

交易风险监控系統-产品部署运维手册

1 引言

交易风险监控系統，以下简称风控系統。

1.1 系統概述

本系統为行方提供交易数据的评估，其意义在于对銀行各渠道的交易是否存在着相应的风险做保障，例如非习惯时间进行交易，短时间内多次交易等，此系統的应用，能够高效的帮助銀行业务人员发现存在不安全的交易行为。并进行及时的操作与管理。

1.2 编写目的

本操作手册是针对风控系統的安装、配置说明。手册中着重介绍风控系統各模块的部署方法。能够帮助行方科技人员清楚的认识到此系統的功能，以及每个功能的部署所注意的事项，便于后续的系統的维护与使用。

2 环境介绍

2.1 网络环境

主要以拓扑图为主，可以配文字说明。（待修改）

2.2 系統环境

以系統的运行软硬件环境及应用本身的目录结构说明，推荐的硬件配置要求：

3 安装前准备

3.1 运行环境相关

确定所提供的Linux系統的版本，与web服务器的种类与版本，例如（是Websphere或Tomcat），确定JDK的版本，编译符合JDK版本的项目部署包。确定数据库的类别，已筛选对应的数据库脚本。

以下是推荐软件及版本：

软件	版本	推荐安装模式
jdk	jdk1.8.0_xxx	每台服务器
zookeeper	zookeeper-3.4.11	3个节点
kafka	kafka_2.11-1.1.0	3个节点
aerospike	aerospike-server-community-3.15.1.3-el6	2个节点
spark	spark-2.3.0-bin-hadoop2.7	3个节点
hadoop	hadoop-2.7.5.tar.gz	3个节点
mongodb	mongodb-linux-x86_64-rhel62-3.6.6	2个节点

3.2 本系统应用

应用	描述
rrm-web	系统管理平台
rtrc-server	风控评估服务
rtrc-platform	大数据平台
rtrc-tools	运维测试人员辅助工具
zkui	zookeeper可视化管理
kafka-manager	kafka可视化管理
aerospike-amc	aerospike可视化管理

4 部署步骤

4.1 运行环境部署

本步骤用户可以根据自身情况，个性化部署。也可以直接使用自身现有环境。

以下是一个部署示例，运行环境采用8台服务器，服务分布如下：z

服务器/服务	server1	server2	server3	server4	server5	server6	server7	server8
jdk	√	√	√	√	√	√	√	√
zookeeper		√				√		√
kafka	√		√		√			
spark	√		√		√			
hadoop	√		√		√			
aerospike				√			√	
mongodb		√				√		√
rtrc-server				√			√	
rrm-web		√				√		
zkui								√

每台服务器都同一采用 `bigdata` 用户部署服务，服务安装基础目录为 `/bigdata`。将安装软件包 `install` 上传到该目录下，开始部署环境。

```
-rw-r--r-- 1 root root 7868123 5月 24 14:12 aerospike-amc-community-4.0.17-linux.tar.gz
-rw-r--r-- 1 root root 7519440 5月 24 14:12 aerospike-server-community-3.15.1.3-el6.tgz
-rw-r--r-- 1 root root 9472492 5月 24 14:12 apache-tomcat-8.5.23.tar.gz
-rw-r--r-- 1 root root 216929574 5月 24 14:13 hadoop-2.7.5.tar.gz
drwxr-xr-x 2 root root 26 5月 24 14:12 init
-rw-r--r-- 1 root root 185515842 5月 24 14:12 jdk-8u144-linux-x64.tar.gz
-rw-r--r-- 1 root root 56969154 5月 24 14:13 kafka_2.11-1.1.0.tgz
-rw-r--r-- 1 root root 119851243 5月 24 14:13 kafka-manager.tar.gz
-rw-r--r-- 1 root root 86822375 5月 24 14:13 mongodb-linux-x86_64-3.6.3.tgz
-rw-r--r-- 1 root root 205255367 5月 24 14:13 platform.zip
-rw-r--r-- 1 root root 323209636 5月 24 14:13 rrm-web.war
-rw-r--r-- 1 root root 137796237 5月 24 14:13 server.zip
-rw-r--r-- 1 root root 226128401 5月 24 14:13 spark-2.3.0-bin-hadoop2.7.tgz
-rw-r--r-- 1 root root 150955881 5月 24 14:13 tools.zip
-rw-r--r-- 1 root root 7787048 5月 24 14:13 zkui.tar.gz
-rw-r--r-- 1 root root 36668066 5月 24 14:13 zookeeper-3.4.11.tar.gz
```

4.1.1 jdk

将jdk解压到基础目录下，此时jdk目录为：`/bigdata/jdk1.8.0_144`。

修改用户环境变量：

```
vi ~/.bash_profile
```

在文件中添加java环境变量：

```
export JAVA_HOME=/bigdata/jdk1.8.0_144
export CLASSPATH=$CLASSPATH:/bigdata/jdk1.8.0_144/lib/dt.jar:/bigdata/jdk1.8.0_144/lib/tools.jar
export PATH=/bigdata/jdk1.8.0_144/bin:$PATH
```

使其生效：

```
source ~/.bash_profile
```

查看jdk配置是否生效：

```
java -version
```

```
java version "1.8.0_144"  
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)  
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
```

