

UNIT-II

Working with Big Data: Google File System, Hadoop Distributed File System (HDFS) – Building blocks of Hadoop (Namenode, Datanode, Secondary Namenode, JobTracker, TaskTracker), Introducing and Configuring Hadoop cluster (Local, Pseudo-distributed mode, Fully Distributed mode), Configuring XML files.

Introduction:

What is Data?

Data is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things.

Data can be categorized into 3 different ways.

- Structured Data
- Semi Structured Data
- Un Structured Data

Structured Data:

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes.

Semi Structured Data:

Semi-structured data can contain both the forms of data. We can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS. Example of semi-structured data is a data represented in an XML file.

Un Structured Data:

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have

wealth of data available with them but unfortunately, they don't know how to derive value out of it since this data is in its raw form or unstructured format.

Big Data:

Big Data is also **data** but with a **huge size**. Big Data is a term used to describe a collection of data that is huge in volume and yet growing exponentially with time. In short such data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.

Examples Of Big Data

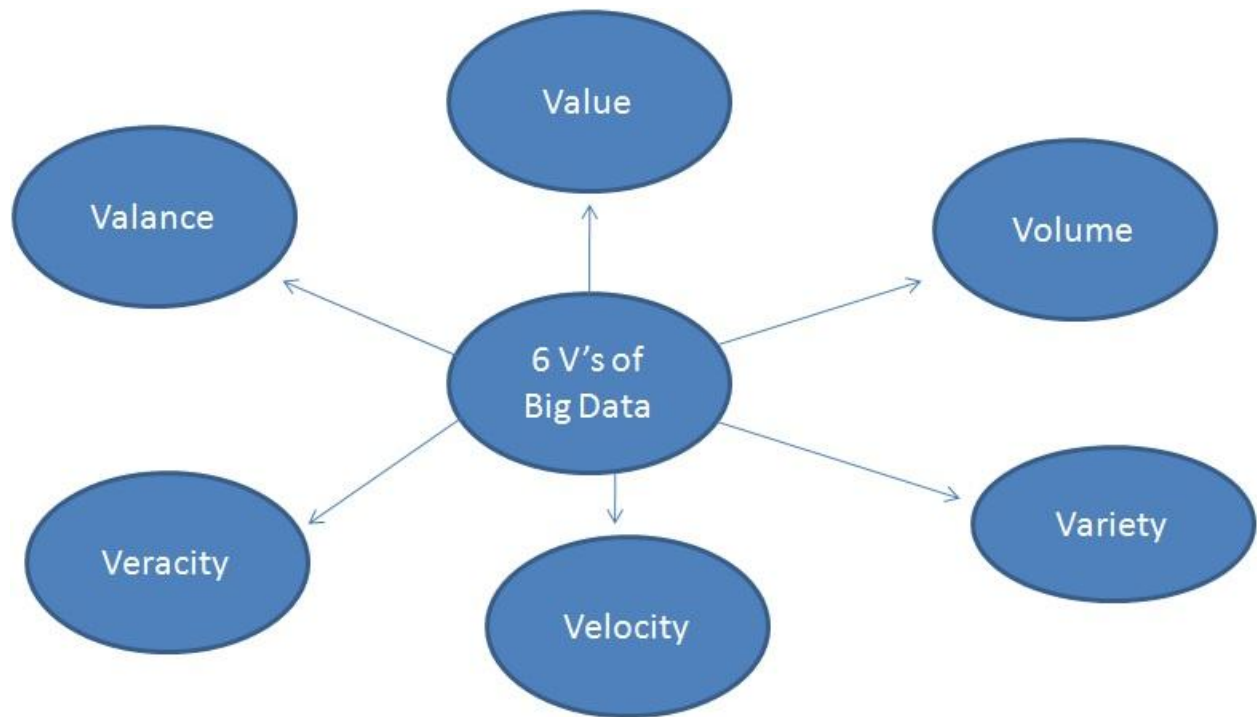
Following are some the examples of Big Data-

- ❖ The **New York Stock Exchange** generates about *one terabyte* of new trade data per day.
- ❖ The statistic shows that *500+terabytes* of new data get ingested into the databases of social media site **Facebook**, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc.
- ❖ A single **Jet engine** can generate *10+terabytes* of data in *30 minutes* of flight time. With many thousand flights per day, generation of data reaches up to many *Petabytes*.

Applications of Big Data:



Characteristics of Big Data:



Google File System (GFS):

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.

