3. Hadoop 1.x Installation

Hadoop has been built to natively work with Linux. If you have other than Linux, then use hypervisor like virtual box, kvm… to create Linux Virtual Machine (VM) and try Hadoop. However, Hadoop works with Windows, MAC, Solaris… Hadoop MR (JT, TT) and HDFS (NN, SNN, DN) components run as daemons in nodes. Daemon is a background process that keeps running. There are different modes of Hadoop deployment.

Single node implementation: setting up Hadoop in one machine

1. Standalone or Local mode: no Hadoop daemons run in this mode. All execution sequence is taken care by Hadoop framework itself. It is useful to test and debug MR jobs.
2. Pseudo-distributed mode: Hadoop daemons run on separate JVM in single machine by simulating cluster environment.

Multi-node implementation: setting up Hadoop on more than one machines

1. Fully distributed or cluster mode: Hadoop daemons run in different physical machines.
2. Virtual cluster mode: Hadoop daemons run in different VMs in virtual cluster.

3.1 System requirements

I am going to use a physical machine with Ubuntu 16. If you don't use Ubuntu, then install virtual box (any hypervisor) and create a VM with Ubuntu 16. Then, complete the following requirements in that node (physical/virtual machine).

```
$ lsb_release -a                    // to check Ubuntu version
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.3 LTS
Release:        16.04
Codename:       xenial
```

1. set static IP and verify the connection to internet. If you use VM, then verify the connection between VM to internet, VM to host machine. If you go for multi-node cluster, then each physical/virtual node must contain static IP. (multi-node is discussed later)

```
$ sudo vi /etc/network/interfaces        // include the following
    auto eth0
    iface eth0 inet static
    address 10.100.55.92                 // your LAN IP
    netmask 255.255.252.0                // appropriate netmask
    gateway 10.100.52.1                  // network gateway
    dns-nameservers 10.20.1.22           // DNS server IP
```

restart network services

```
$ sudo service networking restart  (or)
$ sudo /etc/init.d/networking restart (or)
$ sudo reboot                        // restart the node if nothing works
```

then check network connectivity

```
$ ping google.com
```

2. run update and upgrade

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

3. install latest java and setup path in .bashrc file. Install the same version of java in all the Hadoop nodes in case of cluster mode.

```
$ sudo apt-get install openjdk-8*      // installs latest opensource JDK
$ vi .bashrc                           // set path to java (verify the location of jdk in your node)
        export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
        export PATH=$PATH:$JAVA_HOME/bin
$ source .bashrc
$ java -version                        // verify java and javac have same version
        openjdk version "1.8.0_131"
        OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-
        2ubuntu1.16.04.3-b11)
        OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
$ javac -version
        javac 1.8.0_131
```

4. Install openssh server and client for master-slave communication among nodes

```
$ sudo apt-get install openssh-server
$ sudo apt-get install openssh-client
```

5. Install vim editor

```
$ sudo apt-get install vim
```

If you use Ubuntu VM on windows, then you can use (refer Linux chapter)

- WINSCP to transfer files from windows to Ubuntu VM. WINSCP is a file transfer protocol with nice GUI.
- PUTTY to remotely login from windows to Ubuntu VM to launch commands.

I am going to use opensource Hadoop software from Apache BigTop project for Hadoop installation. Therefore, download a stable release, which is packaged as a gzipped tar file from apache Hadoop release page.

If you want to install Hadoop on Windows machines, then install Cygwin and SSH server in each machine. It provides Unix-like environment and its command line interface in windows. The link http://opensourceforu.com/2015/03/getting-started-with-hadoop-on-windows/ provides step-by-step instructions.

3.2 Single node setup

3.2.1 Standalone or Local mode

By default, Hadoop has been configured to run in standalone mode as a single java process as shown in *Figure 3-1*. In local mode, no Hadoop daemons are running. Since, HDFS and JT are not installed (no concept of block, replication concept) in standalone implementation. So, it works on local file system with default configuration and we need not set any configuration files. All execution sequence is handled by Hadoop framework itself. It is not recommended for development environment. However, it is good enough to practice Hadoop commands and debug MR jobs.

