HATS Tuning Guide
December 2011

IBM
Host Access Transformation Services (HATS) on Multi-platforms Tuning Guide

Document version 2.0

IBM Host Access Transformation Services Performance Team

Notices

Table of Contents

1.0 Introduction

1.1 About this document

The purpose of this document is to provide performance tuning advice and information for IBM Host Access Transformation Services (HATS).

Performance is affected by many factors, including environment, hardware, software, user applications, and workloads. The information in this document was obtained during IBM benchmarking performance tests. These benchmarking performance tests were conducted in specific controlled environments. Stress environments were included.

Some parameter settings might only be necessary in stress or memory-constrained environments.

Warning: Do not arbitrarily modify your server settings to match the settings in this document. Monitor your system in your environment with your applications and make the appropriate settings for your environment.

1.2 Performance Tuning

Performance tuning is the process of selecting configurable parameter values to maximize the capacity and performance of your HATS applications. This document gives you an idea of what can be modified to improve the performance. These are parameters that laboratory measurements have shown are the ones most likely to provide the most performance benefit when tuned appropriately. The recommendations provided herein are general guidelines only. These recommendations might not yield appreciable performance improvements in your environment. In general, you should consult the documentation that comes with your particular hardware and software for performance tuning advice first, and use the information in this document as a supplement for HATS applications.

For the optimal performance of your HATS applications, evaluate the performance of your system configuration in your operating environment with your workloads. Conduct a controlled experiment in which a well-defined workload is applied against a system configuration that does not vary except for the parameter of interest. That is, change only one parameter at a time and analyze how its change affects your performance. These parameter settings might not be appropriate for your environment. The values should not be taken as recommendations for all environments. The best values for your environment and applications might vary.

1.3 Performance Concepts

For HATS applications, system performance is usually described as end-user response times for user requests/transactions and HATS server (WebSphere Application Server) transaction throughput. When tuning for performance, the goal is to maximize server throughput while still achieving acceptable end-user response times.

The following definitions are applicable to discussion in this document:

- **Transaction**

  A unit of work. Typically this is a single, indivisible operation issued by a person or device. For a HATS Web application, a "transaction" starts with a user click in the Web browser and ends with the display of an HTML page.

- **Transaction throughput**

  The number of "transactions per unit of time" the server can handle before bad things start to happen such as connections being refused, the system crashes under load, excessive network errors are experienced, or the average response time exceeds an acceptable value.

  Throughput is the most important measure of the provider of a "server". It is a critical measure of the capacity of an application server.

- **End-User Response Time**

  The time starting from when the client enters a transaction request until the client gets the transaction response. It is important because it represents the quality of service users will see.

  Response time is the single most important measure of user satisfaction.

1.4 How is this document organized?

Although it is assumed that the reader of this document will read the document in its entirety, this section gives an overview of the document's organization. This information should help the reader for follow-up references.

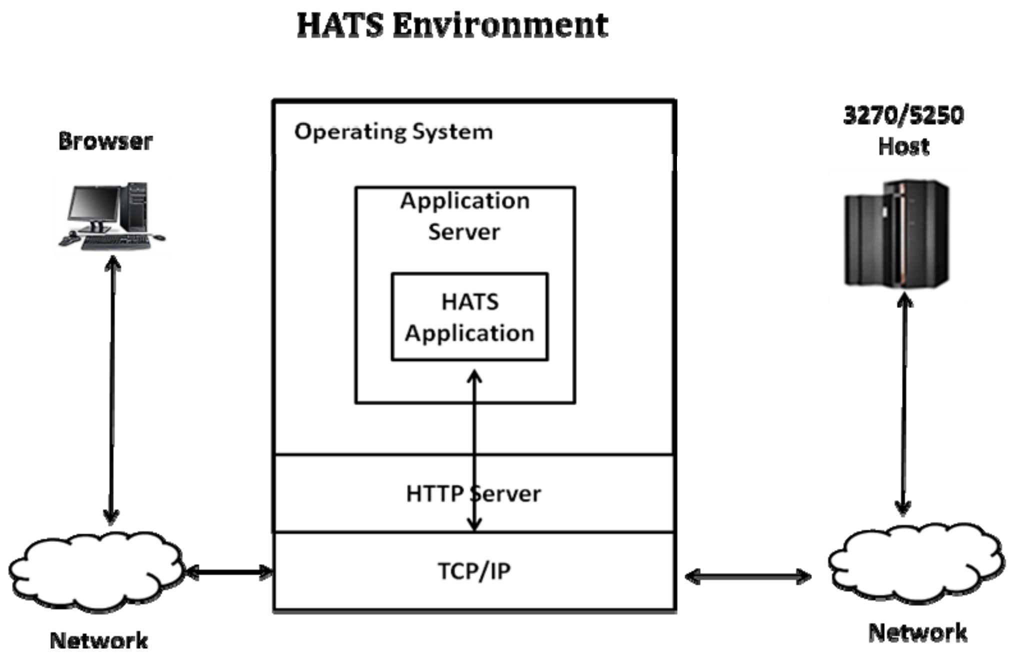
This document is organized as follows:

- Chapter 1: Explains the purpose of the document; provides a definition of performance tuning; and defines the organization of the document.
- Chapter 2: Provides an overview of the HATS environment; suggests a methodology for approaching performance tuning.
- Chapter 3: Provides general tuning hints.
- Chapter 4: Provides advice for tuning your browser environment.
- Chapter 5: Provides advice for tuning your HATS applications.
- Chapter 6: Provides advice for tuning your operating systems environment.
- Chapter 7: Provides advice for tuning your TCP/IP network.
- Chapter 8: Provides advice for tuning your WebSphere Application Server.
- Chapter 9: Provides advice for tuning your HTTP Server.
- Chapter 10: Provides advice for monitoring your performance environment.
- Chapter 11: Provides advice for troubleshooting your performance environment.

2.0 Tuning the performance of the HATS environment

2.1 Understanding the HATS environment

HATS applications are installed on a device running WebSphere Application Server (WAS). Users enter a screen request or initiate a transaction by entering a URL in a browser. The request travels over a TCP/IP network from the client machine to the HATS application, which is a standard WebSphere application.



The diagram shows a typical HATS system environment.

Given the environment above, performance tuning advice is provided for the following:

- Browser
- HATS Applications
- Operating System
- TCP/IP Network
- WebSphere Application Server
- HTTP Server

2.2 HATS Requirements

HATS applications are installed on the WebSphere Application Server. Therefore, HATS inherits the hardware and software requirements of the WebSphere Application Server. Refer to WebSphere Application Server documentation for lists of supported hardware and software. Ensure you are compliant with the supported options for your version and level of the WebSphere Application Server.

HATS applications usually put higher demands on virtual memory compared to traditional WebSphere enterprise applications. For HATS installations, add the following to WebSphere Application server memory requirements:

- Small installations: Minimum of 1 GB memory.
- Large installations: Minimum of 2 GB memory.

In addition to disk storage required by the WebSphere Application server, you should allocate the following for HATS:

- 40 MB of disk space per HATS application; more might be needed if you customize your pages with graphics or other such components.
- 100 MB of disk space for HATS configuration files, logs, and other operational files.

3.0 General tuning tips

Some general tuning tips are:

- Make sure you have adequate hardware.
- Make sure your server has sufficient physical memory and hard drive space. Changing tuning parameters almost always affects memory allocation and/or storage allocation.
- Configure your network interface card for sufficient transmit and receive buffers.
- Ensure that the latest HATS cumulative fix pack is installed. We are constantly looking for ways to improve HATS capacity and performance.
- Tune your TCP/IP network to avoid fragmentation of packets. Choose a packet size that is the largest value the route between your server and the TCP/IP host can handle without fragmentation.
- Make sure your operating systems and application servers are at the latest supported levels. Generally, each new release brings improved performance.
- Cache static data as close as possible to the end user.

3.1 Server Hardware

To get the best performance out of your HATS applications, make sure that you have adequate hardware. This section describes some hardware basics that helps you prepare your server for software tuning as software tuning can only improve performance to hardware capabilities.

There are three critical hardware resources that affect performance:

- Central Processing Unit (CPU)
- Memory
- Network Interface Subsystem

Central Processing Unit

The central processing unit is the hardware resource that has the largest impact on performance.

Consider the following suggestions when selecting application server hardware:
- Use faster, multiple processor machines.
- Use server-level machines.
- Use innovative technologies, such as hyperthreading.
- Move work off the main CPU when possible.

HATS applications are Java based. Java performs best on faster, multiple processor machines.

Modern processors designed for servers are recommended. These processors operate at java-compatible speeds and usually have built-in technologies that support greater performance such as processor cache. Processor cache is memory that is incorporated into the main system processor and can be accessed faster than standard memory. In fact, most modern, designed-for-server processors contain three levels of cache – Level 1 (L1), Level 2 (L2) and Level 3 (L3). These caches act as temporary storage spaces for instructions and data obtained from slower memory or DASD.

Select servers that incorporate innovative technologies that enable increased performance such as hyperthreading and multi-core processors. Hyperthreading technology takes advantage of virtual processors. Multi-core technology support multiple processor execution cores in a single processor package.

Use other system resources to perform specialized operations, such as a System z Application Assist Processor(zAAP) processor or cryptographic processor.

