmap(0x30800000, 4194304) = 0
17629396:       32964661: 0.9815:        munmap(0x31C00000, 4194304) = 0
17629396:       29425739: 0.9829:        msync(0x31400000, 4194304, 32) = 0
17629396:       32964661: 1.0529:        msync(0x30800000, 4194304, 32) = 0
```

However, from **filemon**, the average I/O size underneath msync was 8.3 512-byte blocks (~4 KB), as in Example A-20. Also, the seek ratio was 100%, which means the I/O is purely random, while the seek distance is quite short, only about 110.7 blocks (~55 KB).

Example A-20 filemon report

```
------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------
```

```
VOLUME: /dev/fslv05  description: /data
writes:                 23756   (0 errs)
  write sizes (blks):   avg     8.3 min       8 max    1648 sdev    21.3
  write times (msec):   avg 128.851 min   3.947 max 493.747 sdev 117.692
  write sequences:      23756
  write seq. lengths:   avg     8.3 min       8 max    1648 sdev    21.3
seeks:                  23756   (100.0%)
  seek dist (blks):     init   5504,
                        avg   110.7 min       8 max   65904 sdev  1831.6
  seek dist (%tot blks):init 0.01312,
                        avg 0.00026 min 0.00002 max 0.15713 sdev 0.00437
time to next req(msec): avg   0.107 min   0.000 max 1423.920 sdev  10.018
throughput:             37427.0 KB/sec
utilization:            0.44
PS: we can also get some clue of this using iostat command.
```

Usually such issues are likely caused by file fragmentation. We used the **fileplace** command to confirm this, as in Example A-21. There were 7981 fragments in the 32 MB file.

Example A-21 fileplace -pv output

```
#fileplace -pv m.txt|pg

File: m.txt  Size: 33554432 bytes  Vol: /dev/fslv05
Blk Size: 4096  Frag Size: 4096  Nfrags: 8192
Inode: 6  Mode: -rw-r--r--  Owner: root  Group: system

  Physical Addresses (mirror copy 1)
Logical Extent
  ----------------------------------
----------------
  26732319          hdisk1             1 frags         4096 Bytes,   0.0%
00000671
  26732324-26732330  hdisk1            7 frags        28672 Bytes,   0.1%
00000676-00000682
  26732332          hdisk1             1 frags         4096 Bytes,   0.0%
00000684
  26732337          hdisk1             1 frags         4096 Bytes,   0.0%
00000689
  26732348          hdisk1             1 frags         4096 Bytes,   0.0%
00000700
  26732353          hdisk1             1 frags         4096 Bytes,   0.0%
00000705
  ...
26738415          hdisk1             1 frags         4096 Bytes,   0.0%
00006767
  26738422          hdisk1             1 frags         4096 Bytes,   0.0%
00006774
  26738428          hdisk1             1 frags         4096 Bytes,   0.0%
00006780

  8192 frags over space of 8252 frags:   space efficiency = 99.3%
  7981 extents out of 8192 possible:   sequentiality = 2.6%
```

Problem solution

Further analysis shows that the fragmentation is caused by concurrent writes to an unallocated mmaped file (`m.txt`). You can create the `m.txt` in advance, and the problem will not occur, as in Example A-22. After the adjustment, the I/O bandwidth on V7000 storage is ~350 MBps, as compared to less than 70 MBps before adjustment.

Example A-22 Create and allocate the mmaped file before concurrent write

```
#dd if=/dev/zero of=./m.txt bs=1M count=32
#./testmmap 8
```

From **curt** and **filemon** output in Example A-23, the average elapsed time now is 26.3662; the average I/O is 2048 blocks, which is 1 MB in size, and the seek percent is 25.2%, which means the majority of I/O is sequential. Also you can see from the **fileplace** output at the bottom of Example A-23 that the file sequentially is 100%.

Example A-23 Curt and filemon report after adjustment

```
curt report:
...
System Calls Summary
                    --------------------
   Count   Total Time   % sys  Avg Time  Min Time  Max Time  Tot ETime  Avg ETime
Min ETime  Max ETime  SVC (Address)
              (msec)    time    (msec)    (msec)    (msec)     (msec)     (msec)
(msec)     (msec)
========  ===========  ======  ========  ========  ========   ========  =========
=========  =========  ================
     460    1885.0173   4.29%    4.0979    0.0192    9.0059  12128.4694    26.3662
10.4989    61.4130  msync(32a63e8)
     441     294.1300   0.67%    0.6670    0.0178    1.1844  2540.1148     5.7599
0.6690    33.9758  munmap(32a63d0)
     459      19.4737   0.04%    0.0424    0.0090    0.1565   751.7646     1.6378
0.0467     8.8085  mmap(32a6400)