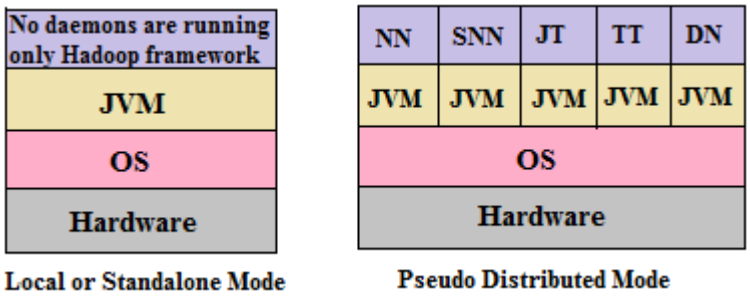


Figure 3-1: Single node setup

Step 1: download Hadoop 1.2.1 & unpack the tar file

- if you have Hadoop file in windows, use winscp to copy from windows to ubuntu or
- you can download Hadoop tarball release from apache software foundation links. https://archive.apache.org/dist/hadoop/core/
- else download using Linux command
  $ wget https://archive.apache.org/dist/hadoop/core/hadoop-1.2.1/hadoop-1.2.1.tar.gz
- untar Hadoop file
  $ tar -zxvf hadoop-1.2.1.tar.gz

Step 2: move to /usr/local/hadoop

```
$ sudo cp -r hadoop-1.2.1 /usr/local/hadoop
```

Step 3: setup environment variable for Hadoop

```
$ vi .bashrc
        export HADOOP_HOME=/usr/local/hadoop
        export PATH=$PATH:$HADOOP_HOME/bin
$ source .bashrc
```

Step 4: configure Hadoop environment variable

You don't have to edit any configuration files except hadoop-env.sh to set JAVA_HOME and enabling IPV4.

```
$ sudo vi /usr/local/hadoop/conf/hadoop-env.sh
        export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
        export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true

$ hadoop version                        // Hadoop version is displayed
```

Step 5: give privileges

Give all access rights to the Hadoop daemons to read and write in local file system location /usr/local/hadoop. Assume username is **itadmin.**

```
$ sudo chown -R itadmin /usr/local/hadoop
$ sudo chmod -R 777 /usr/local/hadoop
```

Step 6: running a sample job

Start launching Hadoop MR job on local files (no HDFS). Create a file input.txt in local file system and launch wordcount job with output directory called "result" (created on local file system, not on HDFS). Output directory should not already exit. Once job is submitted, LocalJobRunner will take care of the job execution as no MR daemons are running.

```
$ vi input.txt
        Hi how are you
$ hadoop jar /usr/local/hadoop/hadoop-examples-1.2.1.jar wordcount input.txt result
$ ls result                          // you will see two files
        part-r-00000                 // output file
        _SUCCESS                     // empty file showing MR execution success
$ cat result/part-r-00000            // to view the word count output
```

You don't have any WUI facility in this mode as no configurations have been set.

3.2.2 Pseudo distributed mode

Hadoop daemons run on different JVM in a node as shown in *Figure 3-1*. In this mode, we can use HDFS and MR. Complete system requirements given in section 3.1. Table 3-1 shows the configuration files we are going edit.

Table 3-1: HDFS and MR configuration files

hadoop-env.sh	specifies location for java and can set configurations for HDFS and MR daemons.
core-site.xml	to specify NN, location for storing FSImage, and location to store blocks in DNs.
mapred-site.xml	to set MR and its configuration.
hdfs-site.xml	to configure NN, SNN, and DN.
masters	to specify SNN for checkpointing.
slaves	to set list of slave machines (one per line) that runs DN and TT.

3.2.2.1 Installation in short

1. generate public-private key pair.
2. download Hadoop and untar the file.

3. move to /usr/local/hadoop location.
4. set environment variable for Hadoop in .bashrc file.
5. configure Java_Home, disable IPv6… in hadoop-env.sh
6. edit configuration files as given in *Table 3-1*.
7. change permission and ownership for the Hadoop location in local file system.
8. create HDFS using format command (creates namespace for NN).
9. launch Hadoop daemons.