3.2 Caching static content

HATS applications generate lots of network traffic. To improve network performance, it is best to cache as much of your static content as possible as close to your end users as possible.

Here are some caching mechanisms you might choose from:

- Proxy Server caching
- Dynamic caching by the HTTP Server

Proxy server caching

A proxy server, such as IBM WebSphere Edge Server, provides capabilities to cache static Web content and serves requests for this content from its cache. This type of caching eases the workload of the WebSphere Application Server, reduces network latency, and improves response time to the browser.

Dynamic caching by the HTTP server

The IBM HTTP Server can be configured to dynamically identify static content, store the static content in a cache and serve all requests for static content from its cache. The HTTP Server can serve static content quicker than the WebSphere Application Server and avoid some TCP hops.

3.3 SSL

Secure Sockets Layer (SSL) data encryption is a heavy user of CPU cycles. If you use SSL encryption to secure data on your TCP/IP connections, consider using hardware encryption devices. Using encryption hardware improves capacity and performance in a secure environment.

WebSphere for System z™ uses System SSL, which takes advantage of z990 cryptographic hardware. If no crypto hardware is installed, System SSL uses general purpose CPU cycles.

Use SSL only when necessary.

3.4 Application start up time

The first time an application is requested, the JSP must be compiled. During the installation of a HATS application, you can request that it be compiled when it is started. Compiling the application when it is started improves the response time of the first request.

4.0 Browser Tuning

4.1 Browser caching

Browsers generally contain a cache which can hold all static content. For the best response Time enable browser caching.

4.2 Browser Cache Size

Make sure the browser cache is large enough to hold your application's static content: graphics, stylesheets, templates, and JavaScript files.

5.0 HATS application tuning

When building your HATS applications, there are some configuration changes that you can make to achieve improved performance.

5.1 Contention resolution

3270 screen settling delays

For 3270E HATS applications, HATS supports two screen settling algorithms: Timing and Fast 3270e. These algorithms are used by HATS to determine when a requested screen has been fully received from the 3270 host application.

Timing

The timing algorithm utilizes two delay parameters: delayStart and delayInterval.

delayStart: Specifies the time (in milliseconds) that the server waits until the first full host screen has arrived.   Recommendation: Tune this value according to the responsiveness of your host and network.

delayInterval: Specifies the maximum time (in milliseconds) that the server waits until a full host screen, which is not the first host screen, has arrived. Recommendation: Tune this value according to the responsiveness of your host and network.

If you need to change the delayStart and delayInterval parameters after they have been deployed to an application server, modify the application main.hco file.

Fast 3270e

The second algorithm implements contention resolution as defined in RFC 2355 for TN3270E connections. In the implementation of contention resolution, HATS is told in the protocol exactly when the full screen has been delivered. Therefore, HATS does not need to use the delayInterval parameter.

For more information about HATS screen settling, please see the “HATS screen-settling reference” appendix in the HATS User’s and Administrator’s Guide.

5.2 Asynchronous update applet

HATS provides the capability for asynchronous outbound data to be updated at the client browser by means of either a client pull method using an AJAX implementation or a server push method using an applet implementation. The Server push (applet) method is now deprecated. If you enable asynchronous update, you are advised to use the Client pull (AJAX) method instead. If your applications do not generate screen updates asynchronously, and you have no problems with partial screen updates, you do not need to enable this feature.

The “asynchronous update” or “refresh” function is configurable in the HATS Toolkit (Project Settings→Other→ Automatic Disconnect and Refresh). To enable the AJAX implementation, select the Enable client pull (AJAX) option. To enable the applet implementation, select the Enable server push (applet) option. To disable both of these functions, select Disabled. The default is Disabled.

For further information on settings for the Client pull (AJAX) method, please reference the HATS Information Center, or the RSDP online help.

5.3 HTML response optimizations

The Web page (HTTP Response) is delivered in HTML. The HTML forms the framework whereby text, images, and objects are displayed in the browser. Use the following tips to optimize the size of your HTML content:

- Do not use sample templates. They were built to show you what you can do, not for optimal performance.
- Optimize your busiest pages. Remove unnecessary elements. If you can remove a design element from your page, do it.
- Simplify complex tables because they display slower. Break tables up into separate tables.
- Minimize the size and number of images in your page. Each image requires a separate HTTP request.
- Crop images; use lower resolutions.
- Use WIDTH and HEIGHT attributes with all images, tables, and applets.
- Remove backgrounds; substitute page/element background colors, table cells, and so forth.
- Combine adjacent graphics.
- Set page size limits. The page size is defined as the sum of the file sizes for all the elements that make up a page, including the defining HTML file as well as all embedded objects (image files, javascript, and so on.)

5.4 Macro tuning

Pausetime

The macro runtime implements a screen settling delay for macros called “pausetime”. Pausetime defaults to 300 ms. This means that there is a 300 ms delay in the macro processing before each screen is processed. The setting for this parameter should depend upon the responsiveness of your network and backend application. If your network and backend application provide good response times, you might be able to significantly reduce the default pausetime value.

If you use Contention Resolution, there are related parameters that you should consider changing:

- Ignorepauseforenhancedtn
- Ignorepausetimeforenhancedtn
- Delayifnotenhancedtn

Please refer to the HATS Advanced Macro Guide for more details on these parameters.

MaxHODEventThreads

The HATS runtime uses a pool of event threads to handle macro processing.   A thread is removed from the pool to process a macro and returned to the pool when the macro processing is complete.   The default and minimum number of event threads is 10.

The maxHODEventThreads parameter is a setting in the runtime.properties or the runtime-debug.properties file. It is not set as a JVM flag.

5.5 Pooled connections

Connection setup to the backend application is an expensive operation. To help lessen the overhead of connection setup, HATS enables developers to establish a pool of connections per application that users can share. Connections in the pool are initialized when the application is started during application server startup. The number of connections initialized at application server startup is controlled by having a non-zero number for the minimum number of idle connections to retain in the pool. When a user accesses the backend application, an existing connection is selected from the pool to service the request. When the user finishes with the connection, it is returned to the pool for reuse.

Connection pooling can improve the response time of an application since there is an insignificant amount of overhead in obtaining a connection from the pool and releasing a connection for reuse.

Connection pooling in conjunction with a transformation connection is generally not recommended but can be used with care.   Ensure that the application navigates back to the check-in screen for the pool. Otherwise, a new connection is started and the old one cleaned up.

5.6 Screen recognition

Screen recognition is the process that the HATS runtime performs when it attempts to match a screen customization with the current application screen.   Here are a few tips that improve the performance of screen recognition:

1. Specify enough criteria to recognize the screen, but don’t overdo it.   If you have specified text recognition and you don’t need other criteria like cursor position, disable it.
2. When specifying text recognition, use exact locations when possible. It is more expensive to find text within an area than it is to find it at an exact location.

5.7 Identified Next Screen

The HATS screen matching algorithm might become inadequate when there are huge numbers of screen customizations. Application developers may be able to improve application performance by identifying the next screen in the application flow. This feature allows the application developer to identify, in a screen customization, the next screen in the application flow. This feature should provide improved performance for applications with large numbers of customized screens.

6.0 Operating systems tuning for HATS

Make sure your operating system is supported by your level of WebSphere Application Server. Also, make sure you have the latest service fixes for your level of operating system. Generally, each new release brings improved performance.

Through IBM's testing of the HATS product, we have learned that **tuning some operating system parameters is almost always necessary for achieving optimal performance** for HATS enterprise applications. We provide tuning advice on the following operating systems:

- Microsoft Windows Server 2003
- IBM System x™
- Novell SUSE 9
- Novell SUSE 10
- IBM AIX®
- IBM System p™
- Sun Microsystems Solaris 10
- IBM System z™
- IBM System i™

Users are not limited to these operating systems. HATS applications are supported on any operating system supported by WebSphere Application Server.

6.1 Windows Server 2003

The parameters in the following table were tested on Microsoft Windows Server 2003 operating system. They might not be applicable to other versions of Windows.

System	
Parameter	**Setting**
Processor Scheduling	Background Services
Memory Usage	Adjust for best performance of System Cache
Virtual memory	One contiguous, static, 4 GB Pagefile

6.2 SuSe on System x

6.2.1 Specific parameters

The parameters in the following table were tested on SuSe 10.

Kernel Tuning	
Parameter	**Setting**
ulimit –n	131072
ulimit –u	39936
ulimit –l	64
Fs.file-max	365859
kernel.shmmni	8000
kernel.shmmax	1.84467E+19
kernel.shmall	2511724800
Vm.nr_hugepages	575

6.2.2 Parameter definitions

ulimit –n - Sets the maximum number of open files.
ulimit –u - Sets the maximum number of users processes.
ulimit –l - Sets the maximum limit for locked memory.
Fs.file-max - Sets the maximum number of file-handles that the Linux kernel will allocate.
kernel.shmmni - Sets the shared memory identifier.
kernel.shmmax - Sets the maximum shared memory. Our recommended setting is the default in SuSe 10.
kernel.shmall - Sets the maximum total amount of memory. Our recommended setting is the default in SuSe 10.
Vm.nr_hugepages - Sets the maximum number of huge pages allocated.

