

# Hadoop Cluster

P<sub>1</sub>



Filipe Sena, Pedro Trindade | G<sub>4</sub> | SCC | December 11, 2015

## Introduction

Hadoop is a framework written in Java to run applications on large clusters of commodity hardware. It incorporates features similar to those of the Google File System (GFS) and of the MapReduce computing paradigm. Hadoop's HDFS is a highly fault-tolerant distributed file system and, like Hadoop in general, designed to be deployed on low-cost hardware.

A Hadoop cluster is a special type of computational cluster designed specifically for storing and analyzing huge amounts of unstructured data in a distributed computing environment.

Those clusters run Hadoop's open source distributed processing software on low-cost commodity computers. Typically one machine in the cluster is designated as the NameNode and another machine as the JobTracker - the masters. The rest of the machines in the cluster act as both DataNode and TaskTracker - the slaves. Hadoop clusters are often referred to as "shared nothing" systems because the only thing that is shared between nodes is the network that connects them.

Hadoop clusters are known for boosting the speed of data analysis applications. They also are highly scalable: If a cluster's processing power is overwhelmed by growing volumes of data, additional cluster nodes can be added to increase throughput. Hadoop clusters also are highly resistant to failure because each piece of data is copied onto other cluster nodes, which ensures that the data is not lost if one node fails.

Our assignment was to deploy a virtualized Hadoop infrastructure, taking into account the requirements: 1 master, 4 slaves, 1 client, 1 administrator and 1 disc server. For this the teacher gave us 10 virtual machines, so that we could learn, practice and complete the assignment.

## Requirements

Our virtualized infrastructure had some requirements in order for us to be able to build it.

### 1. ADMINISTRATOR

- OpenSSH installed;
- 512MB of memory;
- 2 Network Interface Controllers – one public, for us to be able to access it from home, and one private – so that we could administrate the private VMs;
- Ubuntu Server 14.04.

### 2. HADOOP

#### 2a. Client

- OpenSSH installed;
- 256MB of memory;
- 2 Network Interface Controllers – one public, for us to be able to access it from home, and one private – so that we could submit Hadoop jobs;
- Ubuntu Server 14.04.

### 2b. Master

- OpenSSH installed;
- 256MB of memory;
- 2 Network Interface Controllers – one to connect with the administrator (SSH connection), and the other with the Hadoop network;
- Ubuntu Server 14.04.

### 2c. Slave

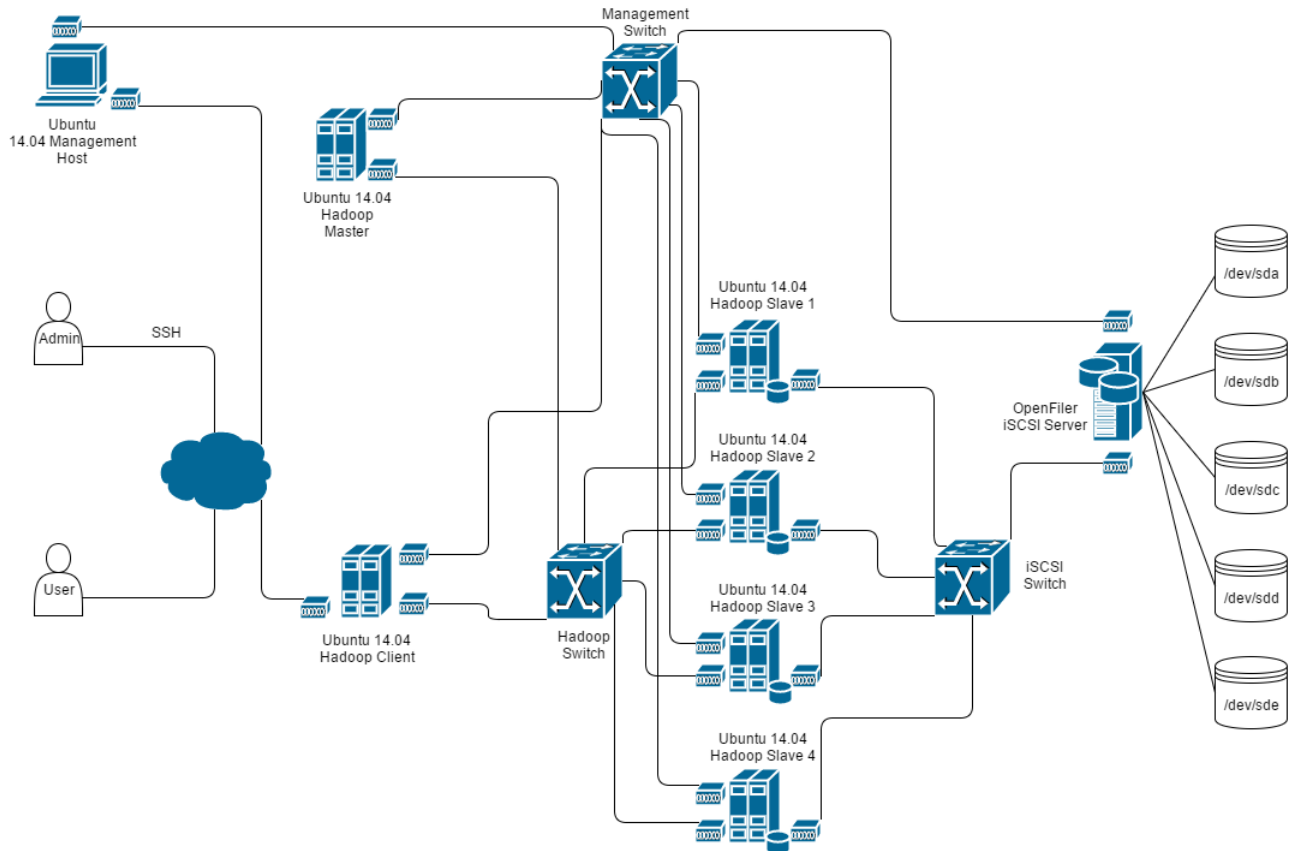
- OpenSSH installed;
- 256MB of memory;
- 3 Network Interface Controllers – the first to connect with the administrator (SSH connection), the second with the Hadoop network and finally the third with the iSCSI server;
- Ubuntu Server 14.04.