3.2.2.2 Installation in detail

Step 1: generate ssh key for passwordless communication

Hadoop daemons rely on SSH to perform cluster-wide operations. So, generate public-private key pair and add key to authorized_keys file for passwordless communication to start/stop services. Otherwise every command to start/stop services will prompt username and password.

```
$ ssh localhost          // asks username and password and loops into same host
$ exit                   // to exit from looping
```

Generate key and add public key to ~/.ssh/authorized_keys file. Once added, you can log into your system itself without password. In production environment, passphrase is important to be more secure.

```
$ ssh-keygen  or ssh-keygen -t rsa          // leave empty for name, pwd…
$ ls ~/.ssh                                 // you can see the public private keys
$ vi ~/.ssh/id_rsa.pub                      // to view generated public key
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
$ chmod 700 ~/.ssh/
$ ssh localhost
```

It loops into same machine itself. First attempt prompts password. From the next try, you can login without entering username and password.

```
$ exit                                      // to log off from loop
```

To configure SSH, define the HADOOP_SSH_OPTS environment variable in hadoop-env.sh.

Step 2: download Hadoop 1.2.1 & unpack the tar file

- if you have Hadoop file in windows, use winscp to copy from windows to ubuntu
- or you can download Hadoop tar from following links.
  https://archive.apache.org/dist/hadoop/core/
- else download using Linux command
  $ wget https://archive.apache.org/dist/hadoop/core/hadoop-1.2.1/hadoop-1.2.1.tar.gz
- untar the Hadoop file
  $ tar -zxvf hadoop-1.2.1.tar.gz

Step 3: move to /usr/local/hadoop

```
$ sudo cp -r hadoop-1.2.1 /usr/local/hadoop
```

$ ls /usr/local/hadoop

```
bin     hadoop-ant-1.2.1.jar          ivy                sbin
build.xml hadoop-client-1.2.1.jar     ivy.xml            share
c++      hadoop-core-1.2.1.jar        lib                src
CHANGES.txt hadoop-examples-1.2.1.jar  libexec          webapps
conf    hadoop-minicluster-1.2.1.jar LICENSE.txt
contrib hadoop-test-1.2.1.jar        NOTICE.txt
docs     hadoop-tools-1.2.1.jar       README.txt
```

/usr/local/hadoop/bin - contains script to start/stop… Hadoop daemons.
/usr/local/hadoop/conf - contains configuration files for Hadoop daemons.
/usr/local/hadoop/logs - state of Hadoop activities is recorded. You can also see system log file /var/log if any error.
/usr/local/hadoop/sbin - supporting scripts for Hadoop services.

Step 4: setup environment variable for Hadoop

$ vi .bashrc

```
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

$ source .bashrc

$ $PATH // to verify hadoop path set up

Step 5: configure Hadoop environment variable

$ sudo vi /usr/local/hadoop/conf/hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

$ hadoop version // to verify Hadoop version

Step 6: edit configuration files

IP of node is 10.100.55.92 and username is itadmin. Configurations are set as properties which contain name-value pairs.

core-site.xml

fs.default.name - to specify IP address of NN
hadoop.tmp.dir - specifies local file system location for storing FSImage, edit logs in NN, SNN, meta-data of JT, to store blocks in DNs…

$ sudo vi /usr/local/hadoop/conf/core-site.xml

```
<configuration>
        <property>
              <name>fs.default.name</name>
              <value>hdfs://10.100.55.92:10001</value>
        </property>
        <property>
              <name>hadoop.tmp.dir</name>
              <value>/usr/local/hadoop/tmp</value>
        </property>
  </configuration>
```

mapred-site.xml

mapred.job.tracker - to specify IP address of node that is going to run JT

$ sudo vi /usr/local/hadoop/conf/mapred-site.xml

```
<configuration>
      <property>
            <name>mapred.job.tracker</name>
            <value>10.100.55.92:10002</value>
      </property>
</configuration>
```

masters - to specify IP of SNN for checkpointing

$ sudo vi /usr/local/hadoop/conf/masters

```
10.100.55.92
```

slaves - to specify list of slaves to run TT and DN

$ sudo vi /usr/local/hadoop/conf/slaves