6.3 AIX

6.3.1 Specific parameters

AIX Kernel Tuning	
Parameter	**Setting**
ulimit –n	131072
ulimit –u	39936
ulimit –l	64
Fs.file-max	365859
kernel.shmmni	8000
kernel.shmmax	1073741824
kernel.shmall	2097152

6.3.2 Parameter definitions

ulimit –n: Sets the maximum number of open files.
ulimit –u: Sets the maximum users processes.
ulimit –l: Sets the maximum locked memory.
Fs.file-max: Sets the maximum number of file-handles that the Linux kernel will allocate.
kernel.shmmni: Sets the shared memory identifier.
kernel.shmmax: Sets the maximum shared memory.
kernel.shmall: Sets the maximum total amount of memory.
Vm.nr_hugepages: Sets the maximum number of huge pages allocated.

6.4 Solaris

The parameters in this section were tested on Solaris 10 only.

6.4.1 Specific parameters

Solaris Kernel Tuning	
Parameter	**Setting**
maxphys	10448576
maxpid	65534
bufhwm	2048
pt_cnt	512
rlim_fd_max	65536
ulimit -n	4000
semsys:seminfo_semune	1024

6.4.2 Parameter definitions

maxphys: Specifies the maximum size of physical I/O requests. If a driver sees a request larger than this size, the driver breaks the request into `maxphys` size chunks.

maxpid: Specifies value of largest possible process ID.

bufhwm: Defines the maximum amount of memory for caching I/O buffers. The buffers are used for writing file system metadata. Buffers are allocated as needed until the amount of memory (in Kbytes) to be allocated exceed `bufhwm`. At this point, metadata is purged from the buffer cache until enough buffers are reclaimed to satisfy the request.

pt_cnt: One of three variables that determine the number of users who can remotely log in to the system. If `pt_cnt` is zero, a default maximum number of remote logins is determined by a percentage of the amount of available physical memory. If `pt_cnt` is non-zero, the system allocates to the greater of `pt_cnt` and the default maximum.

rlim_fd_max: **S**pecifies the “hard” limit on file descriptors that a single process might have open. Overriding this limit requires super user privilege.

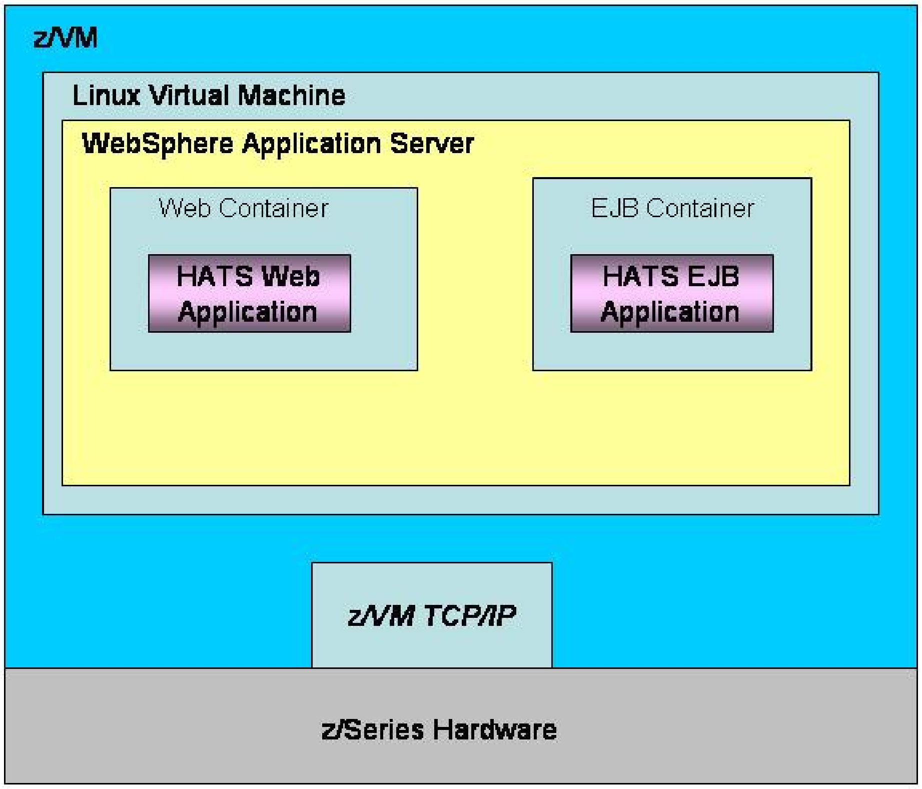
semsys:seminfo_semune: The maximum value of System V semaphore operations per semaphore call. The default value for this option is too low for highly concurrent systems.

6.5 SuSe on z/VM

In this section, we discuss tuning for HATS in a typical WebSphere Application Server for Linux on System z. Linux can run natively on System z hardware or it can run as a guest of z/VM. This section focuses on the Linux as a guest of z/VM configuration.

In the z/VM environment, VM provides basic hardware resource management such as the management of allocation of CPU and memory resources to the Linux guests. The Linux kernel is responsible for everything else including scheduling of threads and so forth

The following chart shows HATS in a basic WebSphere Application Server for Linux on z/VM runtime environment:



In the Linux environment, the WebSphere Application Server runs as a single process in a JVM.

As few services as possible should be started on a Linux Server image.

For the z/VM environment, **at a minimum**, **you will need to tune the following**:
- z/VM
- Linux Kernel
- TCP/IP

z/VM

z/VM is an operating system that manages guest virtual machines such as Linux. For the purpose of this documentation, the only active guest OS was the one being tested. For that reason, the processors were dedicated to the guest being tested. This might not be possible in all scenarios.

To improve the scheduling and dispatch of resources, quick dispatch was turned on. To ensure a certain amount of CPU cycles were being spent on the guest OS, 100% of the absolute CPU share was allotted to the guest under test.

Ensure the necessary amount of storage is allocated to run Linux and HATS under load. The memory considerations of z/VM and other guests must be taken into account when considering storage.

Committing extra resources was ineffective for the test described in this document but could be beneficial in a scenario with multiple guest virtual machines competing for resources.

Linux Kernel

The following kernel parameters usually need tuning:

- Number of file descriptors
- Maximum shared memory segment size
- Maximum number of shared memory segments

Number of File Descriptors: (fs.file-max) specifies the maximum of open files permitted. Default: 351042. Recommendation: 65536.

```
echo "65536" >> /proc/sys/fs/file-max
```

Maximum shared memory segment size (kernel.shmmax). Defines the maximum allowable size of one shared memory segment. Default: 33554432. Recommendation: 1073741824.

```
echo “1073741824” >> /proc/sys/kernel/shmmax
```

Maximum number of shared memory segments: (kernel.shmmni**)** specifies the maximum of number of shared memory segments.
Default: 4096. Recommendation: 8000.

```
echo “8000” >> /proc/sys/kernel/shmmni
```

TCP/IP

- Make sure you have enough local ports available.

  echo “1024 65000” >> /proc/sys/net/ipv4/ip_local_port_range

- Increase the memory available with TCP/IP send and receive buffers.

```
echo  “2621143” >> /proc/sys/net/core/rmem_max
echo  “2621143” >> /proc/sys/net/core/rmem_default

echo  “2621143” >> /proc/sys/net/core/wmem_max
echo  “2621143” >> /proc/sys/net/core/wmem_default

echo  “4096  131072 8388608” >> /proc/sys/net/ipv4/tcp_rmem
echo  “4096  131072 8388608” >> /proc/sys/net/ipv4/tcp_wmem
```

- Disable TCP Selective Acknowledgements (RFC2018).

```
echo “0” >> /proc/sys/net/ipv4/tcp_sack
```

- Disable TCP Timestamps (RFC1323).

```
echo “0” >> /proc/sys/net/ipv4/tcp_timestamps
```

- Decrease TCP KeepAlive timeout.

```
echo “15” >> /proc/sys/net/ipv4/tcp_keepalive_time
```

- Reuse TIME-WAIT sockets.

```
echo “-1”   >> /proc/sys/net/ipv4/tcp_tw_reuse
```

- Enable Path MTU Discovery to find the largest possible MTU (RFC1191).

```
echo “1” >> /proc/sys/net/ipv4/ip_no_pmtu_disc
```

- Increase default user limits:

```
ulimit –n 131072
```

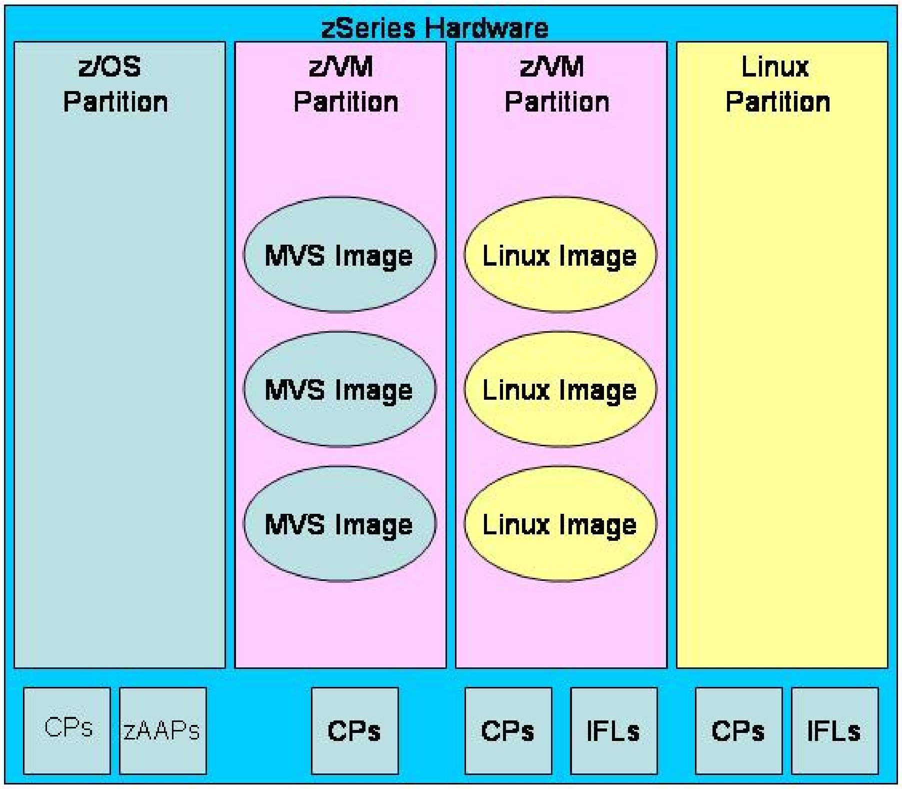
  Specifies the maximum number of files that can be opened at once by a user. Default 1024. Recommended 131072.

```
ulimit –u 39936
```

Specifies the maximum number of user processes. Default 16384. Recommended 39936.