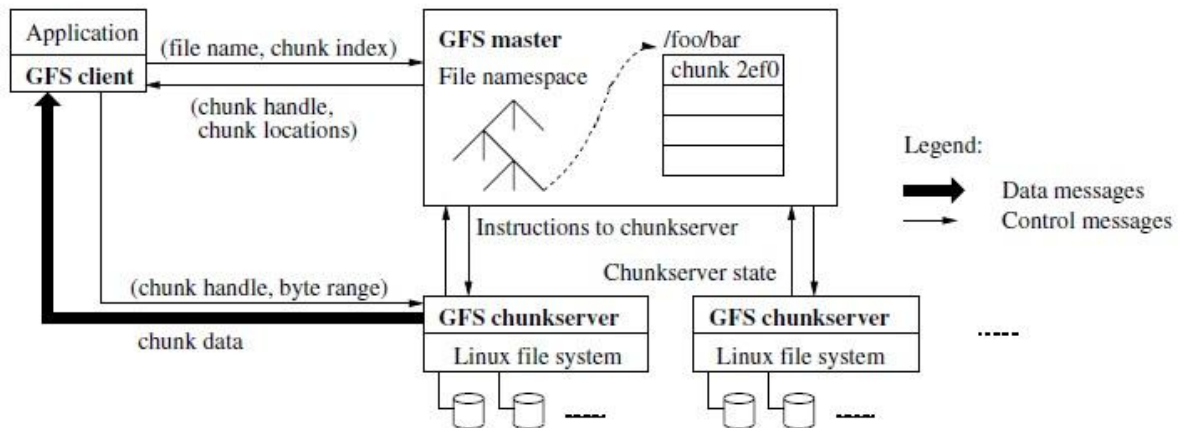
Moreover, GFS has snapshot and record append operations. Snapshot creates a copy of a file or a directory tree at low cost. Record append allows multiple clients to append data to the same file concurrently while guaranteeing the atomicity of each individual client's append.

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.



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 chunk servers. We do not provide the POSIX API and therefore need not hook into the Linux vnode layer.

Neither the client nor the chunk server 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.) Chunk servers 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 chunk servers 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 chunk server 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 chunk server 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 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 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 chunk server about its chunks at master startup and whenever a chunkserver joins the cluster.

In-Memory Data Structures

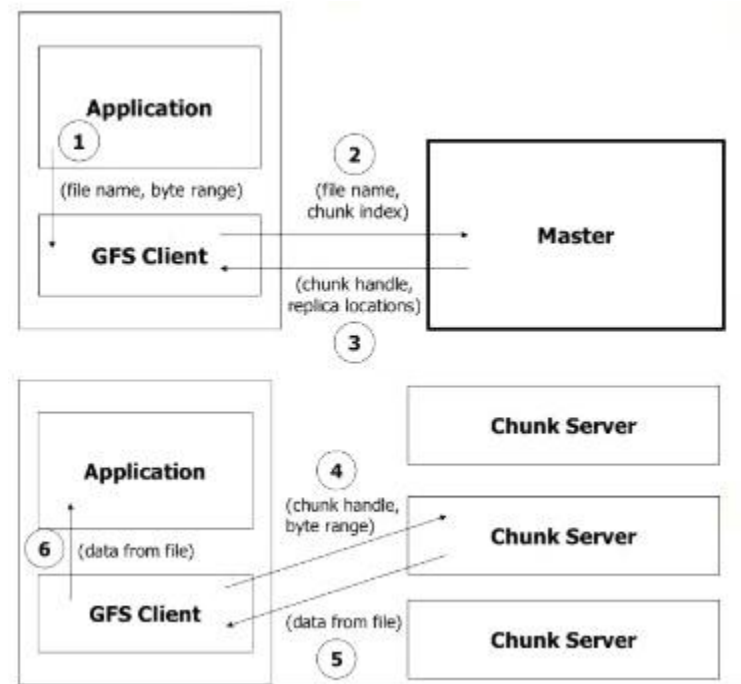
Since metadata is stored in memory, master operations are fast. Furthermore, it is easy and efficient for the master to periodically scan through its entire state in the background.

