BIGDATA HADOOP

In this course, the participants will get you up to speed on Big Data and Hadoop. Topics include how to install, configure and manage a single and multi-node Hadoop cluster, configure and manage HDFS, write MapReduce jobs and work with many of the projects around Hadoop such as Pig, Hive, HBase, Sqoop, and Zookeeper. Topics also include configuring Hadoop in the cloud and troubleshooting a multi-node Hadoop cluster.

LinuxWorld Informatics Pvt Ltd

2016 - 17

BEFORE WE BEGIN

Before we begin our Big Data Hadoop Training , we must know some basic linux because we are going to configure our cluster on Red Hat version 6.

Right click on desktop anywhere and click on open in terminal to open the command line window.

```
[root@desktop83 hadoop]# ls                 // this is the list command which will list contents
[root@desktop83 hadoop]# ifconfig           // this command will show the interfaces and ip address
[root@desktop83 hadoop]# yum install  <package name> // this will installed the desired package
 [root@desktop83 hadoop]# getenforce        //to check selinux security
Enforcing
[root@desktop83 hadoop]# setenforce 0      // to set selinux to permissive

[root@desktop83 hadoop]# iptables -L       //show the firewall rules
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[root@desktop83 hadoop]# iptables -F       // will flush the firewall rules
```

How to Setup Hadoop

Our Training will take place on RHEL 7_2.

You can download Centos 7_2.

NOTE :- Red Hat Linux and Centos share the same kernell and rpm packages.

Step 1:

Get hadoop rpm from apache site, search on google "apache hadoop download"

https://archive.apache.org/dist/hadoop/core/hadoop-1.2.1/hadoop-1.2.1-1.x86_64.rpm

on your system , run

```
# yum install hadoop
```

Step 2:

Get java rpm from oracle site , search on google "jdk download"

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

in LinuxWorld Lab, run

```
# yum install jdk
```

Step 3: SET JAVA PATH

```
[root@server Desktop]# rpm –ql   jdk | grep java$
/etc/.java
/usr/java
/usr/java/jdk1.7.0_51/bin/java
/usr/java/jdk1.7.0_51/jre/bin/java
```

```
[root@server Desktop]# /usr/java/jdk1.7.0_51/bin/java  -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)
[root@server Desktop]# java -version
java version "1.7.0_09-icedtea"
OpenJDK Runtime Environment (rhel-2.3.4.1.el6_3-x86_64)
OpenJDK 64-Bit Server VM (build 23.2-b09, mixed mode)
[root@server Desktop]# echo $JAVA_HOME
/usr

 [root@server Desktop]# JAVA_HOME=/usr/java/jdk1.7.0_51/
[root@server Desktop]# echo $JAVA_HOME
/usr/java/jdk1.7.0_51/

[root@server Desktop]# PATH=$JAVA_HOME/bin:$PATH
```

Note: $JAVA_HOME must be put first then $PATH in above cmd

```
[root@server Desktop]# java -version
java version "1.7.0_51"
Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
Java HotSpot(TM) 64-Bit Server VM (build 24.51-b03, mixed mode)
```

Step 4: SET JAVA HOME AND JAVA PATH PERMANENTLY.

```
[root@server Desktop]# vim /root/.bash_profile
export JAVA_HOME=/usr/java/jdk1.7.0_51/
PATH=$JAVA_HOME/bin:$PATH
```

Step 6: Setup HDFS name and data node MULTINODE CLUSTER

ON NAMENODE SIDE

```
[root@server hadoop]# vim /etc/hadoop/hdfs-site.xml
<configuration>
<property>
<name>dfs.name.dir</name>
<value>/namenode</value>
</property>
</configuration>
```

ON ALL DATANODE SIDES

```
[root@client hadoop]# vim /etc/hadoop/hdfs-site.xml
<configuration>
<property>
<name>dfs.data.dir</name>
<value>/dataname</value>
</property>
</configuration>
```

note: above directory automactically created, no need to create before

Step 7: SETUP THE NAMENODE AND DATANODE CORE SITE

ON NAMENODE SIDE

```
[root@server hadoop]# vim /etc/hadoop/core-site.xml
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://ip of namenode:10001</value>
</property>
</configuration>
[root@server hadoop]# hadoop namenode –format                //This commande will format the namenode.
```

ON ALL DATANODE SIDES

```
[root@server hadoop]# vim /etc/hadoop/core-site.xml
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://ip of namenode:10001</value>
</property>
</configuration>
```

NOTE:- THE CORE SITE ENTRY WILL BE THE SAME FOR BOTH NAMENODE AND DATANODE.

TO START THE NAMENODE RUN THE FOLLOWING COMMAND

```
[root@server hadoop]# hadoop-daemon.sh  start namenode
```

above cmd start some port, run

```
#netstat  -tnlp | grep java
tcp     0     0 127.0.0.1:10001        0.0.0.0:*          LISTEN     14969/java
tcp     0     0 0.0.0.0:50070          0.0.0.0:*          LISTEN     14969/java
```

TO START DATANODES RUN THIS COMMAND ONLY ON DATA NODES

[root@server hadoop]# **hadoop-daemon.sh start datanode**

above cmd start some port, run

#netstat -tnlp | grep java

```
tcp        0      0 0.0.0.0:50010              0.0.0.0:*                  LISTEN      15093/java


tcp        0      0 0.0.0.0:50075              0.0.0.0:*                  LISTEN      15093/java
```

To verify weather the namenode is working or not run this command on the namenode.