```
10.100.55.92
```

HDFS configuration – to configure NN, SNN, DN. It is optional.

$ sudo vi /usr/local/hadoop/conf/hdfs-site.xml

```
<configuration>
      <property>
            <name>dfs.data.dir</name>
            <value>/usr/local/hadoop/tmp/dfs/data</value>
      </property>
      <property>
            <name>dfs.name.dir</name>
            <value>/usr/local/hadoop/tmp/dfs/name</value>
      </property>
      <property>
            <name>dfs.replication</name>
            <value>1</value>
      </property>
      <property>
          <name>dfs.block.size</name>
          <value>62000000</value>
      <property>
</configuration>
```

Step 7: grant access to Hadoop daemons

Give all access rights to Hadoop daemons to read and write in local file system location /usr/local/hadoop. **itadmin** is username

$ sudo chown -R itadmin /usr/local/hadoop/

$ sudo chmod -R 777 /usr/local/hadoop/

Step 8: create HDFS namespace

Execute format command to create new HDFS namespace with new namespaceID.

```
$ hadoop namenode -format
……
Storage directory /usr/local/hadoop/tmp/dfs/name has been successfully
formatted.
……
```

You will see various metrics and a message like above in the end. The formatting process creates an empty HDFS by creating the storage directories and the initial versions of the NNs persistent data structures in NN. DNs are not involved in the initial formatting process.

```
$ ls /usr/local/hadoop/tmp
```

Step 9: launch Hadoop daemons: there are different ways to start/stop HDFS and MR

```
$ start-dfs.sh                          // NN, SNN, DN are started
$ ls /usr/local/hadoop/tmp/dfs          // directories for SNN, DN are created
$ jps                                   // JVM process status: to see running services
    DataNode
    NameNode
    SecondaryNameNode

$ start-mapred.sh                       // JT and TT are started
$ ls /usr/local/hadoop/tmp              // mapred directory is created for JT and TT
$ ls /usr/local/hadoop/tmp/mapred

$ jps                                   // you will see JT and TT services running
    TaskTracker
    DataNode
    NameNode
    JobTracker
    SecondaryNameNode
```

All services are running in same node in single node implementation

3.2.2.3 Simple wordcount job

```
$ vi input.txt                                  // enter some text
$ hadoop fs -ls  /                              // lists meta-data of files from HDFS root
$ hadoop fs -mkdir /data                        // create a directory in HDFS
$ hadoop fs -copyFromLocal input.txt /data      //load input.txt onto HDFS /data directory
$ hadoop fs -ls  /data
$ hadoop jar /usr/local/hadoop/hadoop-examples-1.2.1.jar wordcount
  /data/input.txt   /result
```

result will be a directory and should not already exist in HDFS. wordcount is a job (class name) that contains main method to start job in hadoop-examples-1.2.1.jar.

```
$ hadoop fs -ls /result

Found 3 items

-rw-r--r-- 3 rathinaraja supergroup 0 2017-02-15 20:54 /result/_SUCCESS
drwxr-xr-x - rathinaraja supergroup0 2017-02-15 20:54 /result/_logs
-rw-r--r-- 3 rathinaraja supergroup 13 2017-02-15 20:54 /result/part-
r-00000
```

3.2.2.4 Validating output

- HDFS commands
- WebUI
- MR and HDFS API in java programs
- logs

HDFS commands

```
$ hadoop fs -mkdir new
$ hadoop fs -ls                                  // lists HDFS /user/itadmin location
  Found 1 items
  drwxr-xr-x  - itadmin supergroup 0 2018-01-03 14:53 /user/itadmin/new
$ hadoop fs -ls /                                // lists HDFS root location
  Found 1 items
  drwxr-xr-x   - itadmin supergroup     0 2018-01-03 14:47 /data
  drwxr-xr-x   - itadmin supergroup     0 2018-01-03 14:48 /result
  drwxr-xr-x   - itadmin supergroup     0 2018-01-03 14:53 /user
  drwxr-xr-x   - itadmin supergroup     0 2018-01-03 14:46 /usr
$ hadoop fs -cat /result/part-r-00000            // to view result file in HDFS
$ hdoop fs -ls file:///                          // lists out local file system root
```

copy output from HDFS into local file system and then view using local file system commands

