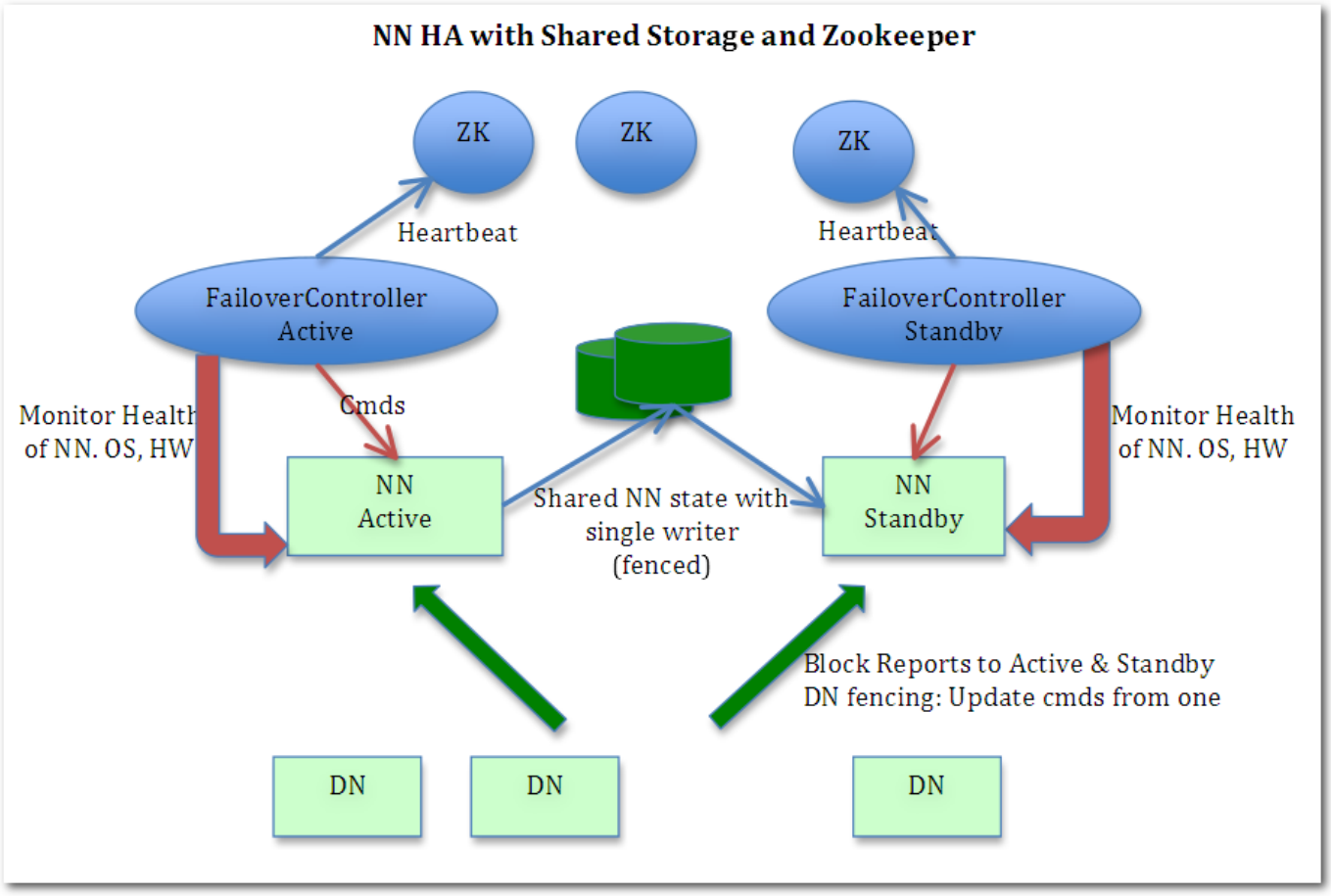
Hadoop高可用部署

一、背景知识

1.HDFS实现HA

JournalNode

可能有些同学没有听说过JournalNode,只听过Hadoop的Datanode,Namenode,因为这个概念是在MR2也就是Yarn中新加的,journalNode的作用是存放EditLog的,在MR1中editlog是和fsimage存放在一起的然后SecondNamenode做定期合并,Yarn在这上面就不用SecondNamanode了.下面是目前的Yarn的架构图,重点关注一下JournalNode的角色.



上面在Active Namenode与StandBy Namenode之间的绿色区域就是JournalNode,当然数量不一定只有1个,作用相当于NFS共享文件系统.Active Namenode往里写editlog数据,StandBy再从里面读取数据进行同步.

NameNode之间共享数据（NFS 、Quorum Journal Node（用得多））

两个NameNode为了数据同步，会通过一组称作JournalNodes的独立进程进行相互通信。当active状态的NameNode的命名空间有任何修改时，会告知大部分的JournalNodes进程。standby状态的NameNode有能力读取JNs中的变更信息，并且一直监控edit log的变化，把变化应用于自己的命名空间。standby可以确保在集群出错时，命名空间状态已经完全同步了。

Hadoop中的NameNode好比是人的心脏，非常重要，绝对不可以停止工作。在hadoop1时代，只有一个NameNode。如果该NameNode数据丢失或者不能工作，那么整个集群就不能恢复了。这是hadoop1中的单点问题，也是hadoop1不可靠的表现，如图1所示。hadoop2就解决了这个问题。