[root@server hadoop]**# jps**

8177 Jps

7933 NameNode

Or go to url, as "50070" is name node management port

http://ip of namenode:50070

To verify weather the namenode is working or not run this command on the namenode.

[root@server hadoop]**# jps**

8177 Jps

4362 DataNode

in CLI,we can also see the report

[root@server hadoop]# **hadoop dfsadmin –report**

You can check hadoop hdfs filesytem,initially there is nothing

# **hadoop fs -ls /**

Create directory in hdfs filesystem

# hadoop fs -mkdir /input

Upload or copy local file into hdfs filesystem

# hadoop fs -copyFromLocal test.txt /input

Note : it uploaded to datanode at the storage folder named “current” in distributed fashion of maximum file size “64MB” as bcoz by default block size is 64MB

You can change block size in **hdfs-site.xml**

```
<property>
<name>dfs.block.size</name>
<value>134217728</value>
<final>true</final>
</property>
```

Bydefault it copy to 3 datanode, as by default replication is 3, you can change it in hdfs-site.xml

```
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
```

List file in hdfs

# hadoop fs -ls /input

# hadoop fs -lsr /

How to Setup MultiNode Map Reduce Cluster

Step 1: On JOBTRACKER SIDE

Setup Mapred-site.xml file:

```
# vim /etc/hadoop/mapred-site.xml
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>ip of jobtracker:9001</value>
</property>
</configuration>
```

Step 2: start jobtracker

```
# hadoop-daemon.sh  start jobtracker
starting jobtracker, logging to /var/log/hadoop/root/hadoop-root-jobtracker-desktop16.example.com.out
```

```
# jps
7247 JobTracker
7325 Jps
```

Note: it start 2 new port, check

```
# netstat -tnlp | grep java
tcp        0      0 0.0.0.0:50030            0.0.0.0:*                LISTEN      7411/java
tcp        0      0 192.168.0.16:9001        0.0.0.0:*                LISTEN      7411/java
```

Where, 50030 is management port for mapreduce,

Check it : http://ip of jobtracker:50030

Step 3: ON TASKTRACKER SIDE

Setup Mapred-site.xml file:

```
# vim /etc/hadoop/mapred-site.xml
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>ip of jobtracker:9001</value>
</property>
</configuration>
```

```
# hadoop-daemon.sh  start tasktracker
starting tasktracker, logging to /var/log/hadoop/root/hadoop-root-tasktracker-desktop16.example.com.out
```

```
# jps
7639 Jps
7569 TaskTracker
6467 DataNode
```

NOTE:- IT IS A GOOD PRACTICE TO CREATE THE TASKTRACKERS ON THE SAME NODES AS DATANODES.

Step 4:Test your setup, by run example file from hadoop rpm

You can get it here

```
# rpm -ql hadoop | grep examples
/usr/share/hadoop/hadoop-examples-1.2.1.jar
```

```
# hadoop jar  /usr/share/hadoop/hadoop-examples-1.2.1.jar   wordcount /input  /output
```

```
14/05/07 14:38:01 INFO input.FileInputFormat: Total input paths to process : 1
14/05/07 14:38:01 INFO util.NativeCodeLoader: Loaded the native-hadoop library
14/05/07 14:38:01 WARN snappy.LoadSnappy: Snappy native library not loaded
14/05/07 14:38:02 INFO mapred.JobClient: Running job: job_201405071431_0001
14/05/07 14:38:03 INFO mapred.JobClient:  map 0% reduce 0%
14/05/07 14:38:12 INFO mapred.JobClient:  map 100% reduce 0%
14/05/07 14:38:20 INFO mapred.JobClient:  map 100% reduce 33%
14/05/07 14:38:21 INFO mapred.JobClient:  map 100% reduce 100%
14/05/07 14:38:22 INFO mapred.JobClient: Job complete: job_201405071431_0001
```

# hadoop job -list all

```
1 jobs submitted
States are:
        Running : 1     Succeded : 2    Failed : 3      Prep : 4
JobId   State   StartTime       UserName        Priority        SchedulingInfo
job_201405071431_0001   2       1399453681859   root    NORMAL  NA
```

# hadoop fs -ls /output

```
Found 3 items
-rw-r--r--   3 root supergroup          0 2014-05-07 14:38 /output/_SUCCESS
drwxr-xr-x   - root supergroup          0 2014-05-07 14:38 /output/_logs
-rw-r--r--   3 root supergroup         34 2014-05-07 14:38 /output/part-r-00000
```

Note: **_SUCCESS** file created means map reduce is successfully done

Note: **part-r-00000** contains output of reducer , final output

You can see the final output of map reduce job

```
# hadoop fs -cat /output/part-r-00000
```

And we can also see by web UI

http://127.0.0.1:50070 -> Browse the filesystem

if you want to list complete details of running or completed job, then use job id with status option

```
# hadoop job -status  job_201405071431_0004
Job: job_201405071431_0004
file: hdfs://192.168.0.16:10001/tmp/hadoop-
root/mapred/staging/root/.staging/job_201405071431_0004/job.xml
tracking URL: http://desktop16.example.com:50030/jobdetails.jsp?jobid=job_201405071431_0004
map() completion: 0.017579561
reduce() completion: 0.0
Counters: 3
    Job Counters
        SLOTS_MILLIS_MAPS=2481
        Launched map tasks=2
        Data-local map tasks=2
```

```
# hadoop job -list
1 jobs currently running
JobId   State   StartTime      UserName       Priority       SchedulingInfo
job_201405071431_0004   1      1399486864042   root    NORMAL  NA
```

if you want to kill running process

```
# hadoop job -kill   job_201405071431_0004
Killed job job_201405071431_0004
```

