

Setting up a Hadoop Cluster with Cloudera Manager and Impala

Comparison between Hive and Impala

University of St. Thomas

Frank Rischner

Abstract

This paper describes the results of my independent study class. The independent study had two purposes: first, the installation and configuration of a Hadoop cluster. The GPS department's current cluster is built on top of Ubuntu Linux and does not use Cloudera Manager. There were also some limitations with the internal networking design. Administering a Hadoop cluster with Cloudera Manager makes the administration job easier, compared with editing XML configuration files by hand. In this project, I moved to CentOS, Cloudera Manager, and a new internal network design.

The second part of this Independent Study project was to explore Cloudera Impala and compare it with Hive. Impala is a new product in the Hadoop ecosystem, developed by Cloudera. Impala provides real-time queries against HDFS data, and is an alternative to the batch-oriented Hive component. Currently it is in the beta phase and provides only limited functionality.

1 Table of Contents

1.	Existing solution	4
2.	Implementation	5
2.1	Installing CentOS.....	5
2.2	Installing Cloudera Manager.....	6
2.3	Installing CDH.....	7
2.4	Adding a host to the cluster.....	12
2.5	Scripts for managing the cluster	13
2.5.1	createuser.sh:	13
2.5.2	deleteuser.sh	14
2.5.3	checkdisks.sh	14
2.5.4	checksystem.sh	14
2.6	Updating Cloudera Manager and Impala.....	14
2.7	Testing the cluster	15
3.	Impala	19
3.1	Sample run with data from neuro rat experiment	19
3.2	Setting up Impala	20
3.3	Running Neuro experiment with Impala	20
4	Conclusions.....	22
5	Bibliography.....	23

1. Existing solution

Currently the Graduate Program in Software at the University of St. Thomas is using a cluster running Hadoop with CDH4 Update 3 under Ubuntu Linux. Each machine has two quad core processors, 12GB of RAM and a 250GB and a 1TB hard disk. The Name Node has 18 GB of RAM and two 1TB disks mirrored in a RAID1.

The Cloudera Manager is not installed on this cluster, so administration is time consuming. The configuration was done by editing the configuration files by hand, which easily can lead to errors.

The current cluster runs MapReduce (MRv1) Version 1, which is highly recommended. For this project Dr. Rubin decided to install YARN as well as MRv1.

In October 2012 Cloudera released Impala [1]. Impala is a real-time query engine on top of the Hadoop stack. Impala supports a large subset of SQL and HiveQL. As opposed to Hive, Impala does not rely on MapReduce. A downside of Impala is that it currently only supports queries not larger than the installed memory, so sufficient memory is needed on the machines.

2. Implementation

To achieve the goal of setting up a new cluster for GPS, first CentOS had to be installed. Usually it is possible to install the CDH on other Linux Systems, but in order to install Impala on top of Hadoop, RedHat or CentOS Linux is required.

2.1 Installing CentOS

For this project, the system is set up to run on five Sun Fire X2000 Machines with 12GB of RAM each. The Master has two 1TB hard drives, configured in a software raid. Slaves have one 250GB drive with the system on it and one 1TB disk.

Since the machines don't have a built-in DVD drive the installation had to be done with a USB thumb drive. The Installation image was created with Unetbootin [2]. The installation was a plain vanilla installation of CentOS 6.3; no extra packages have been installed. Since CentOS came with X11, the GUI had to be turned off by setting the default runlevel to 3 in the file */etc/inittab*.

The Master has two configured network cards; one is a public interface, configured with *140.209.113.60* and one is a private interface configured with *192.168.2.1*. In order to have them working with the classic Linux networking, the service *NetworkManager* had to be turned off and the service *network* had to be enabled at boot time. This has been done by the following commands:

```
$chkcfg NetworkManager off
$chkcfg network on
```

On each machine the *sshd* server had to be enabled and in addition root login over ssh was enabled on the slave machines. In order to run a Hadoop cluster correctly, the time on its machine has to be synchronized. The systems have been set up to use a Network Time Protocol server on the Internet.

Cloudera Manager requires selinux to be disabled. This was done by editing the file */etc/selinux/config*. Also, in order to communicate between the slaves and masters the iptables firewall was disabled on the slave nodes. Otherwise, some services would throw a *NoRouteToHostException*.

After installing and configuring one slave, the slave was cloned with Clonezilla [3]. The cloning process runs in about 15 minutes for the current setup. On the cloned machine the file */etc/udev/rules.d/70-persistent-net* had to be deleted. This is necessary in order to assign the network interfaces to the correct device. On some machines it still happened that device *eth0* and *eth2* were switched, in this case I had to edit the file manually. It also was necessary to edit the hostname and the IP address.

Since the master has the public interface, it was necessary to configure a firewall with iptables. I set the rules for incoming traffic on the public interface on DENY, except ports 22 and 50030 for ssh and the Hadoop JobTracker page. The clients use the master as a gateway to get updates and to update the time. I had to set up a FORWARDING rule which uses NAT. The only traffic that is forwarded to the private network is traffic where the connection has been established from the internal network. On the private network interface I had to enable the mysql port, so the nodes can access the Hive metastore. The following Listing is the set of iptables rules:

```
#!/bin/bash

iptables -P INPUT ACCEPT
iptables -F INPUT
iptables -P OUTPUT ACCEPT
iptables -F OUTPUT
iptables -P FORWARD DROP
iptables -F FORWARD
iptables -t nat -F

iptables -A INPUT -i eth1 -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -i eth0 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport ssh -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --dport 50030 -j ACCEPT
iptables -A INPUT -i eth1 -p tcp --dport 3006 -j ACCEPT

iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -A FORWARD -i eth0 -o eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT

iptables -A INPUT -i eth0 -j DROP
iptables -nvL
```

