

UNIT 2

Working with Big Data:

- 1. Google File System,**
- 2. Hadoop Distributed File System (HDFS)**

Building blocks of Hadoop

- A. Namenode**
 - B. Datanode**
 - C. Secondary Name node**
 - D. JobTracker**
 - E. TaskTracker**
-
- 3. Introducing and Configuring Hadoop cluster**
 - A. Local**
 - B. Pseudo-distributed mode**
 - C. Fully Distributed mode**
 - 4. Configuring XML files.**

1. The Google File System

The Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

GFS provides a familiar file system interface, though it does not implement a standard API such as POSIX. Files are organized hierarchically in directories and identified by path-names. We support the usual operations to create, delete, open, close, read, and write files.

Moreover, GFS has snapshot and record append operations. Snapshot creates a copy of a file or a directory tree at low cost. Record append allows multiple clients to append data to **the same file concurrently while guaranteeing the atomicity of each individual client's** append

Architecture

A GFS cluster consists of a single master and multiple chunk servers and is accessed by multiple clients, as shown in Figure .

Each of these is typically a commodity Linux machine running a user-level server process. It is easy to run both a chunkserver and a client on the same machine, as long as machine resources permit and the lower reliability caused by running possibly flaky application code is acceptable.

Each of these is typically a commodity Linux machine running a user-level server process. It is easy to run both a chunkserver and a client on the same machine, as long as machine resources permit and the lower reliability caused by running possibly flaky application code is acceptable.

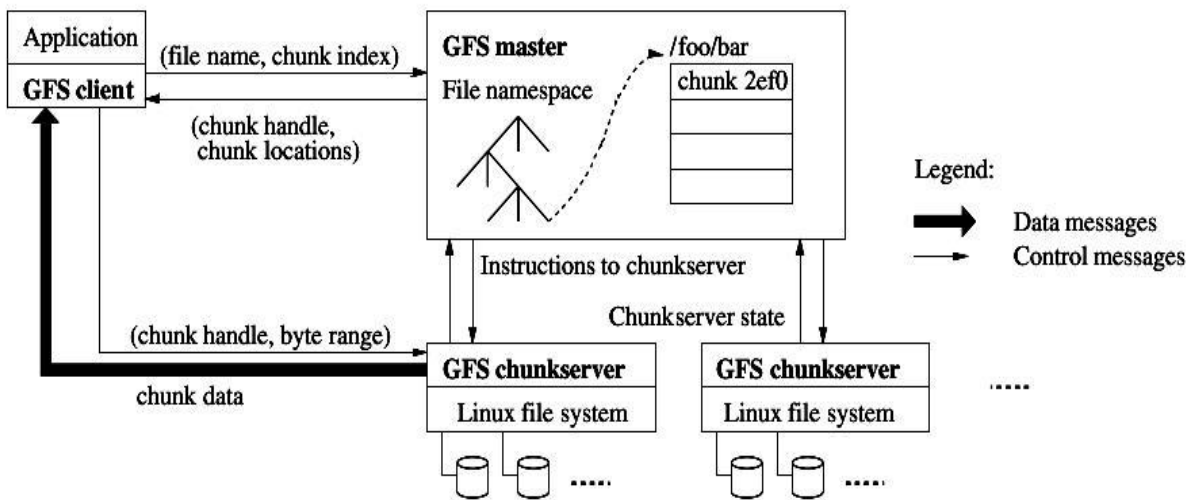


Figure 1: GFS Architecture

Files are divided into fixed-size chunks. Each chunk is identified by an immutable and globally unique 64 bit chunk handle assigned by the master at the time of chunk creation.

Chunkservers store chunks on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple chunkservers. By default, we store three replicas, though users can designate different replication levels for different regions of the file namespace. The master maintains all file system metadata. This includes the namespace, access control information, the mapping from files to chunks, and the current locations of chunks.

It also controls system-wide activities such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between chunkservers. The master periodically communicates with each chunkserver in HeartBeat messages to give it instructions and collect its state.

GFS client code linked into each application implements the file system API and communicates with the master and chunkservers to read or write data on behalf of the application. Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunkservers. We do not provide the POSIX API and therefore need not hook into the Linux vnode layer.

Neither the client nor the chunkserver caches file data. Client caches offer little benefit because most applications stream through huge files or have working sets too large to be cached. Not having them simplifies the client and the overall system by eliminating cache coherence issues. (Clients do cache metadata, however.) Chunkservers need not cache file **data because chunks are stored as local files and so Linux's buffer cache already keeps** frequently accessed data in memory.

Single Master

Having a single master vastly simplifies our design and enables the master to make sophisticated chunk placement and replication decisions using global knowledge. However, we must minimize its involvement in reads and writes so that it does not become a bottleneck. Clients never read and write file data through the master. Instead, a client asks the master which chunkserver it should contact.

Chunk Size

Chunk size is one of the key design parameters. We have chosen 64 MB, which is much larger than typical file system block sizes. Each chunk replica is stored as a plain Linux file on a chunkserver and is extended only as needed.

Lazy space allocation avoids wasting space due to internal fragmentation, perhaps the greatest objection against such a large chunk size.

A large chunk size offers several important advantages.

First, it reduces clients' need to interact with the master because reads and writes on the same chunk require only one initial request to the master for chunk location information.

The reduction is especially significant for our work loads because applications mostly read and write large files sequentially.

Even for small random reads, the client can comfortably cache all the chunk location information for a multi-TB working set. Second, since on a large chunk, a client is more likely to perform many operations on a given chunk, it can reduce network overhead by keeping a persistent TCP connection to the chunkserver over an extended period of time.

Third, it reduces the size of the metadata stored on the master. This allows us to keep the metadata in memory, which in turn brings other advantages .

On the other hand, a large chunk size, even with lazy space allocation, has its disadvantages. A small file consists of a small number of chunks, perhaps just one. The chunkservers storing those chunks may become hot spots if many clients are accessing the same file. In practice, hot spots have not been a major issue because our applications mostly read large multi-chunk files sequentially.

However, hot spots did develop when GFS was first used by a batch-queue system: an executable was written to GFS as a single-chunk file and then started on hundreds of machines at the same time.

Metadata

The master stores three major types of metadata: the file and chunk namespaces, the mapping from files to chunks, **and the locations of each chunk's replicas.**

All metadata is **kept in the master's memory.** The first two types (namespaces and file-to-chunk mapping) are also kept persistent by logging mutations to an operation log stored **on the master's local disk and replicated on remote machines.** Using a log allows us to update the master state simply, reliably, and without risking inconsistencies in the event of a master crash. The master does not store chunk location information persistently. Instead, it asks each chunkserver about its chunks at master startup and whenever a chunkserver joins the cluster.

In-Memory Data Structures

Since metadata is stored in memory, master operations are fast. Furthermore, it is easy and efficient for the master to periodically scan through its entire state in the background.

This periodic scanning is used to implement chunk garbage collection, re-replication in the presence of chunkserver failures, and chunk migration to balance load and disk space usage across chunkservers.

Consistency Model

GFS has a relaxed consistency model that supports our highly distributed applications well but remains relatively simple and efficient to implement. We now discuss **GFS's** guarantees and what they mean to applications.

We also highlight how GFS maintains these guarantees but leave the details to other parts of the paper.

Guarantees by GFS

Operation Log

The operation log contains a historical record of critical metadata changes. It is central to GFS. Not only is it the only persistent record of metadata, but it also serves as a logical time line that defines the order of concurrent operations.