HADOOP SUDOKU PUZZLE SOLVER

In hadoop example we can also run a sudoku puzzle solver , but we need to place the puzzle in same location as the example file.

```
8 5 ? 3 9 ? ? ? ?
? ? 2 ? ? ? ? ? ?
? ? 6 ? 1 ? ? ? 2
? ? 4 ? ? 3 ? 5 9
? ? 8 9 ? 1 4 ? ?
3 2 ? 4 ? ? 8 ? ?
9 ? ? ? 8 ? 5 ? ?
? ? ? ? ? ? 2 ? ?
? ? ? ? 4 5 ? 7 8
```

The above shown example is a sudoku puzzle. You can save the puzzle in .dta format e.g. sudoku.dta and then paste it in /usr/share/hadoop directory and run the following command.

```
hadoop jar /usr/share/hadoop/hadoop-examples-1.2.1.jar sudoku a.dta
Solving a.dta
8 5 1 3 9 2 6 4 7
4 3 2 6 7 8 1 9 5
7 9 6 5 1 4 3 8 2
6 1 4 8 2 3 7 5 9
5 7 8 9 6 1 4 2 3
3 2 9 4 5 7 8 1 6
9 4 7 2 8 6 5 3 1
1 8 5 7 3 9 2 6 4
2 6 3 1 4 5 9 7 8
```

This is the answer to the sudoku puzzle.

If you want to change the priority of one job over other

```
# hadoop job -set-priority   job_201405071431_0004 LOW
Changed job priority.
```

You can list by below command

```
#hadoop job -list all
4 jobs submitted
States are:
    Running : 1    Succeded : 2   Failed : 3     Prep : 4
JobId   State   StartTime      UserName       Priority       SchedulingInfo
job_201405071431_0001   2      1399453681859   root    NORMAL  NA
job_201405071431_0002   3      1399462379102   root    NORMAL  NA
job_201405071431_0003   2      1399462502071   root    NORMAL  NA
job_201405071431_0004   2      1399486864042   root    LOW     NA
```

JOB SCHEDULING IN HADOOP

BY default FIFO scheduler is used in "Apache Hadoop"

We can change to "Fair Scheduler" in mapred-site.xml file

Step 1:

```
# vim /etc/hadoop/mapred-site.xml
```

Step 2:

In mapred-site.xml of job tracker,specify the scheduler used :

```
<property>
<name>mapred.jobtracker.taskScheduler</name>
<value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
```

Identify the pool configuration file :

```
<property>
```

```
<name>mapred.fairscheduler.allocation.file</name>
<value>/etc/hadoop/fair-scheduler.xml</value>
</property>
```

Step 3:

```
# vim /etc/hadoop/fair-scheduler.xml
<allocations>

<pool name="tech">
<minMaps>10</minMaps>
<minReduces>5</minReduces>
<maxRunningJobs>2</maxRunningJobs>
</pool>



<pool name="hr">
<minMaps>10</minMaps>
<minReduces>5</minReduces>
</pool>

<user name="vimal">
<maxRunningJobs>2</maxRunningJobs>
</user>
</allocations>
```

Step 4:

```
# hadoop-daemon.sh  stop jobtracker
stopping jobtracker

# hadoop-daemon.sh  start jobtracker
```

```
starting jobtracker, logging to /var/log/hadoop/root/hadoop-root-jobtracker-desktop16.example.com.out
```

Step 5:

Run job with pool name “tech”

```
# hadoop jar /usr/share/hadoop/hadoop-examples-1.2.1.jar  wordcount  -Dpool.name=tech  /input
/output3
```

#Programming of MAP-REDUCE

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

(*The main concept in MAP REDUCE Programming is how the Reducer takes input from the Mapper)

#sys.stdin ----> To make the program wait for the input which is retrieved from a different program, here, Reducer waits for Mapper to produce output which acts as an input for Reducer

(*Every system should have a python interpreter, for the job to run successfully)

#re.search("abhi", name, I) ----> I converts the given keyword into case sensitive

#hadoop jar /usr/share/hadoop/contrib/streaming/hadoop-streaming-1.2.1.jar -input /passwd -file mapper.py -mapper ./mapper.py   -file reducer.py -reducer ./reducer.py -output /out1234 -------> Command to run the job on a MAP REDUCE Cluster

```
#Programn for Mapper

{

#!/usr/bin/python2

import re

import sys

#fh=open("/etc/passwd", mode="rt")

for i in sys.stdin:

        j=i.strip()

        k=j.split(":")

        name=k[0]

# regex -> regular expression

        if re.search("^abhi", name, re.I):

                print name

        else:

                pass

}

#Program for Reducer

-------------------

{

#!/usr/bin/python2

import sys

j=0

for x in sys.stdin:

        j+=1


print j

}

 ----------------------------------------------------------
```

How to get metadata of name node :

```
# hadoop dfsadmin -metasave mymeta.txt
```

Go to log directory and get the meta data

```
# cd /var/log/hadoop/root/
# cat mymeta.txt
```

If datanode goes down , name node comes into “SafeMode” , means readonly , we can only read hdfs metadata , but not able to write and read storage, until enough data node comes up