2.2 Installing Cloudera Manager

In order to install Cloudera Manager I had to download the binary from Cloudera's website. After making the file executable, I simply run it.

```
# chmod +x cloudera-manager-installer.bin

# ./cloudera-manager-installer.bin
```

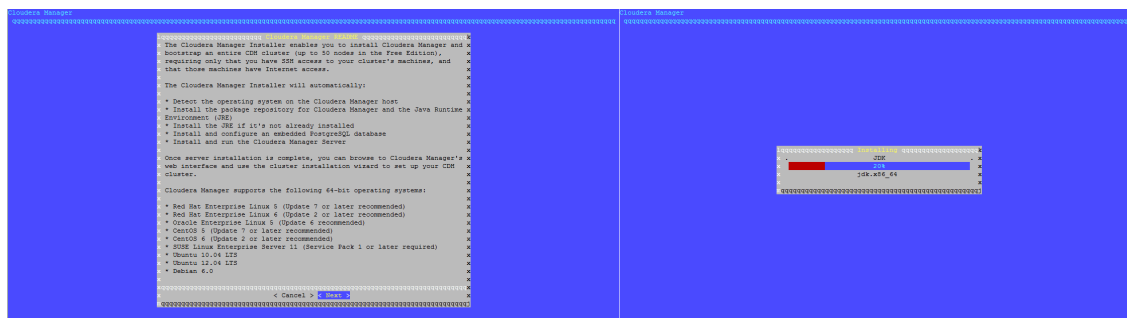
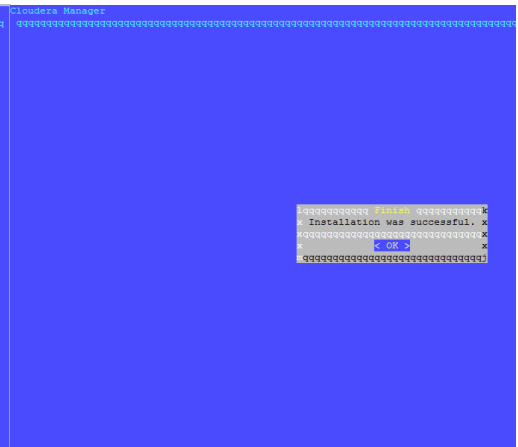


Figure 2-1: Cloudera Manager Installation

Figure 2-2: Installing Components

After accepting the licenses from Cloudera and Oracle (see Figure 2-1), the program installs the Java SDK and the components of the manager.



2.3 Installing CDH

Once Cloudera Manager is installed we can access it in a web browser on port 7180 (see Figure 2-5).

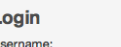
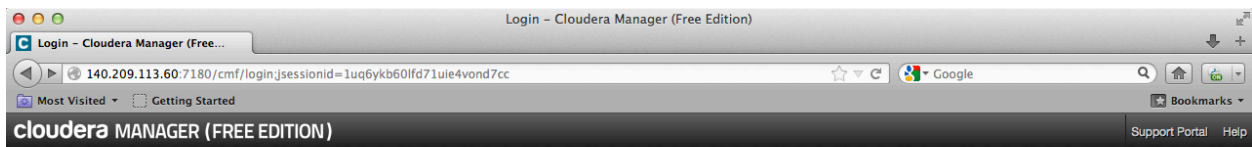


Figure 2-5: Login to Cloudera Manager

The password for the user admin has been changed; please take the password from the attached document with the login data.

The next screen asks for a license file for the full Cloudera Manager. Since we only use a limited amount of machines with this cluster we just installed the free edition. Then we are asked to enter the hostnames of the machines we want to manage (see Figure 2-6). Alternatively it is possible to enter IP addresses. In this project the hostnames hc1.gps.stthomas.edu, n2.gps.stthomas.edu, n3.gps.stthomas.edu, etc. have been used.

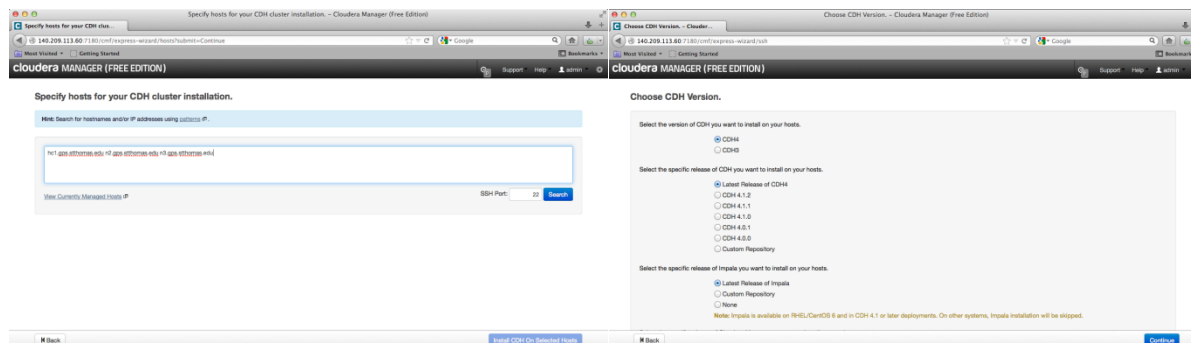


Figure 2-6: Select hosts

Figure 2-7: Selection of the CDH Version

After selecting the distribution, the version and the version of Impala (see Figure 2-6) CDH gets installed on the previous selected hosts (see Figure 2-8).