4.1.2 zookeeper

zookeeper采用副本模式安装，至少需要三个节点。

在基础目录下创建应用文件夹，之后的其他服务同样安装在本目录下（`/bigdata/apps`）：

```
mkdir apps
```

将zookeeper安装包解压到apps文件夹下：

```
tar -zxvf /bigdata/install/zookeeper-3.4.11.tar.gz -C /bigdata/apps/
```

进入解压后的zookeeper目录，复制zookeeper示例配置文件并重命名：

```
cd /bigdata/apps/zookeeper-3.4.11/conf  
cp zoo_sample.cfg zoo.cfg
```

修改 `zoo.cfg` 配置文件，主要注意一下几个参数：

```
tickTime=2000  
dataDir=/bigdata/apps/zookeeper-3.4.11/data  
dataLogDir=/bigdata/apps/zookeeper-3.4.11/log  
clientPort=2181  
initLimit=5  
syncLimit=2  
server.1=server8:2888:3888  
server.2=server2:2888:3888  
server.3=server6:2888:3888
```

其中，`initLimit`是每个server节点连接到leader的超时次数限制 `syncLimit`是每个server节点连接leader的过期次数限制 对于以上两个参数，`tickTime`可以指定它们的单位。比如本例中，`initLimit`的限制为每2000毫秒5次运转，即超时为10秒 `server.x`是集群配置参数。2888端口用于同辈节点之间互相连接，3888用于leader节点选举 `dataDir`是存储内存数据库快照的路径，需要创建data文件夹 `clientPort`用于监听客户端连接

在创建的数据文件夹下创建 `myid` 文件，`server8`、`server2`、`server6`的值分别为1、2、3：

```
touch myid
```

启动zookeeper服务：

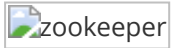
```
./bigdata/apps/zookeeper-3.4.11/bin/zkServer.sh start
```

查看是否安装成功 `jps` 存在QuorumPeerMain进程

zookeeper命令

```
zkServer.sh restart    (重启)
zkServer.sh status      (查看状态)
zkServer.sh stop        (关闭)
```

异常



报这种异常一般有三种情况:

1. zoo.cfg配置文件中, server.x:2888:3888配置出现错误
2. myid文件内容和server.x不对应, 或者myid不在data目录下
3. 系统防火墙是否在启动

4.1.3 mongodb

mongodb安装在server2、server6、server8上, 为副本集模式, 其中server2为主节点, server6为副本节点, server8为仲裁节点。将mongodb安装包解压到应用目录中, 并重命名:

```
tar -zxvf /bigdata/install/mongodb-linux-x86_64-3.6.6.tgz -C /bigdata/apps/
mv /bigdata/apps/mongodb-linux-x86_64-3.6.6/ /bigdata/apps/mongodb-3.6.6/
```

进入mongodb应用目录, 添加配置文件:

```
vi mongo.conf
```

配置内容如下:

```
systemLog:
  destination: file
  path: "/bigdata/apps/mongodb-3.6.6/log/mongod.log"
  logAppend: true
storage:
  dbPath: "/bigdata/apps/mongodb-3.6.6/data/"
  journal:
    enabled: true
processManagement:
  fork: true
  pidFilePath: "/bigdata/apps/mongodb-3.6.6/pid"
net:
  bindIp: 127.0.0.1,server2
  port: 27017
setParameter:
  enableLocalhostAuthBypass: false
replication:
  replSetName: "rtrc_set"
```

配置参考: <https://docs.mongodb.com/manual/reference/configuration-options/>

其中，三台机器的bindIp中的服务器名称对应各台服务器主机名，其余一样。

启动mongodb服务：

启动之前需要在安装目录下创建log和data文件夹

```
/bigdata/apps/mongodb-3.6.6/bin/mongod -f /bigdata/apps/mongodb-3.6.6/mongo.conf
```

启动客户端，配置副本集和用户：

```
./bigdata/apps/mongodb-3.6.6/bin/mongo
```

副本集初始化（只在一个节点上初始化，本例是主节点server2）：

```
rs.initiate()
```

将server6加入到副本集中：

```
rs.add("server6:27017")
```

将仲裁节点server8加入到副本集中：

```
rs.addArb("server8:27017")
```

查看集群状态：

```
rs.status()
```

添加admin用户

```
use admin
db.createUser(
  {
    user: "admin",
    pwd: "111111",
    roles: [ "root", "dbAdminAnyDatabase" ]
  }
)
```

mongodb角色参考：<https://docs.mongodb.com/manual/reference/method/db.createUser/>

验证安装：

```
# bin/mongo -u admin -p 111111 --host 主机地址 --port 端口号 admin
...
执行命令后最终出现提示符：
rtrc_set:PRIMARY> show dbs
admin      0.000GB
config     0.000GB
local      1.582GB
rtrc_set:PRIMARY>
```

停止mongodb:

注意：不要用kill -9 ,在bin下执行命令终止mongod：

```
./mongod -shutdown -dbpath=../data
kill <mongod process ID>
kill -2 <mongod process ID>
```

mongo shell 使用参考：<https://docs.mongodb.com/v3.6/mongo/>

4.1.4 kafka

kafka安装在server1、server3、server5上，将kafka安装包解压到应用目录中：

```
tar -zxvf /bigdata/install/kafka_2.11-1.1.0.tgz -C /bigdata/apps/
```