To Leave from Safe mode now:

```
# hadoop dfsadmin -safemode leave
```

How to start all datanode from namenode with single commands

```
# chmod  +x /usr/sbin/start-all.sh
# chmod  +x /usr/sbin/start-dfs.sh
# chmod  +x /usr/sbin/start-mapred.sh
# ssh-keygen
# ssh-copy-id root@10.0.0.189
# ssh-copy-id root@10.0.0.188
```

Write ip address of all datanode in below file

```
# vim /etc/hadoop/slaves
10.0.0.188
10.0.0.189
# start-all.sh
```

Datanode sends heart beat message in every 3 seconds by default

You can check by :

# tcpdump host ip –n

And till datanode is running it send heart beat and as soon as datanode die , its stops send heart beat and then name node disconnect its session

You can check by :

# netstat –nct

But it keeps entry in "hadoop dfsadmin –report" for some times

---

How name node get ip address of datanode :

when datanode starts it advertise its storage ID, which contains ip address of datanode

it maintain in:

# cat   current/VERSION

```
#Mon Jun 09 21:02:39 IST 2014
namespaceID=2011105450
storageID=DS-1361994480-10.0.0.188-50010-1401986099391
cTime=0
storageType=DATA_NODE
layoutVersion=-41
```

How to scan and check hdfs file system:

# hadoop fsck /

# hadoop fsck / -files

How to change the port number of management http protocol of name node :

In name node

# vim /etc/hadoop/hdfs-site.xml

```
<property>
 <name>dfs.http.address</name>
<value>ip_of namenode:50111</value>
<final>true</final>
 </property>
```

Every Data node send heart beat signal to name node to keep alive, by default time is 3 sec

<u>How to change heart beat time:</u>

Go to data node

# vim /etc/hadoop/hdfs-site.xml

```
<property>
<name>dfs.heartbeat.interval</name>
<value>10</value>
</property>
```

If datanode stop sending heart beat, then name node terminate its "ESTABLISHED" session

You can see by

# netstat –nct

<u>How to find where all blocks replicated and how much replication done</u>

<u>**# hadoop fsck / -files -blocks –location**</u>

<u>**# hadoop fsck / -location**</u>

How to create user and manage into Hadoop

Hadoop uses system user, better go for NIS or LDAP user to mange centrally

```
#useradd vimal
# passwd vimal
```

Create user home directory into hadoop hdfs, hadoop manages user into "/user" folder

```
# hadoop fs -mkdir /user/vimal
```

Change owner of home directory to user and group

```
# hadoop fs  -chown   vimal:vimal   /user/vimal
```

Change permission so that only user can access its home directory

```
# hadoop  fs   -chmod   770 /user/vimal
```

How to enable or disable permission checking

```
# vim /etc/hadoop/hdfs-site.xml
```

```
<property>
<name>dfs.permissions</name>
<value>false</value>
<final>true</final>
</property>
```

To check number of directory,files and space used

```
# hadoop fs -count   /
```

To check quota count and list

```
# hadoop fs -count -q  /
# hadoop fs -count -q /input
 none    inf   none      inf          1          4         1234 hdfs://10.0.0.173:10001/input
```

in above commands output :

"1" is no .of directory

"4" is no. of files

"1234" size of total files in bytes

First "none" is name quota and "inf" remaining name quota limit in no. of files includes directory

Second "none" is space quota and "inf" remaining space quota limit in bytes

---

<u>How to set Quota to limit maximum "10" files or directory in "/input"</u>

```
# hadoop dfsadmin  -setQuota  10 /input
# hadoop fs -count -q    /input
  10            5          none      inf        1           4         1234 hdfs://10.0.0.173:10001/input
```

10 is quota limit on files and 5 is remaining limit bcoz 1 directory and 4 files already exists

```
# hadoop dfsadmin  -setSpaceQuota    10000  /input


# hadoop fs -count -q /input
 10     5      10000       8766     1           4         1234 hdfs://10.0.0.173:10001/input
```

"10000" is quota limit in bytes and remaining space available is "8766" bcoz "1234" space already used

Or u can use size in MB, GB, TB....

```
# hadoop dfsadmin  -setSpaceQuota   10m  /input
```

<u>To clear name quota</u>

```
# hadoop   dfsadmin    -clrQuota    /input
# hadoop fs    -count   –q   /input
      none          inf      10000      8766          1          4         1234 hdfs://10.0.0.173:10001/input
```

<u>To clear Space Quota</u>

```
# hadoop dfsadmin    -clrSpaceQuota          /input
# hadoop fs -count -q /input
      none          inf         none          inf          1          4         1234 hdfs://10.0.0.173:10001/input
```

How to get size of all file

```
# hadoop fs -du /input
```

How to total size of all file

```
# hadoop fs -dus /input
```

How to manually set the block size by file basis during putting file into hdfs

```
# Hadoop    fs    -Ddfs.block.size=67108864  -put   file.txt   /input
```

How to manually set replication per file basis during putting file into hdfs

```
# hadoop fs    -Ddfs.replication=1  -put   file.txt  /
```

How to include and exclude datanode (commission and decommission)

You can control which data node to join our cluster , by default anybody can join – no security

```
# vim /etc/hadoop/hdfs-site.xml
<property>
<name>dfs.hosts</name>
<value>/etc/hadoop/hostsallow</value>
<final>true</final>
</property>
```

Create "/etc/hadoop/hostsallow" and restart namenode, bydefault if file is empty , any data node can connect, but want to allow particular data nodes, then write their ip addresses