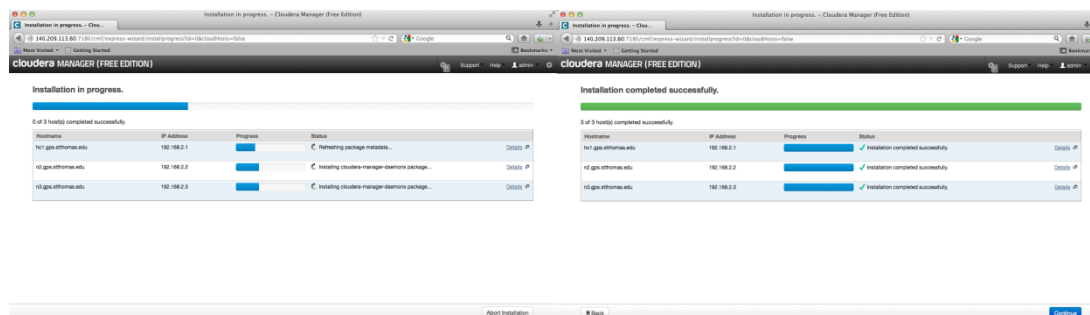


Figure 2-8: Installation on selected Hosts

Figure 2-9: Installation complete

Once the installation is complete (see Figure 2-9), the services which should be installed have to be chosen. For this project HDFS, HBase, YARN, MapReduce, Zookeeper and Oozie have been selected. Impala was added after the initial setup (see Figure 2-10).

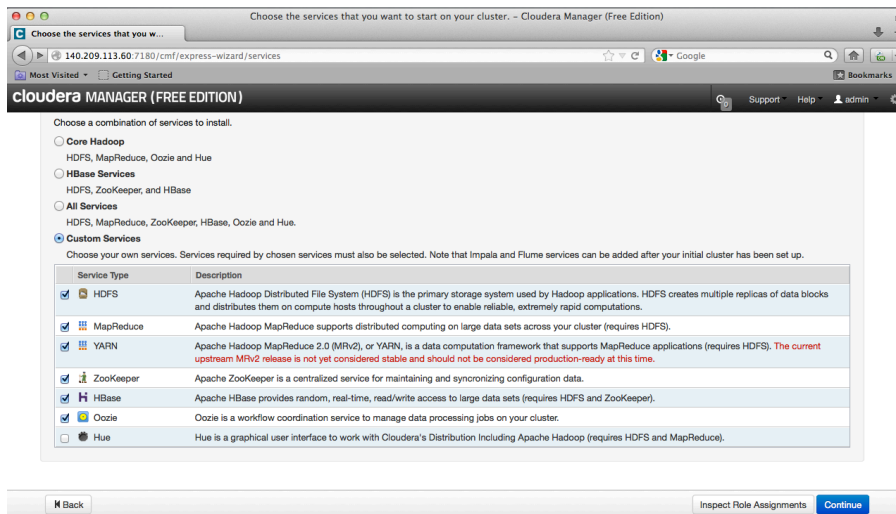


Figure 2-10: Selection of Services

After the selection of the services, roles had to be assigned to each node (see Figure 2-11). Initially I had the name node hc1 also assigned to all the other available roles. During the first tests I discarded this decision and took away the datanode, regionserver, tasktracker, and nodemananager roles, because that running all those roles on a single machine consumed too much memory.

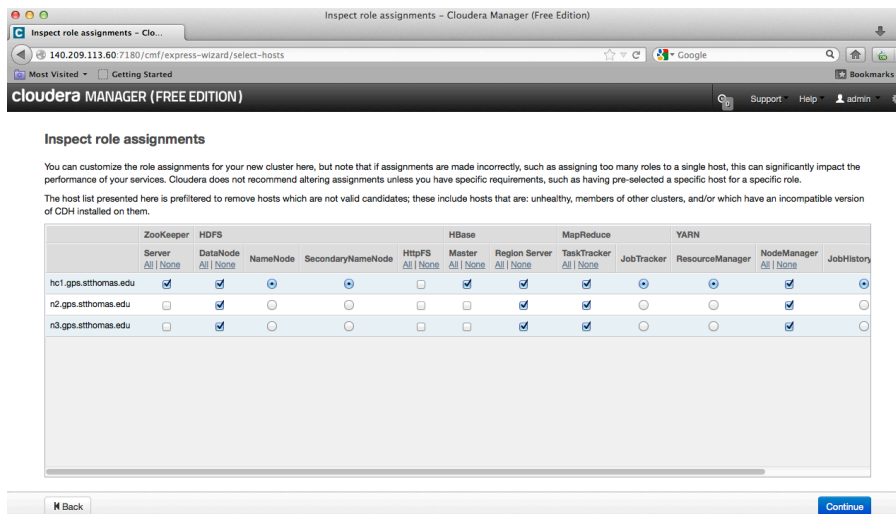


Figure 2-11 Role assignment

After the role assignment, I had to review the configuration changes. By clicking on *Continue* the changes are confirmed and the services get started (see Figure 2-12).