Advantages and disadvantages of large sized chunks in Google File System

Chunks size is one of the key design parameters. In GFS it is 64 MB, which is much larger than typical file system blocks sizes. Each chunk replica is stored as a plain Linux file on a chunk server and is extended only as needed.

Advantages

1. It **reduces clients' need to interact with the master because reads and writes on the same chunk** require only one initial request to the master for chunk location information.

2. Since on a large chunk, a client is more likely to perform many operations on a given chunk, it can reduce network overhead by keeping a persistent TCP connection to the chunk server over an extended period of time.

3. It reduces the size of the metadata stored on the master. This allows us to keep the metadata in memory, which in turn brings other advantages.

Disadvantages

1. Lazy space allocation avoids wasting space due to internal fragmentation.

2. Even with lazy space allocation, a small file consists of a small number of chunks, perhaps just one. The chunk servers storing those chunks may become hot spots if many clients are accessing the same file. In practice, hot spots have not been a major issue because the applications mostly read large multi-chunk files sequentially. To mitigate it, replication and allowance to read from other clients can be done.

2. Hadoop Distributed File System (HDFS) Building blocks of Hadoop :

A. Namenode

B. Datanode

C. Secondary Name node

D. JobTracker

E. TaskTracker

Hadoop is made up of 2 parts:

1. HDFS – Hadoop Distributed File System
2. MapReduce – The programming model that is used to work on the data present in HDFS.

HDFS – Hadoop Distributed File System

HDFS is a file system that is written in Java and resides within the user space unlike traditional file systems like FAT, NTFS, ext2, etc that reside on the kernel space. HDFS was primarily written to store large amounts of data (terabytes and petabytes). HDFS was built inline with **Google's paper on GFS.**

MapReduce

MapReduce is the programming model that uses Java as the programming language to retrieve data from files stored in the HDFS. All data in HDFS is stored as files. Even MapReduce was built inline with another paper by Google.

Google, apart from their papers did not release their implementations of GFS and MapReduce. However, the Open Source Community built Hadoop and MapReduce based on those papers. The initial adoption of Hadoop was at Yahoo Inc., where it gained good momentum and went onto be a part of their production systems. After Yahoo, many organizations like LinkedIn, Facebook, Netflix and many more have successfully implemented Hadoop within their organizations.

Hadoop uses HDFS to store files efficiently in the cluster. When a file is placed in HDFS it is broken down into blocks, 64 MB block size by default. These blocks are then replicated across the different nodes (*DataNodes*) in the cluster. The default replication value is 3, i.e. there will be 3 copies of the same block in the cluster. We will see later on why we maintain replicas of the blocks in the cluster.

A Hadoop cluster can comprise of a single node (single node cluster) or thousands of nodes. Once you have installed Hadoop you can try out the following few basic commands to work with HDFS:

- `hadoop fs -ls`
- `hadoop fs -put <path_of_local> <path_in_hdfs>`
- `hadoop fs -get <path_in_hdfs> <path_of_local>`
- `hadoop fs -cat <path_of_file_in_hdfs>`
- `hadoop fs -rmr <path_in_hdfs>`

the different components of a Hadoop Cluster are:

NameNode (Master) – NameNode, Secondary NameNode, JobTracker

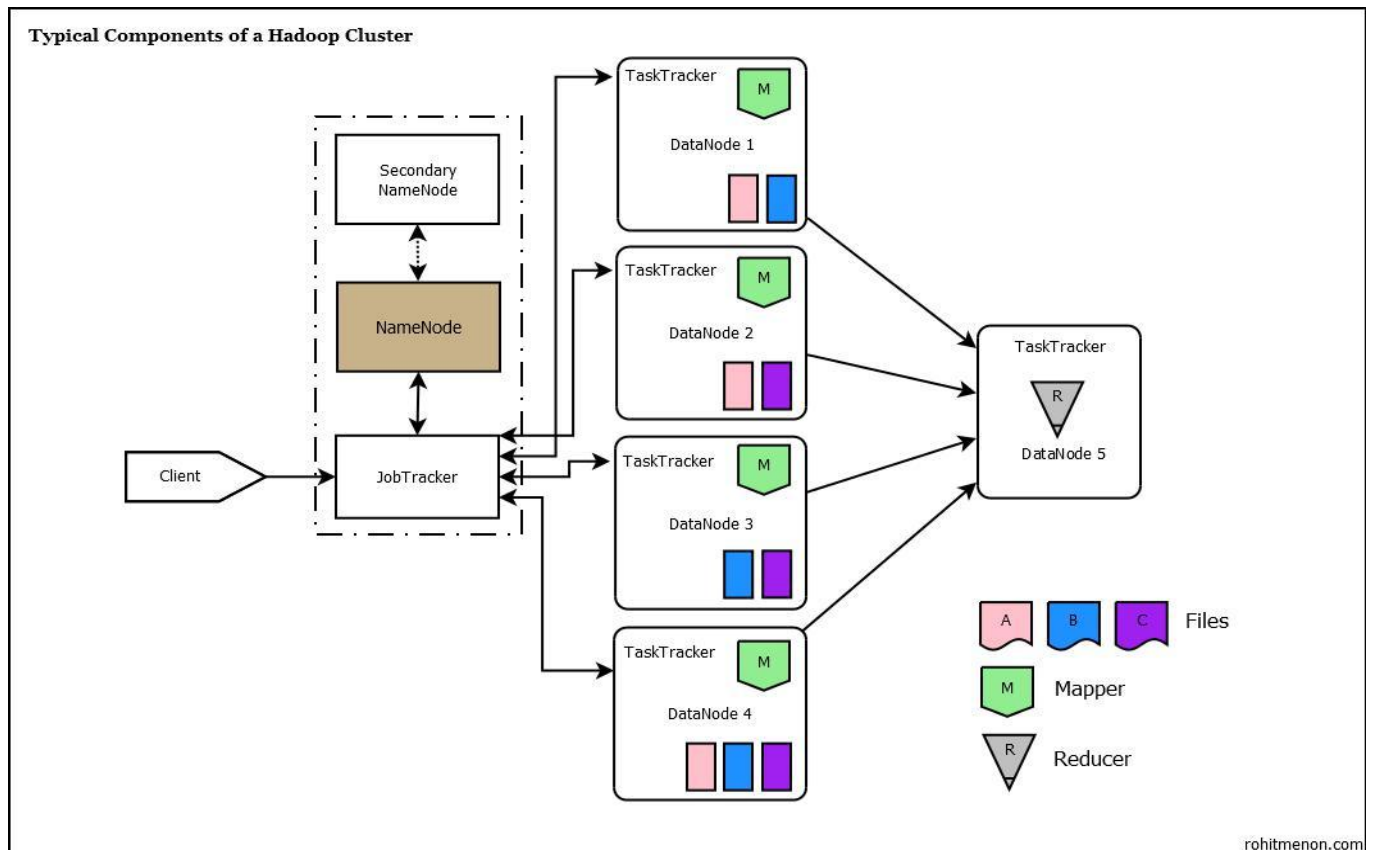
DataNode 1 (Slave) – TaskTracker, DataNode

DataNode 2 (Slave) – TaskTracker, DataNode

DataNode 3 (Slave) – TaskTracker, DataNode

DataNode 4 (Slave) – TaskTracker, DataNode

DataNode 5 (Slave) – TaskTracker, DataNode



The above diagram depicts a 6 Node Hadoop Cluster

In the diagram you see that the *NameNode*, *Secondary NameNode* and the *JobTracker* are running on a single machine. Usually in production clusters having more those 20-30 nodes, the daemons run on separate nodes.

