UNIT II

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.**

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 records append operations. Snapshot creates a copy of a file or a directory treat 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.

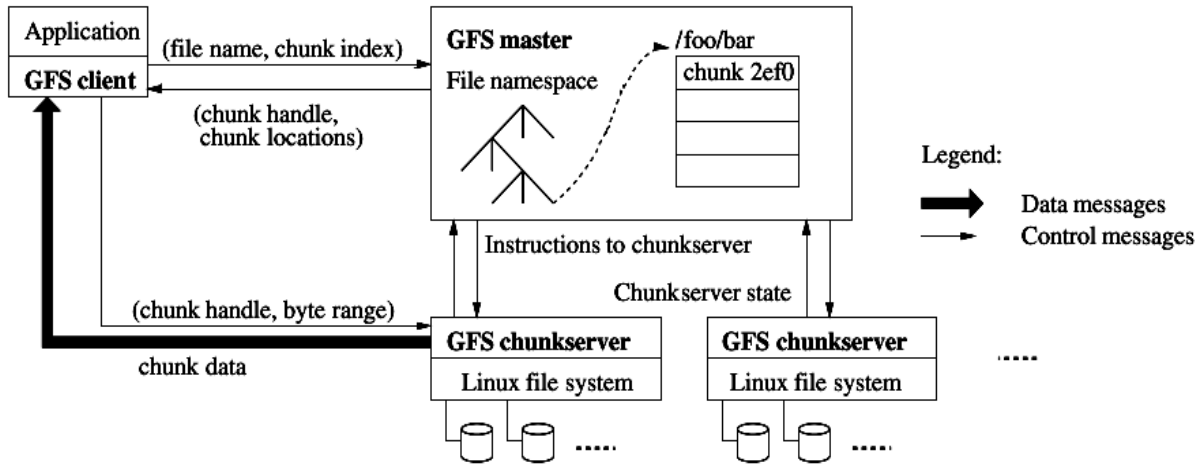


Figure 1: GFS Architecture

Each of these is typically a commodity Linux machine running a user-level server process. It is easy to run both a chunk server 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.

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 Application 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 chunkservers 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.

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

- 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.
- 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.
- 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

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

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 (terrabytes 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

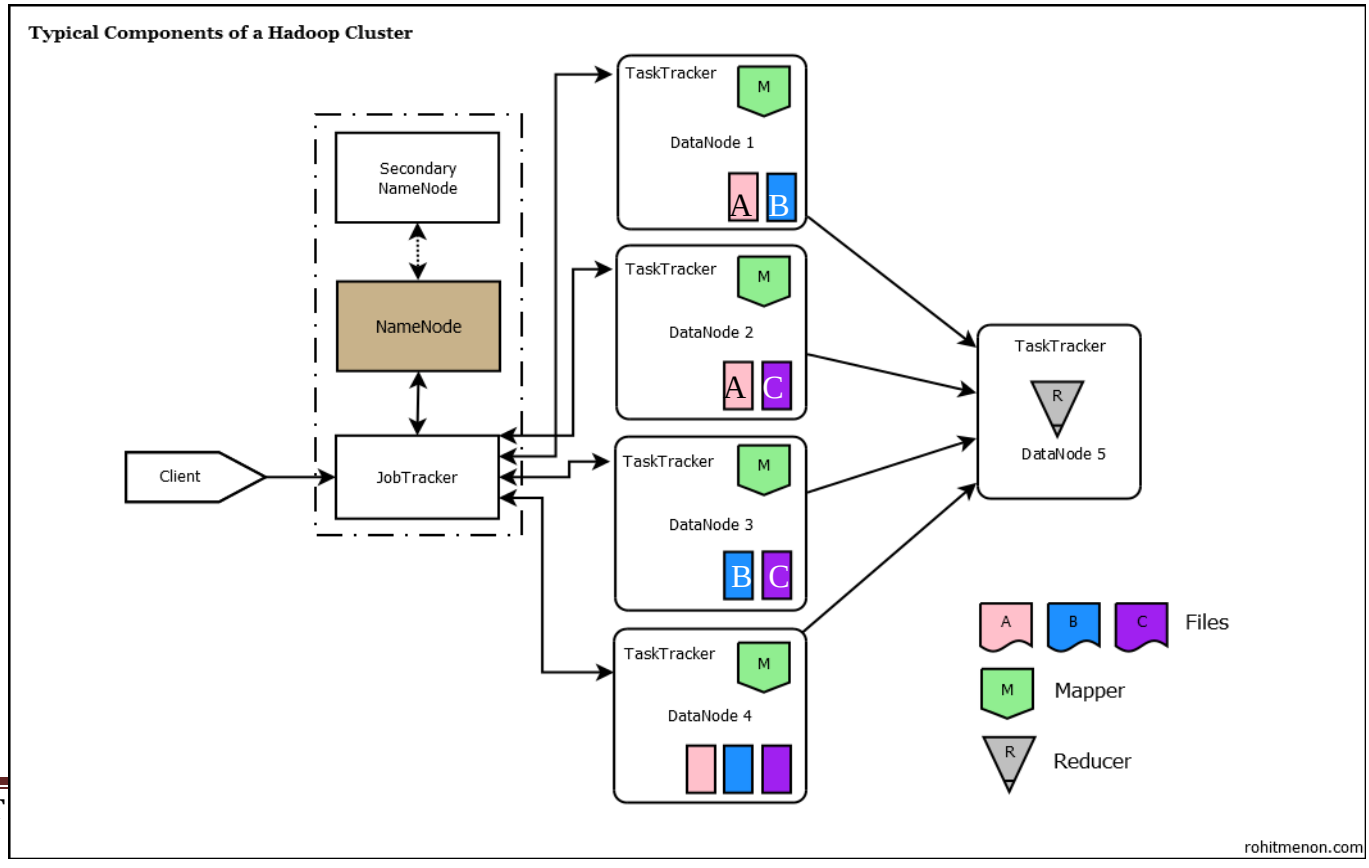
DataNode 1 (Slave) – TaskTracker, DataNode