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

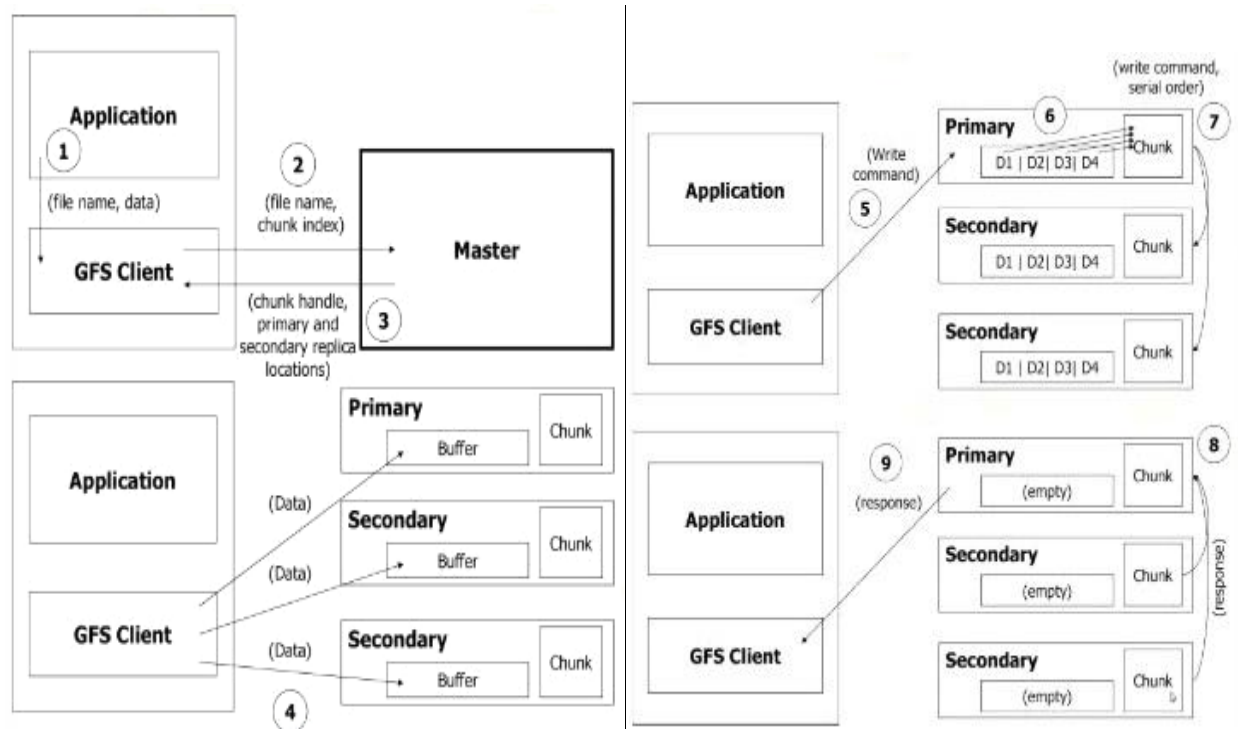
Consistency Model

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.

GFS Read Operation:



GFS Write Operation:



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 inmemory, 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.

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.

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject.

Assumptions and Goals

1)Hardware Failure

Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

2)Streaming Data Access

Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

3)Large Data Sets

Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

4)Simple Coherency Model

HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

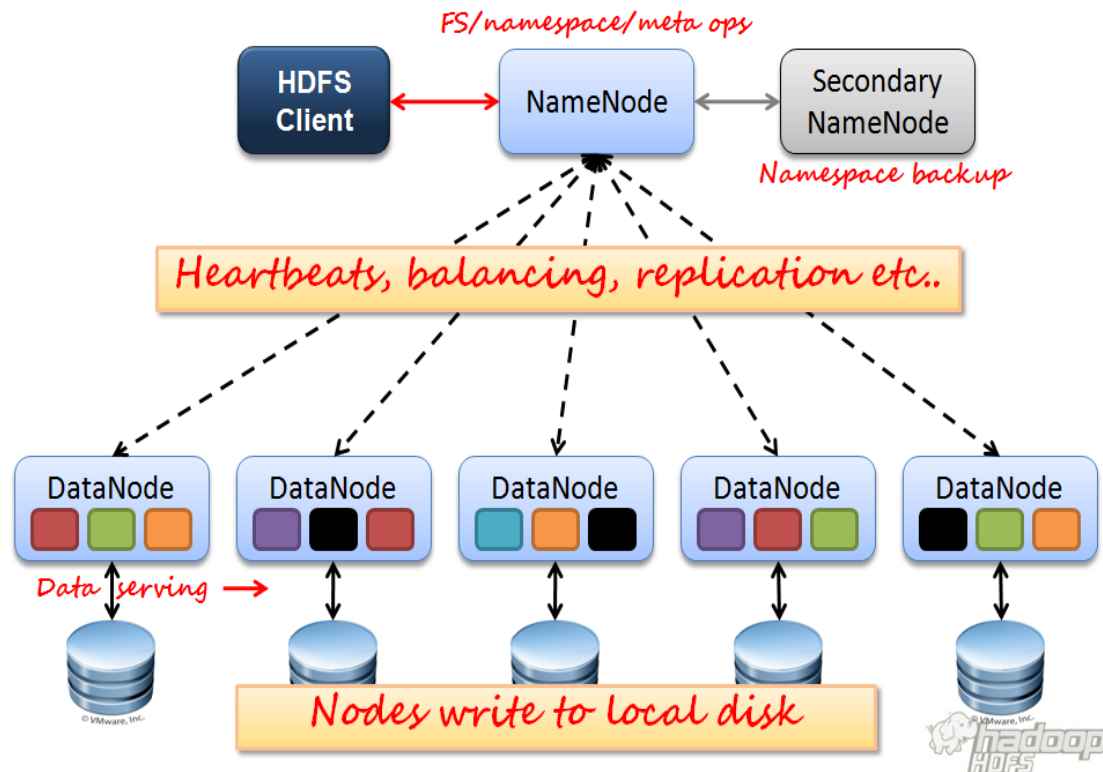
5)“Moving Computation is Cheaper than Moving Data”