```
$ hadoop fs -copyToLocal  /result  ~/dir_name
$ cat /dir_name/result/part-r-00000
```

Web User Interface

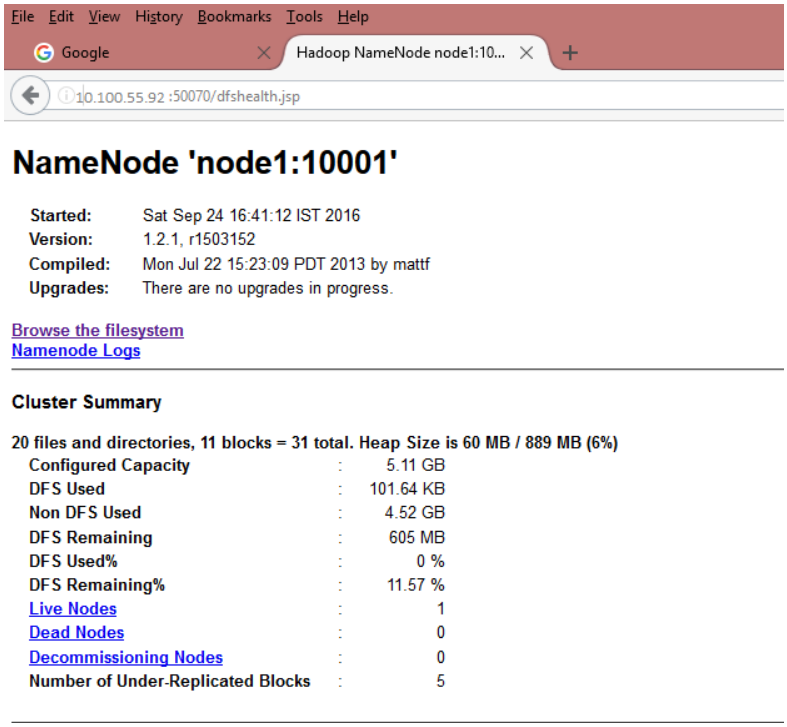
To see result via web interface, open browser and enter IP address + port number of the node where Hadoop services are running. In single node implementation, HDFS and MR services are running in same node.