DataNode 2 (Slave) – TaskTracker, DataNode

DataNode 3 (Slave) – TaskTracker, DataNode

DataNode 4 (Slave) – TaskTracker, DataNode

DataNode 5 (Slave) – TaskTracker, DataNode



A

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, DataNode
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 a follows:

- A Client (usaually a MapReduce program) submits a job to the*JobTracker*.
- 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.
- 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.
- The progress of the operation is relayed back to the *JobTracker* by the*TaskTracker*.
- On completion of the Map task an intermediate file is created on the local filesystem of the *TaskTracker*.
- Results from Map tasks are then passed on to the Reduce task.
- The Reduce tasks works on all data received from map tasks and writes the final output to HDFS.
- 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.

Introducing and Configuring Hadoop cluster

A. Local
B. Pseudo-distributed mode
C. Fully Distributed mode

We all know that Apache Hadoop is an open source framework that allows distributed processing of large sets of data set across different clusters using simple programming. Hadoop has the ability to scale up to thousands of computers from a single server. Thus in these conditions installation of Hadoop becomes most critical. We can install Hadoop in three different modes:

- *Standalone mode - Single Node Cluster*
- *Pseudo distributed mode - Single Node Cluster*

- *Distributed mode. - Multi Node Cluster*

Purpose for Different Installation Modes

When Apache Hadoop is used in a production environment, multiple server nodes are used for distributed computing. But for understanding the basics and playing around with Hadoop, single

node installation is sufficient. There is another mode known as 'pseudo distributed' mode. This mode is used to simulate the multi node environment on a single server.

In this document we will discuss how to install Hadoop on Ubuntu Linux. In any mode, the system should have java version 1.6.x installed on it.

Standalone Mode Installation

Now, let us check the standalone mode installation process by following the steps mentioned below.

Install Java

Java (JDK Version 1.6.x) either from Sun/Oracle or Open Java is required.

- Step 1 - If you are not able to switch to OpenJDK instead of using proprietary Sun JDK/JRE, install sun-java6 from Canonical Partner Repository by using the following command.

  Note: The *Canonical Partner Repository* contains free of cost closed source third party software. But the Canonical does not have access to the source code instead they just package and test it.

  Add the canonical partner to the apt repositories using -
  $ sudo add-apt-repository "deb http://archive.canonical.com/lucid partner"
- Step 2 - Update the source list.
  $ sudo apt-get update
- Step 3 - Install JDK version 1.6.x from Sun/Oracle.
  $ sudo apt-get install sun-java6-jdk
- Step 4 - Once JDK installation is over make sure that it is correctly setup using - version 1.6.x from Sun/Oracle.

  user@ubuntu:~# java -version

  java version "1.6.0_45"

  Java(TM) SE Runtime Environment (build 1.6.0_45-b02)

  Java HotSpot(TM) Client VM (build 16.4-b01, mixed mode, sharing)

Add Hadoop User

- Step 5 - Add a dedicated Hadoop unix user into you system as under to isolate this installation from other software -