filemon report:
...
------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/fslv05  description: /data
writes:                  3264    (0 errs)
  write sizes (blks):    avg  2048.0 min    2048 max    2048 sdev     0.0
  write times (msec):    avg   5.379 min   2.724 max  29.700 sdev   2.657
  write sequences:       822
  write seq. lengths:    avg  8132.2 min    2048 max   32768 sdev  3776.7
seeks:                   822     (25.2%)
  seek dist (blks):      init  22528,
                         avg 25164.7 min    2048 max   67584 sdev 16029.7
time to next req(msec): avg   3.062 min   0.001 max  39.448 sdev   5.873
throughput:              333775.0 KB/sec
utilization:             0.59

fileplace report:
#fileplace -pv m.txt
```

```
File: m.txt  Size: 33554432 bytes  Vol: /dev/fslv05
Blk Size: 4096  Frag Size: 4096  Nfrags: 8192
Inode: 6  Mode: -rw-r--r--  Owner: root  Group: system

  Physical Addresses (mirror copy 1)
Logical Extent
  ----------------------------------
----------------
  26732416-26740351  hdisk1          7936 frags     32505856 Bytes,  96.9%
00000768-00008703
  26740608-26740863  hdisk1           256 frags      1048576 Bytes,   3.1%
00008960-00009215

  8192 frags over space of 8448 frags:   space efficiency = 97.0%
  2 extents out of 8192 possible:   sequentiality = 100.0%
```

PerfPMR

PerfPMR is an official AIX Support tool. It provides a set of scripts to gather AIX performance data, including:

- 600 seconds (default) of general system performance data (monitor.sh 600)
- System trace data
  - Trace data for reporting
  - Trace data for post processing tools (curt, tprof, pprof, and splat)
  - Stand alone tprof and filemon data collection
- PMU events count data
- Hardware and software configuration data
- iptrace and tcpdump data

PerfPMR is available from this public website:

ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr

You can refer to the README in the PerfPMR package for data collection and how to send data to IBM. We suggest you get a baseline PerfPMR data package under normal situations in advance, and collect more performance data when the problem is occurring.

You can get more information about PerfPMR at the following website:

http://www-01.ibm.com/support/docview.wss?uid=aixtools-27a38cfb

Note: We suggest you only use the PerfPMR tools when collecting data on production systems. Do not use the **trace** command on production directly unless you are required to do so.

The hpmstat and hpmcount utilities

IBM Power processor provides on-chip performance monitor units (PMUs) to count the number of performance critical processor events. Use the PMU-based utilities to analyze specific processor performance problems including memory affinity, TLB misses, SLB misses, and so on.

Two commands can be used to count and analyze PMU events, **hpmcount** and **hpmstat**, respectively. The **hpmcount** command is used to collect PMU statistics, while the **hpmstat** command collects data system wide. We have already shown an example using **hpmstat** to monitor SLB misses in 4.2.3, “One TB segment aliasing” on page 129.

Note: The ASO/DSO functionality also uses PMU counters for performance analysis. When you are running the **hpmstat** and **pmcount** commands, the ASO/DSO might stop functioning for a while.

Here is another example using **hpmstat** to identify memory affinity issues. As from the **pmlist** output, Group #108 is used for memory affinity analyzing; refer to Example A-24.

Example A-24 Determine the processor event group for memory affinity

```
#pmlist -g -1|pg
...
Group #108: pm_dsource12
Group name: Data source information
Group description: Data source information
Group status: Verified
Group members:
Counter  1, event 27: PM_DATA_FROM_RL2L3_MOD  : Data loaded from remote L2 or L3
modified
Counter  2, event 25: PM_DATA_FROM_DMEM  : Data loaded from distant memory
Counter  3, event 24: PM_DATA_FROM_RMEM  : Data loaded from remote memory
Counter  4, event 31: PM_DATA_FROM_LMEM  : Data loaded from local memory
Counter  5, event  0: PM_RUN_INST_CMPL  : Run_Instructions
Counter  6, event  0: PM_RUN_CYC  : Run_cycles
```

Thus you can use the **hpmstat** command to monitor memory affinity status, as in Example A-25. In the example, we monitored the PMU event group #108, which contains the memory affinity metrics for 20 seconds. The memory locality value is 0.667, which means 66.7% of memory access is local, which indicates good memory affinity. We can try the RSET command and **vmo** options to get even better memory affinity.

Example A-25 Memory affinity report

```
# hpmstat -r -g 108 20
 Execution time (wall clock time): 20.009228351 seconds