进入kafka应用目录，修改配置文件：

```
vi conf/server.properties
```

主要配置参数如下：

```
# 集群broker的id，每个节点需要不同的值，本例中分别为0/1/2
broker.id=0
# 监听端口，每个节点需要不同的值，hostname:9092
listeners=PLAINTEXT://server1:9092
# 消息文件存储路径（区别于logs）
log.dirs=/bigdata/apps/kafka_2.11-1.1.0/log
# 默认topic分区数
num.partitions=16
# 默认topic副本数
offsets.topic.replication.factor=1
# zookeeper连接
zookeeper.connect=server2:2181,server6:2181,server8:2181
```

静默启动kafka服务：

```
./bigdata/apps/kafka_2.11-1.1.0/bin/kafka-server-start.sh -daemon /bigdata/apps/kafka_2.11-1.1.0/config/server.properties
```

出现异常【java.lang.OutOfMemoryError: unable to create new native thread】切换为root用户启动成功，原因：linux对每个用户都限制了最大进程数（包括线程）ulimit -a 查看当前的各种用户进程限制

```
vi /etc/profile
```

```
ulimit -u 10000  
ulimit -n 4096  
ulimit -d unlimited  
ulimit -m unlimited  
ulimit -s unlimited  
ulimit -t unlimited  
ulimit -v unlimited
```

```
source /etc/profile
```

参考：<https://www.ktanx.com/blog/p/3989> 查看是否启动成功 jps 出现Kafka进程

4.1.5 spark

spark安装在server1、server3、server5上，将spark应用解压到apps中：

```
tar -zxvf /bigdata/install/spark-2.3.0-bin-hadoop2.7.tgz -C /bigdata/apps/
```

进入spark目录，复制配置模板文件并修改：

```
cp spark-env.sh.template spark-env.sh  
vi spark-env.sh
```

配置文件中添加：

```
export JAVA_HOME=/bigdata/jdk1.8.0_144  
export SPARK_MASTER_HOST=server1
```

配置参数参考：<https://spark.apache.org/docs/2.2.1/spark-standalone.html>

配置集群节点：

```
cp slaves.template slaves
```

文件中添加：

```
server1  
server3  
server5
```

启动spark服务：

```
./bigdata/apps/spark-2.3.0-bin-hadoop2.7/sbin/start-all.sh
```

查看是否部署成功，通过浏览器访问<http://server1:8080/>

4.1.6 hadoop

hadoop安装在server1、server3、server5上，安装模式为yarn，各个节点之间需要可以免密登录。将hadoop安装包解压到应用目录中：

```
tar -zxvf /bigdata/install/hadoop-2.7.5.tar.gz -C /bigdata/apps/
```

进入hadoop应用目录，修改相关配置文件：

```
cd etc/hadoop/  
vi slaves
```

文件内容修改如下：

```
server1  
server3  
server5
```

```
vi hadoop-env.sh
```

```
export JAVA_HOME=/bigdata/jdk1.8.0_144
```

```
vi core-site.xml
```

```
<configuration>  
  <property>  
    <name>hadoop.tmp.dir</name>  
    <value>file:/bigdata/apps/hadoop-2.7.5/tmp</value>  
    <description>Abase for other temporary directories.</description>  
  </property>  
  <property>  
    <name>fs.defaultFS</name>  
    <value>hdfs://localhost:8020</value>  
  </property>  
</configuration>
```

默认端口9000会出现【java.net.UnknownHostException: test62: test62: Name or service not known to vm1:9000 failed on connection exception: java.net.ConnectException: Connection refused; For more details see: <http://wiki.apache.org/hadoop/ConnectionRefused>】异常 补充：8020也会 解决：core-site.xml的ip由localhost改为server1等本机ip名

```
vi hdfs-site.xml
```

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/bigdata/apps/hadoop-2.7.5/tmp/dfs/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/bigdata/apps/hadoop-2.7.5/tmp/dfs/data</value>
  </property>
</configuration>
```

```
cp mapred-site.xml.template mapred-site.xml
```

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

```
vi yarn-site.xml
```

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

NameNode初始化：

```
./bin/hdfs namenode -format
```

启动文件系统：

```
./sbin/start-dfs.sh
```

浏览器访问<http://vm1:50070>

启动YARN：

```
./sbin/start-yarn.sh
```

浏览器访问<http://vm1:8088>

停止命令：

```
./sbin/stop-yarn.sh  
./sbin/stop-dfs.sh
```

参考：<http://www.powerxing.com/install-hadoop-in-centos/>

4.1.7 aerospike

aerospike安装在server4、server7上，将aerospike安装包解压到应用目录中：

```
tar -zxvf /bigdata/install/aerospike-server-community-3.15.1.3-el6.tgz -C /bigdata/apps/
```

进入aerospike应用目录，安装初始化，默认配置文件为 `/etc/aerospike/aerospike.conf`，请参照6附录aerospike配置修改，在bigdata下创建data-storage目录：

```
mkdir data-storage  
sudo ./asinstall
```

启动aerospike服务：

```
sudo service aerospike start
```

查看是服务状态：

```
sudo service aerospike status
```

安装aerospike工具：

```
sudo ./aerospike-tools-deps/install.sh
```

amc安装 <https://www.aerospike.com/docs/amc/install/linux/any/index.html>