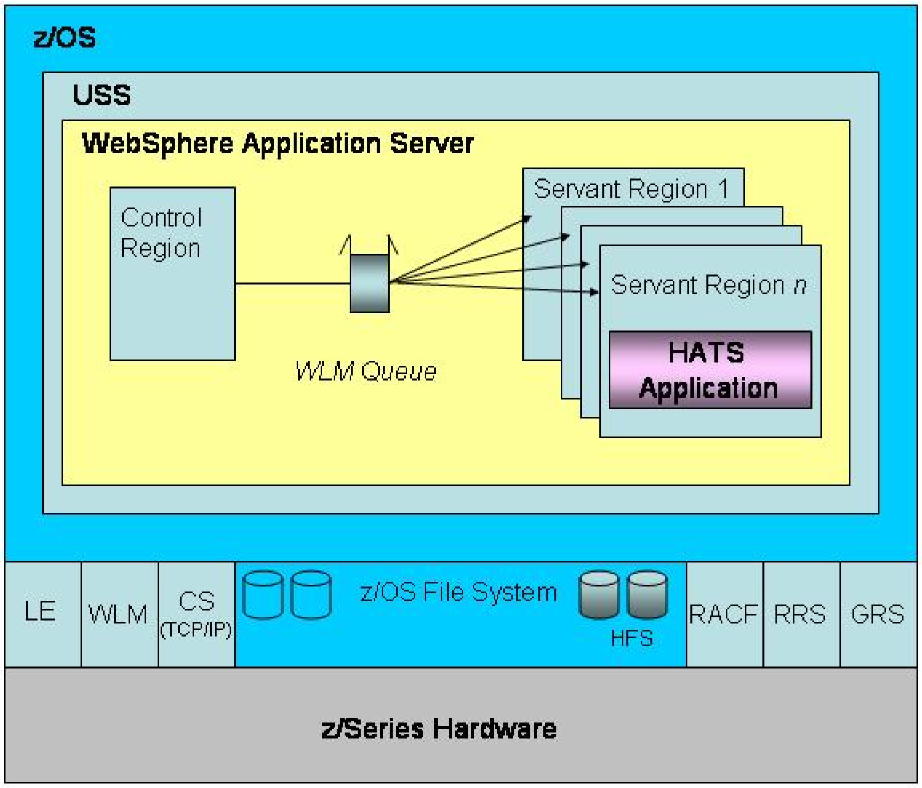
6.6 System z

On System z, HATS applications are supported in four System z operating system environments: System z, MVS as a guest of z/OS, Linux as a guest of z/VM, and Linux running natively in an LPAR.



This section is intended to offer you a basic view of the WebSphere Application Server for the System z runtime environment focusing on important core components that affect HATS performance in this environment.   This section does not address clustered environments or specific details of the basic WebSphere Application Server. Refer to WebSphere documentation to dig deeper into configuration options and details.

The following chart shows HATS in a basic WebSphere Application Server for System z runtime environment:



System z has many subsystems that provide reliability, availability, serviceability, and other qualities of service. As a well-behaved System z product, WebSphere Application Server uses these subsystems. The picture above shows a few key subsystems that WebShpere Application Server uses.

As a Java application, Unix System Services (USS) runs the WebSphere Application Server and the USS Hierarchical File System (HFS) holds the server's configuration files, applications and so forth. Also, WAS uses System z services such as LE (storage and message control), CS (TCP/IP communications), workload management (WLM), resource recovery services (RRS), and global resource serialization (GRS). These services must be tuned for optimal performance of HATS applications running in the WebSphere Application Server.

The WebSphere Application Server executes in two types of address spaces: A Control Region (CR) and one or more servant regions. The CR receives the browser request and queues it to the WLM. WLM load balances requests from its queue across servant regions based on configured application policies.

HATS applications run in the WebSphere Application Server's servant regions.

6.6.1 Unix system services

WebSphere Application Server is a UNIX system services application. It runs in the UNIX kernel under System z. Define to USS how many processes (address spaces), threads (tasks), sockets, and users it is expected to handle.

UNIX kernel configuration parameters are set in the BPXPRMxx parmlib member. Tune the following kernel parameters:

- **MAXTHREADTASKS:** Defines the maximum number of threads that a single process can have active. Set between 1000 and 5000.

- **MAXTHREADS:** Defines the total threads (running, queued, and exited) that a single process can have active concurrently. MAXTHREADS should always be equal to of greater than MAXTHREADTASKS. Set this to twice the value of MAXTHREADTASKS.

- **MAXFILEPROC:** Defines the highest possible number of combined file descriptors (file directories, sockets, and pipe) that a single USS process can have open. Multiply 120 by the number of Web requests per second; if you don't know that information, use 5,000 for low throughput, 10,000 for medium-throughput, and 35,000 for high-throughput environments. Resources are not used for these files until they are allocated so there is no disadvantage to setting it high. For V1R6 and earlier, the limit is 64K. For System z V1R7 and later, the limit for MAXFILEPROC is 128K.

- **MAXSOCKETS:** Defines the maximum number of sockets. Use two times the value of MAXFILEPROC + 500. MAXSOCKETS is set in the NETWORK statement in the BPXPRMxx member. MAXSOCKETS is a USS limitation, TCP/IP has no limits on the number of open sockets. There are no storage concerns when specifying large values for MAXSOCKETS.

```
NETWORK DOMAINNAME (AF_INET) DOMAINNUMBER(2) MAXSOCKETS(50000)
```

- **IPCSHMMPAGES:** Defines the maximum pages for shared memory segments. Use 256.

If you are running more than one server or other applications, check the kernel parameters to ensure they are adequate:

- **MAXPROCSYS:** Defines the number of USS processes that can be active at one time within the system. Make sure this value is high enough to support the maximum number of processes at peak usage.

- **MAXPROCUSER:** Defines the maximum number of processes that a single USS user might have active concurrently. Ensure the value of this parameter is sufficient for the user with the most processes. To negate the effect of this parameter, set the value to be equal to or greater than MAXPROCSYS.

- **MAXUIDS: S**pecifies the number of unique user IDs (UIDs) that can use kernel services at the same time. Every user needs a unique UID, so ensure this value is high enough to support the maximum number of concurrent users.

6.6.2 Language environment (LE)

For best performance, follow these tuning tips.

- Turn LE tracing off.
- Put frequently used LE modules into the link pack area (LPA). This improves performance by reducing the number of copies of the same module loaded in virtual storage and consequently in pageable storage.
- Do not use these options in your production environment:
  - RPTSTG(ON)
  - RPTOPTS(ON)
  - HEAPCHK(ON)
- Turn on LE heappools; Set LEPARM='HEAPP(ON)'

6.6.3 Communications Server (TCP/IP)

Communications Server for System z provides TCP/IP support. See section 7.6 for details.

6.6.4 Resource Access Control Facility (RACF)

Resource Access Control Facility (RACF) administers security in a WebSphere environment. Security uses resources heavily. Enable security only for what you need. Disable facility classes that are not needed.

Turn off the recording of system management facility (SMF) records other than the ones you need.

6.6.5 Workload Manager (WLM)

On System z, a WebSphere Application Server consists of a Controller address space and one or more Servant address spaces. Workload Manager is used to control the number of Servants and for the even distribution of workload across Servants. WLM can also be used to balance workloads across clustered application servers in certain situations.

Two very important WLM tuning tips are:

- Setting WLM goals properly can have a very significant effect on application throughput. WebSphere for System z system address spaces should be given a fairly high priority.

  For more information about setting up WLM performance goals, see *z/OS MVS Planning: Workload Management*.

- Set your application environment to "no limit". This setting is required if you need more than one servant per application server. When the WLM configuration is set to "no limit", you can control the maximum and minimum number of Servants by specifying variables in the application server console.

  See Number of servant regions section for more information on setting the number of servant regions.

6.6.6 Resource Recovery Services (RRS)

WebSphere Application Server uses Resource Recovery Services to handle transactional behaviors and to access all recovery services. Here are a few optimization tips that can impact the operation of RRS.