To update gracefully

```
# hadoop dfsadmin -refreshNodes
```

Means just write ip address in file, then no need to start services, to update it run above cmd

To exclude or Decommission any datanode that already connected

```
# vim /etc/hadoop/hdfs-site.xml
<property>
```

```
<name>dfs.hosts.exclude</name>
<value>/etc/hadoop/hostsdeny</value>
<final>true</final>
</property>
```

Create “/etc/hadoop/hostsdeny” and restart namenode, bydefault if file is empty ,no data node exclude, but want to deny particular data nodes, then write their ip addresses

<u>To update gracefully</u>

# hadoop dfsadmin -refreshNodes

Means just write ip address in file, then no need to start services, to update it run above cmd

<u>Name node takes 10 minutes to make dead a data node if it is faulty , so to make instantly , put in “excludes” option in name node</u>

Tips - To get help form commands to check its internal options

```
[root@server Desktop]#  hadoop fs
[root@server Desktop]#  hadoop  dfs admin
[root@server Desktop]#  hadoop  mradmin
[root@server Desktop]#  Hadoop
```

How to start SNN from NN :

In NN masters file write snn ip address :

```
# cat /etc/hadoop/masters
snn.lw.comp    ---- * Ip of SNN
```

and Then,

from NN node :

```
# hadoop-daemon.sh  --hosts masters start secondarynamenode
```

How to decommission task tracker :

In mapred-site.xml

```
# cat /etc/hadoop/mapred-site.xml

<name>mapred.hosts.exclude</name>

<value>/etc/hadoop/excludes</value>

# hadoop mradmin –refreshNodes
```

How to Configure Secondary NameNode in HDFS :

In secondary Namenode

```
[root@secondarynn ~]# vim /etc/hadoop/core-site.xml
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://nn.lw.com:9001</value>
</property>
</configuration>
```

```
[root@secondarynn~]# vim /etc/hadoop/hdfs-site.xml
<configuration>
<property>
<name>dfs.http.address</name>
<value>Ip of NN:50070</value>
</property>
<property>
<name>dfs.secondary.http.address</name>
<value>Ip of SNN:50090</value>
</property>
<property>
<name>fs.checkpoint.dir</name>
<value>/data/check</value>
</property>

<property>
<name>fs.checkpoint.edits.dir</name>
```

```
<value>/data/edits</value>
</property>
```

To force checkpointing , run below command

[root@secondarynn~]# **hadoop secondarynamenode -checkpoint force**

And if you want to automate check point in some time like per 1 hours

[root@secondarynn~]# **vim /etc/hadoop/hdfs-site.xml**

```
<property>
<name>fs.checkpoint.period</name>
<value>3600</value>        ------ // Comment – 3600 Seconds
</property>
```

[root@secondarynn~]# **hadoop-daemon.sh start secondarynamenode**

How to check pointing without restart namenode or SNN :

[root@nn~]# **hadoop dfsadmin –safemode enter**

[root@nn~]# **hadoop dfsadmin -saveNamespace**

Get the list of all live task tracker

```
# hadoop job -list-active-trackers
#hadoop job -list
#hadoop job -list all
 #hadoop job -status job_201406132048_0010
#hadoop jar /usr/share/hadoop/contrib/streaming/hadoop-streaming-1.2.1.jar  -mapper cat -reducer 'wc -l'
-input /passwd  -output /out5
```

```
# vim /etc/hadoop/mapred-site.xml
<property>
```

```
<name>mapred.hosts.exclude</name>
<value>/tmp/maphosts</value>
</property>

<property>
<name>mapred.hosts</name>
<value>/tmp/maphostsallow</value>
</property>
```

Note: must write dns hostname in above file to work

ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01

Real DataSets

ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/2014/

http://grouplens.org/datasets/movielens/

http://aws.amazon.com/datasets

http://aws.amazon.com/publicdatasets/

Hadoop Version 2

Extract hadoop package

```
# tar -xvzf hadoop-2.6.4.tar.gz
# mv hadoop-2.6.4 /hadoop2
```

Set java path

```
# export  JAVA_HOME=/usr/java/jdk1.7.0_79
# export  PATH=$JAVA_HOME/bin:$PATH
```

Set hadoop path

```
# export  HADOOP_HOME=/hadoop2
# export  PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
```

Verify

```
# hadoop version
```

```
# cat hdfs-site.xml
<configuration>
<property>
<name>dfs.name.dir</name>
<value>file:/data/nn</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:/data/dn</value>
</property>
</configuration>
```

```
# hdfs namenode -format
# hadoop-daemon.sh  start namenode
# hadoop-daemon.sh  start datanode
# hdfs dfsadmin  -report
# hdfs dfs  -ls /
```

```
# mv  mapred-site.xml.template mapred-site.xml
```

MR2 support 3 framework named:

- **Local : only run locally and only require master service to run (resourcemanager)**
- **Classic : run MR1 framework**
- **Yarn : run on multinode cluster require nodemanager, and require container service for mapreduce_shuffle at nodemanger side**

Run MR2 locally :

```
# cat mapred-site.xml
<configuration>
```

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

# yarn-daemon.sh start resourcemanager

Run MR2 with YARN :
# cat mapred-site.xml

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

# cat yarn-site.xml

```
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
</configuration>
```

# yarn-daemon.sh start resourcemanager
# yarn-daemon.sh start nodemanager

Run test job
# yarn jar /hadoop2/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.4.jar wordcount /input.txt /output
Port of RM:
8025 : resource tracker port : NM connect to RM to send heartbeat message
8032 : client connect to RM to submit job
8088 : Web management port
8030: RM scheduler port, NM continuous send status info to RM during job runs