A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

6)Portability Across Heterogeneous Hardware and Software Platforms

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

HDFS Architecture:

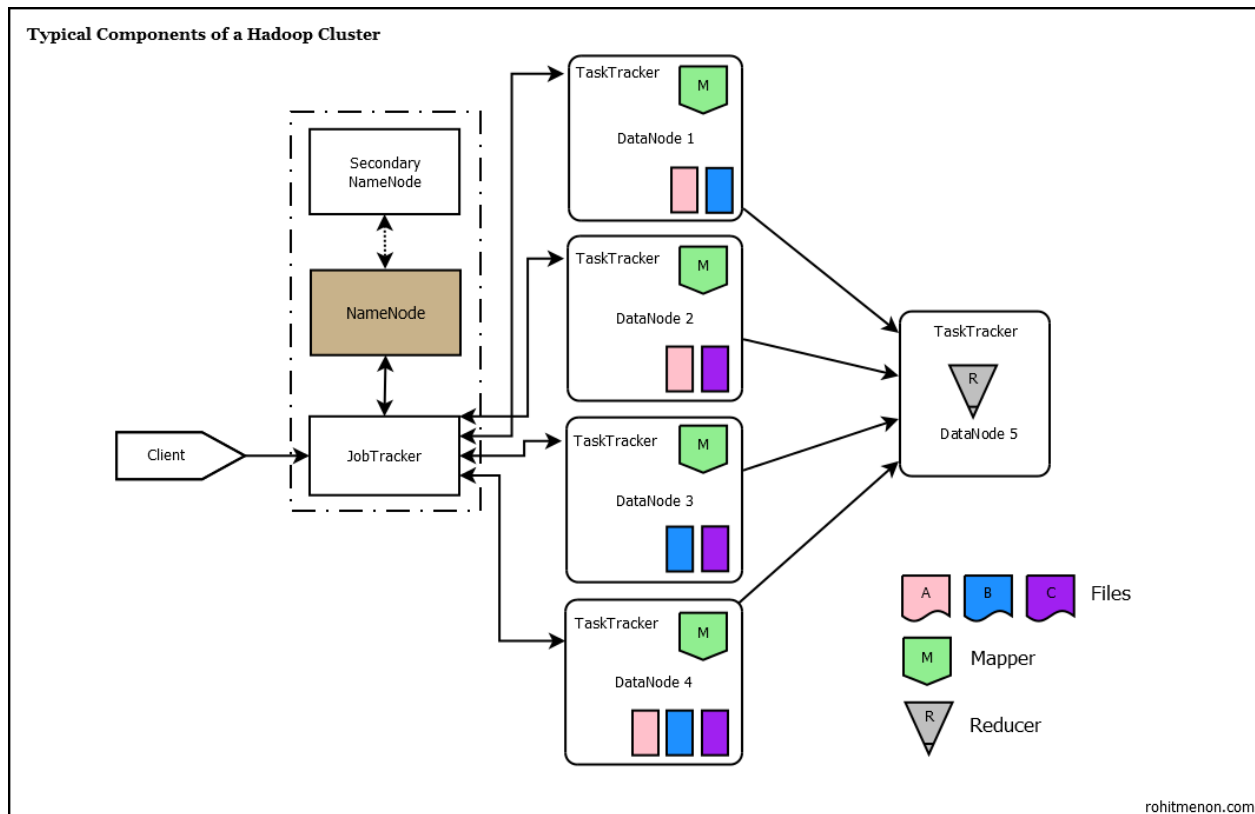


MapReduce

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.



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 than 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 its 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.

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 gets 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 work 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.

Configuring Hadoop cluster:

Hadoop cluster can configure in 3 different modes.

- ❖ Local mode or Standalone mode
- ❖ Pseudo distributed mode
- ❖ Fully distributed mode

Local mode or Standalone mode:

- Standalone mode is the default mode in which Hadoop runs. Standalone mode is mainly used for debugging where you don't really use HDFS.
- You can use input and output both as a local file system in standalone mode.
- You also don't need to do any custom configuration in the files- **mapred-site.xml**, **core-site.xml**, **hdfs-site.xml**.
- Standalone mode is usually the fastest Hadoop mode as it uses the local file system for all the input and output.