 Group: 108
 Counting mode: user+kernel+hypervisor+runlatch
 Counting duration: 1040.320869363 seconds
  PM_DATA_FROM_RL2L3_MOD                                  :           66621
    (Data loaded from remote L2 or L3 modified)
  PM_DATA_FROM_DMEM (Data loaded from distant memory)     :               0
  PM_DATA_FROM_RMEM (Data loaded from remote memory)      :           60922
  PM_DATA_FROM_LMEM (Data loaded from local memory)       :          122057
  PM_RUN_INST_CMPL (Run_Instructions)                     :      7322552457
  PM_RUN_CYC (Run_cycles)                                 :     78132452650

 Normalization base: time

 Counting mode: user+kernel+hypervisor+runlatch

  Derived metric group: Memory
```

```
  [   ] Memory locality                                    :           0.667
  [   ] Memory load traffic                                :        2859.047
MBytes
  [   ] Memory load bandwidth per processor                :           2.748
MBytes/s
  [   ] Number of loads from local memory per loads from remote memory:
2.003
  [   ] Number of loads from local memory per loads from remote and distant
memory:           2.003

  Derived metric group: dL1_Reloads_percentage_per_inst

  [   ] % of DL1 Reloads from Remote L2 or L3 (Modified) per Inst:           0.001
%
  [   ] % of DL1 Reloads from Local Memory per Inst        :           0.002 %
  [   ] % of DL1 Reloads from Remote Memory per Inst       :           0.001 %
  [   ] % of DL1 Reloads from Distant Memory per Inst      :           0.000 %

  Derived metric group: General

  [   ] Run cycles per run instruction                     :          10.670
  [   ] MIPS                                               :           7.039
MIPS

 u=Unverified c=Caveat R=Redefined m=Interleaved
```

B

New commands and new commands flags

In this appendix, we introduce new commands that can be useful during performance tuning of your POWER Systems environment.

The following topics are illustrated in this appendix:

- amepat
- lsconf

amepat

The **amepat** tool now has the additional flag of **-0**. This flag enables the tool to provide a report on different processor types. This flag is available in AIX 7.1 TL2 and above and AIX 6.1 TL8 and above.

The possible options are as follows:

- **-0 proc=P7 -** This reports on software compression with POWER7 hardware.
- **-0 proc=P7+ -** This reports on hardware compression with POWER7+ hardware.
- **-0 proc=ALL** - This reports on both processor types.

Example B-1 demonstrates using the **amepat** command with the **-0** flag to provide a report on POWER7+ hardware with the compression accelerator.

Example B-1 Using new amepat option

```
root@aix1:/ # amepat -O proc=P7+ 5

Command Invoked                : amepat -O proc=P7+ 5

Date/Time of invocation        : Tue Oct  9 07:48:28 CDT 2012
Total Monitored time           : 7 mins 21 secs
Total Samples Collected        : 3

System Configuration:
---------------------
Partition Name                 : aix1
Processor Implementation Mode  : POWER7 Mode
Number Of Logical CPUs         : 16
Processor Entitled Capacity    : 2.00
Processor Max. Capacity        : 4.00
True Memory                    : 8.00 GB
SMT Threads                    : 4
Shared Processor Mode          : Enabled-Uncapped
Active Memory Sharing          : Disabled
Active Memory Expansion        : Enabled
Target Expanded Memory Size    : 8.00 GB
Target Memory Expansion factor : 1.00


System Resource Statistics:               Average             Min             Max
---------------------------             -----------       -----------       -----------
CPU Util (Phys. Processors)             1.41 [ 35%]       1.38 [ 35%]       1.46 [ 36%]
Virtual Memory Size (MB)                5665 [ 69%]       5665 [ 69%]       5665 [ 69%]
True Memory In-Use (MB)                 5881 [ 72%]       5881 [ 72%]       5881 [ 72%]
Pinned Memory (MB)                      1105 [ 13%]       1105 [ 13%]       1106 [ 14%]
File Cache Size (MB)                     199 [  2%]        199 [  2%]        199 [  2%]
Available Memory (MB)                   2302 [ 28%]       2302 [ 28%]       2303 [ 28%]

AME Statistics:                           Average             Min             Max
---------------                         -----------       -----------       -----------
AME CPU Usage (Phy. Proc Units)         0.00 [  0%]       0.00 [  0%]       0.00 [  0%]
Compressed Memory (MB)                     0 [  0%]          0 [  0%]          0 [  0%]
Compression Ratio                         N/A
```

```
Active Memory Expansion Modeled Statistics        :
-----------------------------------------
Modeled Implementation   : POWER7+
Modeled Expanded Memory Size :   8.00 GB
Achievable Compression ratio   :0.00

Expansion    Modeled True      Modeled             CPU Usage
Factor       Memory Size       Memory Gain         Estimate
---------    -------------     ------------------  -----------
     1.00          8.00 GB         0.00 KB [  0%]  0.00 [  0%]
     2.47          3.25 GB         4.75 GB [146%]  1.13 [ 28%]
     4.00          2.00 GB         6.00 GB [300%]  1.13 [ 28%]
     5.34          1.50 GB         6.50 GB [433%]  1.13 [ 28%]
     6.40          1.25 GB         6.75 GB [540%]  1.13 [ 28%]
     8.00          1.00 GB         7.00 GB [700%]  1.13 [ 28%]