  $ sudo adduser hadoop_admin

Download the Hadoop binary and install

- Step 6 - Download Apache Hadoop from the apache web site. Hadoop comes in the form of tar-gx format. Copy this binary into the /usr/local/installables folder. The folder - installables should be created first under /usr/local before this step. Now run the following commands as sudo

  $ cd /usr/local/installables

  $ sudo tar xzf hadoop-0.20.2.tar.gz

  $ sudo chown -R hadoop_admin /usr/local/hadoop-0.20.2

Define env variable - JAVA_HOME

- Step 7 - Open the Hadoop configuration file (hadoop-env.sh) in the location - /usr/local/installables/hadoop-0.20.2/conf/hadoop-env.sh and define the JAVA_HOME as under –

  export JAVA_HOME=path/where/jdk/is/installed

  (e.g. /usr/bin/java)

Installation in Single mode

- Step 8 - Now go to the HADOOP_HOME directory (location where HADOOP is extracted) and run the following command –

  $ bin/hadoop

  The following output will be displayed –

  Usage: hadoop [--config confdir] COMMAND

  Some of the COMMAND options are mentioned below. There are other options available and can be checked using the command mentioned above.

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

The above output indicates that Standalone installation is completed successfully. Now you can run the sample examples of your choice by calling –

$ bin/hadoop jar hadoop-*-examples.jar <NAME> <PARAMS>

Pseudo Distributed Mode Installation

This is a simulated multi node environment based on a single node server.

Here, the first step required is to configure the SSH in order to access and manage the different nodes. It is mandatory to have the SSH access to the different nodes. Once the SSH is configured, enabled

and is accessible we should start configuring the Hadoop. The following configuration files need to be modified:

- conf/core-site.xml
- conf/hdfs-site.xml
- conf/mapred.xml

Open the all the configuration files in vi editor and update the configuration.

Configure core-site.xml file:

$ vi conf/core-site.xml

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/tmp/hadoop-${user.name}</value>
</property>
</configuration>
```

Configure hdfs-site.xml file:

$ vi conf/hdfs-site.xml

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

Configure mapred.xml file:

$ vi conf/mapred.xml

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>localhost:9001</value>
</property>
</configuration>
```

Once these changes are done, we need to format the name node by using the following command. The command prompt will show all the messages one after another and finally success message.

```
$ bin/hadoop namenode –format
```

Our setup is done for pseudo distributed node. Let's now **start the single node cluster** by using the following command. It will again show some set of messages on the command prompt and start the server process.

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

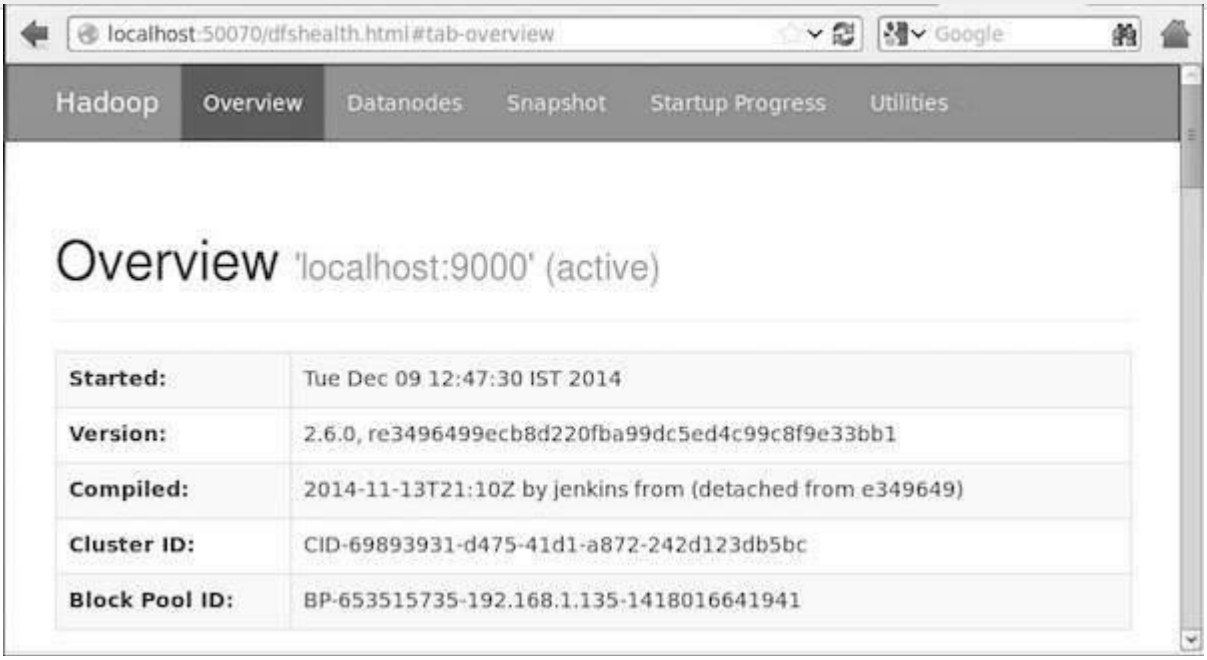
Now we should check the status of Hadoop process by executing the jps command as shown below. It will show all the running processes.

```
$ jps
14799 NameNode
14977 SecondaryNameNode
15183 DataNode
15596 JobTracker
15897 TaskTracker
```