## 3. iSCSI

- OpenSSH installed;
- 1GB of memory;
- 2 Network Interface Controllers – one to connect with the administrator (SSH connection), and the other with the Hadoop slaves;
- OpenFiler running on CentOS.

## Architecture

In this section we describe how we decided to build the cluster.



## Decisions

In this section we describe the main decisions that we made. The IP address that we decided to give to the NICs, the names of the VMs and the names of the discs that used on each slave.

The first decision that we made was either to use OpenFiler or FreeNAS. We decided to use OpenFiler because we wanted the disks to be written in blocks instead of files.

The second decision that we made was defining our network. We decided that the networks we had to build (Hadoop and iSCSI – since SSH was already given to us) would be 192.168.1.0 and 192.168.2.0. We chose this because the 192.168.0.0 is a private network, which means that it can't be accessed from the outside - the public domain. The network specification is made in the 1 and the 2 on the 3<sup>rd</sup> dot of the IP address. So, we represented the Hadoop network with the 1 and the iSCSI network with the 2. The netmask of this network is 255.255.255.0.

Name	SSH	Hadoop	iSCSI	Disk	Disk Directory
Master	172.16.1.4	192.168.1.2	-	-	-
Slave1	172.16.1.5	192.168.1.3	192.168.2.1	/dev/sde	/media/slave1disk
Slave2	172.16.1.6	192.168.1.4	192.168.2.2	/dev/sdd	/media/slave2disk
Slave3	172.16.1.7	192.168.1.5	192.168.2.3	/dev/sdc	/media/slave3disk
Slave4	172.16.1.9	192.168.1.7	192.168.2.5	/dev/sdb	/media/slave4disk
iSCSI Server	172.16.1.10	-	192.168.2.6	-	-

The third decision we had to make was where to mount the disks. Because in the beginning we didn't know what we were doing exactly, we ended up giving different directory names in each slave. We didn't need to do this because the directory is in different machines and as a result, we had more work in configuring the Hadoop files. It's just a better policy if the slaves write in the same directory in each slave (for the sake of debugging).

## Building our Hadoop Cluster

In this section we show how we built the cluster.