```
sudo rpm -ivh aerospike-amc-<version>.rpm
```

4.2 应用部署

4.2.1 数据初始化

本系统数据初始化主要分为配置初始化和数据库初始化。系统相关参数主要配置在zookeeper中，为了便于维护，采用zkui管理。数据库初始化主要是oracle数据库中的系统表初始化。

4.2.1.1 配置初始化

安装部署zkui，将安装包解压到应用目录中：

```
tar -zxvf /bigdata/install/zkui.tar.gz -C /bigdata/apps/
```

进入zkui文件夹，修改配置文件

```
cd zkui
vi config.cfg
```

主要配置参数：

```
serverPort=9090
# zookeeper集群
zkServer=server8:2181,server2:2181,server6:2181
# 用户名
jdbcUser=admin
# 密码
jdbcPwd=111111
```

启动zkui：

```
nohup java -jar zkui-2.0-SNAPSHOT-jar-with-dependencies.jar &
```

验证是否启动成功，通过浏览器访问<http://vm8:9090/login>

导入应用配置并修改为本环境相关参数

4.2.1.2 数据库初始化

登录到oracle数据库：

```
sqlplus / as sysdba
```

导入初始化数据：

```
@init.sql
```

4.2.1.3 其他初始化

hadoop文件系统初始化，在hadoop应用目录下执行以下命令：

```
bin/hadoop fs -rm -r -f /zzcx/realTimeRisk
bin/hadoop fs -rm -r -f /zzcx/hourTxn
bin/hadoop fs -rm -r -f /zzcx/dayTxn
bin/hadoop fs -rm -r -f /zzcx/hourWarningTxn
bin/hadoop fs -rm -r -f /zzcx/dayWarningTxn
bin/hadoop fs -rm -r -f /zzcx/hourRule
bin/hadoop fs -rm -r -f /zzcx/dayRule

bin/hadoop fs -mkdir -p /zzcx/realTimeRisk
bin/hadoop fs -mkdir -p /zzcx/hourTxn
bin/hadoop fs -mkdir -p /zzcx/dayTxn
bin/hadoop fs -mkdir -p /zzcx/hourWarningTxn
bin/hadoop fs -mkdir -p /zzcx/dayWarningTxn
bin/hadoop fs -mkdir -p /zzcx/hourRule
bin/hadoop fs -mkdir -p /zzcx/dayRule
```

```
bin/hadoop fs -chmod -R 777 /zzcx/realTimeRisk
bin/hadoop fs -chmod -R 777 /zzcx/hourTxn
bin/hadoop fs -chmod -R 777 /zzcx/dayTxn
bin/hadoop fs -chmod -R 777 /zzcx/hourWarningTxn
bin/hadoop fs -chmod -R 777 /zzcx/dayWarningTxn
bin/hadoop fs -chmod -R 777 /zzcx/hourRule
bin/hadoop fs -chmod -R 777 /zzcx/dayRule
```

aerospike创建名称空间和导入数据：

在aerospike应用目录下，添加rtrc namespace：

```
sudo service aerospike stop
sudo vi /etc/aerospike/aerospike.conf
```

添加以下内容：

```
namespace rtrc {
    replication-factor 2
    memory-size 4G
    default-ttl 30d # 30 days, use 0 to never expire/evict.
    storage-engine memory
}
```

重启aerospike：

```
sudo service aerospike restart
sudo service amc start
```

4.2.2 rrm-web

本示例中管理系统部署在tomcat中，用户可以根据自身情况选择其他web容器。

安装部署tomcat，将安装包解压到应用目录中：

```
tar -zxvf /bigdata/install/apache-tomcat-8.5.23.tar.gz -C /bigdata/apps/
```

将rrm-web包复制到tomcat中：

```
cp /bigdata/install/rrm-web.war /bigdata/apps/apache-tomcat-8.5.23/webapps
```

启动tomcat：

```
./bigdata/apps/apache-tomcat-8.5.23/bin/startup.sh
```

通过浏览器访问<http://server8:8080/rrm-web>，验证是否安装成功。初始用户名及密码：admin/111111

4.2.3 rtrc-server

在apps目录下创建deployments，用于发布系统相关服务：

```
mkdir deployments
cd deployments
```

将 `server.zip` 解压到应用目录下：

```
unzip /bigdata/install/server.zip -d /bigdata/apps/deployments
```

server目录文件列表如下，其中 `logs` 为日志目录，`start.sh` 为启动脚本。

```
drwxrwxr-x. 2 bigdata bigdata 4096 May 14 09:30 logs
-rw-r--r--. 1 bigdata bigdata 170866698 May 17 14:11 rtrc-server.jar
-rwxr-xr-x. 1 bigdata bigdata 56 May 7 17:37 start.sh
```

进入server目录，执行启动命令

```
./start.sh
```

4.2.4 rtrc-platform

rtrc-platform是一个基础spark的大数据流处理平台，内置四个流应用。

将 `platform.zip` 解压到应用目录下：

```
unzip /bigdata/install/platform.zip -d /bigdata/apps/deployments
```

platform目录文件列表如下，其中 `lib` 目录为流处理平台依赖包，`start.sh` 为启动脚本。

```
drwxr-xr-x 2 bigdata bigdata 12288 May 10 15:31 lib
-rw-r--r-- 1 bigdata bigdata 48614 May 17 14:11 rtrc-platform.jar
-rwxrwxr-x 1 bigdata bigdata 1928 May 10 14:08 startup.sh
```

进入platform目录，修改启动脚本，主要是以下两个参数：