Active Memory Expansion Recommendation:
---------------------------------------
The recommended AME configuration for this workload is to configure the LPAR
with a memory size of 8.00 GB and to configure a memory expansion factor
of 1.00.  This will result in a memory gain of  0%. With this
configuration, the estimated CPU usage due to AME is approximately 0.00
physical processors, and the estimated overall peak CPU resource required for
the LPAR is 1.46 physical processors.

NOTE: amepat's recommendations are based on the workload's utilization level
during the monitored period. If there is a change in the workload's utilization
level or a change in workload itself, amepat should be run again.

The modeled Active Memory Expansion CPU usage reported by amepat is just an
estimate.  The actual CPU usage used for Active Memory Expansion may be lower
or higher depending on the workload.
root@aix1:/ #
```

lsconf

The **lsconf** command now specifically reports if your hosting hardware has the new NX accelerator as found on the POWER7+ processor. Example B-2 shows the output when run on an older POWER7 machine.

Example B-2 Enhanced lsconf output

```
# lsconf
System Model: IBM,8233-E8B
Machine Serial Number: 106011P
Processor Type: PowerPC_POWER7
Processor Implementation Mode: POWER 7
Processor Version: PV_7_Compat
Number Of Processors: 4
Processor Clock Speed: 3300 MHz
CPU Type: 64-bit
Kernel Type: 64-bit
LPAR Info: 15 750_1_AIX6
Memory Size: 8192 MB
Good Memory Size: 8192 MB
Platform Firmware level: AL730_095
```