Hadoop follows a Master-Slave architecture. As mentioned earlier, a file in HDFS is split into blocks and replicated across *Datanodes* in a Hadoop cluster. You can see that the three files A, B and C have been split across with a replication factor of 3 across the different *Datanodes*.

Now let us go through each node and daemon:

NameNode

The NameNode in Hadoop is the node where Hadoop stores all the location information of the files in HDFS. In other words, it holds the metadata for HDFS. Whenever a file is placed in the cluster a corresponding entry of it location is maintained by the NameNode. So, for the files A, B and C we would have something as follows in the NameNode:

File A – DataNode1, DataNode2, DataNode4

File B – DataNode1, DataNode3, DataNode4

File C – DataNode2, DataNode3, DataNode4

This information is required when retrieving data from the cluster as the data is spread across multiple machines. **The NameNode is a Single Point of Failure for the Hadoop Cluster.**

Secondary NameNode

IMPORTANT – The *Secondary NameNode* is not a failover node for the *NameNode*.

The secondary name node is responsible for performing periodic housekeeping functions for the *NameNode*. It only creates checkpoints of the file system present in the *NameNode*.

DataNode

The *DataNode* is responsible for storing the files in HDFS. It manages the file blocks within the node. It sends information to the *NameNode* about the files and blocks stored in that node and responds to the *NameNode* for all filesystem operations.

JobTracker

JobTracker is responsible for taking in requests from a client and assigning *TaskTrackers* with tasks to be performed. The *JobTracker* tries to assign tasks to the *TaskTracker* on the *DataNode* where the data is locally present (Data Locality). If that is not possible it will at least try to assign tasks to *TaskTrackers* within the same rack. If for some reason the node fails the *JobTracker* assigns the task to another *TaskTracker* where the replica of the data exists since the data blocks are replicated across the *DataNodes*. This ensures that the job does not fail even if a node fails within the cluster.

TaskTracker

TaskTracker is a daemon that accepts tasks (Map, Reduce and Shuffle) from the *JobTracker*. The *TaskTracker* keeps sending a heart beat message to the *JobTracker* to notify that it is alive. Along with the heartbeat it also sends the free slots available within it to process tasks. *TaskTracker* starts and monitors the Map & Reduce Tasks and sends progress/status information back to the *JobTracker*.

All the above daemons run within have their own JVMs.

A typical (simplified) flow in Hadoop is as follows:

1. A Client (usually a MapReduce program) submits a job to the *JobTracker*.
2. The *JobTracker* get information from the *NameNode* on the location of the data within the *DataNodes*. The *JobTracker* places the client program (usually a jar file along with the

configuration file) in the HDFS. Once placed, *JobTracker* tries to assign tasks to *TaskTrackers* on the *DataNodes* based on data locality.

3. The *TaskTracker* takes care of starting the Map tasks on the *DataNodes* by picking up the client program from the shared location on the HDFS.
4. The progress of the operation is relayed back to the *JobTracker* by the *TaskTracker*.
5. On completion of the Map task an intermediate file is created on the local filesystem of the *TaskTracker*.
6. Results from Map tasks are then passed on to the Reduce task.
7. The Reduce tasks works on all data received from map tasks and writes the final output to HDFS.
8. After the task complete the intermediate data generated by the *TaskTracker* is deleted.

A very important feature of Hadoop to note here is, that, the program goes to where the data is and not the way around, thus resulting in efficient processing of data.

3. Introducing and Configuring Hadoop cluster

A. Local

B. Pseudo-distributed mode

C. Fully Distributed mode

Hadoop is supported by GNU/Linux platform and its flavors. Therefore, we have to install a Linux operating system for setting up Hadoop environment.

Pre-installation Setup

Before installing Hadoop into the Linux environment, we need to set up Linux using ssh (Secure Shell). Follow the steps given below for setting up the Linux environment.

Creating a User

At the beginning, it is recommended to create a separate user for Hadoop to isolate Hadoop file system from Unix file system. Follow the steps given below to create a user:

- **Open the root using the command “su”.**
- **Create a user from the root account using the command “useradd username”.**
- **Now you can open an existing user account using the command “su username”.**

Open the Linux terminal and type the following commands to create a user.

```
$ su
password:
# useradd hadoop
# passwd hadoop
New passwd:
Retype new passwd
```

SSH Setup and Key Generation

SSH setup is required to do different operations on a cluster such as starting, stopping, distributed daemon shell operations. To authenticate different users of Hadoop, it is required to provide public/private key pair for a Hadoop user and share it with different users.

The following commands are used for generating a key value pair using SSH. Copy the public keys from `id_rsa.pub` to `authorized_keys`, and provide the owner with read and write permissions to `authorized_keys` file respectively.

```
$ ssh-keygen -t rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys $
chmod 0600 ~/.ssh/authorized_keys
```

Installing Java

Java is the main prerequisite for Hadoop. First of all, you should verify the existence of java in **your system using the command “java -version”**. The syntax of java version command is **given** below.

```
$ java -version
```

If everything is in order, it will give you the following output.

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

If java is not installed in your system, then follow the steps given below for installing java.

Step 1

Download java (JDK <latest version> - X64.tar.gz) by visiting the following link <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads1880260.html>. Then jdk-7u71-linux-x64.tar.gz will be downloaded into your system. Step 2

Generally you will find the downloaded java file in Downloads folder. Verify it and extract the jdk-7u71-linux-x64.gz file using the following commands.

```
$ cd Downloads/  
$ ls  
jdk-7u71-linux-x64.gz  
$ tar xzf jdk-7u71-linux-x64.gz  
$ ls  
jdk1.7.0_71  jdk-7u71-linux-x64.gz
```

Step 3

To make java available to all the users, you have to move it to the location “/usr/local/”. Open root, and type the following commands.

```
$ su  
password:  
# mv jdk1.7.0_71 /usr/local/  
# exit
```

Step 4

For setting up PATH and JAVA_HOME variables, add the following commands to ~/.bashrc file.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71  
export PATH=$PATH:$JAVA_HOME/bin
```

Now apply all the changes into the current running system.

```
$ source ~/.bashrc
```

Step 5

Use the following commands to configure java alternatives:

```
# alternatives --install /usr/bin/java java usr/local/java/bin/java 2
# alternatives --install /usr/bin/javac javac usr/local/java/bin/javac 2
# alternatives --install /usr/bin/jar jar usr/local/java/bin/jar 2
# alternatives --set java usr/local/java/bin/java
# alternatives --set javac usr/local/java/bin/javac
# alternatives --set jar usr/local/java/bin/jar
```

Now verify the java -version command from the terminal as explained above.

Downloading Hadoop

Download and extract Hadoop 2.4.1 from Apache software foundation using the following commands.