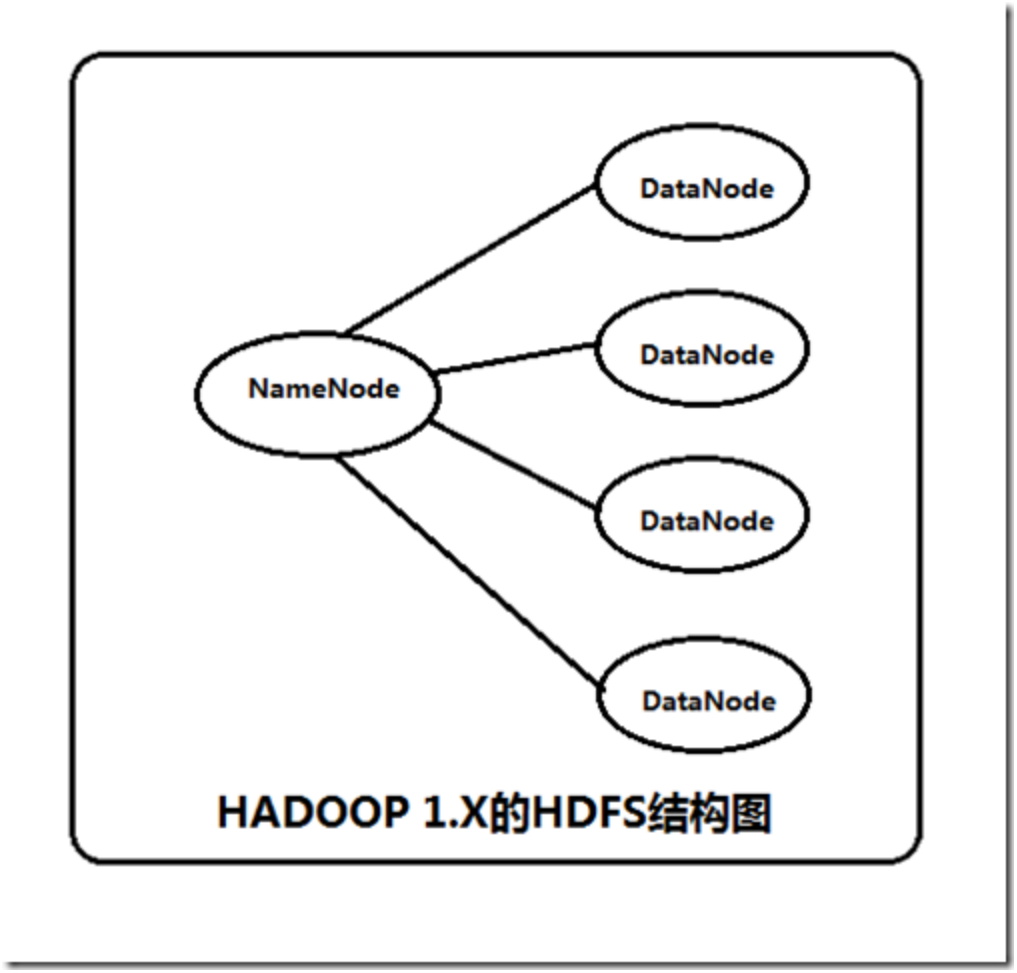


图1

hadoop2.2.0（HA）中HDFS的高可靠指的是可以同时启动2个NameNode。其中一个处于工作状态，另一个处于随时待命状态。这样，当一个NameNode所在的服务器宕机时，可以在数据不丢失的情况下，手工或者自动切换到另一个NameNode提供服务。

这些NameNode之间通过共享数据，保证数据的状态一致。多个NameNode之间共享数据，可以通过Nnetwork File System或者Quorum Journal Node。前者是通过linux共享的文件系统，属于操作系统的配置；后者是hadoop自身的东西，属于软件的配置。

我们这里讲述使用Quorum Journal Node的配置方式，方式是手工切换。

集群启动时，可以同时启动2个NameNode。这些NameNode只有一个是active的，另一个属于standby状态。active状态意味着提供服务，standby状态意味着处于休眠状态，只进行数据同步，时刻准备着提供服务，如图2所示。

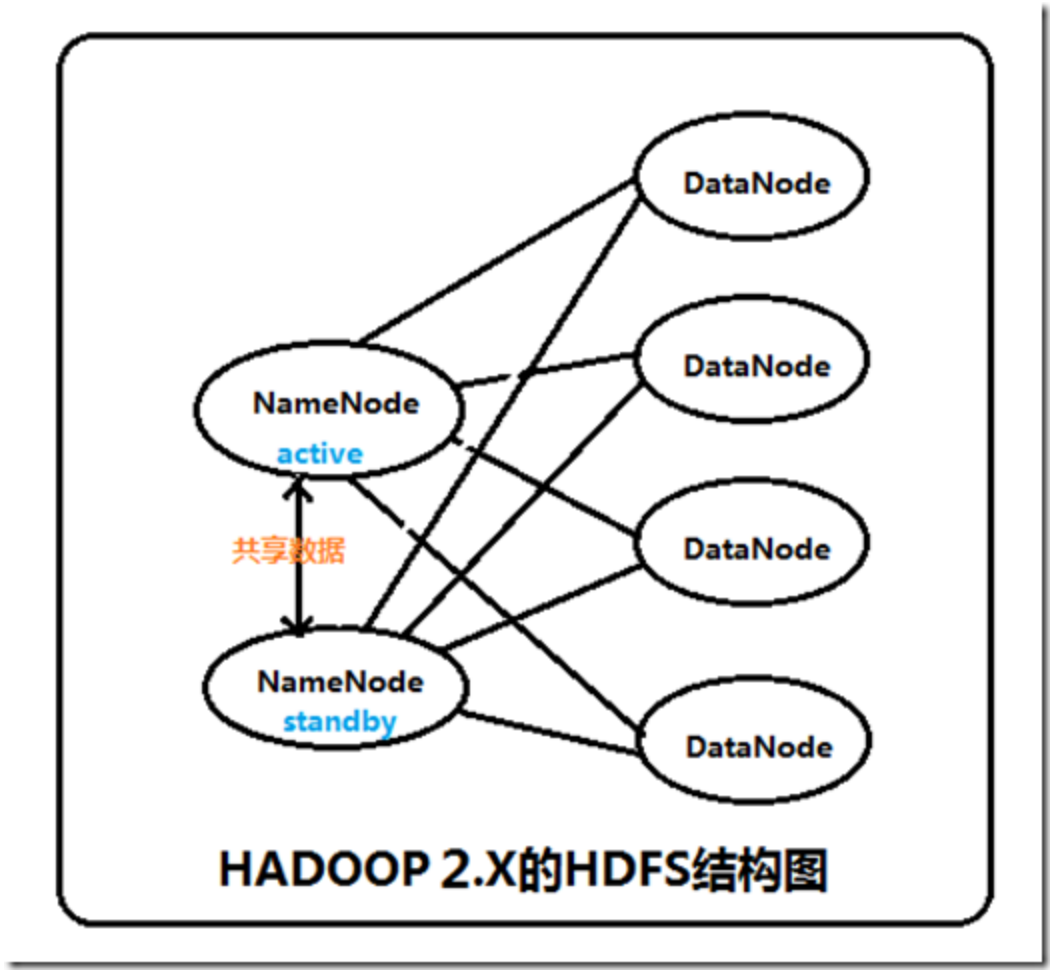


图2

架构

在一个典型的HA集群中，每个NameNode是一台独立的服务器。在任一时刻，只有一个NameNode处于active状态，另一个处于standby状态。其中，active状态的NameNode负责所有的客户端操作，standby状态的NameNode处于从属地位，维护着数据状态，随时准备切换。

两个NameNode为了数据同步，会通过一组称作JournalNodes的独立进程进行相互通信。当active状态的NameNode的命名空间有任何修改时，会告知大部分的JournalNodes进程。standby状态的NameNode有能力读取JNs中的变更信息，并且一直监控edit log的变化，把变化应用于自己的命名空间。standby可以确保在集群出错时，命名空间状态已经完全同步了，如图3所示。