```
# spark安装目录
SPARK_HOME=/bigdata/apps/spark-2.3.0-bin-hadoop2.7
# spark master
master=spark://server1:6066
```

启动流处理平台：

```
./start.sh
```

通过浏览器访问，查看流应用状态。

附：startup.sh脚本调优

startup.sh脚本负责向spark提交四种应用：

#数据落地及离线评估流应用

```
/bigdata/apps/spark-2.3.0-bin-hadoop2.7/bin/spark-submit --class
cn.com.zzcx.rtrc.RunningSparkStreamingApp --master spark://server1:6066 --deploy-mode cluster --
conf "spark.driver.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --conf
"spark.executor.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --conf
"spark.streaming.kafka.maxRatePerPartition=1000" --executor-memory 1G --total-executor-cores 4
/home/bigdata/apps/deployments/platform/rtrc-platform.jar
```

#统计计算流应用

```
/bigdata/apps/spark-2.3.0-bin-hadoop2.7/bin/spark-submit --class
cn.com.zzcx.rtrc.RunningStatCalculateStreamingApp --master spark://server1:6066 --deploy-mode
cluster --conf "spark.driver.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --conf
"spark.executor.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --conf
"spark.streaming.kafka.maxRatePerPartition=1000" --executor-memory 2G --total-executor-cores 16
/home/bigdata/apps/deployments/platform/rtrc-platform.jar
```

#警告数据处理流应用

```
/bigdata/apps/spark-2.3.0-bin-hadoop2.7/bin/spark-submit --class
cn.com.zzcx.rtrc.RunningWarningStreamingApp --master spark://server1:6066 --deploy-mode cluster -
-conf "spark.driver.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --conf
"spark.executor.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --conf
"spark.streaming.kafka.maxRatePerPartition=1000" --executor-memory 1G --total-executor-cores 2
/bigdata/apps/deployments/platform/rtrc-platform.jar
```

#报表数据汇总应用

```
/bigdata/apps/spark-2.3.0-bin-hadoop2.7/bin/spark-submit --class
cn.com.zzcx.rtrc.RunningStructuredStreamingApp --master spark://server1:6066 --deploy-mode
cluster --conf "spark.driver.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --conf
"spark.executor.extraClassPath=/bigdata/apps/deployments/platform/lib/*" --driver-memory 2G --
executor-memory 2G --total-executor-cores 2 /bigdata/apps/deployments/platform/rtrc-platform.jar
```

以上脚本参数中：

- `spark.streaming.kafka.maxRatePerPartition=n`：每个 `kafka partition` 的最大读取速率,此值的设置是为了防止kafka topic中数据长时间未消费后，首次启动流处理首批数据过多的问题，亦限制了以后流处理的最大速度。在每个kafka partition下都有足量数据的情况下： $\text{首批数量} = (\text{partitionNum} * n) * \text{批次时间}$
- `--total-executor-cores`：所有executor核数之和，影响任务并行度。在资源充足的情况下，越高的并行度意味着越高的吞吐量。

spark集群总的 $\text{可分配核数} = \text{worker数量} * \text{服务器cpu核数}$ ，在硬件资源充足的情况下，每台spark服务器可启动多个worker，故 $\text{worker数量} = \text{slaveNum} * \text{SPARK_WORKER_INSTANCES}$ (spark-env.sh中配置项)，在可分配核数充足的情况下，`total-executor-cores`值可设为 `kafka partitionNum` (应用读取的topic各自partition数量之和) 的一半或与之相同。

尤其“统计计算流应用”对实时性要求较高，应尽量提高其并行度，防止批次延时。

测试环境：

```
cpu : Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz 32vcpu
memory : 64G
spark服务器 : 2台
SPARK_WORKER_INSTANCES : 2
topic transaction:64 partition
统计计算流应用topic处理速度 :
total-executor-cores : 32
8000tps
total-executor-cores : 64
16000tps
```

- --driver-memory : driver内存大小
- -executor-memory : 每个executor内存大小

spark环境总的可用内存由 `worker数量*SPARK_WORKER_MEMORY`(spark-env.sh中配置项) 决定, 单应用消耗内存为: `driver-memory+executor-num*executor-memory`, 其中在spark的standalone模式下 `executor-num=max(total-executor-cores,worker数量)`, 由此可估算出应用所需内存, 以便恰当的进行内存分配。

四应用中 报表数据汇总应用 所需内存较高, 故尽量保证其driver-memory与executor-memory不小于2g, 可通过gc时间占比衡量内存是否足够。

4.2.5 zkui(可选)

将zkui安装包解压到apps文件夹下：

```
tar -zxvf /bigdata/install/zkui.tar.gz -C /bigdata/apps/
```

进入zkui文件夹, 修改配置文件 vi config.cfg

主要配置参数：

```
# 若与rtrc-tools部署在同一台服务器, 修改端口为9091, 避免端口冲突
serverPort=9090
zkServer=vm8:2181,vm2:2181,vm6:2181
jdbcUser=admin
jdbcPwd=111111
```

启动命令

```
nohup java -jar zkui-2.0-SNAPSHOT-jar-with-dependencies.jar &
```

验证是否启动成功, 通过浏览器访问<http://vm8:9090/login>

4.2.6 kafka-manager(可选)

将kafka-manager安装包解压到apps文件夹下：