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/
```



Stopping the Single node Cluster: We can stop the single node cluster by using the following command. The command prompt will display all the stopping processes.

```
$ bin/stop-all.sh
stopping jobtracker
localhost: stopping tasktracker
stopping namenode
localhost: stopping datanode
```

localhost: stopping secondarynamenode

Fully Distributed Mode Installation

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

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

Before we start the distributed mode installation, we must ensure that we have the pseudo distributed setup done and we have at least two machines, one acting as master and the other acting as a slave. Now we run the following commands in sequence.

- $ bin/stop-all.sh- Make sure none of the nodes are running

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).**

Now we open the two files - conf/master and conf/slaves. The conf/master defines the name nodes of our multi node cluster. The conf/slaves file lists the hosts where the Hadoop Slave will be running.

- **Edit the conf/core-site.xml** file to have the following entries -

```
<property>
        <name>fs.default.name</name>
        <value>hdfs://master:54310</value>
</property>
```

- **Edit the conf/mapred-site.xml** file to have the following entries -

```
<property>
        <name>mapred.job.tracker</name>
        <value>hdfs://master:54311</value>
</property>
```

- **Edit the conf/hdfs-site.xml** file to have the following entries -

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

- **Edit the conf/mapred-site.xml** file to have the following entries -

```
<property>
        <name>mapred.local.dir</name>
        <value>${hadoop-tmp}/mapred/local</value>
  </property>
  <property>
        <name>mapred.map.tasks</name>
        <value>50</value>
  </property>
  <property>
        <name>mapred.reduce.tasks</name>
        <value>5</value>
```

```
</property>
```

Now start the master by using the following command.

bin/start-dfs.sh

Once started, check the status on the master by using jps command. You should get the following output –

```
14799 NameNode
15314 Jps
16977 secondaryNameNode
```

On the slave, the output should be as shown as:

```
15183 DataNode
15616 Jps
```

Now start the MapReduce daemons using the following command.

$ bin/start-mapred.sh

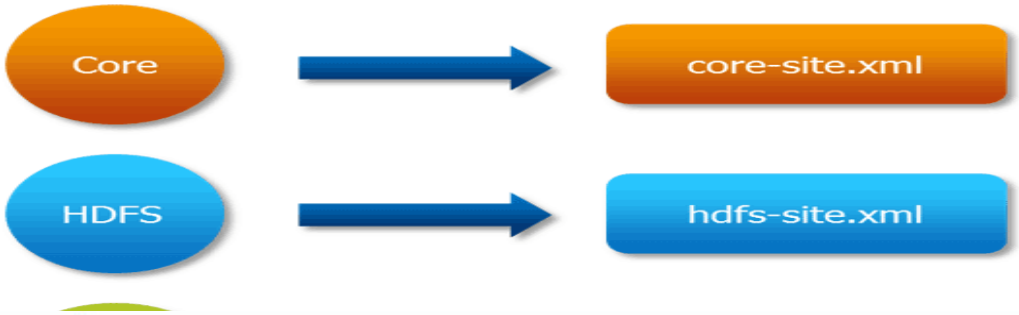
Once started, check the status on the master by using jps command. You should get the following output:

```
16017 Jps
14799 NameNode
15596 JobTracker
14977 SecondaryNameNode
```

On the slaves, the output should be as shown below.