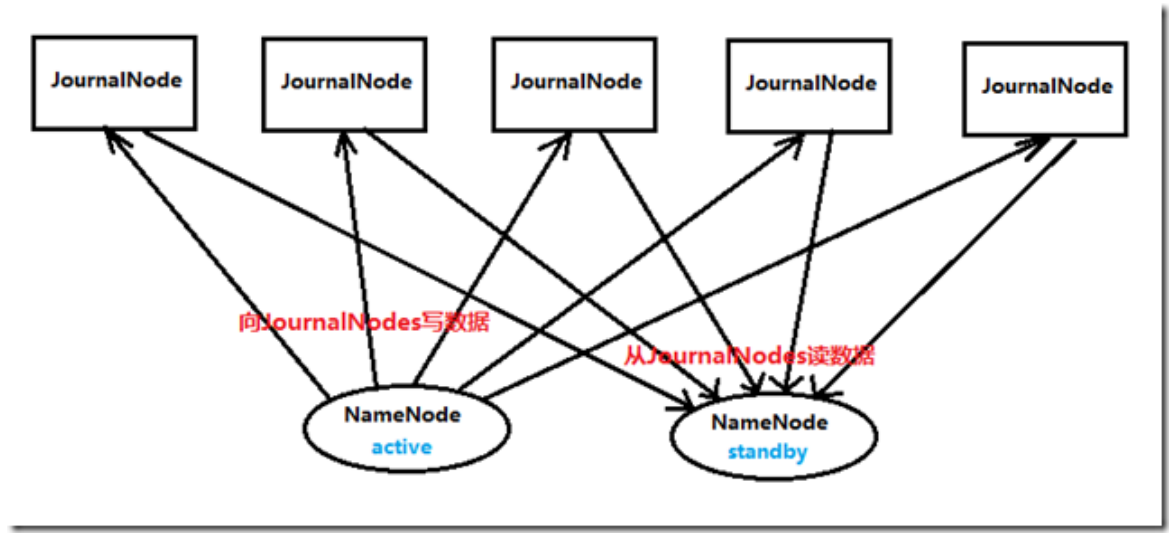


图3

为了确保快速切换，standby状态的NameNode有必要知道集群中所有数据块的位置。为了做到这点，所有的datanodes必须配置两个NameNode的地址，发送数据块位置信息和心跳给他们两个。

对于HA集群而言，确保同一时刻只有一个NameNode处于active状态是至关重要的。否则，两个NameNode的数据状态就会产生分歧，可能丢失数据，或者产生错误的结果。为了保证这点，JNs必须确保同一时刻只有一个NameNode可以向自己写数据。

硬件资源

为了部署HA集群，应该准备以下事情：

* NameNode服务器：运行NameNode的服务器应该有相同的硬件配置。

* JournalNode服务器：运行的JournalNode进程非常轻量，可以部署在其他的服务器上。注意：必须允许至少3个节点。当然可以运行更多，但是必须是奇数个，如3、5、7、9个等等。当运行N个节点时，系统可以容忍至少(N-1)/2(N至少为3)个节点失败而不影响正常运行。

在HA集群中，standby状态的NameNode可以完成checkpoint操作，因此没必要配置Secondary NameNode、CheckpointNode、BackupNode。如果真的配置了，还会报错。

2.YARN实现HA

大家都知道在hadoop2中对HDFS的改进很大，实现了NameNode的HA；也增加了ResourceManager。但是ResourceManager也可以实现HA。你没看错，确实是ResourceManager的HA。注意是在Apache Hadoop 2.4.1版本中开始加入的，可不是任意一个版本。

我们不讲单点问题的危害，直接讲如何配置ResourceManager的HA。

HA的架构

如果大家理解HDFS的HA，那么ResourceManager的HA与之是相同道理的：也是Active/Standby架构，任意时刻，都一个是Active，其余处于Standby状态的ResourceManager可以随时转换成Active状态。状态转换可以手工完成，也可以自动完成。手工完成时通过命令行的管理命令(命令是“yarn rmadmin”)。自动完成是通过配置自动故障转移(automatic-failover)，使用集成的failover-controller完成状态的自动切换。

自动故障转移是依赖于ZooKeeper集群，依赖ZooKeeper的ActiveStandbyElector会嵌入到ResourceManager中，当Active状态的ResourceManager失效时，处于Standby状态的ResourceManager就会被选举为Active状态的，实现切换。注意：这里没有ZooKeeperFailoverController进程，这点和HDFS的HA不同。

对于客户端而言，必须知道所有的ResourceManager中。因此，需要在yarn-site.xml中配置所有的ResourceManager。那么，当一个Active状态的ResourceManager失效时，客户端怎么办哪？客户端会采用轮询机制，轮询配置在yarn-site.xml中的ResourceManager，直到找到一个active状态的ResourceManager。如果我们想修改这种寻找ResourceManager的机制，可以继承类org.apache.hadoop.yarn.client.RMFailoverProxyProvider，实现自己的逻辑。然后把类的名字配置到yarn-site.xml的配置项yarn.client.failover-proxy-provider中。

配置

在yarn-site.xml中配置如下

```
<property>        <name>yarn.resourcemanager.ha.enabled</name>        <value>true</value>   </property>   <property>
<name>yarn.resourcemanager.cluster-id</name>              <value>cluster1</value>     </property>     <property>
<name>yarn.resourcemanager.ha.rm-ids</name>               <value>rm1,rm2</value>     </property>     <property>
<name>yarn.resourcemanager.hostname.rm1</name>              <value>master1</value>     </property>     <property>
<name>yarn.resourcemanager.hostname.rm2</name>              <value>master2</value>     </property>     <property>
<name>yarn.resourcemanager.zk-address</name>   <value>zk1:2181,zk2:2181,zk3:2181</value> </property>
```

命令

查看状态的命令

```
yarn rmadmin  -getServiceState rm1
```

状态切换的命令

```
yarn rmadmin  -transitionToStandby rm1
```

二、安装部署

主机名	IP	安装的软件	运行的进程
bigdata-realtime-monitor20.gz01	100.69.202.49	jdk、hadoop	NameNode、DFSZKFailoverController(zkfc)、ResourceManager
bigdata-realtime-monitor21.gz01	100.69.204.11	jdk、hadoop	NameNode、DFSZKFailoverController(zkfc)、ResourceManager
bigdata-realtime-monitor22.gz01	100.69.199.31	jdk、hadoop、zookeeper	DataNode、NodeManager、JournalNode、QuorumPeerMain
bigdata-realtime-monitor23.gz01	100.69.204.12	jdk、hadoop、zookeeper	DataNode、NodeManager、JournalNode、QuorumPeerMain
bigdata-realtime-monitor24.gz01	100.69.206.30	jdk、hadoop、zookeeper	DataNode、NodeManager、JournalNode、QuorumPeerMain