```
tar -zxvf /bigdata/install/kafka-manager.tar.gz -C /bigdata/apps/
```

进入kafka-manager目录, 修改配置文件 vi conf/application.conf

```
kafka-manager.zkhosts="vm2:2181,vm6:2181,vm8:2181"
basicAuthentication.enabled=false
basicAuthentication.username="admin"
basicAuthentication.password="111111"
```

启动命令

```
nohup ./kafka-manager -Dconfig.file=../conf/application.conf >/dev/null 2>&1 &

已写入脚本 start.sh
./start.sh

-Dhttp.port=7778 可以用于指定端口 默认为9000
```

通过浏览器访问<http://vm8:7778/>，添加kafka集群配置（可能有旧配置未删除）

4.2.7 aerospike-amc(可选)

将aerospike-amc安装包移动到apps文件夹下：

```
mv /bigdata/install/aerospike-amc-community-4.0.19-1.x86_64.rpm /bigdata/apps/
```

安装

```
sudo rpm -ivh aerospike-amc-community-4.0.19-1.x86_64.rpm
```

启动命令

```
sudo service amc start
默认端口为8081
```

通过浏览器访问<http://vm4:8081/>

5 端口配置

5.1 应用端口

从	至	默认	作用	Notes
Browser	rrm-web	8080	Web UI	
Browser	rtrc-tools	9090	Web UI	
Browser	zkui	8090	Web UI	
Browser	kafka-manager	9000	Web UI	
Browser	rtrc-schedule	8808	Web UI	
Browser	amc	8099	Web UI	

5.2 环境端口

环境	从	至	默认	作用	Notes
MongoDB	Client	Server	27017	连接MongoDB数据库	
Spark	Browser	Standalone Master	8080	Web UI	
Spark	Browser	Standalone Worker	8081	Web UI	
Spark	Browser	Application	4040	Web UI	
Spark	Driver / Standalone Worker	Standalone Master	7077	Submit job to cluster / Join cluster	
Zookeeper	Client	Server	2181	连接Zookeeper服务端	
Kafka	Client	Server	9092	连接Kafka服务端	
Aerospike	Client	Server	3000	连接Aerospike服务端	
Hadoop	Client	Server	8020	连接Hadoop服务端	

6 附录

6.1 Aerospike配置

- Aerospike安装后默认配置文件在/etc/aerospike/aerospike.conf。
- 修改/etc/aerospike/telemetry.conf。

`disable = true` #原来是false，如果是true,AS服务器会定期往aerospike的官网发送一些数据

关闭集群自动发现节点功能，直接列出所有AS节点：

```

network {
    service {
        address any
        port 3000
    }
    heartbeat {
        mode mesh
        port 3002 # Heartbeat port for this node.

        #列出集群中所有节点,每一行 [ IP ] [ 端口 ] 的形式列出:
        mesh-seed-address-port server2 3002
        mesh-seed-address-port server3 3002
        interval 250
        timeout 10
    }
}

...

```

缓存集群中的数据库配置：

风控系统的缓存数据可以根据配置分布在两个数据库中：全部缓存到内存（data-in-memory:true）和只把索引存入内存(data-in-memory:false)。

data-in-memory:所有数据和索引都存在内存，需要预先估算内存大小，估算公式：

```

内存空间 = ( 64+每条数据字节数 ) * 估计数据记录数 / 0.7
硬盘空间 = 内存空间

```

data-in-memory:只存索引数据，内存估算方法：

```

内存需要量 = 64 * 估计数据记录数 / 0.7
硬盘空间 = 每条记录字节数 * 估计数据记录数 / 0.7

```

系统初始化时建议建立两个库：

```

namespace rtrc {
    #数据库的名字
    replication-factor 2    #数据复制个数，生产环境集群中一般大于等于2
    high-water-memory-pct 80 #分配的内存使用率
    high-water-disk-pct 90  #分配的硬盘空间使用率
    stop-writes-pct 80     #停止写入
    memory-size 8G         #分配的内存
    default-ttl 0d         #默认生命长度，例如30d表示30天，0 表示永不会被驱逐
    #storage-engine memory  #纯粹用内存做缓存，关机丢数
    storage-engine device { #存储介质配置，关机不丢数
        file /home/bigdata/data/aerospike_rtrc.data #这个文件存储数据
        filesize 32G #文件分配的大小，每个上限2T，可配置多个文件
        #数据默认加载到内存中吗？设置为true的时候，数据会全部加载到内存中
        data-in-memory true
    }
}

```