```
$ su
password:
# cd /usr/local
# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/
hadoop-2.4.1.tar.gz
# tar xzf hadoop-2.4.1.tar.gz
# mv hadoop-2.4.1/* to hadoop/
# exit
```

Hadoop Operation Modes

Once you have downloaded Hadoop, you can operate your Hadoop cluster in one of the three supported modes:

- **Local/Standalone Mode** : After downloading Hadoop in your system, by default, it is configured in a standalone mode and can be run as a single java process.
- **Pseudo Distributed Mode** : It is a distributed simulation on single machine. Each Hadoop daemon such as hdfs, yarn, MapReduce etc., will run as a separate java process. This mode is useful for development.
- **Fully Distributed Mode** : This mode is fully distributed with minimum two or more machines as a cluster. We will come across this mode in detail in the coming chapters.

Installing Hadoop in Standalone Mode

Here we will discuss the installation of **Hadoop 2.4.1** in standalone mode.

There are no daemons running and everything runs in a single JVM. Standalone mode is suitable for running MapReduce programs during development, since it is easy to test and debug them.

Setting Up Hadoop

You can set Hadoop environment variables by appending the following commands to `~/.bashrc` file.

```
export HADOOP_HOME=/usr/local/hadoop
```

Before proceeding further, you need to make sure that Hadoop is working fine. Just issue the following command:

```
$ hadoop version
```

If everything is fine with your setup, then you should see the following result:

Hadoop 2.4.1

Subversion <https://svn.apache.org/repos/asf/hadoop/common> -r 1529768

Compiled by hortonmu on 2013-10-07T06:28Z

Compiled with protoc 2.5.0

From source with checksum 79e53ce7994d1628b240f09af91e1af4

It means your Hadoop's standalone mode setup is working fine. By default, Hadoop is configured to run in a non-distributed mode on a single machine. Example

Let's check a simple example of Hadoop. Hadoop installation delivers the following example MapReduce jar file, which provides basic functionality of MapReduce and can be used for calculating, like Pi value, word counts in a given list of files, etc.

```
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar
```

Let's have an input directory where we will push a few files and our requirement is to count the total number of words in those files. To calculate the total number of words, we do not need to write our MapReduce, provided the .jar file contains the implementation for word count. You can try other examples using the same .jar file; just issue the following commands to check supported MapReduce functional programs by `hadoop-mapreduce-examples-2.2.0.jar` file.

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar
```

Step 1

Create temporary content files in the input directory. You can create this input directory anywhere you would like to work.

```
$ mkdir input
$ cp $HADOOP_HOME/*.txt input
$ ls -l input
```

It will give the following files in your input directory:

```
total 24
-rw-r--r-- 1 root root 15164 Feb 21 10:14 LICENSE.txt
-rw-r--r-- 1 root root 101 Feb 21 10:14 NOTICE.txt
-rw-r--r-- 1 root root 1366 Feb 21 10:14 README.txt
```

These files have been copied from the Hadoop installation home directory. For your experiment, you can have different and large sets of files. Step 2

Let's start the Hadoop process to count the total number of words in all the files available in the input directory, as follows:

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar
wordcount input output
```

Step 3

Step-2 will do the required processing and save the output in output/part-r00000 file, which you can check by using:

```
$cat output/*
```

It will list down all the words along with their total counts available in all the files available in the input directory.

```
"AS 4
"Contribution" 1
"Contributor" 1
"Derivative 1
"Legal 1
```



```

"License"      1
"License");   1
"Licensor"    1
"NOTICE"      1
"Not          1
"Object"      1
"Source"      1
"Work"       1
"You"        1
>Your")     1
"[]"        1
"control"    1
"printed"    1
"submitted"  1
(50%)       1
(BIS),      1
(C)         1
(Don't)     1
(ECCN)      1
(INCLUDING  2
(INCLUDING, 2
.....

```

Installing Hadoop in Pseudo Distributed Mode

Follow the steps given below to install Hadoop 2.4.1 in pseudo distributed mode. Step 1: Setting Up Hadoop

You can set Hadoop environment variables by appending the following commands to `~/.bashrc` file.

```

export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME

```

```
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME export
YARN_HOME=$HADOOP_HOME

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin export
HADOOP_INSTALL=$HADOOP_HOME
```

Now apply all the changes into the current running system.

```
$ source ~/.bashrc
```

Step 2: Hadoop Configuration

You can find all the Hadoop configuration files in the location “**\$HADOOP_HOME/etc/hadoop**”. **It is required to make changes in those configuration files** according to your Hadoop infrastructure.

```
$ cd $HADOOP_HOME/etc/hadoop
```

In order to develop Hadoop programs in java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing **JAVA_HOME** value with the location of java in your system.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
```

The following are the list of files that you have to edit to configure Hadoop.

core-site.xml

The **core-site.xml** file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and size of Read/Write buffers.

Open the core-site.xml and add the following properties in between <configuration>, </configuration> tags.

```
<configuration>

  <property>
    <name>fs.default.name </name>
    <value> hdfs://localhost:9000 </value>
```

```
</property>
```

```
</configuration>
```

hdfs-site.xml

The **hdfs-site.xml** file contains information such as the value of replication data, namenode path, and datanode paths of your local file systems. It means the place where you want to store the Hadoop infrastructure.

Let us assume the following data.

```
dfs.replication (data replication value) = 1
```

(In the below given path /hadoop/ is the user name.

hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)

```
namenode path = //home/hadoop/hadoopinfra/hdfs/namenode
```

(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)

```
datanode path = //home/hadoop/hadoopinfra/hdfs/datanode
```

Open this file and add the following properties in between the <configuration> </configuration> tags in this file.

```
<configuration>
```

```
<property>
```

```
<name>dfs.replication</name>
```

```
<value>1</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.name.dir</name>
```

```
<value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.data.dir</name>
```

```
<value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>
</property>

</configuration>
```

Note: In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

yarn-site.xml

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>

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

</configuration>
```

mapred-site.xml

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, it is required to copy the file from **mapred-site.xml.template** to **mapred-site.xml** file using the following command.

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

Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration>tags in this file.

```
<configuration>

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

```
</configuration>
```

Verifying Hadoop Installation

The following steps are used to verify the Hadoop installation.

Step 1: Name Node Setup

Set up the namenode using the command “hdfs namenode -format” as follows.

```
$ cd ~
```

```
$ hdfs namenode -format
```

The expected result is as follows.

```
10/24/14 21:30:55 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode STARTUP_MSG: host =
localhost/192.168.1.11
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 2.4.1
...
...
10/24/14 21:30:56 INFO common.Storage: Storage directory
/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.
10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to
retain 1 images with txid >= 0
10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0
10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at localhost/192.168.1.11
*****/
```

Step 2: Verifying Hadoop dfs

The following command is used to start dfs. Executing this command will start your Hadoop file system.

```
$ start-dfs.sh
```

The expected output is as follows:

```
10/24/14 21:37:56
```

```
Starting namenodes on [localhost]
```

```
localhost: starting namenode, logging to /home/hadoop/hadoop
```

```
2.4.1/logs/hadoop-hadoop-namenode-localhost.out
```

```
localhost: starting datanode, logging to /home/hadoop/hadoop
```

```
2.4.1/logs/hadoop-hadoop-datanode-localhost.out
```

```
Starting secondary namenodes [0.0.0.0]
```

Step 3: Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

```
$ start-yarn.sh
```

The expected output as follows:

```
starting yarn daemons
```

```
starting resourcemanager, logging to /home/hadoop/hadoop
```

```
2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out
```