0.配置SSH免密登录

```
vi /etc/hosts.allow


su monitor

vi ~/.ssh/authorized_keys


chmod 700 ~/.ssh/authorized_keys
```

1.安装JDK

```
touch /etc/profile.d/monitor.sh
chown monitor:monitor  /etc/profile.d/monitor.sh


mkdir -p /home/data/hadoop

mkdir -p /home/data/hadoop/data

mkdir -p /home/data/hadoop/name

mkdir -p /home/data/hadoop/temp

mkdir -p /home/data/hadoop/journaldata

mkdir -p /home/data/hadoop/history/done_intermediate

mkdir -p /home/data/hadoop/history/done
```

```
mkdir -p /home/data/hadoop/yarn_logs


mkdir -p /home/data/hadoop/yarn_logs/monitor/logs/



mkdir -p /data1/hadoop/userlogs

mkdir -p /data2/hadoop/userlogs

mkdir -p /data3/hadoop/userlogs
mkdir -p /data4/hadoop/userlogs
mkdir -p /data5/hadoop/userlogs

mkdir -p /data6/hadoop/userlogs

mkdir -p /data7/hadoop/userlogs
mkdir -p /data8/hadoop/userlogs

mkdir -p /data9/hadoop/userlogs

mkdir -p /data10/hadoop/userlogs

mkdir -p /data11/hadoop/userlogs

mkdir -p /home/data/zookeeper

mkdir -p /home/data/zookeeper/data

mkdir -p /home/data/zookeeper/logs

mkdir -p /home/data/mysql

mkdir -p /home/apps


chown monitor:monitor -R  /home/apps

chown monitor:monitor -R /home/data/


mv /bin/java /bin/java1.7


su monitor
cd /home/apps
scp monitor@bigdata-realtime-monitor20.gz01:/home/apps/jdk-8u102-linux-x64.tar.gz .
tar -xzf jdk-8u102-linux-x64.tar.gz


#初始化环境变量

vi /etc/profile.d/monitor.sh

====================

export JAVA_HOME="/home/apps/jdk1.8.0_102"

export HADOOP_HOME="/home/apps/hadoop-2.7.2"

export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

export ZOO_HOME=/home/apps/zookeeper-3.4.8

export ZOO_LOG_DIR=/home/data/zookeeper/logs

export PATH=$PATH:$ZOO_HOME/bin

====================


2.搭建zookeeper集群
```

其他节点复制安装：

在bigdata-realtime-monitor22.gz01上执行：

scp -r zookeeper-3.4.8 monitor@bigdata-realtime-monitor23.gz01:/home/apps/

scp -r zookeeper-3.4.8 monitor@bigdata-realtime-monitor24.gz01:/home/apps/

cd /home/apps

wget http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.4.8/zookeeper-3.4.8.tar.gz

tar -xzf zookeeper-3.4.8

#修改配置文件

cp zookeeper-3.4.8/conf/zoo_sample.cfg zookeeper-3.4.8/conf/zoo.cfg

vi /home/apps/zookeeper-3.4.8/conf/zoo.cfg

==========

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/home/data/zookeeper/data
# the port at which the clients will connect
clientPort=2181


server.1=100.69.199.31:2888:3888
server.2=100.69.204.12:2888:3888
server.3=100.69.206.30:2888:3888
```

==========

#修改各节点的myid

在100.69.199.31上执行：

echo 1 > /home/data/zookeeper/data/myid

在100.69.204.12上执行：

echo 2 > /home/data/zookeeper/data/myid

在100.69.206.30上执行：

echo 3 > /home/data/zookeeper/data/myid

#分别在3台server上启动zookeeper

cd /home/apps/zookeeper-3.4.8

bin/zkServer.sh start

#jps查看进程

30056 QuorumPeerMain

QuorumPeerMain是zookeeper进程，说明启动正常。

#查看状态

bin/zkServer.sh status

[monitor@bigdata-realtime-monitor23 zookeeper-3.4.8]$ bin/zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /home/apps/zookeeper-3.4.8/bin/../conf/zoo.cfg

Mode: leader

#停止Zookeeper

bin/zkServer.sh stop

3.安装Hadoop

cd /home/apps

tar -zxf hadoop-2.7.2.tar.gz

chown monitor:monitor -R hadoop-2.7.2

#修改环境变量

vi /etc/profile.d/monitor.sh

==========

export JAVA_HOME="/home/apps/jdk1.8.0_102"

export HADOOP_HOME="/home/apps/hadoop-2.7.2"

export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

==========

source /etc/profile

#修改/home/apps/hadoop-2.7.2/etc/hadoop下的hadoop-env.sh

vi /home/apps/hadoop-2.7.2/etc/hadoop/hadoop-env.sh

========

export JAVA_HOME=/home/apps/jdk1.8.0_102

========

source /home/apps/hadoop-2.7.2/etc/hadoop/hadoop-env.sh

#修改core-site.xml

vi /home/apps/hadoop-2.7.2/etc/hadoop/core-site.xml

==========

```
<configuration>

         <property>
                <name>fs.defaultFS</name>
                <value>hdfs://ns1</value>
        </property>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/home/data/hadoop/temp</value>
        </property>
        <property>
                <name>ha.zookeeper.quorum</name>

<value>bigdata-realtime-monitor22.gz01:2181,bigdata-realtime-monitor23.gz01:2181,bigdata-realtime-monito
r24.gz01:2181</value>
        </property>

</configuration>
```

==========

#修改hdfs-site.xml

先修改磁盘的权限：chown monitor:monitor -R /data*

```
<configuration>
 <!--指定hdfs的nameservice为ns1，需要和core-site.xml中的保持一致 -->
 <property>
  <name>dfs.nameservices</name>
  <value>ns1</value>
 </property>
 <!-- ns1下面有两个NameNode，分别是nn1，nn2 -->
 <property>
  <name>dfs.ha.namenodes.ns1</name>
  <value>nn1,nn2</value>
 </property>
 <!-- nn1的RPC通信地址 -->
 <property>
  <name>dfs.namenode.rpc-address.ns1.nn1</name>
  <value>bigdata-realtime-monitor20.gz01:9000</value>
 </property>
 <!-- nn1的http通信地址 -->
 <property>
  <name>dfs.namenode.http-address.ns1.nn1</name>
  <value>bigdata-realtime-monitor20.gz01:50070</value>
 </property>
```