```
Firmware Version: IBM,AL730_095
Console Login: enable
Auto Restart: true
Full Core: false
NX Crypto Acceleration: Not Capable
```

Example B-2 on page 339 provides a way to verify hardware support for the NX accelerator without having access to the HMC.

C

Workloads

This appendix gives an overview of the workloads used throughout this book for the various scenarios. Some workloads were created using simple tool and utilities, some developed purposely by the team and others are based on real products.

The following topics are discussed in this appendix:

- IBM WebSphere Message Broker
- Oracle SwingBench
- Self-developed C/C++ application
- 1TB segment aliasing demo program illustration
- “latency” test for RSET, ASO and DSO demo program illustration

IBM WebSphere Message Broker

WebSphere Message Broker is an information integrator that allows interaction and data exchange between applications regardless of the message formats or protocols that they support.

A number of samples ship with the Message Broker product. We used one of these to generate repeatable CPU and network load. The sample we used was the Web Service SOAP Nodes sample. The Simple Object Access Protocol (SOAP) is a lightweight, XLM-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used to query and return information and involve service across the Internet.

This sample demonstrates the use of a number of SOAP nodes to provide and support a simple Web service.

There were a few characteristics which made this sample workload appealing:

- Ease and speed of product installation and configuration
- Ability to set the duration for a workload
- Variation of message payload: 2k, 20k and 200k were available options. This allowed us to vary workload footprint where required
- Client/server workload. We could either have both halves of the workload within an LPAR (communicating over the loopback), or span LPARs (and CECs) where required
- Scalability. Ability to scale the workload across 1 or more threads
- TPS-style statistics, both during and a summary on completion
- Reproducible workload.

We actively used this workload to facilitate our tests in a number of areas, including ASO, AME, WPARs and generating network sustained network traffic between our two physical machines.

For more information on WebSphere Message Broker, please refer to the product website:

http://www-01.ibm.com/software/integration/wbimessagebroker/

Please refer to the Message Broker information center for a more detailed explanation of the sample application:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v8r0m0/index.jsp?topic=%2Fcom.ibm.etools.mft.samples.SOAPNodes.doc%2Fdoc%2Foverview.htm

Oracle SwingBench

Swingbench is free load generator for Oracle database 10g and 11g. Based on a Java framework, it can work on wide variety of platforms.

Swingbench is usually used for demonstration and test. It offers several type of loads:

- OrderEntry: A typical OLTP load with some select, update, insert (60% read, 40% write).
- SalesHistory: This test is composed of complex queries against large table (100% read).
- CallingCircle: To simulate a typical online telco application.
- StressTest: Create some random inserts, updates, select against a table.

The installation of swingbench is easy. You need to install an Oracle database (10g or 11g) and create an empty database (with the **dbca** command). You can after use the script,

provided with swingbench to generate tables and load the data (oewizzard for OrderEntry, shwizzard for SalesHistory and ccwizzard for CallingCircle). When everything is done, you can start the swingbench binary (Java 6 needed for Swingbench 2.4).

For AME and LPAR placement test, we have used OrderEntry scenario with 200 concurrent users (Figure C-1).

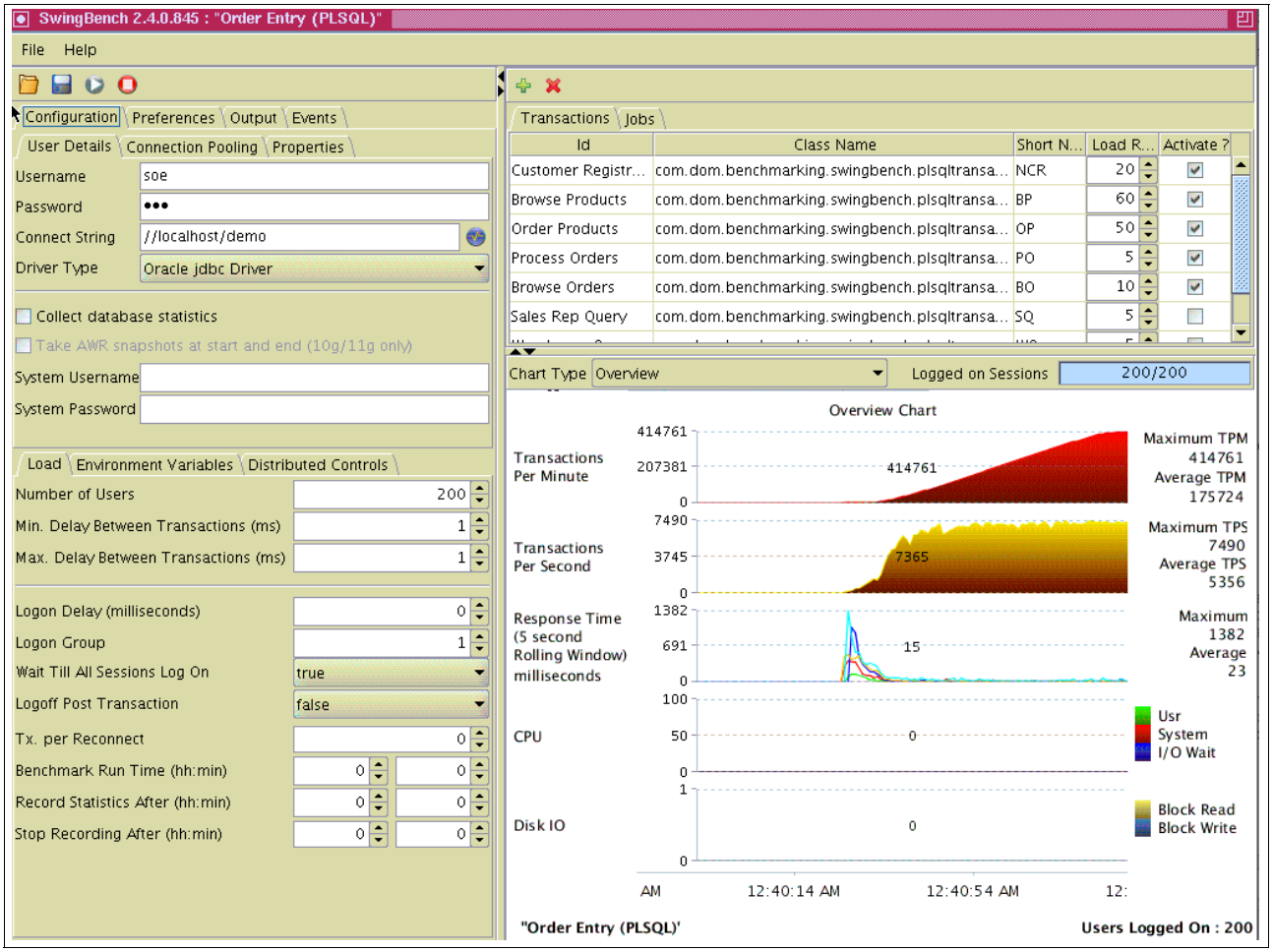


Figure C-1 Swinbench 2.4 graphical interface

Please refer to the following website to download or have information about the product:

http://www.dominicgiles.com/swingbench.html

Self-developed C/C++ application

In this section, we introduce the C/C++ sample applications used for the demonstration of Power System, PowerVM, and AIX features in this book.

1TB segment aliasing demo program illustration

Example C-1 on page 344 shows the testing steps. Since LSA is good for 64 bit applications with large memory footprint and low spatial locality, the sample is intentionally designed in such a way. We can see 30% performance gain on average when LSA is enabled in this test.

Example C-1 sample test procedures

```
# ./lsatest
usage: ./lsatest <length_in_MB> <step_in_Bytes> <iteration>

Specify a large memory footprint, and specify a large step size, then the spatial
locality will not be good. As below, we specify 16384MB memory footprint, and
20,000,000 Byte step size.