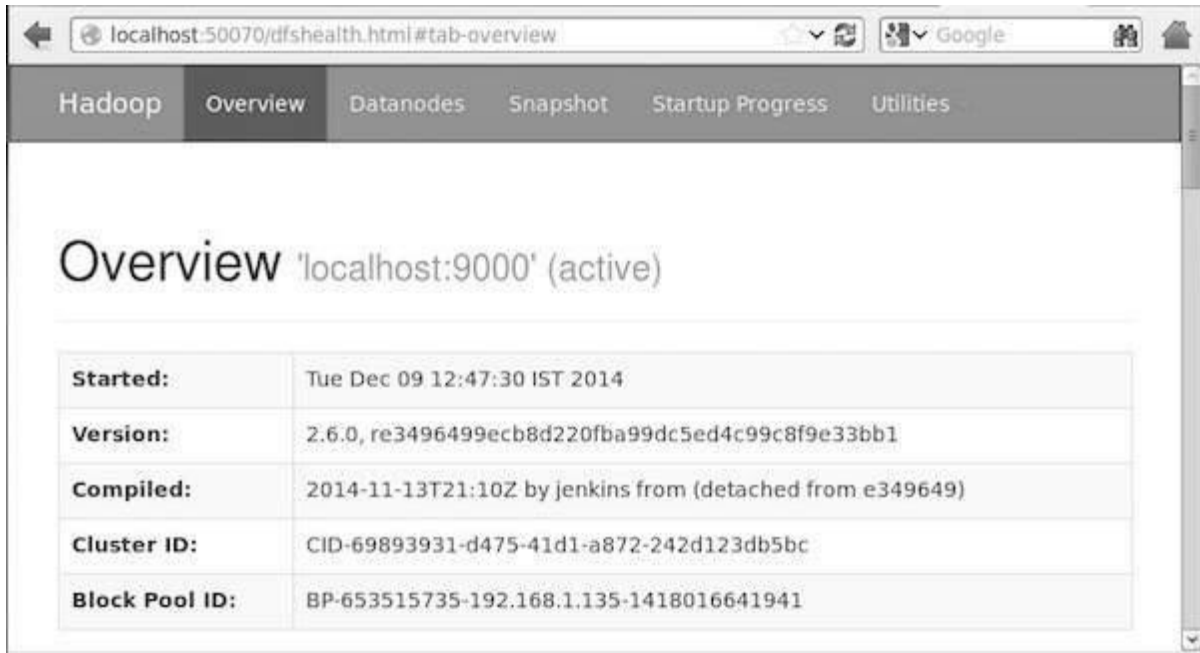
```
localhost: starting nodemanager, logging to /home/hadoop/hadoop
```

```
2.4.1/logs/yarn-hadoop-nodemanager-localhost.out
```

Step 4: Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following url to get Hadoop services on browser.

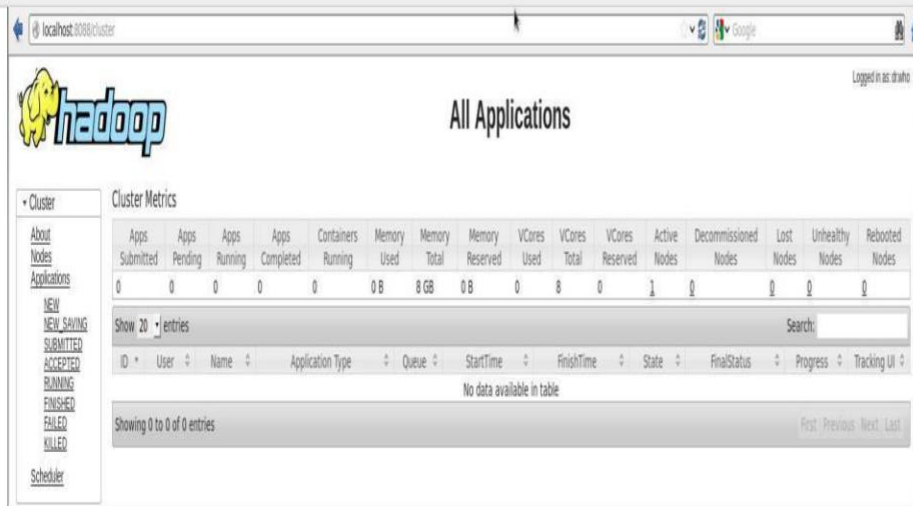
```
http://localhost:50070/
```



Step 5: Verify All Applications for Cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

<http://localhost:8088/>



C .Fully Distributed mode

Compatibility Requirements

S.No	Category	Supported
1	Languages	Java, Python, Perl, Ruby etc.
2	Operating System	Linux (Server Deployment) Mostly preferred, Windows (Development only), Solaris.
3	Hardware	32 bit Linux (64 bit for large deployment)

Installation Items

S.No	Item	Version
1	jdk-6u25-linux-i586.bin	Java 1.6 or higher
2	hadoop-0.20.2-cdh3u0.tar.gz	Hadoop 0.20.2

Note: Both Items are required to be installed on Namenode and Datanode machines

Installation Requirements

S.No	Requirement	Reason
1	Operating system – Linux recommended for server deployment (Production env.)	
2	Language – Java 1.6 or higher	
3	Ram – at least 3 GB/node	
4	Hard disk – at least 1 TB	For namenode machine.
5	Should have root credentials	For changing some system files you need admin permissions.

High level Steps

Step #	Activity	Check
1	Binding IP address with the host name under /etc/hosts	
2	Setting passwordless SSH	
3	Installing Java	
4	Installing Hadoop	
5	Setting JAVA HOME and HADOOP HOME variables	
6	Updating .bash_profile file for hadoop	
7	Creating required folders for namenode and datanode	
8	Configuring the .xml files	
9	Setting the masters and slaves in all the machines	
10	Formatting the namenode	
11	Starting the Dfs services and mapred services	
12	Stopping all services	

Binding IP address with the host names

Before starting the installation of hadoop, first you need to bind the IP address of the machines along with their host names under /etc/hosts file.

First check the hostname of your machine by using following command :

```
$ hostname
```

Open /etc/hosts file for binding IP with the hostname

```
$ vi /etc/hosts
```

Provide ip & hostname of the all the machines in the cluster

e.g: 10.11.22.33 hostname1

10.11.22.34 hostname2

Setting Passwordless SSH login

SSH is used to login from one system to another without requiring passwords. This will be required when you run a cluster, it will not prompt you for the password again and again.

First log in on Host1 (**hostname of namenode machine**) as hadoop user and generate a pair of authentication keys. Command is:

```
hadoop@Host1$ ssh-keygen -t rsa
```

Note: Give the hostname which you got in step 5.3.1. Do not enter any passphrase if asked. Now use ssh to create a directory ~/.ssh as user hadoop on Host2 (**Hostname other than namenode machine**).

```
hadoop@Host1$ ssh hadoop@Host2 mkdir -p .ssh
```

```
hadoop@Host2's password:
```

Finally append Host1's new public key to hadoop@Host2: .ssh/authorized_keys and enter Host2's password one last time:

```
hadoop@Host1$ cat /home/hadoop/.ssh/id_rsa.pub | ssh hadoop@Host2 ' cat >> .ssh/authorized_keys '
```

```
hadoop@Host2's password:
```

From now on you can log into Host2 as hadoop from Host1 without password:

```
hadoop@Host1$ ssh hadoop@Host2
```

```
Host2@hadoop$
```

NOTE: Do the following changes:

- Change the permissions of .ssh to 700
- Change the permissions of .ssh/authorized_keys to 640

Prepare for installation

Check for previous installed versions of java and hadoop on your machine

```
$ rpm -qa | grep java
```

It will display **fully qualified paths** of the version installed.