How to Configure Multi node MR2 cluster with YARN :

At RM (master) side conf:

```
# vim yarn-site.xml
<property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>masterip:8025</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>masterip:8030</value>
  </property>
```

At RM(master) side conf:

```
# vim mapred-site.xml
```

No property required

At NM(slave) side conf:

```
# vim yarn-site.xml
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>

  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>masterip:8025</value>
  </property>
# vim mapred-site.xml
```

No configuration required

At Client side:

(this is compulsory to conf at side, otherwise client job will run on local mode)

```
# vim  mapred-site.xml
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
```

```
# vim  yarn-site.xml
<property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>masterip:8025</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>masterip:8030</value>
  </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>masterip:8032</value>
  </property>
```

```
# vim  core-site.xml
<property>
        <name>fs.default.name</name>
        <value>hdfs://namenodeip:9001</value>
    </property>
```

Check no. of NM connected to RM:

```
# yarn version
# yarn node -list –all
```

Total number of job (application) running:

```
# yarn application  -list
```

Or

```
# hadoop job -list all
```

Hive

Download binary:
# wget http://mirror.fibergrid.in/apache/hive/hive-1.2.1/apache-hive-1.2.1-bin.tar.gz
# tar -xvzf apache-hive-1.2.1-bin.tar.gz
# mv apache-hive-1.2.1-bin /hive
# export HIVE_HOME=/hive
# export PATH=$HIVE_HOME/bin:$PATH

To support Hive in hadoop version 2, you need to enable below feature:
# export HADOOP_USER_CLASSPATH_FIRST=true

Create database:

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

Drop database:

```
DROP DATABASE|SCHEMA [IF EXISTS] database_name [RESTRICT|CASCADE];
```

Table Created:

```
hive> CREATE TABLE IF NOT EXISTS tablename ( id int, name String,
salary String, destination String)
COMMENT ‘any details’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE;
```

Data load into table:

```
hive> LOAD DATA LOCAL INPATH '/home/vimal/db.txt'
OVERWRITE INTO TABLE tablename;
```

Drop Table:

```
DROP TABLE [IF EXISTS] table_name;
```

Set current db display on prompt:
hive> set hive.cli.print.current.db=true

Run OS cmd in hive prompt:
hive (default)> !date ;

Run HDFS cmd inside hive :
hive (default)> dfs -ls /;
hive (default)> dfs -help;

Describe DB:
hive (default)> desc database lw;

Change location of DB created:
hive > create database lw1 location '/otherlocation';

Describe in details table:
hive> desc extended tablename;
hive> desc formatted tablename;

Create external table :
hive> create external table myextable (id int) row format delimited fields terminated by '\t' stored as textfile location '/sharelocation';

hive> create table passwd (uname string, pass string, uid int, gid int, gecos string, homedir string, shell string) row format delimited fields terminated by ":" lines terminated by "\n" stored as textfile;
hive> load data local inpath "/etc/passwd" overwrite into table passwd;
hive> select uid from passwd where shell="/bin/bash";

Count job that to be executed in mapreduce:
hive> select count(uid) from passwd where shell="/bin/bash";

How to save output into some directory :
hive> insert overwrite directory '/p1' select count(uid) from passwd where shell="/bin/bash”;

save output into CSV format:
hive> insert overwrite local directory '/output' row format delimited fields terminated by ',' select uname,uid from passwd where shell="/bin/bash";

Pig

Download Binary:

```
# wget http://mirror.fibergrid.in/apache/pig/latest/pig-0.15.0.tar.gz
# tar -xvzf pig-0.15.0.tar.gz
# mv pig-0.15.0  /pig
# export  PIG_HOME=/pig
# export  PATH=$PIG_HOME/bin:$PATH
```

At client side, where PIG installed:

```
# vim mapred-site.xml
<property>
  <name>mapreduce.jobhistory.address</name>
  <value>RMip:10020</value>
  <description>Host and port for Job History Server (default 0.0.0.0:10020)</description>
</property>
```

At RM side, start history server:

```
# /hadoop2/sbin/mr-jobhistory-daemon.sh   start historyserver
```

Start PIG grunt Shell:

```
#pig –x mapreduce
Grunt>
```

```
Relation_name = LOAD 'Input file path' USING function as schema;
```

```
grunt> user = LOAD 'hdfs://NNip:9001/passwd' USING PigStorage(':')  as (name:chararray, passwd:chararray,
uid:int, gid:int, comment:chararray, homedirectory:chararray, shell:chararray);
grunt> DUMP user;
```

```
STORE Relation_name INTO ' required_directory_path ' [USING function];
```

```
grunt> STORE user INTO 'hdfs://NNip:9001/output';
grunt> Dump Relation_Name
```

```
grunt> Describe Relation_name
```

```
grunt> explain Relation_name;
```

```
grunt> illustrate Relation_name;
```

```
grunt> Group_data = GROUP Relation_name BY age;
```

grunt> groupuserbyshell = group user by shell;
grunt> dump groupuserbyshell;