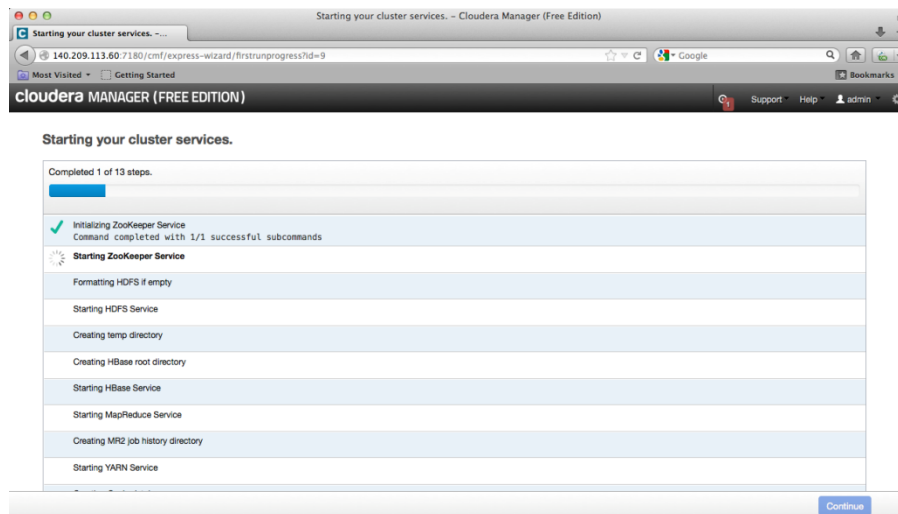


Figure 2-12: Starting the services

I only made a few changes to the default configuration. The HDFS Blocksize has been set to 64MB instead of the default value of 128MB. As I mentioned earlier, the nodes have two hard disks. In order to make use of both of them, I had to change the values for `dfs.datanode.data.dir` to `/disks/sdb1/dfs.dn` and `/disks/sda3/dfs.dn`. I also mentioned that YARN has been installed. It is not recommended to have both services running at the same time, so I turned YARN off. The Interface of Cloudera manager is really intuitive, so it is easy to turn on or off services (see Figure 2-13).

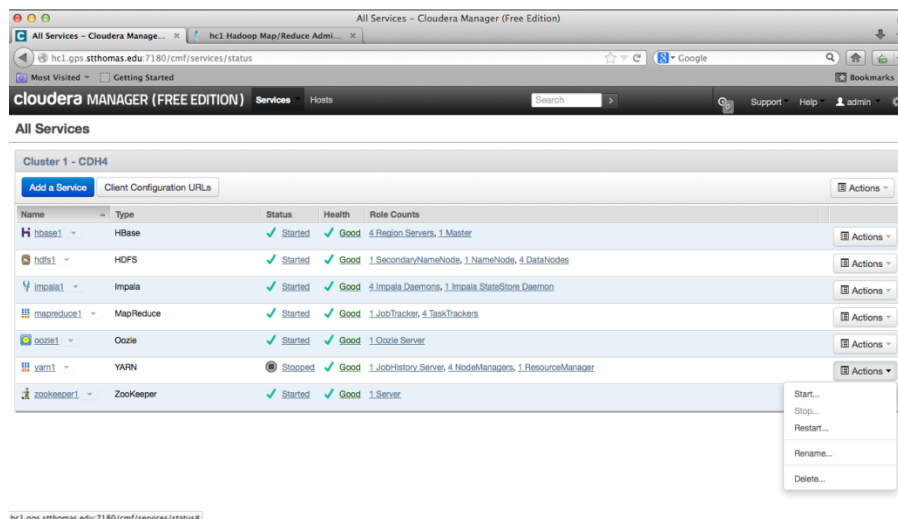


Figure 2-13 CM All Services

For the Impala configuration please see Chapter 3.2.

At this point Pig and Hive services are not managed by Cloudera Manager. These services have to be configured by editing the xml files. For Hive, a good description of how to setup the Hive metastore can be found at <https://ccp.cloudera.com/display/CDH4DOC/Hive+Installation>.

First the package mysql-conector has to be installed. After the Installation a symbolic link in the Hive directory has to be created.

```
$ sudo yum install mysql-connector-java
$ ln -s /usr/share/java/mysql-connector-java.jar /usr/lib/hive/lib/mysql-connector-java.jar
```

In mysql, the database for the Hive metastore has to be created and the hive-schema tables have to be imported:

```
$ mysql -u root -p
Enter password:
mysql> CREATE DATABASE hivemetastoredb;
mysql> USE hivemetastoredb;
mysql> SOURCE /usr/lib/hive/scripts/metastore/upgrade/mysql/hive-schema-0.9.0.mysql.sql;
```

After the tables have been imported a user for the metastore is created by the following commands:

```
mysql>CREATE USER 'hive'@'%' IDENTIFIED BY 'hive';
```

Since all nodes have to access the metastore, access has to be granted for all hosts in the network:

```
mysql> GRANT ALL PRIVILEGES ON hivemetastoredb.* TO hive@'192.168.2.%' WITH GRANT OPTION;
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

Each node has to have the following entries in the file */etc/hive/conf/hive-sites.xml*:

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://hcl1.gps.stthomas.edu/hivemetastoredb</value>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>

<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>hive</value>
</property>

<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>hive</value>
</property>
```

These entries serve to establish the connection to the hive metastore.

Adding a host to the existing cluster using Cloudera Manager is straightforward, the steps are similar to the ones in Section 2.3.

After a reboot of the system the udev configuration file should be back. If the interface configuration does not match, the file has to be edited so that the interface eth0 is the first Ethernet device.