Remove all the previous version of Java and Hadoop installed on the machine.

```
$ rpm -e softwarename or path-name
```

NOTE: All the installations and extractions are being done in /home/hadoop/

Installing Java

Use the JDK bin file (jdk-6u25-linux-i586.bin) for installing java on your machine . Copy the .bin file in /home/hadoop/

Execute the command “./jdk-6u25-linux-i586.bin” in /home/hadoop/ (which will unzip the contents into folder **jdk1.6.0_25)**

Extract the hadoop package

Syntax

```
$ tar -xzf <hadoop-tar-package>
```

```
$ tar -xzf hadoop-0.20.0-cdh3u9.tar.gz
```

Configuring HADOOP_HOME

Check whether HADOOP_HOME is set up to the folder containing hadoop_core_VERSION.jar using

```
$ echo $HADOOP_HOME
```

If not set then set it

```
$ export HADOOP_HOME=/home/hadoop/hadoop-version
```

For e.g.

```
$ cd /home/hadoop/
```

```
$ export HADOOP_HOME=/home/hadoop/hadoop-0.20.2-cdh3u0/
```

Setting JAVA_HOME

```
$ cd /home/hadoop/hadoop-0.20.0-cdh3u0/conf/
```

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



```
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 at remote nodes. The java implementation to use. Required. export JAVA_HOME=/home/hadoop/jdk1.6.0_25 Set the Java CLASSPATH elements. Optional. export HADOOP_CLASSPATH="${HADOOP_HOME}/lib/*:${HADOOP_HOME}/lib/**/*.jar" The maximum amount of heap to use, in MB. Default is 1024. export HADOOP_HEAPSIZE=1024
```

hadoop-env.sh file for setting

JAVA_HOME Press :wq to save and exit the file

You need to change the bash file also .

```
$ vi ~/.bash_profile
```

```
# Set the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# Use specific environments and startup programs
export HADOOP_HOME="/home/hadoop/hadoop-0.20.2-cdh3u0"
export JAVA_HOME="/home/hadoop/jdk1.6.0_25"
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$PATH
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib
#Adding flume path
export PATH=$PATH:/home/hadoop/flume/flume-0.9.2/bin
```

[bash profile file for setting environment variables and jar files](#)

Check for hadoop installation confirmation

Run hadoop command to confirm whether the installation is successful.

```
$ cd <hadoop-home-directory>
```

Standard Path

```
$ cd /home/hadoop/ hadoop-0.20.0-cdh3u0/
```

```
$ bin/hadoop
```

On successful installation you should get the following message.

```
[hadoop@hadoop-0.20.2-cdh3u0]$ bin/hadoop
Usage: hadoop [--config confdir] COMMAND
where COMMAND is one of:
  namenode -format      format the DFS filesystem
  secondarynamenode    run the DFS secondary namenode
  namenode              run the DFS namenode
  datanode              run a DFS datanode
  dfsadmin             run a DFS admin client
  mradmin              run a Map-Reduce admin client
  fsck                 run a DFS filesystem checking utility
  fs                   run a generic filesystem user client
  balancer              run a cluster balancing utility
  fetchdt              fetch a delegation token from the NameNode
  jobtracker           run the MapReduce job Tracker node
  pipes                run a Pipes job
  tasktracker          run a MapReduce task Tracker node
  job                  manipulate MapReduce jobs
  queue                get information regarding JobQueues
  version              print the version
  jar <jar>            run a jar file
  distcp <srcurl> <dsturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
  oiv                  apply the offline fsimage viewer to an fsimage
  classpath            prints the class path needed to get the
                      Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
  or
  CLASSNAME            run the class named CLASSNAME
Most commands print help when invoked w/o parameters.
hadoop@hadoop-0.20.2-cdh3u0$
```

CONFIGURING HADOOP IN FULLY DISTRIBUTED MODE

Create the `dfs.name.dir` local directories on namenode machine

```
$ cd /home/hadoop/
```

```
$ mkdir -p data/1/dfs/nn
```

Creating the directories for storing the Data blocks and the temporary directory for storing process ids on datanode machines

```
$ cd /home/hadoop/
```

```
$ mkdir -p data/1/dfs/dn data/2/dfs/dn data/3/dfs/dn $
```

```
mkdir -p /home/hadoop/ tmp
```

Creating the directories for storing the temporary data (Task Tracker) and the system files for Map/Reduce jobs

```
$ cd /home/hadoop/
```

```
$ mkdir -p data/1/mapred/local data/2/mapred/local data/3/mapred/local $
```

```
mkdir -p /home/hadoop/mapred/system
```

Give full permission to all folder under `/home/hadoop/`

```
$ cd /home/hadoop/
```

```
$ chmod 777 *
```

Navigate to `/home/hadoop/hadoop-0.20.0-cdh3u0/conf` directory

```
$ cd /home/hadoop/hadoop-0.20.0-cdh3u0/conf
```

Set up the configuration files under `/home/hadoop/hadoop-0.20.0-cdh3u0/conf/`

Core-Site.xml

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

Parameters of `core-site.xml`

fs.default.name □ URL for the Name Node

hadoop.tmp.dir □ URL for the temporary data

dfs.replication □ This property specifies the number of times the file has to be replicated on cluster.

```

<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://120.20.201.241:9020</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/hadoop/tmp</value>
</property>
<property>
<name>mapred.job.tracker</name>
<value>120.20.201.241:9010</value>
</property>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
</configuration>

```

[core-site.xml](#)

hdfs-site.xml

\$ vi hdfs-site.xml

Parameters of hdfs-site.xml

dfs.name.dir □ This property specifies the directories where the NameNode stores its metadata and edit logs. Represented by the /home/hadoop/data/1/dfs/nn path examples.

dfs.data.dir □ This property specifies the directories where the DataNode stores blocks. Represented by the /home/hadoop/data/1/dfs/dn, /home/hadoop/data/2/dfs/dn , /home/hadoop/data/3/dfs/dn

```

<configuration>
<property>
<name>dfs.name.dir</name>
<value>/home/hadoop/data/1/dfs/nn</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>/home/hadoop/data/1/dfs/dn,/home/hadoop/data/2/dfs/dn,/home/hadoop/data/3/dfs/dn</value>
</property>
</configuration>

```

[hdfs-site.xml](#)

Press :wq to save and exit the file

mapred-site.xml

\$ vi mapred-site.xml

Parameters of mapred-site.xml

mapred.local.dir □ This property specifies the directories where the TaskTracker will store temporary data and intermediate map output files while running Map Reduce jobs.