Here is the summarized view of the standalone mode:

- ❖ Used for debugging purpose
- ❖ HDFS is not being used
- ❖ Uses local file system for input and output

- ❖ No need to change any configuration files
- ❖ Default Hadoop Modes

Pseudo distributed mode:

- The **pseudo-distribute mode** is also known as a **single-node cluster** where both NameNode and DataNode will reside on the same machine.
- In pseudo-distributed mode, all the Hadoop daemons will be running on a single node. Such configuration is mainly used while testing when we don't need to think about the resources and other users sharing the resource.
- In this architecture, a separate JVM is spawned for every Hadoop components as they could communicate across network sockets, effectively producing a fully functioning and optimized mini-cluster on a single host.

Here is the summarized view of pseudo distributed Mode:

- ❖ Single Node Hadoop deployment running on Hadoop is considered as pseudo distributed mode
- ❖ All the master & slave daemons will be running on the same node
- ❖ Mainly used for testing purpose
- ❖ Replication Factor will be ONE for blocks
- ❖ Changes in configuration files will be required for all the three files- **mapred-site.xml**, **core-site.xml**, **hdfs-site.xml**

Fully distributed mode:

- This is the **production mode of Hadoop** where multiple nodes will be running. Here data will be distributed across several nodes and processing will be done on each node.
- Master and Slave services will be running on the separate nodes in fully-distributed Hadoop Mode.

Here is the summarized view of fully-distributed mode:

- ❖ Production phase of Hadoop
- ❖ Separate nodes for master and slave daemons

- ❖ Data are used and distributed across multiple nodes

HADOOP INSTALLATION -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.

To install hadoop first we need to install java.

JAVA INSTALLATION

Step1: Place software into system Downloads folder.

i.hadoop-2.7.2.tar.gz

ii.jdk-8u77-linux-i586.tar.gz

Step 2: Extract files in Downloads folder and rename hadoop2.7.2 as hadoop.

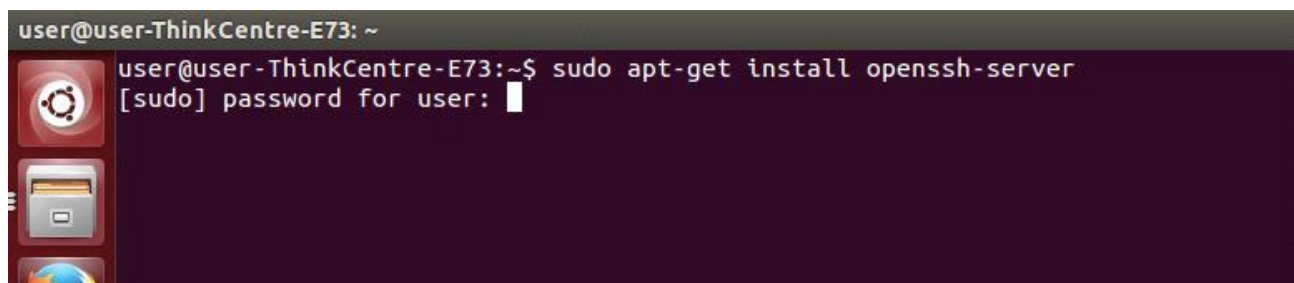
Step 3: Updating ubuntu

syntax : user@user-Thinkceter-E73:- \$ sudo apt update

Step 4: Installing Openssh server

syntax:user@user-Thinkceter-E73:- \$ sudo apt-get install openssh-server

Step5: Enter password in terminal.

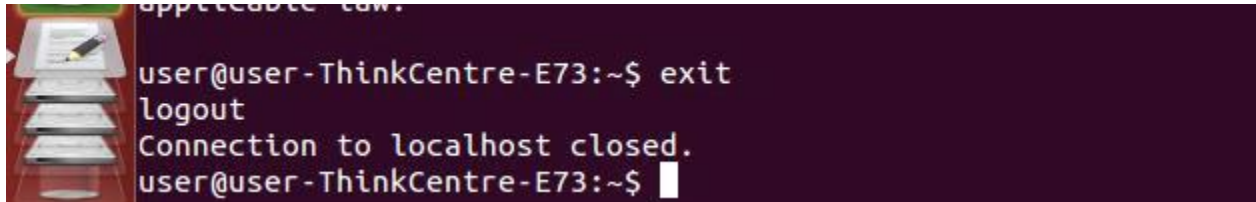


Step 6: Follow the instructions prompted.

Step 7: To check local host connection type ssh localhost in terminal.

syntax: user@user-Thinkceter-E73:- \$ssh localhost

Step 8: After connecting to local host type exit.



```
user@user-ThinkCentre-E73:~$ exit
logout
Connection to localhost closed.
user@user-ThinkCentre-E73:~$
```

Step 9: open bashrc file

syntax: user@user-Thinkceter-E73:- \$sudo gedit .bashrc

Step 10: Add these two lines to the end of the above file.