Figure 2-15: Adding host, enter the name

12

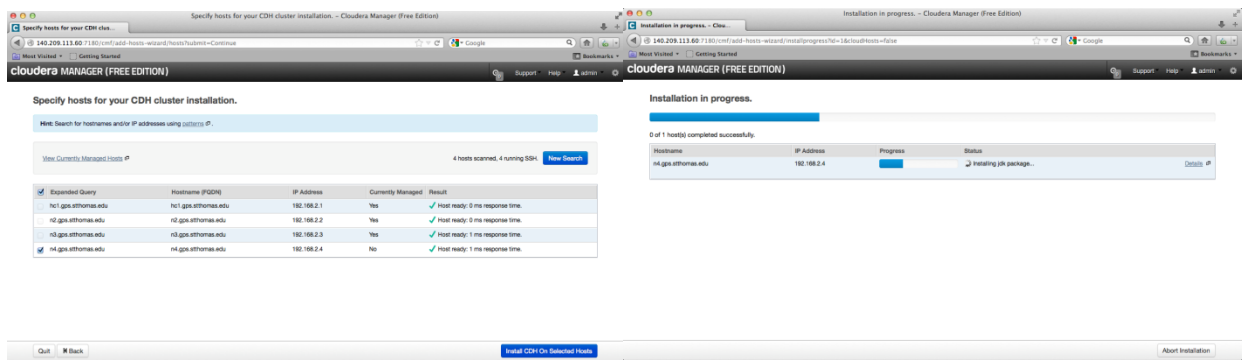


Figure 2-16: Select Hosts

Figure 2-17: Installing Components

Once the components are installed and the machine is added to the cluster, the roles have to be applied for every service (see Figure 2-18: Role assignment). For example, if the new host should run hbase the hosts needs to be added in the HBase service.

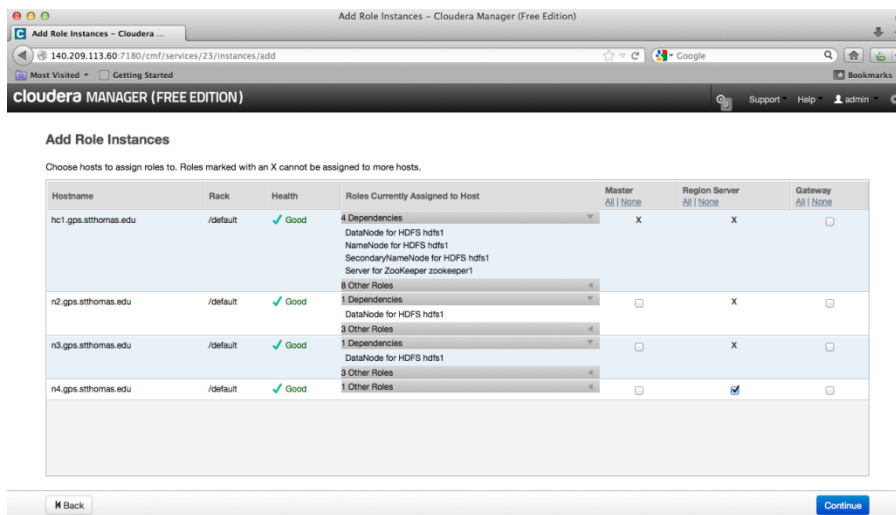


Figure 2-18: Role assignment

2.5 Scripts for managing the cluster

Even though Cloudera Manager provides almost everything to manage the cluster, Dr. Rubin gave me some of the scripts he used to manage the old cluster.

2.5.1 createuser.sh:

This script is for creating a user on the master machine. It takes the name for the user as a parameter. The home directory is created and the `.bashrc` file gets copied from the home directory of the user `hdfs`. After changing the permissions of the newly created home directory, an `hdfs` home directory under the path `/user/` is created. Here the permissions will be changed again as will the owner of the newly created

directory.

The last step is changing the password for the new user.

2.5.2 deleteuser.sh

Just as the *createuser.sh* script did, the *deleteuser.sh* script takes a username as a parameter. The purpose of this script is to remove a user from the system. The script deletes the home directory, the hdfs directory and then the user himself.

2.5.3 checkdisks.sh

Checkdisks.sh is a script which is put in each hdfs home directory on the slaves. It uses the command *smartctl -a -d ata /dev/diskname*. In order to make it run, we had to make sure the package *smartmontools* is installed. The purpose of this script is to check the health of the disks.

2.5.4 checksystem.sh

Checksystem.sh is the script which calls *chckdisks.sh* on each slave node. This is done by the command: `pdsh -R ssh -w n[2-24] "sudo /home/hdfs/checkdisks.sh"`. In order to make this work, first the package *pdsh* had to be installed [4].

To accomplish this, the following steps had to be done:

Download the latest rpmforge-release rpm from:

http://apt.sw.be/redhat/el6/en/x86_64/rpmforge/RPMS/

Install rpmforge-release rpm:

```
# rpm -Uvh rpmforge-release*rpm
```

Install pdsh rpm package:

```
# yum install pdsh
```

The script requires ssh login without a password. For the user hdfs I created a private/public key pair and copied the public key in the file `~/.ssh/authorized_keys` on each node. I used the *ssh-copy-id* command to perform this task. It is important that the `.ssh` directory has the permission 700 and each file in it the permission 600.

By default the home directory of the user *hdfs* is `/var/lib/hadoop-hdfs/`. This directory caused some problems with the login over ssh, so the home directory of *hdfs* has been moved to `/home/hdfs/`.

Since the script requires the remote execution of a sudo command, the sudoers configuration file `/etc/sudoers` had to be modified. Sudo usually requires a TTY terminal. In order to be able to execute a remote sudo command the line *Defaults requiretty* had to be commented out.

2.6 Updating Cloudera Manager and Impala

On January 18th 2013 Cloudera released Cloudera Manager 4.1.3 and Impala 0.4. At this point we had only five machines in the cluster, so we decided to do the update while we have a small cluster.