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

grunt> shell_user = FILTER user by shell == '/bin/bash';
grunt> DUMP shell_user;

```
grunt> Relation_name2 = FOREACH Relatin_name1 GENERATE (required data);
```

grunt> Get_data = FOREACH user GENERATE name,uid;
grunt> DUMP Get_data;

```
grunt> Relation_name2 = ORDER Relatin_name1 BY (ASC|DESC);
```

grunt> order_by_uid = ORDER user BY uid DESC;
grunt> DUMP order_by_uid;

```
grunt> Result = LIMIT Relation_name required number of tuples;
```

grunt> l = LIMIT order_by_uid 4 ;
grunt> DUMP l;

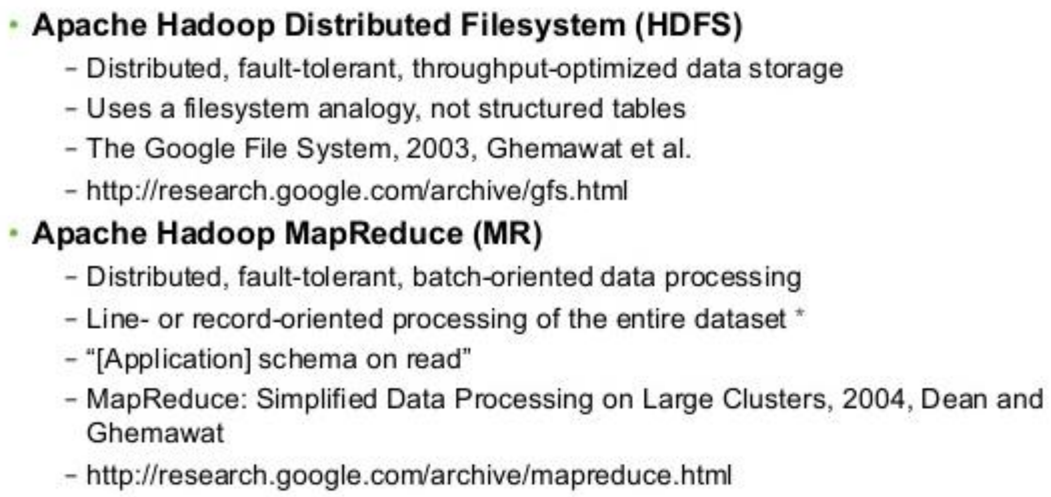
Local mode	MapReduce mode
$ pig -x local **Sample_script.pig**	$ pig -x mapreduce **Sample_script.pig**

```
grunt> exec /sample_script.pig
```

```
$ pig -x mapreduce hdfs://localhost:9000/pig_data/Sample_script.pig
```

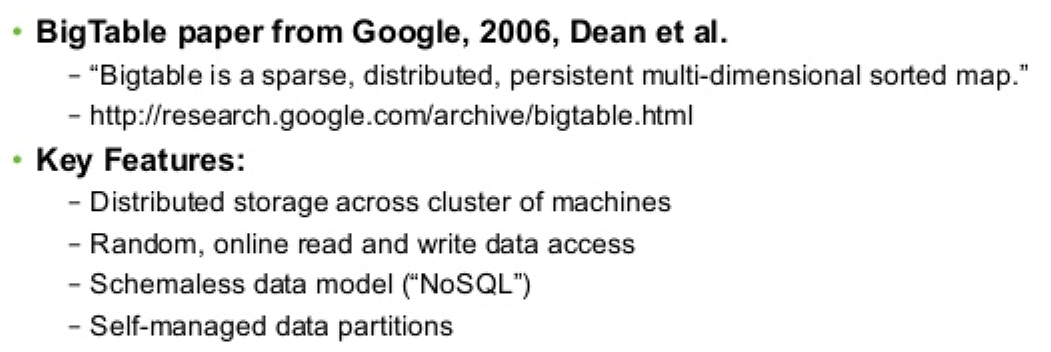
Calculate total user with bash shell:
grunt> shell_user = FILTER user by shell == '/bin/bash';
grunt> group_all = Group shell_user All;
grunt> c = foreach group_all Generate COUNT(shell_user.shell);
grunt> dump c;

HBase



- **Apache Hadoop Distributed Filesystem (HDFS)**
  - Distributed, fault-tolerant, throughput-optimized data storage
  - Uses a filesystem analogy, not structured tables
  - The Google File System, 2003, Ghemawat et al.
  - http://research.google.com/archive/gfs.html
- **Apache Hadoop MapReduce (MR)**
  - Distributed, fault-tolerant, batch-oriented data processing
  - Line- or record-oriented processing of the entire dataset *
  - "[Application] schema on read"
  - MapReduce: Simplified Data Processing on Large Clusters, 2004, Dean and Ghemawat
  - http://research.google.com/archive/mapreduce.html

About Hbase



- **BigTable paper from Google, 2006, Dean et al.**
  - "Bigtable is a sparse, distributed, persistent multi-dimensional sorted map."
  - http://research.google.com/archive/bigtable.html
- **Key Features:**
  - Distributed storage across cluster of machines
  - Random, online read and write data access
  - Schemaless data model ("NoSQL")
  - Self-managed data partitions

- HBase is a *distributed* database
- There is a single HBase master node and multiple region servers
- HBase is a *column-oriented* data store
- HBase is a type of "NoSQL" database
- HBase utilizes ZooKeeper (a distributed coordination service) to manage region assignments to region servers, and to recover from region server crashes by loading the crashed region server's regions onto other functioning region servers.
- Regions contain an in-memory data store (MemStore) and a persistent data store (HFile)
- All regions on a region server share a reference to the write-ahead log (WAL) which is used to store new data that hasn't yet been persisted to permanent storage and to recover from region server crashes
- HBase clusters expand by adding RegionServers that are hosted on commodity class servers.
- As a table grows, more and more regions are created and spread across the entire cluster.
- When clients request a specific row key or scan a range of row keys, HBase tells them the regions on which those keys exist, and the clients then communicate directly with the region servers where those regions exist. This

design minimizes the number of disk seeks required to find any given row, and optimizes HBase toward disk transfer when returning data. This is in contrast to relational databases, which might need to do a large number of disk seeks before transferring data from disk, even with indexes.