- Use the coupling facility (CF) logger if possible. DASD logger can limit throughput because it is I/O-sensitive. The CF logger provides the best throughput and does not require any DASD I/O.
- If using the CF logger is not possible, use well performing DASD.
- Size the RRS logs so that they are not offloaded to a secondary system log stream too often. Offloading too often will impact overall throughput.
- Only archive logs if absolutely necessary. Archiving logs introduces extra DASD I/Os.

6.6.7 WebSphere Environment

You can use the WebSphere administrative console to adjust the following settings and values that can enhance the performance of HATS applications running on a System z server:

- Workload Profile
- Number of Servant Regions
- Java Heap Sizes
- Garbage Collection

Workload profile

On System z, setting the workload profile to LONGWAIT is important for application processing that involves sending or receiving information across a network. The LONGWAIT setting allows each servant region to have a pool of 40 threads to handle concurrent HATS requests. The default pool size is based on the number of CPs. This parameter setting is important when sizing the number of servant regions.

> Servers→Application Servers→*servername*→ORB Service→Advanced Settings→Workload Profile, set value to LONGWAIT.

Number of servant regions

On System z, HATS applications run in the WebSphere Application Server Servant regions. Each Servant has a pool of 40 threads, if you specified LONGWAIT as the workload profile. By default, there is only one servant to handle all concurrent HATS requests. You can increase the number of servant regions to handle your application workload.

> **Multiple instances enabled:** By default, only one servant is initialized. To have more than one servant, and allow multiple servants to be initialized; this parameter must be enabled.
>
> > **Servers→Application Servers→***servername***→Server Instance→Multiple Instances Enabled**, check the box.
>
> **Minimum Number of Instances:** Specifies the number of servant regions to be kept running once server regions are initialized during application server start up. Use this environment variable when the response time for the workload requires that several servant regions are needed to be ready to process work. The default for J2EE servers is 1. The maximum value is 20. Recommendation: Estimate this value for your workload.

Servers→Application Servers→*servername*→Server Instance→Server Instance→Minimum Number of Instances, set to a valid number.

Maximum Number of Instances: Specifies the total number of servants allowed by workload management to run concurrently in the server's application environment. Set this value to limit the number of server regions created by workload management for a server. The default is zero, which means there is no limit.

Servers→Application Servers→*servername*→Server Instance→Server Instance→ Maximum Number of Instances, set to a valid number.

Java heap sizes

The Java heap parameters influence performance and the behavior of garbage collection. Larger heap sizes take longer to fill and longer to garbage collect. With large heap sizes, applications run longer before a garbage collection occurs and it takes a longer time to perform the garbage collection. Our recommendation is to monitor garbage collection and adjust your heap sizes so that no more than 15% of the total execution time is spent in garbage collection. A good way to see what is going on with garbage collection is to use the JVM's verbose garbage collection option, which is enabled by adding –verbosegc to the JVM arguments. Verbosegc provides an output file that gives the date and time of each garbage collection cycle. From the output file, you can calculate how much of the total execution time is spent in garbage collection.

On the System z platform, there is a JVM in both the controller and the servant. Usually, the JVM in the controller does not need to be tuned. This information applies to the JVM in the servant.

There are two JVM parameters that specify how much memory is allocated for the JVM heap. The settings of these two parameters affect the performance of your HATS applications:

Initial heap size: Specifies how much memory is allocated for the heap when the JVM starts. Properly tuning this parameter reduces the overhead of garbage collection, improving server response time and throughput. For most HATS applications, the default setting for this option is too low, resulting in a high number of minor garbage collections. Recommendation: This setting is workload specific, but it should be at least 512 MB. If the heap size changes frequently, you might improve performance if you specify the same value for the initial and maximum heap parameters.

Servers→Application Servers→*servername*→Process Definitions→JVM→Initial Heap Size, set to a value appropriate for your installation, but at least 512 MB.

Maximum heap size: Specifies the maximum heap size the Java interpreter uses for dynamically allocated objects and arrays. Properly tuning this parameter also reduces the overhead of garbage collection, improving server response time and throughput. For most HATS applications, the default setting for this option is too low, resulting in a high number of minor garbage collections. Recommendation: This setting is workload specific, but it should be at least 768 MB.

Servers→Application Servers→*servername***→Process Definitions→JVM→Max Heap Size,** set to a value appropriate for your installation.

WebSphere variables: You should add the following variables under Manage WebSphere Variables and set them to the recommended values as listed:

Servers→Application Servers→*servername***→Custom Properties,** add the following:

protocol_http_tcp_no_delay=1 property for non-secure connections or
protocol_https_tcp_no_delay=1 property for secure connections.
protocol_http_max_persist request=1
protocol_http_timeout_input=0
protocol_http_timeout_output=0
protocol_http_timeout_persistentSession=0
wlm_stateful_session_placement_on=1

Garbage collection tuning

The JVM memory management function, or garbage collection, provides one of the biggest opportunities for improving JVM performance.

Garbage collection should normally consume from 2% to 5% of the total execution of a properly functioning application. It takes monitoring to understand the effects of garbage collection.

Concurrent mark: If you have pause time problems that are denoted by erratic application response times, which are caused by normal garbage collection, set gcpolicy to optavgpause.

```
-Xgcpolicy=optavgpause
```

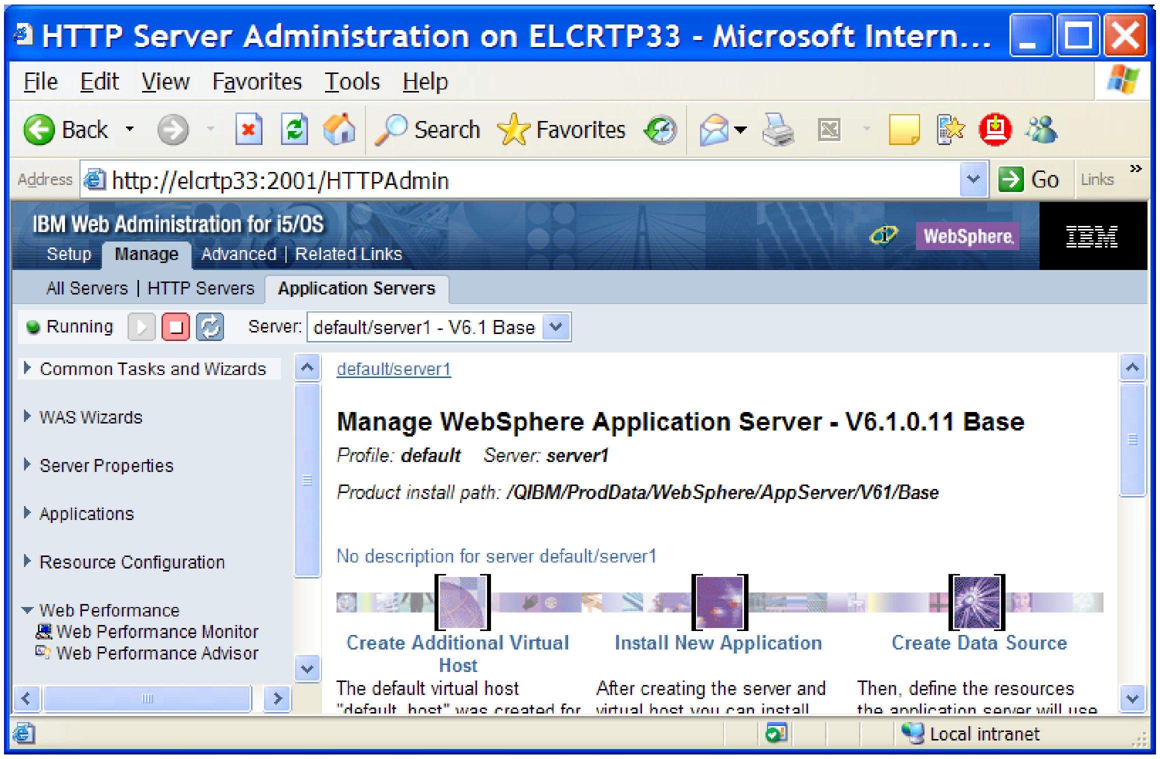
6.7 System i

Tuning system value settings on System i:

System value	**setting**	**Description**
QPRCMLTTSK	1 or 2 on Power 5 system	Allows more than one set of task data on each CPU
QTHDRSCADJ	1	Determines if the system will make adjustments to the affinity of threads
QTHDRSCAFN	Group: *NoGroup Level: *Normal	Specifies whether or not secondary threads will have affinity to the same group of processors
QMAXACTLVL	*NOMAX	Limits the number of threads that can compete at the same time for memory and processors
QPFRADJ	0	Specifies whether the system will adjust values automatically for maximum number of active jobs and pool sizes

Additional Miscellaneous Tuning information on System i

For System i HTTP Server on V5R4 and V6R1 operating system levels there is a Web Performance Advisor available for use within HTTP Server Administration.