Eg./home/hadoop/data/1/mapred/local,/home/hadoop/data/2/mapred/local,
/home/hadoop/data/3/mapred/local.

mapred.system.dir □ Path on the HDFS where the Map Reduce framework stores system files
e.g./home/hadoop/mapred/system/.

mapred.job.tracker □ Host or IP and port of Job Tracker.

```
<configuration>
<property>
<name>mapred.local.dir</name>
<value>/home/hadoop/data/1/mapred/local,/home/hadoop/data/2/mapred/local,/home/hadoop/data/3/mapred/local</value>
</property>
<property>
  <name>mapred.job.tracker</name>
  <value>172.21.129.241:9010</value>
</property>
<property>
  <name>mapred.system.dir</name>
  <value>/home/hadoop/mapred/system</value>
</property>
</configuration>
```

[mapred-site.xml](#)

Press :wq to save and exit the file

Set the correct owner and permissions of the local directories:

Directory	Owner	Permissions
dfs.name.dir	hdfs:hadoop	drwx-----
dfs.data.dir	hdfs:hadoop	drwx-----
mapred.local.dir	mapred:hadoop	drwxr-xr-x

```
$ chmod 700 /home/hadoop/data/1/dfs/nm/
$  chmod  700  /home/hadoop/data/1/dfs/dn/  /home/hadoop/data/2/dfs/dn/
```

```
/home/hadoop/data/3/dfs/dn/  
$ chmod 755 /home/hadoop/data/1/mapred/local/  
/home/hadoop/data/2/mapred/local/ /home/hadoop/data/3/mapred/local/
```

Setting up the masters and slaves

vi conf/masters

hostname of machine acting as a SecondaryNamenode

vi slaves

hostname of machines acting as a Datanode & TaskTrackers

Formatting the namenode

You need to format the namenode every time you start the dfs services. This is because every time you start the services it causes some files to be written in the namenode folder which may get duplicated when you run the services for the second time. Do not format a running Hadoop namenode, otherwise it will cause all your data in the HDFS filesystem to be erased.

```
$ cd /home/hadoop/hadoop-0.20.0-cdh3u0/
```

```
$ bin/hadoop namenode -format
```

Note : Give “Y” when it asks for re-format

Starting dfs service

Run the command

```
$ /bin/start-dfs.sh
```

on the machine you want the namenode to run on. This will bring up HDFS with the namenode running on the machine you ran the previous command on, and datanodes on the machines listed in the conf/slaves file.

```
$ cd /home/hadoop/hadoop-0.20.0-cdh3u0/
```

```
$ ./bin/start-dfs.sh
```

NOTE: For any problems check the log files under all the machines under /home/hadoop/hadoop-0.20.2-cdh3u0/logs/ and refer to the troubleshooting guide for the same.

Starting mapred service

For mapred services run the following command on the machine you want jobtracker to run on (in my case it was namenode machine)
you can choose other machine also.

\$./bin/start-mapred.sh

Checking the DFS service report

\$./bin/hadoop dfsadmin -report

Checking on web interface **DFS SERVICE**

<http://ip-address of namenode machine:50070/>

Checking on web interface **Mapred Job**

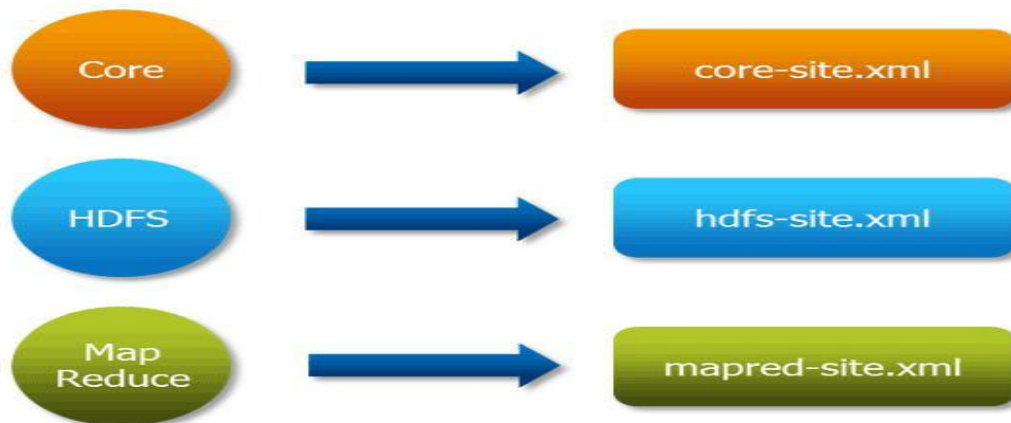
<http://ip of namenode machine:50030/>

Stopping dfs and mapred services

cd /home/hadoop/hadoop-0.20.0-cdh3u0/

\$./bin/stop-all.sh

4 Configuring XML files.



Hadoop Cluster Configuration Files

Configuration Filenames	Description of Log Files
hadoop-env.sh	Environment variables that are used in the scripts to run Hadoop.
core-site.xml	Configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.
hdfs-site.xml	Configuration settings for HDFS daemons, the namenode, the secondary namenode and the data nodes.
mapred-site.xml	Configuration settings for MapReduce daemons : the job-tracker and the task-trackers.
masters	A list of machines (one per line) that each run a secondary namenode.
slaves	A list of machines (one per line) that each run a datanode and a task-tracker.

All these files are available under ‘conf’ directory of Hadoop installation directory.

Here is a listing of these files in the File System:

```
ubuntu@ip-10-251-81-223:~$ ls
hadoop-1.2.0  hadoop-1.2.0.tar.gz  keypairs
ubuntu@ip-10-251-81-223:~$ unalias -a
ubuntu@ip-10-251-81-223:~$ cd
ubuntu@ip-10-251-81-223:~$ ls
hadoop-1.2.0  hadoop-1.2.0.tar.gz  keypairs
ubuntu@ip-10-251-81-223:~$ cd hadoop-1.2.0/
ubuntu@ip-10-251-81-223:~/hadoop-1.2.0$ ls
bin          hadoop-ant-1.2.0.jar      ivy          sbin
build.xml   hadoop-client-1.2.0.jar  ivy.xml     share
c++         hadoop-core-1.2.0.jar    lib         src
CHANGES.txt hadoop-examples-1.2.0.jar libexec     webapps
conf        hadoop-minicluster-1.2.0.jar LICENSE.txt
contrib     hadoop-test-1.2.0.jar   NOTICE.txt
docs       hadoop-tools-1.2.0.jar  README.txt
ubuntu@ip-10-251-81-223:~/hadoop-1.2.0$ cd conf/
ubuntu@ip-10-251-81-223:~/hadoop-1.2.0/conf$ ls
capacity-scheduler.xml  hadoop-policy.xml      slaves
configuration.xml       hdfs-site.xml         ssl-client.xml.example
core-site.xml           log4j.properties     ssl-server.xml.example
fair-scheduler.xml     mapred-queue-acls.xml taskcontroller.cfg
hadoop-env.sh          mapred-site.xml       task-log4j.properties
hadoop-metrics2.properties  masters
ubuntu@ip-10-251-81-223:~/hadoop-1.2.0/conf$
```

Let’s look at the files and their usage one by one!

hadoop-env.sh

This file specifies environment variables that affect the JDK used by **Hadoop Daemon** (bin/hadoop).

As Hadoop framework is written in *Java* and uses *Java Runtime environment*, one of the important environment variables for Hadoop daemon is **\$JAVA_HOME** in **hadoop-env.sh**. This variable directs Hadoop daemon to the *Java path* in the system.

```
# The java implementation to use. Required.
export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk-amd64
```

This file is also used for setting another *Hadoop daemon execution environment* such as **heap size** (**HADOOP_HEAP**), **hadoop home** (**HADOOP_HOME**), **log file location** (**HADOOP_LOG_DIR**), etc.

Note: For the simplicity of understanding the cluster setup, we have configured only necessary parameters to start a cluster.