Scenario I, LSA disabled:
#vmo -o esid_allocator=0
#./lsatest 16384 20000000 0

Scenario II, LSA enabled:
#vmo -o esid_allocator=1
#./lsatest 16384 20000000 0
```

Example C-2 shows the sample program. The sample program creates a piece of shared memory with specified size, and then traverses the shared memory using the step size specified. It calculates the average latency of memory access at the end of each round of load test.

Example C-2 lsatest sample program

```
#cat lsatest.cpp
/*
 * The following [enclosed] code is sample code created by IBM
 * Corporation.  This sample code is not part of any standard IBM product
 * and is provided to you solely for the purpose of demonstration.
 * The code is provided 'AS IS',
 * without warranty of any kind.  IBM shall not be liable for any damages
 * arising out of your use of the sample code, even if they have been
 * advised of the possibility of such damages.
 */
/*
Problem report: chenchih@cn.ibm.com
To compile(64bit is a must): xlC -q64 lsatest.cpp -o lsatest
*/

#include <time.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/shm.h>

#define REPEATTIMES 8
#define ITERATIONS       1048576
#define DEBUG printf

long delta(timeval * start, timeval* end)
{
    long dlt;
    dlt = (end->tv_sec - start->tv_sec)*1000000 + (end->tv_usec - start->tv_usec);
    return dlt;
}
```

```
int loadtest(char *addr, long length, long step)
{
    void **p = 0;
    timeval start, end;
    long totaltime, besttime;
    double latency;
    long i,j;

    if(step % sizeof(void*) != 0)
    {
        DEBUG("The step should be aligned on pointer boudry\n");
        return -1;
    }

    for (i = length; i >= step; i -= step)
    {
        p = (void **)&addr[i];
        *p = &addr[i - step];
    }

    p = (void **)&addr[i];
    *p = &addr[length]; /*rewind*/
    besttime = ~0UL >> 1;


    for (i = 0; i < REPEATTIMES; i++) {
#define ONE     p = (void **)*p;
#define FOUR    ONE ONE ONE ONE
#define SIXTEEN FOUR FOUR FOUR FOUR
#define SIXTYFOUR  SIXTEEN SIXTEEN SIXTEEN SIXTEEN
#define QUARTER_ONE_KI  SIXTYFOUR SIXTYFOUR SIXTYFOUR SIXTYFOUR
#define ONE_KI QUARTER_ONE_KI QUARTER_ONE_KI QUARTER_ONE_KI QUARTER_ONE_KI

    j = ITERATIONS;
    gettimeofday(&start, NULL);
    while (j > 0)
    {
        ONE_KI
        j -= 1024;
    }
    gettimeofday(&end, NULL);

    totaltime = delta(&start, &end);
    if(totaltime < besttime)
        besttime = totaltime;
    }

    besttime*=1000;
    latency = (double)besttime/(double)ITERATIONS;
    printf("latency is %.5lf ns\n", latency);
    return 0;
}

int main(int argc, char* argv[])
```

```
{
    void *addr, *startaddr;
    int fd, myid;
    key_t key;
    int iteration, endless = 0;
    long len, step;
    char estr[256];

    if(argc < 4)
    {
        printf("usage: %s <length_in_MB> <step_in_Bytes> <iteration>\n", argv[0]);
        return -1;
    }

    len = atol(argv[1]);
    len = len * 1024 * 1024;
    step = atol(argv[2]);
    iteration = atoi(argv[3]);

    if(iteration == 0)
        endless = 1;

    fd = open("/tmp/mytest", O_CREAT|O_RDWR);
    close(fd);
    key = ftok("/tmp/mytest", 0x57);
    if(key != -1)
        printf("key = %x\n", key);
    else
    {
        perror("ftok");
        return -1;
    }
    myid = shmget(key, len, IPC_CREAT|0644);
    startaddr = (void*)0x070000000000000011;
    addr = shmat(myid, startaddr, 0);
    printf("Allocated %ld bytes at location 0x%p\n", len, addr);

    if (addr == NULL)
    {
        sprintf(estr,"shmat Failed for size %i\n",len);
        perror(estr);
        return 1;
    }
    while(endless || (iteration > 0))
    {
        loadtest((char*)addr, len - 1024, step);
        iteration--;
    }
}
```

Note: This sample creates a shared memory region for test. The shared memory key starts with “0x57”. You can use **ipcs -mab** to display the shared memory region, and **ipcrm** to delete it after you are done with the test.

“latency” test for RSET, ASO and DSO demo program illustration

As one of the major benefits we can get from RSET and ASO is cache affinity and memory affinity, we create a sample in such a way.

Example C-3 gives the sample code we used for the RSET and ASO demonstration. It is a memory intensive application, which creates a piece of heap memory with size specified, and multiple threads traverse the same heap memory in the step size specified.

In Example C-3, when one thread complete one round of the memory load test, we consider it as one transaction finished. In each round of the memory load test, there is 16777216 times of memory loads, as in the sample code (REPEATTIMES*ITERATIONS). After the time specified, the program exits and logs the overall throughput in “transaction.log”. The program also logs the average memory load latency during the memory load tests.

Example C-3 Memory latency test sample code