HTTP Server Administration on ELCRTP33 - Microsoft Intern...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://elcrtp33:2001/HTTPAdmin
Go Links
IBM Web Administration for i5/OS
Setup Manage Advanced | Related Links
WebSphere
IBM
All Servers | HTTP Servers Application Servers
Running
Server: default/server1 - V6.1 Base
Common Tasks and Wizards
WAS Wizards
Server Properties
Applications
Resource Configuration
Web Performance
Web Performance Monitor
Web Performance Advisor
default/server1
Create Additional Virtual Host
Install New Application
Create Data Source
Local intranet

Manage WebSphere Application Server - V6.1.0.11 Base

Profile: **default** Server: **server1**

Product install path: **/QIBM/ProdData/WebSphere/AppServer/V61/Base**

No description for server default/server1

The default virtual host "default_host" was created for

After creating the server and virtual host you can install

Then, define the resources the application server will use

If using the Web Performance Advisor is not an option, we recommend some changes using the HTTP Administration Tool. These parameters are:

Server Properties > System Resources > HTTP Connections tab		
Suggested value	**Default value**	**Description**
1	5	Time to wait between requests
500	100	Maximum requests per connection

Server Properties > System Resources > Advanced tab		
Suggested value	**Default value**	**Description**
1000	100	Number of threads to process requests
20	4	Accept Threads

Please note that these tuning parameters were designed for a run of 1000 concurrent sessions and are correlated to the thread pool numbers that were specified in WebSphere Application Server tuning.

7.0 Network Tuning for HATS

HATS applications perform best when certain network parameters are tuned properly. For best performance, minimize packet fragmenting and acknowledgements as they slow down network transfers. A HATS application response contains an HTML file, several JavaScript applications, images, templates, stylesheets, and other content. The size of the response, including all of its parts, is usually over 100KB if nothing is done to optimize its size over the network. Responses that contain lots of data fields, tables, graphics, and so forth can become very large, over 300KB. This means that the tuning of the network and TCP/IP is critical to network communications and HATS application response times.

- Make sure you have enough local ports available.
- Increase the memory available with TCP/IP send and receive buffers.
- Disable TCP Selective Acknowledgements (RFC2018).
- Disable TCP Timestamps (RFC1323).
- Decrease TCP KeepAlive timeout.
- Reuse TIME-WAIT sockets.
- Enable Path MTU Discovery to find the largest possible MTU (RFC1191).

7.1 Windows Server 2003

Windows TCP/IP parameters are set in the Windows registry. However, careless registry editing can cause irreversible damage. It is highly recommended that you see your Windows Administrator for changes to the registry.

7.1.1 Specific parameters

Registry Location	**Parameter**	**Recommended Setting**
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\	TCPWindowSize	62420
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\	TCP1323Opts	3
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\	TCPTimedWaitDelay	30
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\	MaxUserPort	65534
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\	MaxHashTableSize	8192
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\	NumTcbTablePartitions	8
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\ Interface\xx\	TCPAckFrequency	13
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\	EnablePMTUDiscovery	1
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\ MemoryManagement\	DisablePagingExecutive	1

7.1.2 Parameter Definitions

This section provides brief descriptions of the Windows TCP/IP parameters. To learn more about these TCP/IP parameters, please see the following link: http://www.microsoft.com.

TCPWindowSize: Determines the largest TCP receive window that the system offers. The receive window is the number of bytes a sender can transmit without receiving an acknowledgment. This entry overrides TCP's negotiated maximum receive window size and replaces it with the value of this entry.
TCP uses a receive window that is four times the size of the maximum TCP segment size (MSS) negotiated during connection setup, up to a maximum size of 64 KB. TCP for Windows 2000 also supports windows scaling, as detailed in RFC 1323, TCP Extensions for High Performance. Scaling enables TCP to provide a receive window of up to 1 GB. For Ethernet networks, the default value of this entry is 0x4470 (17,520, or 12 segments of 1,460 bytes each). For other networks, the default value is 0xFFFF (65,535) unless 0xFFFF is larger than:

- Four times the maximum TCP data size on the network; and
- 0x2000 (8,192) rounded up to an even multiple of the network TCP data size

TCP1323Opts: Determines whether TCP uses the timestamping and window scaling features described in RFC 1323, TCP Extensions for High Performance. Window scaling permits TCP to negotiate a scaling factor for the TCP receive window size, allowing for a very large TCP receive window of up to 1 GB. The TCP receive window is the amount of data the sending host can send at one time on a connection.

TCPTimedWaitDelay: Determines the time that must elapse before TCP/IP can release a closed connection and reuse its resources. This interval between closure and release is known as the TIME_WAIT state or twice the maximum segment lifetime (2MSL) state. During this time, reopening the connection to the client and server costs less than establishing a new connection. By reducing the value of this entry, TCP/IP can release closed connections faster and provide more resources for new connections. Adjust this parameter if the running application requires rapid release, the creation of new connections, or an adjustment because of a low throughput caused by multiple connections in the TIME_WAIT state.

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\TCPIP\Parameters registry subkeyword, and create a new REG_DWORD value named TcpTimedWaitDelay.

Default value: 240 seconds (4 minutes). **Recommended value:** 30 seconds.

MaxUserPort - Determines the highest port number that TCP/IP can assign when an application requests an available user port from the system.

Default value: None. **Recommended value:** At least decimal 65534.

MaxHashTableSize - Determines the size of the hash table in which TCP control blocks (TCBs) are stored. TCP stores control blocks in a hash table so it can find them very quickly.

Default value: 512 **Recommended value:** 8192

NumTcbTablePartitions - Directs the Dynamic Host Configuration Protocol (DHCP) client to ignore Media Sense events from the interface.
Media Sense provides a mechanism for network adapters to notify the protocol stack when network media is connected or disconnected. The protocol stack can then update its network configuration parameters. Media Sense is especially useful on portable computers, which are commonly disconnected from one network and connected to a different network.

TCPAckFrequency - Determines how often TCP/IP sends an acknowledgment message. If the value is 2, TCP/IP will send an acknowledgment after 2 segments have been received or when 1 segment has been received but no other segment has been received for 200 milliseconds.

If the value is 3, TCP/IP will send an acknowledgment after 3 segments have been received or when 1 or 2 segments have been received but no other segment has been received for 200 milliseconds, and so on.

If you need to improve response time by removing the TCP/IP acknowledgment delays, set this value to 1. In this case, TCP/IP will immediately send an acknowledgment to every segment. If your connections are used primarily to move large amounts of data and the 200 millisecond delay is insignificant, you may want to increase this value to reduce the overhead of acknowledgments.

By default, Windows uses a value of 2 (acknowledges every other segment). The valid range for this value is 0 to 255 where 0 indicates that the default value (2) should be used.

EnablePMTUDiscovery – Determines whether TCP uses a fixed, default maximum transmission unit (MTU) or attempts to detect the actual MTU.

By discovering the Path MTU and limiting TCP segments to this size, TCP can eliminate fragmentation at routers connecting networks with different MTUs. Fragmentation reduces TCP throughput and increases network congestion.

DisablePagingExecutive - Specifies whether user-mode and kernel-mode drivers and kernel-mode system code can be paged to disk when not in use.

Value	**Meaning**
0	Drivers and the kernel can be paged to disk as needed.
1	Drivers and the kernel must remain in physical memory.

7.2 SuSe on System x

7.2.1 Specific parameters

These parameters can be set either at the command line or in the sysclt.conf file in the /etc directory. Setting the parameters in the sysclt.conf file will keep the parameters stored across reboot.

TCP/IP Tuning		
Location	**Parameter**	**Setting**
/proc/sys/net/core/	rmem_default	126976
/proc/sys/net/core/	rmem_max	8388608
/proc/sys/net/core/	wmem_default	126976
/proc/sys/net/core/	wmem_max	8388608
/proc/sys/net/core/	somaxconn	3000
/proc/sys/net/core/	netdev_max_backlog	3000
/proc/sys/net/ipv4/	ip_no_pmtu_disc	1
/proc/sys/net/ipv4/	tcp_sack	0
/proc/sys/net/ipv4/	tcp_timestamps	0
/proc/sys/net/ipv4/	tcp_window_scaling	1
/proc/sys/net/ipv4/	tcp_max_syn_backlog	16384
/proc/sys/net/ipv4/	tcp_keepalive_time	30
/proc/sys/net/ipv4/	ip_local_port_range	32768 61000
/proc/sys/net/ipv4/	tcp_mem	1048576 1048576 1048576
/proc/sys/net/ipv4/	tcp_rmem	4096 131072 8388608
/proc/sys/net/ipv4/	tcp_wmem	4096 131072 8388608
/proc/sys/net/ipv4/	tcp_fin_timeout	30
/proc/sys/net/ipv4/	tcp_keepalive_intvl	15
/proc/sys/net/ipv4/	tcp_keepalive_probes	5
/proc/self/	mapped_base	1431654400

7.2.2 Parameter Definitions

rmem_default - Controls the default socket receive buffer size.
rmem_max - Controls the maximum socket receive buffer size.
wmem_default - Controls the default socket send buffer size.
wmem_max - Controls the maximum socket send buffer size.
somaxconn - Determines the maximum number of pending TCP connections.
netdev_max_backlog - variable netdev_max_backlog describes the maximum number of incoming packets that can be queued up for upper-layer processing.
ip_no_pmtu_disc - Disables PMTU (Path Maximum Transfer Unit) discovery if enabled. PMTU discovery tries to discover the maximum transfer unit to specific hosts, including all the intermediate hops on the way there.
tcp_sack – Enables selective acknowledgements as defined in RFC 2883
tcp_timestamps - Tells the kernel to use timestamps as defined in RFC 1323. A TCP option that can be used to calculate the round trip measurement in a better way than the

retransmission timeout method can.
tcp_window_scaling - Enables window scaling as it is defined in RFC 1323. Enabling tcp_window_scaling enables a special TCP option which makes it possible to scale these windows to a larger size, and hence reduces bandwidth losses due to not utilizing the whole connection.
tcp_max_syn_backlog - Tells your machine how many synchronization (SYN) requests to keep in memory while waiting for the third packet to arrive.