```
15183 DataNode
15897 TaskTracker
16284 Jps
```

Configuring XML 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.

Hadoop Cluster Configuration Files

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.xsl           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.



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"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put 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.

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. -->

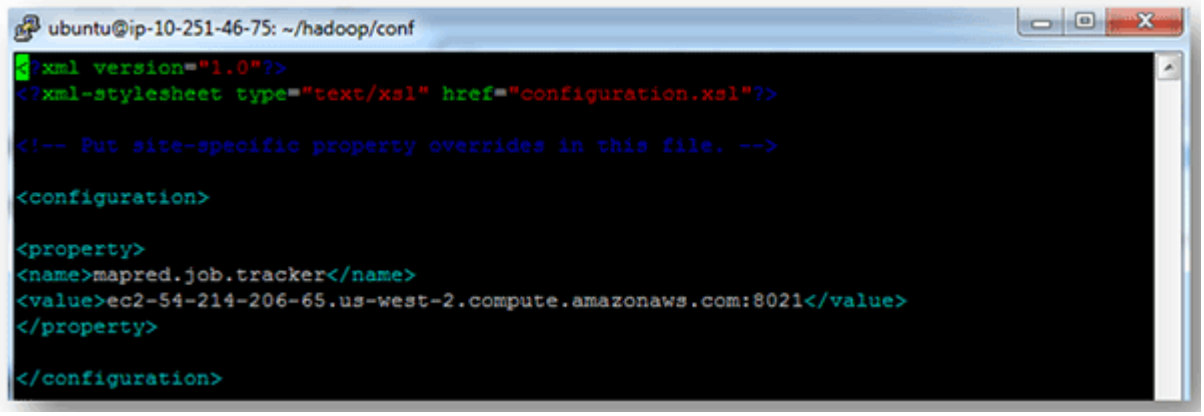
<configuration>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
<property>
<name>dfs.permissions</name>
<value>false</value>
</property>

</configuration>
```

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.

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.

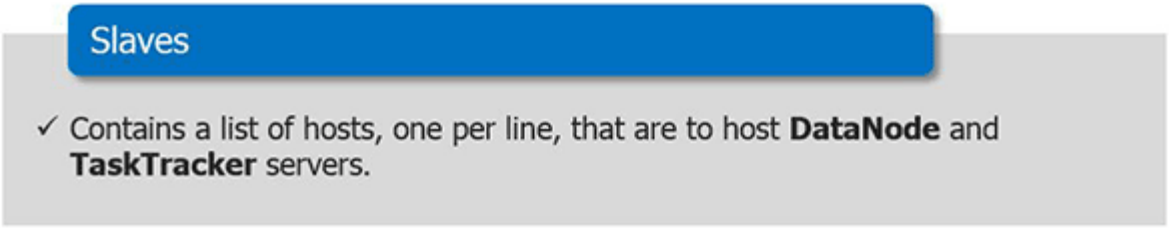


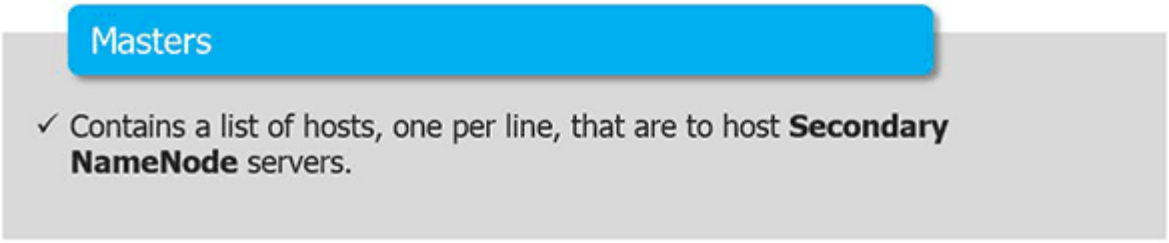
You can replicate all of the four files explained above to all the Data Nodes and Secondary Namenode. These files can then be configured for any node specific configuration e.g. in case of a different **JAVA HOME** on one of the Datanodes.

The following two file 'masters' and 'slaves' determine the master and salve 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.

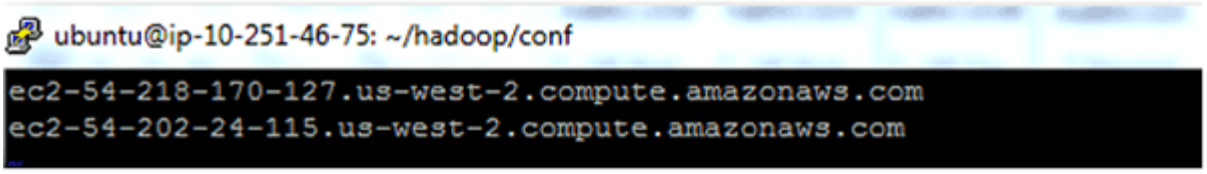




The 'masters' file on **Slave Nodes** is blank.

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.



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.