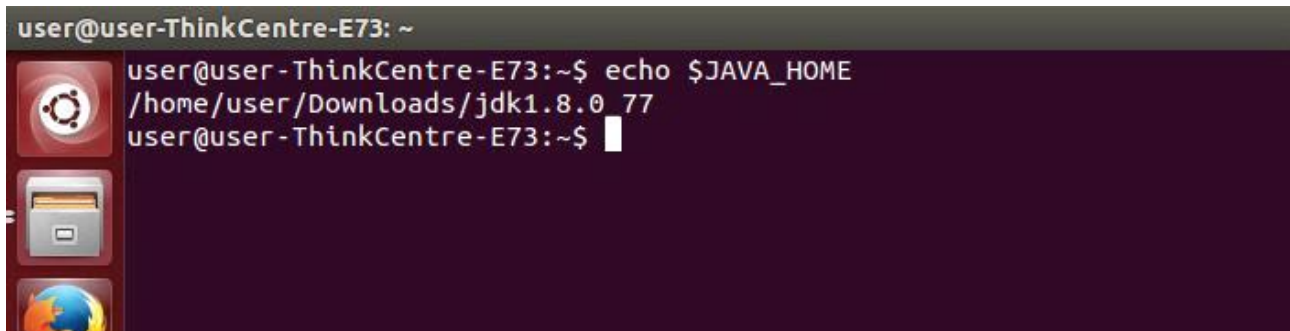
```
export JAVA_HOME=/home/user/Downloads/jdk1.8.0_77
export PATH=$PATH:$JAVA_HOME/bin
```

Step 11: Now apply all the changes into the current running system.

Close terminal. And open again or

user@user-Thinkceter-E73:- \$ source ~/.bashrc

Step 12 : For verification of java path type echo \$java_home in it.



```
user@user-ThinkCentre-E73: ~
user@user-ThinkCentre-E73:~$ echo $JAVA_HOME
/home/user/Downloads/jdk1.8.0_77
user@user-ThinkCentre-E73:~$
```

HADOOP INSTALLATION

Step 13: Open `hadoop/etc/hadoop/hadoop-env.sh` file and add this line at the end of the file.

```
export JAVA_HOME=/home/user/Downloads/jdk1.8.0_77
```

save and exit.

Step 14: open bash rc file using the following command.

```
user@user-Thinkceter-E73:- $ sudo gedit .bashrc
```

At the end of the file add these two lines

```
export PATH=$PATH:/home/user/Downloads/hadoop/bin  
export PATH=$PATH:/home/user/Downloads/hadoop/sbin
```

save and exit

step 15: Now close terminal and again open terminal.

for verification of hadoop installation

```
user@user-Thinkceter-E73:- $ hadoop
```

It will display with version.

HADOOP INSTALLATION -PSEUDO DISTRIBUTION MODE

- It is a distributed simulation on single machine.
- This mode is useful for development.
- Each Hadoop daemon such as hdfs, yarn, MapReduce etc., will run as a separate java process.

Steps for Hadoop installation in Pseudo Distributed Mode

1. Download JAVA Software
2. Download HADDOP Software
3. install java
4. Install ssh
5. set up ssh certificate
6. install hadoop
7. configure hadoop
 - a) bashrc
 - b) haddop-env.sh
 - c) core-site.xml
 - d) hdfs-site.xml
 - e) mapred-site.xml.template
8. Format hadoop Filesystem
9. Start hadoop
10. Testing / running
11. Stopping hadoop

step 1 Downloading java

Hyperlink:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

File name: jdk-8u77-linux-i586.tar.gz

Step 2: Downloading Hadoop software

Hyperlink:

<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html>

(or)

Hyperlink:

<http://www.apache.org/dyn/closer.cgi/hadoop/common/>

FILE NAME: hadoop-2.7.2.tar.gz

Step 3 : INSTALLING JAVA

- Extract jre file in to DOWNLOADS folder

- Add the below 2 lines to .bashrc file

```
export JAVA_HOME=/home/user/Downloads/jdk1.8.0_77
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

Step 4 :installing SSH

At Prompt type the below command to install ssh server

```
user@user-Thinkceter-E73:- $ sudo apt-get install openssh-server
```

Step 5 :set up ssh certificate

To generate key for secured data transmission at \$ prompt type

```
$ ssh-keygen -t rsa
```

copy the generated key into authorized keys.