The update process is pretty straightforward. I followed the steps in [5]. The following three commands have to be run with root privileges.

Stopping the Cloudera manager service:

```
# service cloudera-scm-server stop
```

Cleaning up the yum cache directories:

```
# yum clean all
```

Updating Cloudera Manager:

```
# yum update --disablerepo='*' --enablerepo=cloudera-manager
```

After Cloudera Manager was updated, CDH had to be updated. The process is similar to installing CDH with Cloudera Manager. I had to use the web interface and then confirm the update. After the installation of the updates, Cloudera Manager performed the *Host Inspector* test, in which it checks all installed components and their versions.

2.7 Testing the cluster

To make sure the cluster works the way it should, I took the homework from the SEIS 785 Big Data class, let it run on the new cluster and compared the results.

The functionality of the HBase Service was especially important to Dr. Rahimi. I followed the examples from the lecture slides [6] for the Hive/Hbase integration. First I had to create an HBase table:

```
# hbase shell
hbase(main):001:0> create 'hbaseClassRatings', 'cf1'
hbase(main):002:0> scan 'hbaseClassRatings'
```

The lecture slides provide the old way to start Hive with HBase Integration. This way worked on the old cluster as well as on the virtual machines provide for the class. Both run CDH 3.

```
# hive --auxpath /usr/lib/hive/lib/hive-hbase-handler-0.9.0-
cdh4.1.2.jar:/usr/lib/hbase/hbase.jar:/usr/lib/hbase/lib/zookeeper.jar
-hiveconf hbase.master=localhost:60000
```

In CDH 4 the way to start Hive with HBase integration has changed. The jars in the *auxpath* are no longer separated by a colon; now they have to be separated by a comma. The *hiveconf* parameter has changed too. Instead of setting the *hbase.master* value, now the value for *hbase.zookeeper.quorum* has to be set.

```
# hive --auxpath /usr/lib/hive/lib/hive-hbase-handler-0.9.0-
cdh4.1.2.jar,/usr/lib/hbase/hbase.jar,/usr/lib/hbase/lib/zookeeper.jar
-hiveconf hbase.zookeeper.quorum=hcl.gps.stthomas.edu
```

The lecture slides now directed me to create a staging table into which the data is first loaded. After creating the table, I had to load the data from a local path.

To check the data was loaded, I did a simple `select * from Ratings_staging;`

```
hive> CREATE TABLE Ratings_staging
> (sid INT, CRN string, rating INT)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;

hive> LOAD DATA LOCAL INPATH
> '/home/frank/hive/classratings.txt'
> INTO TABLE Ratings_staging;
```

After this I created the table which is stored in HBase, called *hiveClassRatings*. Important here is the *STORED BY* and the *WITH SERDEPROPERTIES* part, since they define the HBaseStorageHandler and the mapping to the Hbase columns. The statement TBLPROPERTIES defines the name of the table which is used.

```
hive> CREATE EXTERNAL TABLE hiveClassRatings
> (sid INT, CRN string, rating INT)
> STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
> WITH SERDEPROPERTIES ("hbase.columns.mapping" =
> ":key,cf1:CRN,cf1:rating")
> TBLPROPERTIES("hbase.table.name" = "hbaseClassRatings");
```

Since there is no data loaded into the table, no result is returned.

```
hive> select * from hiveClassRatings;
OK
Time taken: 0.483 seconds
```

In order to load the data into the HBase table, an Insert Overwrite command was set followed by a Select from the staging table.

```
hive> INSERT OVERWRITE TABLE hiveClassRatings SELECT
> sid, CRN, rating FROM Ratings_staging;
```

To check if the result is loaded, I again did a simple `select * from hiveClassRatings`.

```
hive> select * from hiveClassRatings;
OK
1      SEIS733 4
10     SEIS733 4
11     SEIS733 3
12     SEIS730 5
13     SEIS785 4
2      SEIS785 1
3      SEIS785 3
4      SEIS785 5
5      SEIS785 3
6      SEIS733 2
7      SEIS730 4
8      SEIS785 4
```



```
9          SEIS785 4
Time taken: 0.692 seconds
```

The Pig/Hive integration was also tested based on the example from the lecture slides. First I created an HBase table with the name *names*. Then I loaded the data from HDFS into Pig. After the loading I stored the data into HBase by using the HBaseStorage class.

```
# hbase shell

hbase(main):001:0> create 'names', 'info'

# pig
grunt> raw_data = LOAD 'HBase/input.csv' USING PigStorage( ',' ) AS (
>> listing_id: chararray,
>> fname: chararray,
>> lname: chararray );

grunt> STORE raw_data INTO 'hbase://names' USING
org.apache.pig.backend.hadoop.hbase.HBaseStorage ('info:fname info:lname');
```

To check if the data was stored correctly into HBase I scanned the table *names* and got the expected results.

```
hbase(main):002:0> scan 'names'
ROW                                COLUMN+CELL
 1                                column=info:fname,
timestamp=1359237771297, value= John
 1                                column=info:lname,
timestamp=1359237771297, value= Smith
 2                                column=info:fname,
timestamp=1359237771298, value= Jane
 2                                column=info:lname,
timestamp=1359237771298, value= Doe
 3                                column=info:fname,
timestamp=1359237771298, value= George
 3                                column=info:lname,
timestamp=1359237771298, value= Washington
 4                                column=info:fname,
timestamp=1359237771298, value= Ben
 4                                column=info:lname,
timestamp=1359237771298, value= Franklin
4 row(s) in 0.1750 seconds
```