```
/*
 * The following [enclosed] code is sample code created by IBM
 * Corporation.  This sample code is not part of any standard IBM product
 * and is provided to you solely for the purpose of demonstration.
 * The code is provided 'AS IS',
 * without warranty of any kind.  IBM shall not be liable for any damages
 * arising out of your use of the sample code, even if they have been
 * advised of the possibility of such damages.
 */
/*
Problem report: chenchih@cn.ibm.com
To compile(64bit is a must): xlC_r -q64 latency.cpp -o latency
*/
#include <pthread.h>
#include <time.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/time.h>
#include <unistd.h>
#include <sys/shm.h>
#include <stdio.h>
#include <string.h>

#define REPEATTIMES 1
#define ITERATIONS      16777216
#define DEBUG printf
#define MAX_NUMBER_OF_THREADS   256

long long g_threadcounter[MAX_NUMBER_OF_THREADS];

long delta(timeval * start, timeval* end)
{
    long dlt;
    dlt = (end->tv_sec - start->tv_sec)*1000000 + (end->tv_usec - start->tv_usec);
    return dlt;
}

int initialize(char* addr, long length, long step)
{
```

```
    void **p = 0;
    long i,j;

    if(step % sizeof(void*) != 0)
    {
        DEBUG("step should be aligned on pointer boudry\n");
        return -1;
    }

    for (i = length; i >= step; i -= step)
    {
        p = (void **)&addr[i];
        *p = &addr[i - step];
    }

    p = (void **)&addr[i];
    *p = &addr[length]; /*rewind*/
    return 0;
}

double loadtest(char *addr, long length, long step)
{
    void **p = 0;
    timeval start, end;
    long long totaltime, besttime;
    double latency;
    long i,j;
    p =(void**) &addr[length]; //start point.
    besttime = ~0UL >> 1;


    for (i = 0; i < REPEATTIMES; i++)
    {
#define ONE     p = (void **)*p;
#define FOUR    ONE ONE ONE ONE
#define SIXTEEN FOUR FOUR FOUR FOUR
#define SIXTYFOUR  SIXTEEN SIXTEEN SIXTEEN SIXTEEN
#define QUARTER_ONE_KI  SIXTYFOUR SIXTYFOUR SIXTYFOUR SIXTYFOUR
#define ONE_KI QUARTER_ONE_KI QUARTER_ONE_KI QUARTER_ONE_KI QUARTER_ONE_KI

    j = ITERATIONS;
    gettimeofday(&start, NULL);
    while (j > 0)
    {
        ONE_KI
        j -= 1024;
    }
    gettimeofday(&end, NULL);

    totaltime = delta(&start, &end);
    if(totaltime < besttime)
        besttime = totaltime;
    }

    besttime*=1000;
```

```
    latency = (double)besttime/(double)ITERATIONS;
    return latency;
}

struct threadarg
{
    void *ptr;
    long len;
    long step;
    long thid;
};


extern "C" void* testfunc(void* arg)
{
    char *addr;
    long len, step, thid;
    double latency;
    struct threadarg *tharg;
    tharg = (struct threadarg *)arg;

    addr = (char*) tharg->ptr;
    len = tharg->len;
    step = tharg->step;
    thid = tharg->thid;

    while(1)
    {
        latency = loadtest(addr, len - 1024, step);
        if(g_threadcounter[thid] % 8 == 0)
            printf("in thread %d, latency is %.5lf ns\n", thread_self(), latency);
        g_threadcounter[thid]++;
    }
    return NULL;
}

int main(int argc, char* argv[])
{
    void *addr, *startaddr;
    int threadnum = 0;
    int duration;
    long len, step, total;
    pthread_t tid = 0, *tlist;
    pthread_attr_t attr;
    char estr[256];
    struct threadarg *parg;
    char* ptr;
    timeval current, end, start;
    int ret;

    if(argc < 5)
    {
      printf("usage: %s <length_in_MB> <step_in_Bytes> <thread number>
<duration_in_seconds>\n", argv[0]);
      return -1;
```