```
$cat /home/user/.ssh/id_rsa.pub >>/home/user/.ssh/authorized_keys
```

Step 6 : Install hadoop

- Extract Hadoop zip file in to DOWNLOADS folder and rename it as hadoop
- Add the below 3 lines to .bashrc file

```
HADOOP_HOME=/home/user/Downloads/hadoop

export PATH=$PATH:/home/user/Downloads/hadoop/bin

export PATH=$PATH:/home/user/Downloads/hadoop/sbin
```

Step 7 :Configure hadoop

The 3 important configurations file are

- Core-site.xml
- Hdfs-site.xml
- Mapred-site.xml are compulsory for pseudo distributive mode of installation and

- Hadoop-env.sh is optional

NOTE: all these files resides in Hadoop/etc/hadoop folder.

Step 7a) configure hadoop-env.sh

Add these 2 lines at the end of the file

```
export HADOOP_HOME=/home/user/Downloads/Hadoop

export JAVA_HOME=/home/user/Downloads/jdk1.8.0_77
```

step 7 b) configure coresite.xml

Open coresite.xml file and add these lines

```
<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:9000</value>

</property>

<property><name>dfs.permissions</name>

<value>>false</value></property>
```

```
</configuration>
```

Step 7c) Configure hdfs-site.xml file

Open hdfs-site.xml and type the following at the end of the file .

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

Step 7d) Configure mapred-site.xml

- In Hadoop/etc/Hadoop folder mapred-site.xml.template exist.
- So rename the file as mapred-site.xml
- Now open the file and type the following at the end of the file .

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

Step 8 :Format hadoop Filesystem

```
syntax: $ Hadoop namenode -format
```

Step 9: Start hadoop

```
$ start-all.sh
```

Step 10 : Testing / running

Browse the web interface for the NameNode and the JobTracker.

By default they are available at:

NameNode - <http://localhost:50070/>

JobTracker - <http://localhost:50030/>

Resourcemanager- <http://localhost:8088>

Step 11: Stopping hadoop

```
$ stop-all.sh
```

Hadoop Installation - FULLY DISTRIBUTED MODE:

Hadoop cluster environment using three systems (one master and two slaves); given below are their IP addresses.

- Hadoop Master: 192.168.1.15 (hadoop-master)
- Hadoop Slave: 192.168.1.16 (hadoop-slave-1)
- Hadoop Slave: 192.168.1.17 (hadoop-slave-2)

Following are the steps to install hadoop in fully distributed mode.

1. Installing java
2. Creating user accounts:

Create a system user account on both master and slave systems to use the Hadoop installation.

```
$useradd hadoop
$passwd hadoop
```

3. Mapping the nodes:

You have to edit **hosts** file in **/etc/** folder on all nodes, specify the IP address of each system followed by their host names.

```
$vi /etc/hosts
enter the following lines in the /etc/hosts file.
192.168.1.109 hadoop-master
192.168.1.145 hadoop-slave-1
192.168.56.1 hadoop-slave-2
```

4. Configuring Key Based Login:

Setup ssh in every node such that they can communicate with one another without any prompt for password.

```
$ su hadoop
$ ssh-keygen -t rsa
$ ssh-copy-id -i ~/.ssh/id_rsa.pub tutorialspoint@hadoop-master
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp1@hadoop-slave-1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp2@hadoop-slave-2
$ chmod 0600 ~/.ssh/authorized_keys
$ exit
```


5. Installing hadoop:

In the Master server, download and install Hadoop using the following commands.

```
$mkdir /opt/hadoop
$ cd /opt/hadoop/
$ wget http://apache.mesi.com.ar/hadoop/common/hadoop-1.2.1/hadoop-1.2.0.tar.gz
$ tar -xzf hadoop-1.2.0.tar.gz
$ mv hadoop-1.2.0 hadoop
$ chown -R hadoop /opt/hadoop
$ cd /opt/hadoop/hadoop/
```

6. Configuring Hadoop:

You have to configure Hadoop server by making the following changes as given below.

core-site.xml

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://hadoop-master:9000</value>
  </property>
  <property>
    <name>dfs.permissions</name>
    <value>>false</value>
  </property>
</configuration>
```

hdfs-site.xml

Open the **hdfs-site.xml** file and edit it as shown below.

```
<configuration>
  <property>
    <name>dfs.data.dir</name>
    <value>/opt/hadoop/hadoop/dfs/name/data</value>
    <final>>true</final>
  </property>

  <property>
    <name>dfs.name.dir</name>
    <value>/opt/hadoop/hadoop/dfs/name</value>
    <final>>true</final>
  </property>

  <property>
```

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

mapred-site.xml

Open the **mapred-site.xml** file and edit it as shown below.

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>hadoop-master:9001</value>
  </property>
</configuration>
```

hadoop-env.sh

Open the **hadoop-env.sh** file and edit JAVA_HOME, HADOOP_CONF_DIR, and HADOOP_OPTS as shown below.