```
 <!-- nn2的RPC通信地址 -->
 <property>
  <name>dfs.namenode.rpc-address.ns1.nn2</name>
  <value>bigdata-realtime-monitor21.gz01:9000</value>
 </property>
 <!-- nn2的http通信地址 -->
 <property>
  <name>dfs.namenode.http-address.ns1.nn2</name>
  <value>bigdata-realtime-monitor21.gz01:50070</value>
 </property>
 <!-- 指定NameNode的元数据在JournalNode上的存放位置 -->
 <property>
  <name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://bigdata-realtime-monitor22.gz01:8485;bigdata-realtime-monitor23.gz01:8485;bigdata-real
time-monitor24.gz01:8485/ns1</value>
 </property>
 <!-- 指定JournalNode在本地磁盘存放数据的位置 -->
 <property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/home/data/hadoop-2.7.2/journaldata</value>
 </property>
 <!-- 开启NameNode失败自动切换 -->
 <property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
 </property>
 <!-- 配置失败自动切换实现方式 -->
 <property>
  <name>dfs.client.failover.proxy.provider.ns1</name>
  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
 </property>
 <!-- 配置隔离机制方法，多个机制用换行分割，即每个机制暂用一行-->
 <property>
  <name>dfs.ha.fencing.methods</name>
  <value>
   sshfence
   shell(/bin/true)
  </value>
 </property>
 <!-- 使用sshfence隔离机制时需要ssh免登陆 -->
 <property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/home/monitor/.ssh/id_rsa</value>
 </property>
 <!-- 配置sshfence隔离机制超时时间 -->
 <property>
  <name>dfs.ha.fencing.ssh.connect-timeout</name>
  <value>30000</value>
 </property>
   <property>
               <name>dfs.data.dir</name>

<value>/data1,/data2,/data3,/data4,/data5,/data6,/data7,/data8,/data9,/data10,/data11</value>vi
        </property>
</configuration>
```

先将mapred-site.xml.template改名为mapred-site.xml然后修改mapred-site.xml

mv mapred-site.xml.template mapred-site.xml

vi mapred-site.xml

==========

```
<configuration>
 <!-- 指定mr框架为yarn方式 -->
 <property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
 </property>
</configuration>
```

vi yarn-site.xml

==========

```
<configuration>
<!-- Site specific YARN configuration properties -->
        <property>
           <name>yarn.resourcemanager.ha.enabled</name>
           <value>true</value>
        </property>
        <property>
           <name>yarn.resourcemanager.cluster-id</name>
           <value>yrc</value>
        </property>

        <property>

           <name>yarn.resourcemanager.ha.rm-ids</name>

           <value>rm1,rm2</value>

        </property>

        <property>

           <name>yarn.resourcemanager.hostname.rm1</name>

           <value>bigdata-realtime-monitor20.gz01</value>

        </property>

        <property>

           <name>yarn.resourcemanager.hostname.rm2</name>

           <value>bigdata-realtime-monitor21.gz01</value>
```

```
        </property>

        <property>

           <name>yarn.resourcemanager.zk-address</name>


<value>bigdata-realtime-monitor22.gz01:2181,bigdata-realtime-monitor23.gz01:2181,bigdata-realtime-monitor24.gz01:2181</value>

        </property>

        <property>

           <name>yarn.nodemanager.aux-services</name>

           <value>mapreduce_shuffle</value>

        </property>
  <property>

           <name>yarn.nodemanager.resource.memory-mb</name>

           <value>40960</value>

        </property>

        <property>

           <name>yarn.scheduler.minimum-allocation-mb</name>

           <value>2048</value>

        </property>

        <property>

           <name>yarn.scheduler.maximum-allocation-mb</name>

           <value>40960</value>

        </property>

        <property>

           <name>yarn.nodemanager.resource.cpu-vcores</name>

           <value>12</value>

        </property>

        <property>

           <name>yarn.scheduler.maximum-allocation-vcores</name>

           <value>36</value>

        </property>
<!-- 需要指定webapp.address否则访问不到sparkUI-->
```

```
<property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>bigdata-realtime-monitor20.gz01:8088</value>
```

```
        </property>

</configuration>
```

#修改capacity-scheduler.xml

```
<property>
    <name>yarn.scheduler.capacity.maximum-am-resource-percent</name>
    <value>0.9</value>
    <description>
      Maximum percent of resources in the cluster which can be used to run
      application masters i.e. controls number of concurrent running
      applications.
    </description>
  </property>
```

==========

#格式化HDFS(在bigdata-realtime-monitor20.gz01上执行)

hdfs namenode -format

#注意：格式化之后需要把temp目录拷给bigdata-realtime-monitor21.gz01（不然bigdata-realtime-monitor21.gz01的namenode起不来）

scp -r /home/data/hadoop/temp/ monitor@bigdata-realtime-monitor21.gz01:/home/data/hadoop/temp/

#.格式化ZKFC(在bigdata-realtime-monitor20.gz01上执行)

hdfs zkfc -formatZK

启动HDFS(在bigdata-realtime-monitor20.gz01上执行)

start-dfs.sh

#启动YARN(在bigdata-realtime-monitor20.gz01上执行)

start-yarn.sh

注意：bigdata-realtime-monitor21.gz01的resourcemanager需要手动单独启动：

yarn-daemon.sh start resourcemanager

如果报SSH错误：

```
The authenticity of host 'bigdata-realtime-monitor23.gz01 (100.69.204.12)' can't be established.

ECDSA key fingerprint is 57:ec:36:5d:14:6c:9e:d7:88:43:58:81:79:a2:48:11.

Are you sure you want to continue connecting (yes/no)? The authenticity of host
'bigdata-realtime-monitor24.gz01 (100.69.206.30)' can't be established.
```

则用ssh monitor@bigdata-realtime-monitor23.gz01登录一次即可

此外，namenode、datanode也可以单独启动：

```
hadoop-daemon.sh start namenode
hadoop-daemon.sh start datanode
```

至此，部署完毕。

hadoop运维操作

删除所有hadoop服务器运行的java进程:

```
ps aux|grep java|grep -v grep |awk '{print $2}'|xargs kill -9

salt -N hadoop  cmd.run "ps aux|grep java|grep -v grep |awk '{print $2}'|xargs kill -9"
salt -N hadoop  cmd.run "ps aux|grep java|grep -v grep "
```

同步配置文件：