```

namespace rtrcdisk {
    replication-factor 1      #数据库的名字
    high-water-memory-pct 80  #数据复制个数，生产环境集群中一般大于等于2
    high-water-disk-pct 90    #分配的内存使用率
    stop-writes-pct 80       #分配的硬盘空间使用率
    memory-size 4G           #停止写入
    default-ttl 0            #分配的内存
    storage-engine device {   #默认生命长度，例如30d表示30天，0 表示永不会被驱逐
        file /home/bigdata/data/aerospike_rtrc.dat  #存储介质配置，可以分配多个文件，每个
        filesize 100G      #这个文件存储数据
        data-in-memory false #文件分配的大小，每个上限2T
    }
}

```

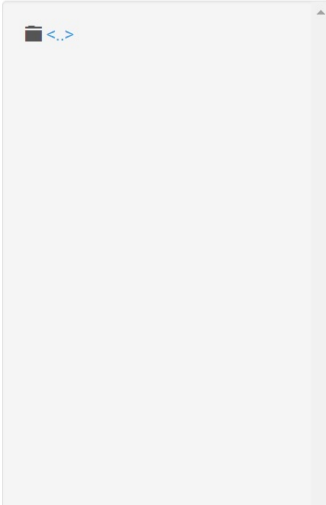
默认情况下：统计和名单类数据以及数据量较小的数据存在于内存中。

签约数据和其它占用存储较高的数据存储在硬盘中。

可以根据用户需求和硬件资源以及实际场景调整缓存策略，配置方法：

zookeeper的节点：/rtrc/static/cache/ 如图：

[/ rtrc / static / cache](#)



Name	Value
rtrc.svc.cache.aerospike.host	192.168.1.222
rtrc.svc.cache.aerospike.maxConcurrentThreads	1
rtrc.svc.cache.aerospike.namespace	rtrc
rtrc.svc.cache.aerospike.namespace2	rtrcdisk
rtrc.svc.cache.aerospike.port	3000
rtrc.svc.cache.aerospike.providerClass	cn.com.zzcx.rtrc.common.cache.provider.AerospikeCacheProvider
rtrc.svc.cache.aerospike.setsInDisk	useraccount,usercredit

其中rtrc.svc.cache.aerospike.namespace的value配置为内存数据缓存的数据库名，

rtrc.svc.cache.aerospike.namespace2的value配置为硬盘数据缓存的数据库名。

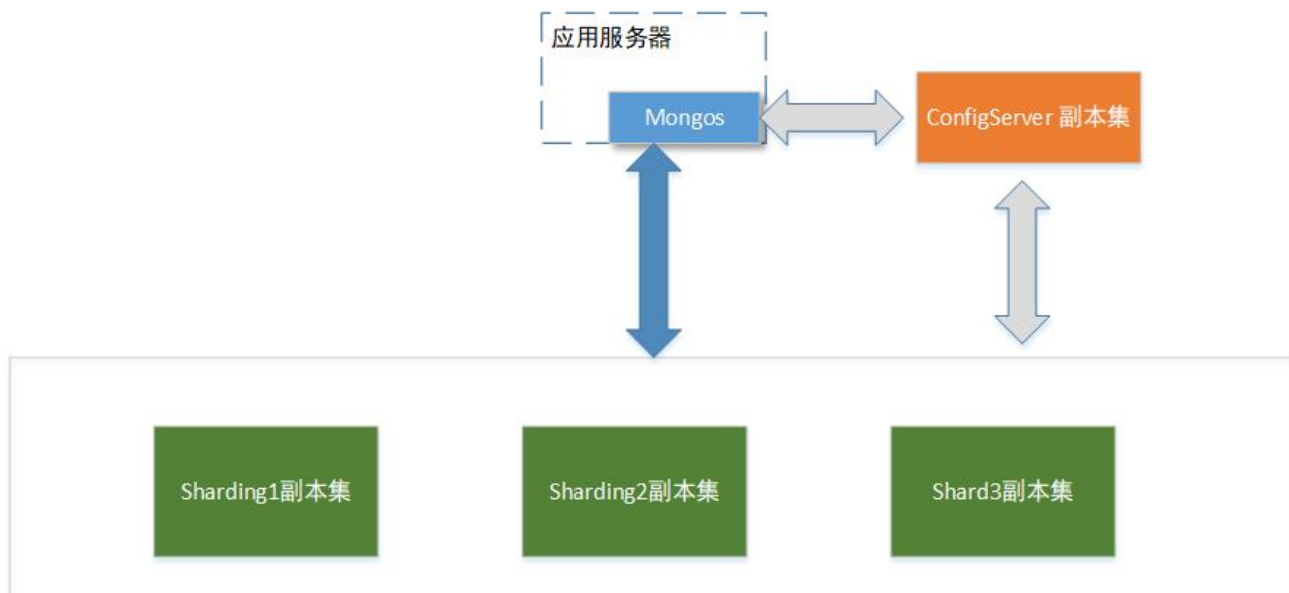
rtrc.svc.cache.aerospike.setsInDisk的value配置的是需要在硬盘数据库中存储的sets名。

6.2 mongodb sharding集群部署

参考资料：

<https://docs.mongodb.com/v3.6/tutorial/deploy-shard-cluster/>

1.规划集群



- 集群中至少要有有一个Mongos节点，用作路由，可以运行于应用服务器上，可以独立运行多个mongos,这些mongos之间不需要通讯。
- 集群中至少需要一个ConfigServer副本集和一个Sharding副本集，根据数据量可以部署多个Sharding副本集。
- 每个副本集分别至少由三个节点构成，ConfigServer中不允许包含arbiter节点，Sharding副本集可以包含arbiter节点。

每个节点上拷贝一份mongodb的发布版本。

假定mongodb目录在/ssddata/mongo下。

2.生成keyfile

```
# openssl rand -base64 756 > keyfile
# chmod 400 keyfile
```

把keyfile分发到Mongodb集群中每一个节点上，确保运行mongod和mongos的用户是这个文件的拥有者并且可以访问到这个文件。

3.部署config server副本集

每个config server的mongodb配置文件：

文件名：confsrv.conf