```
export JAVA_HOME=/opt/jdk1.7.0_17
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
export HADOOP_CONF_DIR=/opt/hadoop/hadoop/conf
```

Installing Hadoop on Slave Servers:

Install Hadoop on all the slave servers by following the given commands.

```
$ su hadoop
$ cd /opt/hadoop
$ scp -r hadoop hadoop-slave-1:/opt/hadoop
$ scp -r hadoop hadoop-slave-2:/opt/hadoop
```

Configuring Hadoop on Master Server

Open the master server and configure it by following the given commands.

```
$ su hadoop
$ cd /opt/hadoop/Hadoop
```

Configuring Master Node

```
$ vi etc/hadoop/masters  
hadoop-master
```

Configuring Slave Node

```
$ vi etc/hadoop/slaves  
hadoop-slave-1  
hadoop-slave-2
```

Format Name Node on Hadoop Master

```
$ su hadoop  
$ cd /opt/hadoop/hadoop  
$ bin/hadoop namenode -format
```

Starting Hadoop Services

The following command is to start all the Hadoop services on the Hadoop-Master.

```
$ cd $HADOOP_HOME/sbin  
$ start-all.sh
```

Adding a New DataNode in the Hadoop Cluster

Given below are the steps to be followed for adding new nodes to a Hadoop cluster.

Adding User and SSH Access

On a new node, add "hadoop" user and set password of Hadoop user to "hadoop123" or anything you want by using the following commands.

```
$useradd Hadoop  
$passwd Hadoop
```

Setup Password less connectivity from master to new slave.

Execute the following on the master

```
$ mkdir -p $HOME/.ssh
```

```
$ chmod 700 $HOME/.ssh
$ ssh-keygen -t rsa -P '' -f $HOME/.ssh/id_rsa
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
$ chmod 644 $HOME/.ssh/authorized_keys
Copy the public key to new slave node in hadoop user $HOME directory
$ cp $HOME/.ssh/id_rsa.pub hadoop@192.168.1.103:/home/hadoop/
```

Execute the following on the slaves

Login to hadoop. If not, login to hadoop user.

```
su hadoop ssh -X hadoop@192.168.1.103
```

Copy the content of public key into file "**\$HOME/.ssh/authorized_keys**" and then change the permission for the same by executing the following commands.

```
$cd $HOME
$mkdir -p $HOME/.ssh
$chmod 700 $HOME/.ssh
$cat id_rsa.pub >>$HOME/.ssh/authorized_keys
$chmod 644 $HOME/.ssh/authorized keys
```

Check ssh login from the master machine. Now check if you can ssh to the new node without a password from the master.

```
ssh hadoop@192.168.1.103 or hadoop@slave3
```

Configuring XML Files:

Configuration Filenames	Description Of Log Files
core-site.xml	Configuration settings for Hadoop Core such as I/O settings that are common to HDFS and Map Reduce.
hdfs-site.xml	Configuration setting for HDFS daemons, the name node, the secondary name node and the data nodes.
mapred-site.xml	Configuration settings for Map Reduce daemons : the job – tracker and the task-trackers.
yarn-site.xml	Configuration settings for Resource Manager and Node Manager.

core-site.xml

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://$namenode.full.hostname:8020</value>
  <description>Enter your NameNode hostname </description>
</property>
<property>
  <name>hdp.version</name>
  <value>${hdp.version}</value>
  <description>Replace with the actual HDP version </description>
</property>
```

hdfs-site.xml

- **Name Node:**

```
<property> <name>dfs.namenode.name.dir</name>  
<value> /grid/hadoop/hdfs/nn, /grid1/hadoop/hdfs/nn </value>  
<description>Comma-separated list of paths. Use the list of directories from  
$DFS_NAME_DIR. For example, /grid/hadoop/hdfs/nn, /grid1/hadoop/hdfs/nn.  
</description>  
</property>
```

- **Data Node:**

```
<property> <name>dfs.datanode.data.dir</name>  
<value> file:///grid/hadoop/hdfs/dn, file:///grid1/hadoop/hdfs/dn </value>  
<description>Comma-separated list of paths. Use the list of directories from  
$DFS_DATA_DIR. For example, file:///grid/hadoop/hdfs/dn,  
file:///grid1/hadoop/hdfs/dn.  
</description>  
</property>
```

mapred-site.xml

```
<property>  
<name>mapreduce.jobhistory.address</name>  
<value>$jobhistoryserver.full.hostname:10020</value>  
<description>Enter your JobHistoryServer hostname.  
</description>  
</property>
```

yarn-site.xml

```
<property><name>yarn.resourcemanager.scheduler.class</name>  
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler  
</value>  
</property>
```