This variable takes an integer value and is per default set to different values depending on how much memory you have.
tcp_keepalive_time - Tells the TCP/IP stack how often to send TCP keepalive packets to keep a connection alive if it is currently unused. This value is only used when keepalive is enabled.
ip_local_port_range - Defines the local port range that is used by TCP and UDP traffic to choose the local port. You will see in the parameters of this file two numbers: The first number is the first local port allowed for TCP and UDP traffic on the server, the second is the last local port number.

tcp_mem - The first value specified tells the kernel the low threshold. Below this point, the TCP stack does not bother about putting any pressure on the memory usage by different TCP sockets. The second value tells the kernel at which point to start pressuring memory usage down. This so called memory pressure mode is continued until the memory usage enters the lower threshold again, and at which point it enters the default behavior of the low threshold again. The final value tells the kernel the maximum number of memory pages it can use. If this value is reached, TCP streams and packets start getting dropped until we reach a lower memory usage again. This value includes all TCP sockets currently in use.

tcp_rmem - The first value tells the kernel the minimum receive buffer for each TCP connection, and this buffer is always allocated to a TCP socket, even under high pressure on the system. The second value specified tells the kernel the default receive buffer allocated for each TCP socket. The third and last value specified in this variable specifies the maximum receive buffer that can be allocated for a TCP socket.

tcp_wmem - The first value in this variable tells the minimum TCP send buffer space available for a single TCP socket. This space is always allocated for a specific TCP socket opened by a program as soon as it is opened. The second value in the variable tells us the default buffer space allowed for a single TCP socket to use. If the buffer tries to grow larger than this, it may get hampered if the system is currently under heavy load and don't have a lot of memory space available. The third value tells the kernel the maximum TCP send buffer space. This defines the maximum amount of memory a single TCP socket may use.

tcp_fin_timeout - tells kernel how long to keep sockets in the state FIN-WAIT-2 if the socket closed on your side.
tcp_keepalive_intvl – Tells the kernel how long to wait for a reply on each keepalive

probe. This value is in other words extremely important when you try to calculate how long time will go before your connection
will die a keepalive death.
tcp_keepalive_probes - Tells the kernel how many TCP keepalive probes to send out before it decides a specific connection is broken.

7.3 AIX

TCP/IP	
Parameter	**Setting**
somaxconn	8192
ulimit -n	unlimited
Rfc2414	1
tcp_nodelayack	1
tcp_recvspace	49152
tcp_sendspace	49152

Somaxconn – Determines the maximum number of pending TCP connections.
ulimit –n - Removes the limit on the maximum number of file descriptors.
Rfc2414 - Increase in the permitted initial window for TCP from one segment.
tcp_nodelayack - Prompts TCP to send an immediate acknowledgement, rather than the usual 200 ms delay. Sending an immediate acknowledgement might add a little more overhead, but in some cases, greatly improves performance.
tcp_recvspace - Specifies how many bytes of data the receiving system can buffer in the kernel on the receiving sockets queue.
tcp_sendspace - Specifies how much data the sending application can buffer in the kernel before the application is blocked on a send call.

7.4 SuSe on System p

Parameter	**Setting**
/proc/sys/net/core/rmem_default	87380
/proc/sys/net/core/rmem_max	8388608
/proc/sys/net/core/wmem_default	87380
/proc/sys/net/core/wmem_max	8388608
/proc/sys/net/ipv4/ip_no_pmtu_disc	1
/proc/sys/net/ipv4/tcp_sack	0
/proc/sys/net/ipv4/tcp_timestamps	0
/proc/sys/net/ipv4/tcp_window_scaling	1
/proc/sys/net/ipv4/tcp_max_syn_backlog	16384
/proc/sys/net/ipv4/tcp_keepalive_time	30
/proc/sys/net/ipv4/ip_local_port_range	1024 65000
/proc/sys/net/ipv4/tcp_mem	1048576 1048576 1048576
/proc/sys/net/ipv4/tcp_rmem	4096 131072 8388608
/proc/sys/net/ipv4/tcp_wmem	4096 131072 8388608
/proc/sys/net/core/somaxconn	2048

/proc/sys/net/core/rmem_default - Controls the default socket receive buffer size.

/proc/sys/net/core/rmem_max - Controls the maximum socket receive buffer size.

/proc/sys/net/core/wmem_default - Controls the default socket send buffer size.

/proc/sys/net/core/wmem_max - Controls the maximum socket send buffer size.

/proc/sys/net/ipv4/ip_no_pmtu_disc - Disables PMTU (Path Maximum Transfer Unit) discovery if enabled. PMTU discovery tries to discover the maximum transfer unit to specific hosts, including all the intermediate hops on the way there.

/proc/sys/net/ipv4/tcp_sack – Enables selective acknowledgements as defined in RFC 2883.

/proc/sys/net/ipv4/tcp_timestamps - Tells the kernel to use timestamps as defined in RFC 1323. This is A TCP option that can be used to calculate the Round Trip Measurement in a better way than the retransmission timeout method can.

/proc/sys/net/ipv4/tcp_window_scaling - Enables window scaling as it is defined in RFC 1323. Enabling tcp_window_scaling enables a special TCP option which makes it possible to scale these windows to a larger size, and hence reduces bandwidth losses due to not utilizing the whole connection.

/proc/sys/net/ipv4/tcp_max_syn_backlog - Tells your box how many SYN requests to keep in memory that we have yet to get the third packet in a 3-way handshake from.

This variable takes an integer value and is per default set to different values depending on how much memory you have.

/proc/sys/net/ipv4/tcp_keepalive_time - Tells the TCP/IP stack how often to send TCP keepalive packets to keep an connection alive if it is currently unused. This value is only used when keepalive is enabled.

/proc/sys/net/ipv4/ip_local_port_range - Defines the local port range that is used by TCP and UDP traffic to choose the local port. You will see in the parameters of this file two numbers: The first number is the first local port allowed for TCP and UDP traffic on the server, the second is the last local port number.

/proc/sys/net/ipv4/tcp_mem - The first value specified in the tcp_mem variable tells the kernel the low threshold. Below this point, the TCP stack do not bother at all about putting any pressure on the memory usage by different TCP sockets. The second value tells the kernel at which point to start pressuring memory usage down. This so called memory pressure mode is continued until the memory usage enters the lower threshold again, and at which point it enters the default behavior of the low threshold again. The final value tells the kernel the maximum number of memory pages it can use. If this value is reached, TCP streams and packets start getting dropped until we reach a lower memory usage again. This value includes all TCP sockets currently in use.

/proc/sys/net/ipv4/tcp_rmem - The first value tells the kernel the minimum receive buffer for each TCP connection, and this buffer is always allocated to a TCP socket, even under high pressure on the system.
The second value specified tells the kernel the default receive buffer allocated for each TCP socket.
The third and last value specified in this variable specifies the maximum receive buffer that can be allocated for a TCP socket.

/proc/sys/net/ipv4/tcp_wmem - The first value in this variable tells the minimum TCP send buffer space available for a single TCP socket. This space is always allocated for a specific TCP socket opened by a program as soon as it is opened. The second value in the variable tells us the default buffer space allowed for a single TCP socket to use. If the buffer tries to grow larger than this, it may get hampered if the system is currently under heavy load and don't have a lot of memory space available. The third value tells the kernel the maximum TCP send buffer space. This defines the maximum amount of memory a single TCP socket may use.

/proc/sys/net/core/somaxconn - Determines the maximum number of pending TCP connections.

7.5 Solaris

The following parameters can be set on the command line using the ndd -set command. Setting these parameters at the command line does not permanently set the parameters. After a reboot the parameters will be reset to the system defaults. To set these parameters permanently place the following parameters in the file labeled system in the /etc directory.

TCP/IP Tuning	
Parameter	**Setting**
ndd -set /dev/tcp tcp_time_wait_interval	60000
ndd -set /dev/tcp tcp_keepalive_interval	7200000
ndd -set /dev/tcp tcp_conn_req_max_q	8192
ndd -set /dev/tcp tcp_conn_req_max_q0	8192
ndd -set /dev/tcp tcp_max_buf	4194304
ndd -set /dev/tcp tcp_cwnd_max	2097152
ndd -set /dev/tcp tcp_recv_hiwat	400000
ndd -set /dev/tcp tcp_xmit_hiwat	400000
ndd –set /dev/tcp tcp_naglim_def	1

Tcp _time_wait_interval - Notifies TCP/IP on how long to keep the connection control blocks closed. After the applications complete the TCP/IP connection, the control blocks are kept for the specified time. When high connection rates occur, a large backlog of the TCP/IP connections accumulates and can slow server performance.

Tcp_keepalive_interval - Ensures that a connection stays in an active and established state.