```
cp /home/apps/hadoop-2.7.2/etc/hadoop/hdfs-site.xml /srv/salt/
cp /home/apps/hadoop-2.7.2/etc/hadoop/mapred-site.xml /srv/salt/
cp /home/apps/hadoop-2.7.2/etc/hadoop/yarn-site.xml /srv/salt/
cp /home/apps/hadoop-2.7.2/etc/hadoop/capacity-scheduler.xml /srv/salt/


salt -N hadoop cp.get_file salt://hdfs-site.xml /home/apps/hadoop-2.7.2/etc/hadoop/yarn-site.xml
salt -N hadoop cp.get_file salt://mapred-site.xml  /home/apps/hadoop-2.7.2/etc/hadoop/mapred-site.xml
salt -N hadoop cp.get_file salt://yarn-site.xml /home/apps/hadoop-2.7.2/etc/hadoop/yarn-site.xml
salt -N hadoop cp.get_file salt://capacity-scheduler.xml
/home/apps/hadoop-2.7.2/etc/hadoop/capacity-scheduler.xml


salt -N hadoop  cmd.run 'chown monitor:monitor -R /home/apps/hadoop-2.7.2/etc/hadoop/'
salt -N hadoop  cmd.run 'mkdir -p /home/data/hadoop/history/done_intermediate'
salt -N hadoop  cmd.run 'mkdir -p /home/data/hadoop/history/done'
salt -N hadoop  cmd.run 'mkdir -p /home/data/hadoop/yarn_logs'
salt -N hadoop  cmd.run 'mkdir -p /home/data/hadoop/yarn_logs/monitor/logs/'
salt -N hadoop  cmd.run 'chown monitor:monitor -R /home/data/hadoop/'


#启动zookeeper

salt -N zk3.4.8 cmd.run "su - monitor -c \"/home/apps/zookeeper-3.4.8/bin/zkServer.sh start\""
salt -N zk3.4.8 cmd.run "su - monitor -c \"jps\""

#启动HDFS
su monitor && start-dfs.sh

#启动YARN
su monitor && start-yarn.sh

启动job history
su monitor && mr-jobhistory-daemon.sh start historyserver

salt -N hadoop  cmd.script salt://test.sh
```

3.Web UI

HDFS UI： http://100.70.100.30:8002/

YARN UI： http://100.70.100.30:8003/

如果需要在YARN UI上查看Log,还需要配置Hosts

==========

```
#广州hadoop集群hosts
100.69.202.49 bigdata-realtime-monitor20.gz01.diditaxi.com
100.69.204.11 bigdata-realtime-monitor21.gz01.diditaxi.com
100.69.199.31 bigdata-realtime-monitor22.gz01.diditaxi.com
100.69.204.12 bigdata-realtime-monitor23.gz01.diditaxi.com
100.69.206.30 bigdata-realtime-monitor24.gz01.diditaxi.com
100.69.207.50 bigdata-realtime-monitor25.gz01.diditaxi.com
100.69.200.50 bigdata-realtime-monitor26.gz01.diditaxi.com
100.69.200.11 bigdata-realtime-monitor27.gz01.diditaxi.com
100.70.100.52 bigdata-realtime-monitor28.gz01.diditaxi.com
100.70.100.12 bigdata-realtime-monitor29.gz01.diditaxi.com
100.70.100.16 bigdata-realtime-monitor30.gz01.diditaxi.com
100.69.137.13 bigdata-realtime-monitor31.gz01.diditaxi.com
100.69.137.50 bigdata-realtime-monitor32.gz01.diditaxi.com
100.69.206.31 bigdata-realtime-monitor33.gz01.diditaxi.com
100.69.121.50 bigdata-realtime-monitor34.gz01.diditaxi.com
100.69.203.30 bigdata-realtime-monitor35.gz01.diditaxi.com
100.70.100.51 bigdata-realtime-monitor36.gz01.diditaxi.com
100.70.100.53 bigdata-realtime-monitor37.gz01.diditaxi.com
100.70.100.50 bigdata-realtime-monitor38.gz01.diditaxi.com
100.69.203.31 bigdata-realtime-monitor39.gz01.diditaxi.com
100.69.202.11 bigdata-realtime-monitor40.gz01.diditaxi.com
100.70.100.54 bigdata-realtime-monitor41.gz01.diditaxi.com
100.69.200.49 bigdata-realtime-monitor42.gz01.diditaxi.com
100.70.100.13 bigdata-realtime-monitor43.gz01.diditaxi.com
100.70.100.14 bigdata-realtime-monitor44.gz01.diditaxi.com
100.70.100.15 bigdata-realtime-monitor45.gz01.diditaxi.com
100.69.137.12 bigdata-realtime-monitor46.gz01.diditaxi.com
100.69.135.50 bigdata-realtime-monitor47.gz01.diditaxi.com
```

==========

在Nginx中配置代理服务器

==========

```
server {
        listen       8002;
        server_name  localhost;
        access_log  logs/host.access.8002.log  main;
        location / {
                  proxy_pass http://100.69.202.49:50070;
        }
   }
    server {
        listen       8003;
        server_name  localhost;
        access_log  logs/host.access.8003.log  main;
        location / {
                  proxy_pass http://100.69.202.49:8088;
        }
   }
```

==========

4 HDFS

查看DataNode是否正常启动

```
hdfs dfsadmin -report
```

查看NameNode的主备状态

```
hdfs haadmin -getServiceState nn1
hdfs haadmin -getServiceState nn2
```

上传文件命令

```
hdfs dfs -put README.txt /
```

查看文件列表

```
hdfs dfs -ls /
```

删除文件

```
hdfs dfs -rm /README.txt
```

创建文件夹

```
hadoop fs -mkdir -p /user/monitor
hadoop fs -rmdir /input
```

重启Datanode

hadoop-daemon.sh stop datanode

hadoop-daemon.sh start datanode

新NameNode如何加入？

```
hadoop-daemon.sh stop
```

当有NameNode机器损坏时，必然存在新NameNode来替代。把配置修改成指向新NameNode，然后以备机形式启动新NameNode，这样新的NameNode即加入到Cluster中：

```
1) ./hdfs namenode -bootstrapStandby
2) ./hadoop-daemon.sh start namenode
```

存储均衡start-balancer.sh

示例：start-balancer.sh -t 10%

10%表示机器与机器之间磁盘使用率偏差小于10%时认为均衡，否则做均衡搬动。“start-balancer.sh”调用“hdfs start balancer”来做均衡，可以调用stop-balancer.sh停止均衡。

执行YARN命令

yarn node -list

```
monitor@bigdata-realtime-monitor20 hadoop-2.7.2]$ yarn node -list

Total Nodes:3

         Node-Id      Node-State Node-Http-Address Number-of-Running-Containers

bigdata-realtime-monitor23.gz01.diditaxi.com:50409         RUNNING
bigdata-realtime-monitor23.gz01.diditaxi.com:8042                               0

bigdata-realtime-monitor24.gz01.diditaxi.com:54782         RUNNING
bigdata-realtime-monitor24.gz01.diditaxi.com:8042                               0

bigdata-realtime-monitor22.gz01.diditaxi.com:32841         RUNNING
bigdata-realtime-monitor22.gz01.diditaxi.com:8042                               0
```

查看APP logs

yarn logs -applicationId application_1472017562505_0001

查看指定NodeManager的状态

yarn node -status

查看rm1的主备状态，即查看它是主（active）还是备（standby）