To check the HBase assignment (assignment 10) it was necessary to create a symbolic link in the */usr/lib/hadoop-0.20-mapreduce/lib/* directory to the *HBase.jar* and the *Zookeeper.jar* files. Otherwise the *MakeTable.jar* file would not execute.

```
cd /usr/lib/hadoop-0.20-mapreduce/lib/
ln -s /usr/lib/hbase/hbase.jar .
ln -s /usr/lib/zookeeper/zookeeper.jar .
```

After creating the symbolic links I could successfully run the sample code.

```
# MYCLASSPATH=/usr/lib/hadoop-0.20-mapreduce/hadoop-core.jar:/usr/lib/hadoop-0.20-mapreduce/hadoop-common.jar:/usr/lib/hbase/hbase.jar:/usr/lib/zookeeper/zookeeper.jar

# javac -classpath $MYCLASSPATH MakeTable.java

# jar cvf MakeTable.jar MakeTable.class

# hadoop jar MakeTable.jar MakeTable
```

Please note the file *hadoop-common.jar* had to be included too, since *hadoop-core.jar* does not have the Writable classes included.

As the last check I ran parts of the questions from exam 2, which also returned the correct results.

3. Impala

Impala is a general-purpose SQL query engine, which runs directly on HDFS or HBase. Other than Hive it does not use MapReduce for its execution. Most Hadoop components are written in Java; Impala is written in C++. Impala utilizes the Hive metadata interface and connects to the Hive metastore, where as Impala does not support Derby. [7] [8]

Since currently no DDL statements are supported by Impala, Hive is needed to create a table before using it. With the General Availability (GA) DDL statements will be supported.

Users submit queries via ODBC or Beeswax Thrift API directly to a random Impala Daemon. The query then gets distributed to all nodes with relevant data. If one node fails or the execution on one node fails the whole query will fail. Each node has an Impala Daemon. A State Store Daemon is run on the NameNode. The State Store provides the name service and the metadata distribution [7].

Impala currently supports a subset of HiveQL and SQL. Order by queries can currently only be used with a Limit. The beta version, which we are using on the cluster, supports neither User Defined Functions nor SerDes, which makes it difficult to just replace Hive with Impala.

Impala can be faster by 4 to 30 times in comparison with Hive, depending on the complexity and the amount of disk I/Os.

3.1 Sample run with data from neuro rat experiment

Dr. Rubin gave me the data and scripts of his neuroscience rat experiment. First we wanted to make sure his old scripts run on the new cluster.

I created a new Linux group with the name *neuro* and added the relevant users to it. Then I created a local folder for the data and the scripts under */neuro/* and changed the group to the newly created *neuro* group. In my home directory I created a folder *NeuroHadoop* which I copied the script files into.

I copied the files from the old cluster to the new one. The readme file recommended setting the variables for *JAVA_HOME*, *HIVE_HOME* and *HADOOP_HOME*; I did set the first two but I did not set *HADOOP_HOME* because this interfered with Hive. The variables have been set in */etc/profile*.

In CDH 4.1.2 with YARN, the jar which includes *org.apache.hadoop.io* is no longer the *hadoop-core.jar* file. Now it is included in *hadoop-common.jar*, which had to be downloaded from the Cloudera repository [9] and put into */usr/lib/hadoop-0.20-mapreduce/*. Also two symbolic links to this file had to be created in the same directory. This step had to be repeated on each node.

On my desktop machine I created a new Eclipse project with the name *NeuroHadoop*. In it I copied the src files and also created a folder called *hlib*. *Hlib* contains the jar files to build the project against. In this case I copied the files *commons-logging-1.1.1.jar*, *hadoop-common-2.0.0-cdh4.1.2.jar*, *hadoop-core-2.0.0-mr1-cdh4.1.2.jar* and *hive-exec-0.9.0-cdh4.1.2.jar*.

In the project's settings I included these four jar files to the build path. The last step was to copy the *build.xml* file into the projects root directory. The *build.xml* gives several options when it is running; one

of them is `upload.sh`. `Upload.sh` loads the created jar files on the master. Since this part of the project was done on a Windows machine, I did only the jar option and uploaded the *build/lib* directory of the project manually.

I followed the steps in the readme file and finally executed the main script with the `nohup` command. `Nohup` is a tool to run shell commands even without a TTY [10].

After the script was executed I compared the results to the ones Dr. Rubin sent me from the old cluster, they matched.

Please note that the experiment was run with only a subset of data.

3.2 Setting up Impala

In order to get the Impala service started some modifications had to be made on the hdfs service. The installation guide [8] suggested setting the value for *dfs.datanode.data.dir.perm* to 750. During the setup I ran into issues with this setting and followed the suggestion of Cloudera Manager itself. The value was now set to 755. The value for *dfs.block.local-path-access.user* was set to the impala user.

The impala user was added to the hadoop group by executing the command

```
# usermod -a -G hadoop impala.
```

Other than hive, impala does not automatically connect to an instance once the shell is started. To start the shell I had to execute the *impala-shell* command.

```
# impala-shell
```

In order to connect to an Impala daemon the connect command is required. It is followed by the host name of the service we want to connect to. It is also possible to use an alias or an IP address.

```
[Not connected] > connect n2.gps.stthomas.edu
Connected to n2.gps.stthomas.edu:21000
[n2.gps.stthomas.edu:21000] >
```

3.3 Running Neuro experiment with Impala

After the Neuroscience experiment ran as expected on the new cluster, I went over all Hive scripts to see which ones would run with Impala. Please remember that UDF are currently not supported, nor are create table statements.