```
}

memset(g_threadcounter, 0, sizeof(g_threadcounter));

len = atol(argv[1]);
len = len * 1024 * 1024;
step = atol(argv[2]);
threadnum = atoi(argv[3]);
duration = atoi(argv[4]);

total = len;
addr = malloc(total);
if (addr == NULL)
{
  sprintf(estr,"malloc failed for size %i\n",len);
  perror(estr);
  return 1;
}
ptr = (char*)addr;

initialize(ptr, total - 1024, step);

tlist = new pthread_t[threadnum];
pthread_attr_init(&attr);

for(int i =0; i < threadnum; i++)
{
    parg = new threadarg;
    parg->ptr = addr;
    parg->len = total;
    parg->step  = step;
    parg->thid = i;
    ret = pthread_create(&tid, &attr, testfunc, parg);
    if(ret != 0)
    {
        printf("pthread_create error, err=%d\n", ret);
        return -1;
    }
    tlist[i] = tid;
}

gettimeofday(&current, NULL);

end.tv_sec = current.tv_sec + duration;
end.tv_usec = current.tv_usec;

long long mycounter = 0, savedcounter = 0, savedsec;
int fd;
char outdata[1024];
fd = open("transaction.log", O_RDWR|O_CREAT|O_APPEND);
savedsec = current.tv_sec;


while(1)
{
```

```
        sleep(60);

        mycounter=0;
        gettimeofday(&current, NULL);
        for(int i = 0; i < threadnum; i++)
          mycounter+=g_threadcounter[i];

        if(current.tv_sec >= end.tv_sec)
        {
            sprintf(outdata, "The total throughput is %lld. \n", mycounter);
            write(fd, outdata, strlen(outdata));
            break;
        }
        else
        {
            sprintf(outdata, "The current TPS is %.2lf\n", (double)(mycounter -
savedcounter)/(double)(current.tv_sec - savedsec));
            write(fd, outdata, strlen(outdata));
            savedcounter = mycounter;
            savedsec = current.tv_sec;
        }
    }

    close(fd);
    /*for(int i=0; i < threadnum; i++)  {
        void* result;
        pthread_join(tlist[i], &result);
    }*/

    return 0;
}
```

The test steps are as shown in Example C-4. We start two latency instances in different directories, each allocates 16384MB heap memory, and creates 30 threads to traverse the heap memory with step size equal to 1024 bytes, and runs for 7200 seconds. To simplify, we put all the parameters in a script named proc1 and proc2.

Example C-4 Memory latency test steps

```
# ./latency
usage: ./latency <length_in_MB> <step_in_Bytes> <thread number> <duration_in_seconds>
#cat proc1
./latency 16384 1024 30 7200
#cat proc2
./latency 16384 1024 30 7200
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *IBM PowerVM Getting Started Guide,* REDP-4815-00
- *Virtualization and Clustering Best Practices Using IBM System p Servers,* SG24-7349
- *IBM PowerVM Virtualization Active Memory Sharing,* REDP-4470
- *IBM PowerVM Virtualization Introduction and Configuration,* SG24-7940-04
- *IBM PowerVM Virtualization Managing and Monitoring,* SG24-7590-03
- *IBM PowerVM Best Practises,* SG24-8062
- *Exploiting IBM AIX Workload Partitions,* SG24-7955
- *IBM PowerVM Introduction and Configuration,* SG24-7940
- POWER7 and POWER7+ Optimization and Tuning Guide, SG24-8079
- *AIX 7.1 Difference Guide,* SG24-7910

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- IBM Power Systems

  http://www.ibm.com/systems/power/advantages/index_midsize.html
- IBM hardware information center

  http://http://pic.dhe.ibm.com/infocenter/powersys/v3r1m5/index.jsp

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Power Systems Performance Guide: Implementing and Optimizing

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

IBM Power Systems Performance Guide
Implementing and Optimizing

Leverages IBM Power virtualization

Helps maximize system resources

Provides sample scenarios

This IBM Redbooks publication addresses performance tuning topics to help leverage the virtualization strengths of the POWER platform to solve clients' system resource utilization challenges, and maximize system throughput and capacity. We examine the performance monitoring tools, utilities, documentation, and other resources available to help technical teams provide optimized business solutions and support for applications running on IBM POWER systems' virtualized environments.

The book offers application performance examples deployed on IBM Power Systems utilizing performance monitoring tools to leverage the comprehensive set of POWER virtualization features: Logical Partitions (LPARs), micro-partitioning, active memory sharing, workload partitions, and more. We provide a well-defined and documented performance tuning model in a POWER system virtualized environment to help you plan a foundation for scaling, capacity, and optimization.

This book targets technical professionals (technical consultants, technical support staff, IT Architects, and IT Specialists) responsible for providing solutions and support on IBM POWER systems, including performance tuning.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-8080-00 ISBN 0738437662