Tcp_conn_req_max_q - Change this parameter when a high rate of incoming connection requests result in connection failures

tcp_conn_req_max_q0 – Determines the default maximum number of incomplete pending TCP connections for a TCP listener.

tcp_max_buf – Sets maximum buffer size in bytes. It controls how large the send and receive buffers are.

tcp_cwnd_max – Sets the maximum value of TCP congestion window (cwnd) in bytes. For more information on TCP congestion window, refer to RFC 1122 and RFC 2581.

tcp_recv_hiwat – Sets the default receive window size in bytes. See `tcp_max_buf` also.

tcp_xmit_hiwat – Sets the default send window size in bytes. See `tcp_max_buf` also.

tcp_naglim_def - This algorithm is used to control congestion on Ethernet networks by reducing the amount of small packets that are exchanged between a server and a client. It also has the potential side effect of increasing network latency for those small packets. Web Servers generally disable this algorithm at the socket call level. This value can be disconnected by setting this value to 1.

7.6 System z

Use the following tips to tune TCP/IP:

- Set NODELAYACKS. This parameter can significantly improve throughput by shortening the delay in sending acknowledgement responses to clients. This parameter is set on the PORT statement of TCPCONFIG.

  ```
  TCPCONFIG
    PORT 80    TCP  NODELAYACKS
  ```

- Increase the sizes of the TCP/IP send and receive buffers from the default of 16KB to at least 64 KB. Larger sizes might not be unreasonable for fast networks and large data transfers.

  TCPMAXRCVBUFRSIZE: Limits the buffer size that can be requested by any application. By limiting the receive buffer size, this parameter can avoid an inbound flood of data arriving over multiple concurrent applications. The maximum buffer size is only significant as the number of applications using it become larger. Maximum is 512KB.

  TCPRCVBUFRSIZE: Sets the default receive buffer size given to a TCP application on the System z host.

  TCPSENDBFRSIZE: Sets the default for the size of the buffers used to hold outbound data prior to transmission. It can be increased up to 256 KB.

  The send and receive buffer sizes are controlled with the TCPCONFIG statement.

  ```
    TCPCONFIG  TCPMAXRCVBUFRSIZE 524288
               TCPSENDBFRSIZE 65535
               TCPRCVBUFRSIZE 65535
  ```

- Increase the default listen backlogs:

  SOMAXCONN: Sets the maximum number of connection requests that can be queued to a socket listener.

  SOMAXCONN is specified in the TCP/IP Profile.

  ```
    SOMAXCONN 8096
  ```

- Reduce the finwait2time.

FINWAIT2TIME: Specifies the number of seconds a TCP connection should remain in the FINWAIT2 state. The default value is 600 seconds.

FINWAIT2TIME is specified on the TCPCONFIG statement.

```
TCPCONFIG FINWAIT2TIME 60
```

8.0 WebSphere Application Server tuning for HATS

HATS applications are installed as J2EE applications on a base WebSphere Application Server. The first time an application is requested, the JSP must be compiled. During the installation of a HATS application, you can request that it be compiled when the application server starts it. Compiling the application when it is started improves the response time of the first request.

Tuning the base WebSphere Application Server is absolutely necessary to achieve optimal performance from HATS applications.

This section does not cover every possible tuning parameter available for WebSphere Application Server. Instead, it presents the most important recommendations based on our benchmark performance measurements. Except where noted, these recommendations are valid for all HATS supported platforms and the parameters can be set using the WebSphere Administrative Console.

At a minimum, you need to tune the following:

- Java heap sizes
- Web Container threads
- Thread Pool Sizes
- Web Container
- Garbage Collection

For more details on tuning WebSphere Application Server, see the Tuning Section of the InfoCenter located at http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp

8.1 Specific parameters

WebSphere Application Server Tuning			
Component	**Parameter**		**Setting**
Java Virtual Machine	**Process Definition**	**Initial java heap size**[1]	Same as maximum java heap size
		Maximum java heap[1] **size**	1024 MB to 1400 MB
Web Container Transport Chains	**HTTP Inbound Channel**	**Use persistent connections**	Enabled[2]
		Maximum persistent requests	-1

Web Container	**Thread Pool**	**Minimum size**	100
		Maximum size	500
		Allow Thread allocation beyond maximum	Enabled

Notes:
[1] Initial and Maximum java heap sizes should be set according to your application requirements. You should monitor the garbage collection to set an appropriate heap size for your applications. The following heap sizes were used in our stress performance benchmarking:

Platform	**Initial/Maximum Heap Sizes**
Windows	**1400 MB**
AIX	**1024 MB**
SUSE on System p	**1024 MB**
SUSE on System x	**1400 MB**
Solaris	**1400 MB**
SUSE on System z	**1024 MB**
System i	**1024 MB**

[2] For user workloads in which there were less than 900 concurrent users, persistent connections were used. For larger numbers of concurrent users, persistent connections were disabled.

JVM Tuning	
Parameter	**Explanation**
-server	Allows the Server HotSpot be first option. (default and 2GB memory)
-Xquickstart	Provides faster server startup.

8.2 Parameter Definitions

8.2.1 Java Heap Sizes

The Java heap parameters influence performance and the behavior of garbage collection. Larger heap sizes take longer to fill and longer to garbage collect. With large heap sizes, applications run longer before a garbage collection occurs and it takes a longer time to perform the garbage collection. On the other hand, smaller heap sizes fill quickly and cause more frequent garbage collections. It is necessary to experiment to find best heap size specifications for your environment. Our recommendation is to monitor garbage collection and adjust your heap sizes so that no more than 15% of the total execution time is spent in garbage collection.

There are two JVM parameters that specify how much memory is allocated for the JVM heap. The settings of these two parameters will affect the performance of your HATS applications:

Initial Heap Size: Specifies how much memory is allocated for the heap when the JVM starts. Properly tuning this parameter reduces the overhead of garbage collection, improving server response time and throughput. For most HATS applications, the default setting for this option is too low, resulting in a high number of minor garbage collections. Recommendation: This setting will be workload specific, but it should be at least 512 MB. If the heap size changes frequently, you might improve performance if you specify the same value for the initial and maximum heap parameters.

> **Servers→Application Servers→***servername***→Process Definitions→JVM→Initial Heap Size,** set to a value appropriate for your installation, but at least 512 MB. Setting this parameter to the maximum heap size often gives the best performance.

Maximum Heap Size: Specifies the maximum heap size the Java interpreter will use for dynamically allocated objects and arrays. Properly tuning this parameter also reduces the overhead of garbage collection, improving server response time and throughput. For most HATS applications, the default setting for this option is too low, resulting in a high number of minor garbage collections. Recommendation: This setting will be workload specific, but it should be at least 768 MB.

> **Servers→Application Servers→***servername***→Process Definitions→JVM→Max Heap Size,** set to a value appropriate for your installation.

WebSphere Variables: You should add the following variables under Manage WebSphere Variables and set them to the recommended values as listed:

> **Servers→Application Servers→***servername***→Custom Properties,** add the protocol_http_tcp_no_delay=1 property for non-secure connections or protocol_https_tcp_no_delay=1 property for secure connections.
> protocol_http_max_persist request=1

```
protocol_http_timeout_input=0
protocol_http_timeout_output=0
protocol_http_timeout_persistentSession=0
wlm_stateful_session_placement_on=1
```

8.2.2 Thread Pool Sizes

HATS applications run in the WebSphere Application Server Web Container. The Web container uses a pool of threads to communicate with the Web server. For best performance, limit the number of threads being created and destroyed by setting appropriate minimum and maximum thread sizes. The following parameters are related to the size of the thread pool and these parameters may be modified to affect the performance you will see with your HATS applications:

Minimum Thread Size: Specifies the starting size of the thread pool for the application server. Each application server has its own thread pool or cache from which it uses threads to communicate with the Web server.

Maximum Thread Size: Specifies the maximum number of concurrent transport connections between the Web server and the servlet engine. Each connection represents a request for a servlet. Recommendation: Set this value to the maximum number of concurrent users you want to support.

8.2.3 Garbage collection tuning

The JVM memory management function, or garbage collection, provides one of the biggest opportunities for improving JVM performance.

Garbage collection should normally consume from 2% to 5% of the total execution of a properly functioning application. It takes monitoring to understand the effects of garbage collection.

Concurrent mark: If you have pause time problems that are denoted by erratic application response times, which are caused by normal garbage collection, you should set gcpolicy to optavgpause.

```
-Xgcpolicy=optavgpause
```

9.0 Monitoring tools

To monitor performance refer to the following links:

Tivoli Performance Viewer (TPV) is embedded in the WebSphere Application Server Administrative Console. This tool can be used to view current activity and summary reports, or log Performance Monitoring Infrastructure (PMI) performance data.

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/tprf_tpvmonitor.html

10.0 Troubleshooting performance issues

To troubleshoot performance issues refer to the following links:

WebSphere Application Server Information Center – Topic: Troubleshooting performance problems:
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/welc6toptuning.html

Appendix A. References

1. Combined Systems Information Center.(2003, 2007) IBM
   Retrieved April 9, 2008 from
   http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/tcp_streaming_workload_tuning.htm&tocNode=int_55975

2. WebSphere Application Server v6.1 Information Center. (2005)
   Retrieved April 9, 2008 from
   http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp

3. Ipsysctl Tutorial 1.0.4. () Chapter 3. IPv4 variable reference.
   Retrieved April 9, 2008 from http://ipsysctl-tutorial.frozentux.net/chunkyhtml/variablereference.html

Appendix B. Trademark Acknowledgments