```
 yarn rmadmin -getServiceState rm1
```

将rm1从主切为备。
yarn rmadmin -transitionToStandby rm1

更多的yarn命令可以参考：

https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YarnCommands.html。

重启YARN

stop-yarn.sh

start-yarn.sh

yarn-daemon.sh stop resourcemanager
yarn-daemon.sh start resourcemanager

重启HDFS

stop-dfs.sh

start-dfs.sh

5.YARN的内存和CPU配置

Hadoop YARN同时支持内存和CPU两种资源的调度，本文介绍如何配置YARN对内存和CPU的使用。

YARN作为一个资源调度器，应该考虑到集群里面每一台机子的计算资源，然后根据application申请的资源进行分配Container。Container是YARN里面资源分配的基本单位，具有一定的内存以及CPU资源。

在YARN集群中，平衡内存、CPU、磁盘的资源的很重要的，根据经验，每两个container使用一块磁盘以及一个CPU核的时候可以使集群的资源得到一个比较好的利用。

内存配置

关于 内存 相关的配置可以参考hortonwork公司的文档 Determine HDP Memory Configuration Settings 来配置你的集群。
YARN以及MAPREDUCE所有可用的内存资源应该要除去系统运行需要的以及其他的hadoop的一些程序，总共保留的内存=系统内存+HBASE内存。

可以参考下面的表格确定应该保留的内存：

每台机子内存	系统需要的内存	HBase需要的内存
4GB	1GB	1GB
8GB	2GB	1GB
16GB	2GB	2GB
24GB	4GB	4GB
48GB	6GB	8GB
64GB	8GB	8GB
72GB	8GB	8GB
96GB	12GB	16GB
128GB	24GB	24GB
255GB	32GB	32GB
512GB	64GB	64GB

计算每台机子最多可以拥有多少个container，可以使用下面的公式：

containers = min (2*CORES, 1.8*DISKS, (Total available RAM) / MIN_CONTAINER_SIZE)

说明：

- `CORES` 为机器CPU核数
- `DISKS` 为机器上挂载的磁盘个数
- `Total available RAM` 为机器总内存
- `MIN_CONTAINER_SIZE` 是指container最小的容量大小，这需要根据具体情况去设置，可以参考下面的表格：

每台机子可用的RAM	container最小值
小于4GB	256MB
4GB到8GB之间	512MB
8GB到24GB之间	1024MB
大于24GB	2048MB

每个container的平均使用内存大小计算方式为：

RAM-per-container = max(MIN_CONTAINER_SIZE, (Total Available RAM) / containers))

通过上面的计算，YARN以及MAPREDUCE可以这样配置：

配置文件	配置设置	默认值	计算值
yarn-site.xml	yarn.nodemanager.resource.memory-mb	8192 MB	= containers * RAM-per-container
yarn-site.xml	yarn.scheduler.minimum-allocation-mb	1024MB	= RAM-per-container
yarn-site.xml	yarn.scheduler.maximum-allocation-mb	8192 MB	= containers * RAM-per-container
yarn-site.xml (check)	yarn.app.mapreduce.am.resource.mb	1536 MB	= 2 * RAM-per-container
yarn-site.xml (check)	yarn.app.mapreduce.am.command-opts	-Xmx1024m	= 0.8 * 2 * RAM-per-container
mapred-site.xml	mapreduce.map.memory.mb	1024 MB	= RAM-per-container
mapred-site.xml	mapreduce.reduce.memory.mb	1024 MB	= 2 * RAM-per-container
mapred-site.xml	mapreduce.map.java.opts		= 0.8 * RAM-per-container
mapred-site.xml	mapreduce.reduce.java.opts		= 0.8 * 2 * RAM-per-container

举个例子：对于128G内存、32核CPU的机器，挂载了7个磁盘，根据上面的说明，系统保留内存为24G，不适应HBase情况下，系统剩余可用内存为104G，计算containers值如下：

containers = min (2*32, 1.8* 7 , (128-24)/2) = min (64, 12.6 , 51) = 13

计算RAM-per-container值如下：

RAM-per-container = max (2, (124-24)/13) = max (2, 8) = 8

这样集群中下面的参数配置值如下：

配置文件	配置设置	计算值
yarn-site.xml	yarn.nodemanager.resource.memory-mb	= 13 * 8 =104 G
yarn-site.xml	yarn.scheduler.minimum-allocation-mb	= 8G
yarn-site.xml	yarn.scheduler.maximum-allocation-mb	= 13 * 8 = 104G
yarn-site.xml (check)	yarn.app.mapreduce.am.resource.mb	= 2 * 8=16G
yarn-site.xml (check)	yarn.app.mapreduce.am.command-opts	= 0.8 * 2 * 8=12.8G
mapred-site.xml	mapreduce.map.memory.mb	= 8G
mapred-site.xml	mapreduce.reduce.memory.mb	= 2 * 8=16G
mapred-site.xml	mapreduce.map.java.opts	= 0.8 * 8=6.4G
mapred-site.xml	mapreduce.reduce.java.opts	= 0.8 * 2 * 8=12.8G

你也可以使用脚本 yarn-utils.py 来计算上面的值：

```
python yarn-utils.py -c 32 -m 128 -d 7 -k False
```

返回结果如下：

```
Using cores=32 memory=128GB disks=7 hbase=False
 Profile: cores=32 memory=106496MB reserved=24GB usableMem=104GB disks=7
 Num Container=13
 Container Ram=8192MB
 Used Ram=104GB
 Unused Ram=24GB
 yarn.scheduler.minimum-allocation-mb=8192
 yarn.scheduler.maximum-allocation-mb=106496
 yarn.nodemanager.resource.memory-mb=106496
 mapreduce.map.memory.mb=8192
 mapreduce.map.java.opts=-Xmx6553m
 mapreduce.reduce.memory.mb=8192
 mapreduce.reduce.java.opts=-Xmx6553m
 yarn.app.mapreduce.am.resource.mb=8192
 yarn.app.mapreduce.am.command-opts=-Xmx6553m
 mapreduce.task.io.sort.mb=3276
```

对应的xml配置为：