- HBase isn't suitable for every problem.
- First, make sure you have enough data. If you have hundreds of millions or billions of rows, then HBase is a good candidate. If you only have a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle.

HBase features of note are:

- Strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.

What is The Difference Between HBase and Hadoop/HDFS?

HDFS is a distributed file system that is well suited for the storage of large files. Its documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files.

HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed lookups.

#start-hbase.sh
#hbase shell
List table name

- List

Create table name "lw" with column family "name", "mob" , "gender"

- Create 'lw' , 'name' , 'mob' , 'gender'

Insert into db "lw",
Syntax is "**put <table name> , <key> , <column family : qualifier>, </value>**"

- Put 'lw' , 'row1' , 'name:a' , 'amit'
- Put 'lw' , 'row2' , 'name:b'
- Put 'lw', 'row3', 'name:c' , 'rahul'

Check data inserted

- Scan test

Data files :
# cd /tmp/hbase-hadoop/hbase/-ROOT-/

- Sqoop tool to import from rdbms into hbase

Sqoop Configuration

- Step 1: Get software
- Get Sqoop from lw lab server or get from web at
- http://mirror.reverse.net/pub/apache/sqoop/1.4.5/sqoop-1.4.5.bin__hadoop-1.0.0.tar.gz
-
- Step 2: Extract and install sqoop
- **# mkdir /sqoop**
- **# tar –xvzf sqoop-1.4.5.bin__hadoop-1.0.0.tar.gz -C /sqoop**
-
- Step 3 : Setup Environment variable
-
- **# export SQOOP_HOME=/sqoop/sqoop-1.4.5.bin__hadoop-1.0.0/**
- **# PATH=$SQOOP_HOME/bin:$PATH**
- **# export HADOOP_COMMON_HOME=/usr**
- **# export HADOOP_MAPRED_HOME=/etc/Hadoop**
-
- *Note: "/usr" is with respect to where "bin/hadoop" command exists*
- *"/etc/hadoop" is the location where mapred-site.xml file exists*
- *Make above variable permanent in /root/.bashrc file*
-
- Check setup in working or not, by below cmd
- **# sqoop version**
- **# sqoop help**
-
- Step 4: Setup JDBC driver RDBMS
- Download JDBC driver, according to RDBMS use, for example if you want to use mysql, then go to
- mysql site and download mysql connector for JDBC, get the below link :
- http://dev.mysql.com/downloads/connector/j/
- And download : **mysql-connector-java-5.1.35.tar.gz**
- Extra mysql-connector-java, and jdbc jar
-
-
- **# tar –xvzf mysql-connector-java-5.1.35.tar.gz**
- **# cd mysql-connector-java-5.1.35**
- Get the jar of jdbc driver and copy in Sqoop lib folder ,
- **# cp mysql-connector-java-5.1.35-bin.jar /sqoop/sqoop-1.4.5.bin__hadoop-1.0.0/lib**
-
- Step 5: Step mysql DB
- **# yum install mysql-server**
- **# service mysqld restart**
- **# mysql-admin –u root password 'lw'**
- **# mysql –u root –p**
- **Mysql> create database lw;**

- **Mysql> use lw;**
- **Mysql> create table student (id int(5) , name char(200));**
- **Mysql> insert into student values (1, 'raj');**
- **Mysql > quit;**
- Step 6: Connect sqoop with mysql RDBMS and run sqoop command
- Get the list of databases created in mysql
- **# sqoop list-databases --connect jdbc:mysql://127.0.0.1/**
- *Note it only show anonymous database named 'test' and 'information_schema', bcoz we havn't pass*
- *the user and password*
- **# sqoop list-databases --connect jdbc:mysql://127.0.0.1/ --username root --password lw**
- *Now it show all database, include 'lw' that we create above*
- To get list of tables in "lw" DB
- **# sqoop list-tables --connect jdbc:mysql://127.0.0.1/lw --username root --password lw**
-
- *Note: use –P instead of –password option, it prompt for password*
-
- **How to import data from RDBMS to HDFS**
- To get data from 'lw' database from 'student' table and put into /testsqoop folder in hdfs with only
- 'one' mapper
-
- **# sqoop import --connect jdbc:mysql://127.0.0.1/lw --username root --password lw --table**
- **student --target-dir /testsqoop -m 1**
-
- But if you want to use more than one mapper, like below command, it give error, bcoz it means we
- want to mapper to work in parallel, so we need to split our table into partition, so we need to give
- option "split-by" by column name id tables
-
- **# sqoop import --connect jdbc:mysql://127.0.0.1/lw --username root --password lw --table**
- **student --target-dir /testsqoop2 -m 2**
-
- **# sqoop import --connect jdbc:mysql://127.0.0.1/lw --username root --password lw --table**
- **student --target-dir /testsqoop2 -m 2 --split-by id**