```
security:
#keyfile文件路径
keyFile: "/ssddata/mongo/keyfile"
```

```
sharding:
  clusterRole: configsvr
replication:
  replSetName: rscfg
storage:
#数据库文件路径
  dbPath: "/ssddata/mongo/cfg1/data"
#  wiredTiger:
#    engineConfig:
#      cacheSizeGB: 1
net:
# 绑定的主机名或者IP,端口
  bindIp: localhost,server3
  port: 27017
systemLog:
  destination: file
# 日志路径
  path: "/ssddata/mongo/cfg1/log/mongo.log"
  logAppend: true
processManagement:
  fork: true
#pid文件位置
  pidFilePath: "/ssddata/mongo/cfg1/pid"
```

分别启动每个节点：

```
# mongod --config confsrv.conf
```

连接到其中一个节点：

```
# mongo --host localhost --port 27017
```

在mongo的提示符下创建副本集：

```
rs.initiate(
{
  _id: "rscfg",
  configsvr: true,
  members: [
    { _id : 0, host : "server1:20001" },
    { _id : 1, host : "server2:20002" },
    { _id : 2, host : "server3:20003" }
  ]
}
)
```

在mongo的提示符下创建用户：

```
admin = db.getSiblingDB("admin")
admin.createUser(
{
  user: "admin",
  pwd: "111111",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
}
)
```

在mongo中验证：

```
db.getSiblingDb("admin").auth("admin","111111")
```

4.部署sharding副本集

sharding的配置文件shard.conf：

```
security:
  keyFile: "/ssddata/mongo/keyfile"
sharding:
  clusterRole: shardsvr
replication:
  replSetName: rs-sharding1
storage:
  dbPath: "/ssddata/mongo/shard11/data"
#   wiredTiger:
#     engineConfig:
#       cacheSizeGB: 1
net:
  bindIp: localhost,server3
  port: 27017
systemLog:
  destination: file
  path: "/ssddata/mongo/shard11/log/mongo.log"
  logAppend: true
processManagement:
  fork: true
  pidFilePath: "/ssddata/mongo/shard11/pid"
```

三个节点分别启动mongod:

```
mongod -f shard11.conf
```

用mongo连接其中一个节点：

```
mongo --host localhost --port 27017
```

初始化副本集(其中的host替换为具体的主机IP)：

如果不需要arbiter，那么去掉arbiterOnly属性即可。

```
rs.initiate(
{
  _id : "rs-sharding1",
  members: [
    { _id : 0, host : "server3:21001" },
    { _id : 1, host : "server3:21002" },
    { _id : 2, host : "server3:21003" ,arbiterOnly:true}
  ]
}
)
```

5.部署Mongos并配置config server

在mongos的节点上：

mongos.conf

```
sharding:
  configDB: rscfg/server3:20001,server3:20002,server3:20003
net:
  bindIp: localhost,server3
  port: 27017
systemLog:
  destination: file
  path: "/ssddata/mongo/mongos/log/mongo.log"
  logAppend: true
processManagement:
  fork: true
  pidFilePath: "/ssddata/mongo/mongos/pid"
```

其中，configDB属性代表config server的三给节点的IP和端口，“rscfg”是config server副本集的名称。

启动mongos:

```
mongos -f mongos.conf
```

用mongo client连接mongos

```
mongo --host mongos-host-name --port xxxx
```

创建集群admin用户：

```
admin = db.getSiblingDB("admin")
admin.createUser(
{
  user: "admin",
  pwd: "111111",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
}
)
```

创建集群管理员：

```
db.getSiblingDB("admin").createUser(
{
  "user" : "ravi",
  "pwd" : "changeme2",
  roles: [ { "role" : "clusterAdmin", "db" : "admin" } ]
}
)
```

集群中的管理员角色详细说明参见：

<https://docs.mongodb.com/v3.6/reference/built-in-roles/#cluster-admin-roles>

6. 添加sharding

用mongo client连接mongos:

```
...
```

```
sh.addShard( "rs-sharding1/server3:21001")
```

```
...
```

"/"前的"rs-sharding1"是分片副本集的名称，"/"后是分片服务器的节点之一。

如果有多个sharding集群，或者日后随时增加sharding的时候，可以通过以上命令增加分片

7. 开启数据库的分片功能

从mongo shell连接到mongos,用sh.enableSharding()命令开启目标数据库的分片能力，开启分片能力以后就可以把数据库中的集合进行分片了。

```
sh.enableSharding("rtrc")
```

8. 对集合进行分片

对集合分片分为Hashed Sharding和Ranged Sharding，具体文档参见：

<https://docs.mongodb.com/v3.6/core/sharding-shard-key/>

下面以Hashed Key为例：

从mongo shell连接到mongos,如果当前的集合没有记录，那么可以直接用以下命令对集合分片：

```
sh.shardCollection("rtrc.online_trafficdata", { "trans_code" : "hashed" } )
```

如果集合中已经有数据，那么需要在shard key上先建立一个hashed index，参见：

<https://docs.mongodb.com/v3.6/core/index-hashed/#index-type-hashed>

x.关于用户

整个sharding集群建立的用户，也就是通过连接mongos建立的用户，可以用于集群数据库管理。

每个sharding可以建立自己的本地用户，用于直连这个sharding，但是这个用户不能通过mongos连接，只能用于sharding。