```
<property>
   <name>yarn.nodemanager.resource.memory-mb</name>
   <value>106496</value>
  </property>
  <property>
   <name>yarn.scheduler.minimum-allocation-mb</name>
   <value>8192</value>
  </property>
  <property>
   <name>yarn.scheduler.maximum-allocation-mb</name>
   <value>106496</value>
  </property>
  <property>
   <name>yarn.app.mapreduce.am.resource.mb</name>
   <value>8192</value>
  </property>
  <property>
   <name>yarn.app.mapreduce.am.command-opts</name>
   <value>-Xmx6553m</value>
  </property>
```

另外，还有一下几个参数：

- `yarn.nodemanager.vmem-pmem-ratio` ：任务每使用1MB物理内存，最多可使用虚拟内存量，默认是2.1。
- `yarn.nodemanager.pmem-check-enabled` ：是否启动一个线程检查每个任务正使用的物理内存量，如果任务超出分配值，则直接将其杀掉，默认是true。
- `yarn.nodemanager.vmem-pmem-ratio` ：是否启动一个线程检查每个任务正使用的虚拟内存量，如果任务超出分配值，则直接将其杀掉，默认是true。

第一个参数的意思是当一个map任务总共分配的物理内存为8G的时候，该任务的container最多内分配的堆内存为6.4G，可以分配的虚拟内存上限为8*2.1=16.8G。另外，照这样算下去，每个节点上YARN可以启动的Map数为104/8=13个，似乎偏少了，这主要是和我们挂载的磁盘数太少了有关，人为的调整 `RAM-per-container` 的值为4G或者更小的一个值是否更合理呢？当然，这个要监控集群实际运行情况来决定了。

CPU配置

YARN中目前的CPU被划分成虚拟CPU（CPU virtual Core），这里的虚拟CPU是YARN自己引入的概念，初衷是，考虑到不同节点的CPU性能可能不同，每个CPU具有的计算能力也是不一样的，比如某个物理CPU的计算能力可能是另外一个物理CPU的2倍，这时候，你可以通过为第一个物理CPU多配置几个虚拟CPU弥补这种差异。用户提交作业时，可以指定每个任务需要的虚拟CPU个数。

在YARN中，CPU相关配置参数如下：

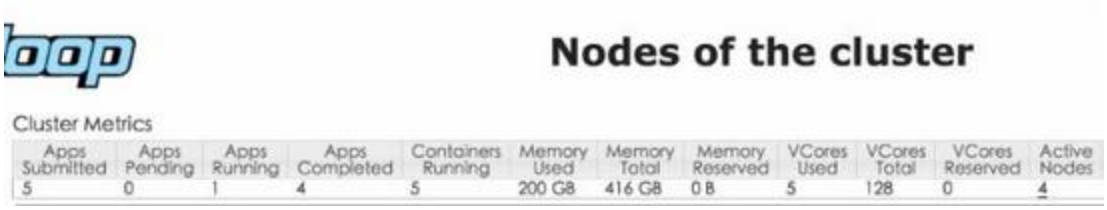
- `yarn.nodemanager.resource.cpu-vcores` ：表示该节点上YARN可使用的虚拟CPU个数，默认是8，注意，目前推荐将该值设值为与物理CPU核数数目相同。如果你的节点CPU核数不够8个，则需要调减小这个值，而YARN不会智能的探测节点的物理CPU总数。
- `yarn.scheduler.minimum-allocation-vcores` ：单个任务可申请的最小虚拟CPU个数，默认是1，如果一个任务申请的CPU个数少于该数，则该对应的值改为这个数。
- `yarn.scheduler.maximum-allocation-vcores` ：单个任务可申请的最多虚拟CPU个数，默认是32。

对于一个CPU核数较多的集群来说，上面的默认配置显然是不合适的，在我的测试集群中，4个节点每个机器CPU核数为32，可以配置为：

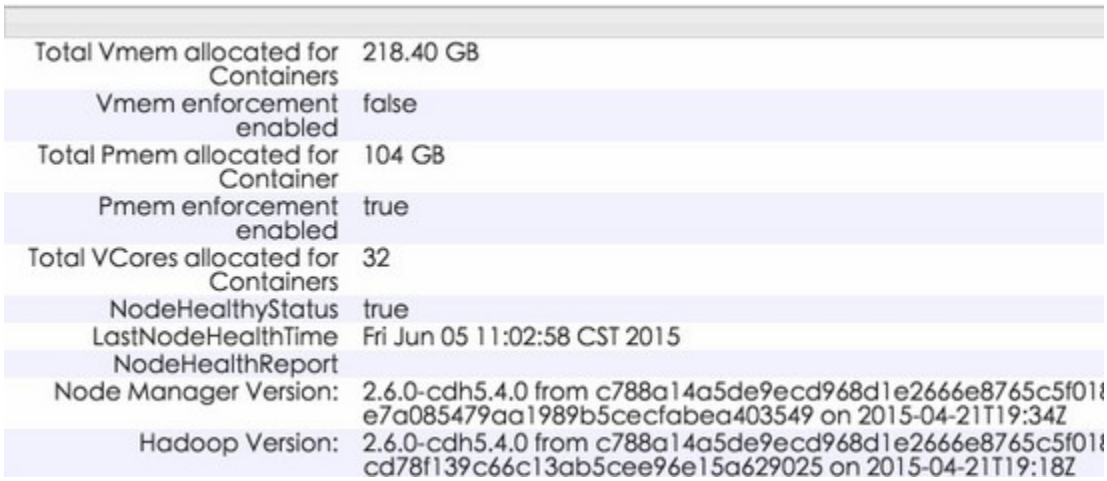
```
<property>
  <name>yarn.nodemanager.resource.cpu-vcores</name>
  <value>32</value>
  </property>
  <property>
  <name>yarn.scheduler.maximum-allocation-vcores</name>
  <value>128</value>
  </property>
```

总结

根据上面的说明，我的测试集群中集群节点指标如下：



每个节点分配的物理内存、虚拟内存和CPU核数如下：



实际生产环境中，可能不会像上面那样设置，比如不会把所有节点的CPU核数都分配给Spark，需要保留一个核留给系统使用；另外，内存上限也会做些设置。

```
tips 查询CPU核数
# 总核数 = 物理CPU个数 X 每颗物理CPU的核数

# 总逻辑CPU数 = 物理CPU个数 X 每颗物理CPU的核数 X 超线程数


# 查看物理CPU个数

cat /proc/cpuinfo| grep "physical id"| sort| uniq| wc -l


# 查看每个物理CPU中core的个数(即核数)

cat /proc/cpuinfo| grep "cpu cores"| uniq


# 查看逻辑CPU的个数

cat /proc/cpuinfo| grep "processor"| wc -l
```

参考链接：

http://itfish.net/article/56265.html

https://yq.aliyun.com/articles/33704