The `nh.sh` script already tracks the execution time of each hive script. My approach was to run the Hive script under Hive, and directly after it to do the same with Impala. Both execution durations would be logged into a file. The old setup uses a custom SerDe for storing the rat data. Impala does currently not support custom SerDes. In order to get some results it was necessary to change the way the scripts store the data. Now they store the results as a text file and not as a binary file. Please see the attached source code to see the changes.

After running a demo with the modified original scripts and checking the results, I decided to follow my initial approach. I copied the *nh.sh* script to *nhImp.sh*. Both scripts send the execution time of the several jobs to a result file.

The original hive script had an alter table statement in it. Since Impala does not support DDL statements at the moment, I created a separate script for this, called *alterrataverage.q*. I let this run in both versions of the jobs.

Impala seems to be very picky in terms of type conversion, so I had to cast the *AVG(convolution)* value to a float value by *CAST(AVG(convolution) AS FLOAT)*.

I let each job run five times and then took the average time. The important results are the *Ratsaverage* and the *RatsAverageImpala* jobs. The Impala job was about 100 seconds faster than the Hive job which is not the expected result. I expected Impala to be at least 100 percent faster but it was only faster by a factor of 1.2. I checked on the Hive version and realized that I did not do the casting. After I changed this and did some runs, the difference between those two was a little clearer. Now the factor was 1.28. The fastest run of the *Insertratsaverage.q* was done in 379 seconds with Impala.

Job	Hive	Impala	Hive with the casting of the AVG value
Clean	17 s	17 s	17 s
Putdata	11.4 s	11.3 s	13 s
SettingsJob	17 s	20 s	17 s
ConvolutionJob	963 s	992.83 s	1345.5 s
Session	7.4 s	7.1 s	7.5 s
Rats	47.8 s	47 s	46 s
Ratsaverage/RatsaveragImpala	580.2 s	474 s	607 s
Ratsstats	69.2 s	67.2 s	68 s
Ratssubset	140.6 s	141.6 s	140 s
Phasebucket	28.6 s	31 s	26.5 s
Result	99.6 s	99.4 s	98.5 s

While there were no significant differences in the execution time of the *insertratsaverage.q* script with either Hive or Impala, a *select count(*) from result* with hive took 19.117 seconds. The same query took 0.20 seconds with Impala which is about 95 times faster.

I tried to re-run the Impala version several times, but could not get a different result.

4 Conclusions

Overall my experiences with Cloudera Manager have been absolutely positive. The installation and configuration process is well documented and straightforward. I only recommend using Cloudera Manager, rather than just editing the configuration files by hand. I changed only a few values compared to the default settings. Starting, stopping or restarting a specific service or all services of the cluster is done by clicking one button.

In my opinion the only thing missing at this time is the ability to configure Hive and Pig with Cloudera Manager, which hopefully will be part of the manager in future releases.

Impala seems to be really fast, but at the moment it is far from replacing Hive. The missing ability to use SerDes and User Defined Functions makes it difficult to replace Hive with Impala at the moment. One thing I realized while running the scripts is that Impala does not deal well with tabs in a script.

In the previous chapter I mentioned the execution of the script was not significantly faster, but a simple *select count(*)* query was much faster. At least in the future the currently missing UDFs will be supported [7].

5 Bibliography

- [1] "Cloudera's Project Impala rides herd with Hadoop elephant in real-time," The Register, [Online]. Available: http://www.theregister.co.uk/2012/10/24/cloudera_hadoop_impala_real_time_query/.
- [2] "UNetbootin - Homepage and Downloads," [Online]. Available: <http://unetbootin.sourceforge.net/>.
- [3] "Clonezilla - About," [Online]. Available: <http://www.clonezilla.org>.
- [4] "pdsh-2.27-1.el6.rf.x86_64.rpm Download Mirrors," [Online]. Available: http://pkgs.org/centos-6-rhel-6/repoforge-x86_64/pdsh-2.27-1.el6.rf.x86_64.rpm/download/.
- [5] "Upgrade from Cloudera Manager 4 to the Latest Version of Cloudera Manager 4 - Cloudera Support," Cloudera, [Online]. Available: <http://ccp.cloudera.com/display/FREE41DOC/Upgrade+from+Cloudera+Manager+4+to+the+Latest+Version+of+Cloudera+Manager+4>.
- [6] D. S. Rahimi, *SEIS 785 Lecture Slides: 785-10_HBase-2(1).pdf*, 2012.
- [7] "Cloudera Impala: A Modern SQL Engine for Hadoop," Cloudera, [Online]. Available: <http://www.cloudera.com/content/cloudera/en/resources/library/recordedwebinar/cloudera-impala-a-modern-sql-engine-for-hadoop-slides.html>.
- [8] "Installing Impala with Cloudera Manager Free Edition - Cloudera Support," Cloudera, [Online]. Available: <https://ccp.cloudera.com/display/FREE41DOC/Installing+Impala+with+Cloudera+Manager+Free+Edition>.
- [9] "Index of cloudera-repos/org/apache/hadoop/hadoop-common/2.0.0-cdh4.1.2," [Online]. Available: <https://repository.cloudera.com/artifactory/cloudera-repos/org/apache/hadoop/hadoop-common/2.0.0-cdh4.1.2/>.
- [10] "nohup(1) - Linux man page," [Online]. Available: <http://linux.die.net/man/1/nohup>.
- [11] "Installing Apache Ant," [Online]. Available: <http://ant.apache.org/manual/install.html#getBinary>.