### IP SETUP

One of the first things we needed to do was setting up our network. Since we decided the network IP addresses of each machine we only had to implement it in the NICs of all the VM's. For this step, and since we had a lot of NICs in the machines to setup, we decided to build a simple script and share it to all the machines using SCP – Secure Copy – that is available since we had OpenSSH installed. The script looks like the following:

```
GNU nano 2.2.6 File: addnic.sh
execute run "./addnic.sh eth ip" or "sh addnic.sh eth ip"
sudo ip link set dev $1 down
sudo ip addr flush $1
sudo ip addr add $2/24 dev $1
sudo ip link set dev $1 up
sudo ip addr
```

### IP TABLES

In order for the system to run we needed to disable the IP tables. So we build another simple script to address the situation and shared it in all the machines (master and slaves).

```
GNU nano 2.2.6 File: iptables-disable.sh
iptables -F INPUT ACCEPT
iptables -F OUTPUT ACCEPT
iptables -F FORWARD ACCEPT
iptables -F
```

## ISCSI CONFIGURATION

The configuration of the disk server was made according to the tutorial provided by the teacher. We had to define the network that could access the disks, that network being 192.168.2.0 as showed in the following image:

Network Access Configuration				
Delete	Name	Network/Host	Netmask	Type
<input type="checkbox"/>	iSCSI	192.168.2.0	255.255.255.0	Share

## MOUNT THE ISCSI SERVER DISK IN THE SLAVES

For this part we needed to mount the disks that were available in the slave nodes, since these nodes were already logged in the iSCSI server. We looked in the Ubuntu Documentation how to mount the disk and it turns out, it was quite simple. We just had to follow these steps:

1. Build a partition in the disk: `sudo fdisk /dev/X` with the arguments: p, 1, w - with X being sdb, sdc, sdd, sde. Note: after building the partition, its name is /dev/X1;
2. Format that partition: `sudo mkfs -t ext3 /dev/X1`;
3. Create a mounting point: `sudo mkdir /media/slaveYdisk`, with Y being the number of the slave - 1, 2, 3, 4;
4. Manually mount the disk into that point: `sudo mount /dev/X1 /media/slaveYdisk`. And the disk is mounted in that directory. So if we want to write in the disk, we just have to give the “/media/slaveYdisk” directory.

## CONFIGURE THE SSH

The Hadoop master must be able to run jobs on the Hadoop slaves. Because of that, we need to establish a SSH connection between them without needing a password.

We also created another user in the VMS called “hduser” that has owning permissions on the Hadoop folder and the “/media/slaveYdisk” directory, in order to either the Hadoop master or slave be able to write in those folders.

Finally we also needed to setup the /etc/hosts file in all the Hadoop machines for them to be able to find themselves in the network.

### SSH Connection without password

In the master we did these following commands:

- `ssh-keygen -t rsa -P ""`
- `ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@slaveX`

In the slaves and master nodes, we did this command:

- `cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys`

This makes sure that the slaves have the master public key, and so, the master is able to connect with the slaves without a password. Of course since we did the last command in the master too, Hadoop will be able to execute the command “ssh master” in the master machine without a password.

### User with permissions

For all the nodes we did this:

- `sudo addgroup hadoop`
- `sudo adduser --ingroup hadoop hduser`

Then, we installed Hadoop in the directory `/usr/local`:

- `cd /usr/local`
- `sudo tar xzf hadoop-1.2.1.tar.gz`
- `sudo mv hadoop-1.2.1 hadoop`
- `sudo chown -R hduser:hadoop hadoop`

After that, in the slaves we needed to give them permissions to where they were going to write:

- `sudo chown -R hduser:hadoop /media/slaveXdisk`

### JAVA\_HOME

Everyone knows that Hadoop runs on java, so in order for everything to work we needed to setup the java\_home in the file `/usr/local/hadoop/conf/hadoop-env.sh` in all the nodes:

```
GNU nano 2.2.6
# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME. All others are
# optional. When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use. Required.
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

## Master Configuration

In this assignment we decided to use only one machine as the Hadoop master.

### `$HADOOP_HOME/CONFIG/MASTERS`

In this machine we had to decide if we wanted to run 1 or 2 masters. We decided to go only for one. So we had to edit the “masters” file in the conf folder to look like this:

```
hduser@ubuntu: /usr/local/hadoop
GNU nano 2.2.6
master
```

### `$HADOOP_HOME/CONFIG/SLAVES`

Since we opted for 4 slaves we had to find a way to tell the master that we will have 4 slaves. That is done exactly like this:

```
hduser@ubuntu: /usr/local/hadoop
GNU nano 2.2.6      File: conf/slaves
slave1
slave2
slave3
slave4
```

### `/ETC/HOSTS`

After the changes that we made in the previous points we needed to let the master know where those names are addressed to. We did that by modifying the `/etc/hosts` file. The changes look just like this:

```
hduser@ubuntu: /usr/local/hadoop
GNU nano 2.2.6      File: /etc/hosts
127.0.0.1    localhost
172.16.1.1   ubuntu
192.168.1.2   master
192.168.1.3   slave1
192.168.1.4   slave2
192.168.1.5   slave3
192.168.1.7   slave4

# The following lines are desirable for IPv6 capable hosts
::1          localhost ip6-localhost ip6-loopback
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
```

This file is the same for all the nodes, but if we were trying to perfect our work, we would remove the slaves from this file in the slave nodes. Since the slaves don't need to know the IP addresses of the other slaves.



## \$HADOOP\_HOME/CONF/CORE-SITE.XML

The core-site.xml in the master must look like this:

```
hduser@ubuntu: /usr/local/hadoop
GNU nano 2.2.6 File: conf/core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/usr/local/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:54310</value>
  <description>The name of the default file system. A URI whose
scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl) naming
the FileSystem implementation class. The uri's authority is used to
determine the host, port, etc. for a filesystem.</description>
</property>
</configuration>
```

## \$HADOOP\_HOME/CONF/HDFS-SITE.XML

The hdfs-site.xml file must be exactly like this, since we have 4 slaves we'll need the data to be replicated in all of them, so that they are able to execute the job in every node.

```
hduser@ubuntu: /usr/local/hadoop
GNU nano 2.2.6 File: conf/hdfs-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>dfs.replication</name>
  <value>4</value>
  <description>Default block replication.
The actual number of replications can be specified when the file is created.
The default is used if replication is not specified in create time.
</description>
</property>
</configuration>
```

## \$HADOOP\_HOME/CONF/MAPRED-SITE.XML

The mapred-site.xml is like this because we want to setup the job tracker in the master, since the master controls what the slaves are doing. This file is the same in the slaves so that they are able to find the job tracker.

```
hduser@ubuntu: /usr/local/hadoop
GNU nano 2.2.6 File: conf/mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
  <name>mapred.job.tracker</name>
  <value>master:54311</value>
  <description>The host and port that the MapReduce job tracker runs
at. If "local", then jobs are run in-process as a single map
and reduce task.
</description>
</property>
</configuration>
```

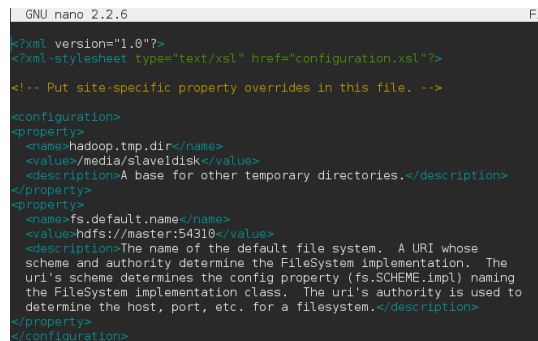
## Slaves Configuration

We decided to have 4 slaves to execute the jobs. We needed them to know where the master was and also to have other properties in their configuration files.

The slaves, as it was said before, have their `/etc/hosts` file like the master, so they have the IP address of the master in the network.

### `$HADOOP_HOME/CONF/CORE-SITE.XML`

In this file we needed to configure the “`hadoop.tmp.dir`” to use the mounted disk. For this report we printed the picture of the first slave:



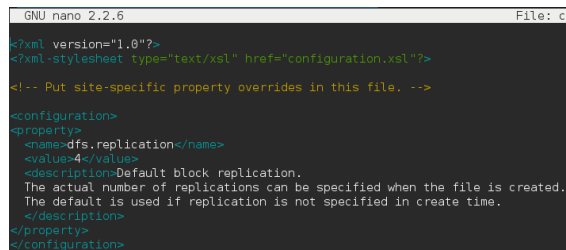
```
GNU nano 2.2.6                                Fil
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/media/slavedisk</value>
    <description>A base for other temporary directories.</description>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://master:54310</value>
    <description>The name of the default file system. A URI whose
scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl) naming
the FileSystem implementation class. The uri's authority is used to
determine the host, port, etc. for a filesystem.</description>
  </property>
</configuration>
```

### `$HADOOP_HOME/CONF/HDFS-SITE.XML`

This file is exactly like the master's:



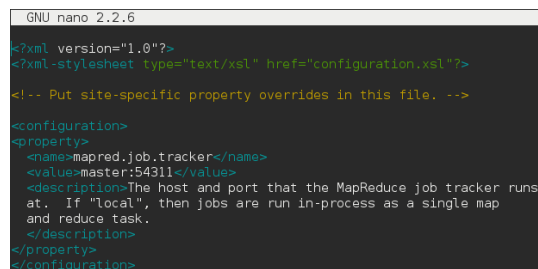
```
GNU nano 2.2.6                                File: cc
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>4</value>
    <description>Default block replication.
The actual number of replications can be specified when the file is created.
The default is used if replication is not specified in create time.
  </description>
  </property>
</configuration>
```

### `$HADOOP_HOME/CONF/MAPRED-SITE.XML`

This file is also like the master's, since the slaves have to know where the job tracker is, in order for them to receive jobs to execute:



```
GNU nano 2.2.6
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>master:54311</value>
    <description>The host and port that the MapReduce job tracker runs
at. If "local", then jobs are run in-process as a single map
and reduce task.
  </description>
  </property>
</configuration>
```

## Execution

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop jar hadoop-examples-*.jar grep /usr/local/hadoop/input /usr/local/hadoop/output 'dfs[a-z.]+'
Warning: $HADOOP_HOME is deprecated.
15/12/07 18:36:56 INFO util.NativeCodeLoader: Loaded the native-hadoop library
15/12/07 18:36:56 WARN snappy.LoadSnappy: Snappy native library not loaded
15/12/07 18:36:56 INFO mapred.FileInputFormat: Total input paths to process : 7
15/12/07 18:36:58 INFO mapred.JobClient: Running job: job_201512071832_0001
15/12/07 18:36:59 INFO mapred.JobClient: map 0% reduce 0%
15/12/07 18:37:50 INFO mapred.JobClient: map 14% reduce 0%
15/12/07 18:38:40 INFO mapred.JobClient: map 28% reduce 0%
15/12/07 18:38:48 INFO mapred.JobClient: map 42% reduce 0%
15/12/07 18:39:19 INFO mapred.JobClient: map 57% reduce 0%
15/12/07 18:39:29 INFO mapred.JobClient: Task Id : attempt_201512071832_0001_m_000000_0, Status : FAILED
java.lang.Throwable: Child Error
    at org.apache.hadoop.mapred.TaskRunner.run(TaskRunner.java:271)
Caused by: java.io.IOException: Task process exit with nonzero status of 137.
    at org.apache.hadoop.mapred.TaskRunner.run(TaskRunner.java:258)
15/12/07 18:39:29 WARN mapred.JobClient: Error reading task outputConnection refused
15/12/07 18:39:29 WARN mapred.JobClient: Error reading task outputConnection refused
15/12/07 18:39:36 INFO mapred.JobClient: map 71% reduce 0%
15/12/07 18:40:06 INFO mapred.JobClient: map 85% reduce 0%
15/12/07 18:40:10 INFO mapred.JobClient: map 100% reduce 0%
```

## Conclusion

While we were solving this assignment we encountered several difficulties, such as not being able to reset the machines for ourselves, lots of errors and the fact that the virtual machines had little memory (256MB). We also made a lot of mistakes like killing a virtual machine, making her unable to recover (VM8) and also being unable to execute a job until the end. As we can see in the previous section we couldn't make the reduce part work. This is actually weird and in spite of our best efforts to try and solve the problem, there were no exceptions in the logs so that we could analyze the problem and solve it.

There was also a part that we could have improved, but we didn't have time for that, since this semester we had so much work in all the courses. That part was the public client. At the moment, in order for the cluster to work, we need to submit the job in the master. In a real-life cluster, such a thing shouldn't be possible, because the master is a private machine in the network, so the jobs should be submitted in the client. We encountered a website explaining easily how to do this, but as we said before it was impossible for us to implement it, due to the lack of time (check bibliography - 4).

Another improvement that we could make is the machines being able to reset to a stable state to execute jobs when the network restarts without having to manually do it. We deliberately left the machines like that because we wanted to make sure we knew exactly what to do in each step.

In the end, all of this contributed to the improvement of our skills as data center managers and to gain a better knowledge of what's behind a simple Hadoop cluster.

## Bibliography

<https://help.ubuntu.com/community/InstallingANewHardDrive>

<http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/>

<http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/>

<https://pravinchavan.wordpress.com/2013/06/18/submitting-hadoop-job-from-client-machine/>