The following three files are the important configuration files for the runtime environment settings of a Hadoop cluster.

core-site.sh

This file informs Hadoop daemon where NameNode runs in the cluster. It contains the configuration settings for Hadoop Core such as I/O settings that are common to *HDFS* and *MapReduce*.

```
<?xml version="1.0"?>
<!-- For site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://ec2-54-214-206-65.us-west-2.compute.amazonaws.com:8020</value>
  </property>
</configuration>
```

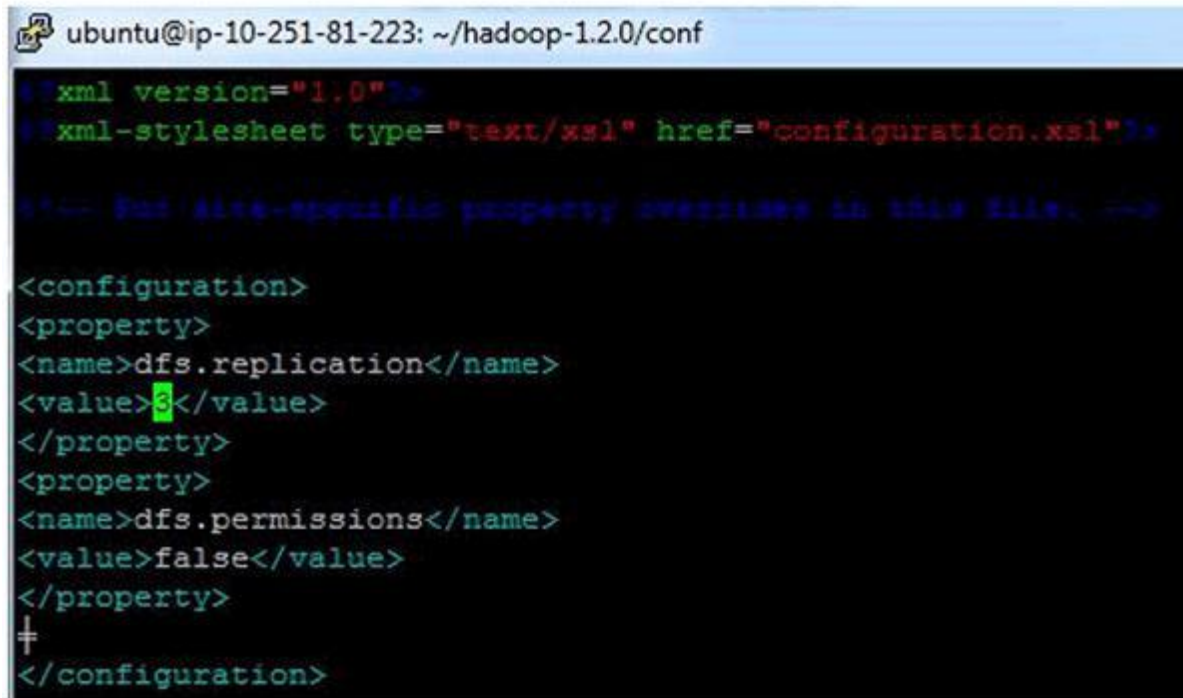
Where *hostname* and *port* are the machine and port on which NameNode daemon runs and listens. It also informs the Name Node as to which IP and port it should bind. The commonly used port is **8020** and you can also specify IP address rather than hostname.

hdfs-site.sh

This file contains the configuration settings for *HDFS daemons; the Name Node, the Secondary Name Node, and the data nodes*.

You can also configure **hdfs-site.xml** to specify default block replication and permission checking on HDFS. The actual number of replications can also be specified when the file is created. The default is used if replication is not specified in create time.

The value “true” for property ‘dfs.permissions’ enables permission checking in HDFS and the value “false” turns off the permission checking. Switching from one parameter value to the other does not change the mode, owner or group of files or directories.



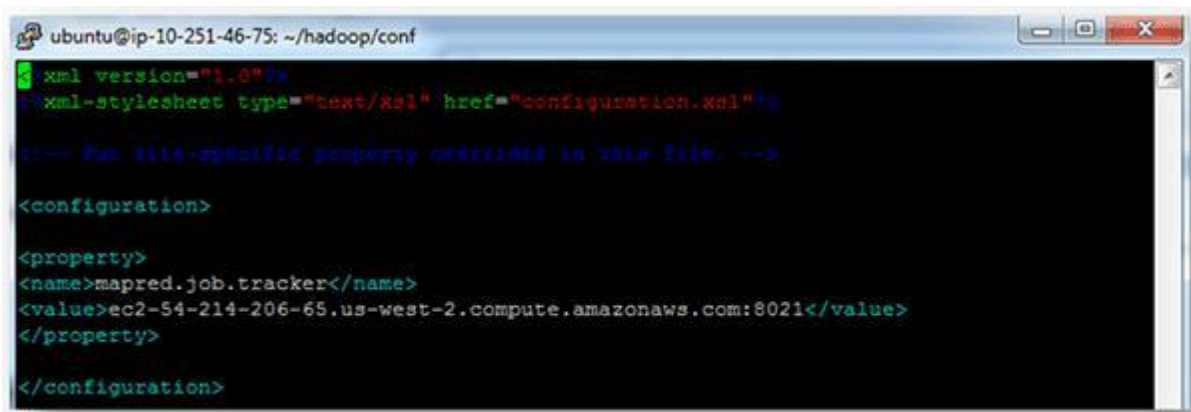
```
ubuntu@ip-10-251-81-223: ~/hadoop-1.2.0/conf
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

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

<configuration>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
<property>
<name>dfs.permissions</name>
<value>>false</value>
</property>
+
</configuration>
```

mapred-site.sh

This file contains the configuration settings for MapReduce daemons; *the job tracker and the task-trackers*. The `mapred.job.tracker` parameter is a *hostname* (or IP address) and *port* pair on which the Job Tracker listens for RPC communication. This parameter specify the location of the Job Tracker to Task Trackers and MapReduce clients.



```
ubuntu@ip-10-251-46-75: ~/hadoop/conf
<?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>ec2-54-214-206-65.us-west-2.compute.amazonaws.com:8021</value>
</property>

</configuration>
```

You can replicate all of the four files explained above to all the Data Nodes and Secondary Namenode. These files can then be configured for any node specific configuration e.g. in case of a different **JAVA HOME** on one of the Datanodes.

The following two file ‘masters’ and ‘slaves’ determine the master and slave Nodes in Hadoop CLUSTER.

Masters

This file informs about the Secondary Namenode location to **hadoop daemon**. The ‘masters’ file at Master server contains a hostname Secondary Name Node servers.

Slaves

- ✓ Contains a list of hosts, one per line, that are to host **DataNode** and **TaskTracker** servers.

Masters

- ✓ Contains a list of hosts, one per line, that are to host **Secondary NameNode** servers.

The ‘masters’ file on Slave Nodes is blank.

Slaves

The ‘slaves’ file at Master node contains a list of hosts, one per line, that are to host Data Node and Task Tracker servers.



```
ubuntu@ip-10-251-46-75: ~/hadoop/conf
ec2-54-218-170-127.us-west-2.compute.amazonaws.com
ec2-54-202-24-115.us-west-2.compute.amazonaws.com
```

The ‘slaves’ file on Slave server contains the IP address of the slave node. Notice that the ‘slaves’ file at Slave node contains only its own IP address and not of any other Data Nodes in the cluster.