```
IP of NN:50070          // after start-dfs.sh you can see data.
IP of DN:50075          // to see DN
IP of SNN:50090         // to see SNN
IP of JT:50030          // after start-mapred.sh you can get jobs execution details
IP of TT:50060          // to see what is going on in TT
```

To check result, go to NN_IP:50070 in browser as shown in *Figure 3-2,* browse files and check the result folder. There you see "part-r-00000".

To see job status, go to JT IP:50030 as shown in *Figure 3-3* and browse completed jobs and see how many mappers, reducers launched and their attempts, how many bytes read, written… all these parameters are counters.

10.100.55.92:50070/



10.100.55.92:50030/

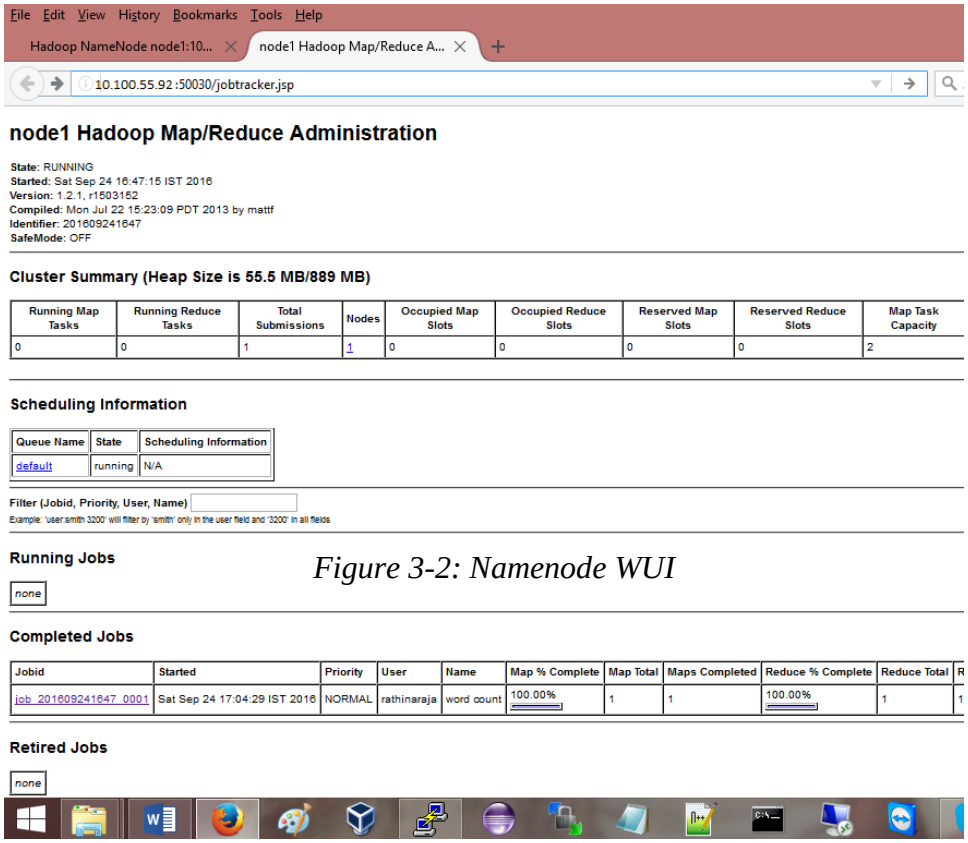


Figure 3-2: Namenode WUI

Figure 3-3: MapReduce WUI

We can use HDFS and JT API's in java program to interact with running Hadoop daemons and display information.

Once HDFS and MR services stopped, it can be restarted again. If you format NN again, then new namespaceID is created. So, DNs having old namespaceID will not match with new namespace version leading to cluster down. So, if there is any problem in starting NN, then delete /usr/local/hadoop/tmp directory and re-format NN. As a consequence, you will lose old data blocks and meta-data form HDFS.

```
$ sudo rm -r /usr/local/hadoop/tmp
$ hadoop namenode -format
```

To practice from the beginning, remove Hadoop file and freshly follow from the first step

```
$ sudo rm -r  /usr/local/Hadoop
```

3.3 Multi-node setup: fully distributed mode and virtual cluster mode deployment

All master processes like NN, JT, SNN can be deployed in same machine only for testing and training purposes. For a small production deployment, at least SNN should be in a different machine from NN.

Fully distributed mode can be done on either cluster of physical machines as shown in *Figure 3-4.* Virtual cluster contains set of VMs running Hadoop services. For large production deployments, NN should be on dedicated machine so should JT and SNN.

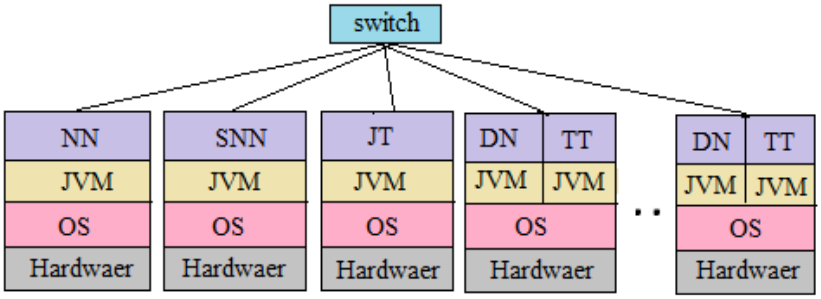


Figure 3-4: fully distributed or cluster mode on physical machines

3.3.1 Installation process

Requirements discussed in *Section 3-1* should be satisfied in both of the multi-node implementation flavours. Most of us would not be having multiple machines to experiment fully distributed mode. So, we can create 5 VMs and assign IP as given in *Table 3-2* for our experiment. You must have at least quad core processor, 32 GB RAM, and over 500 GB HDD to launch 5 VM instances in your physical machine. Values such as IP, hostname, and domain name in this table are based on my cluster. You have to fill this table according to your cluster configurations.

Table 3-2: Multi-node details

VM/physical machine	Daemon	IP	hostname	domain name
VM/server 1	NN	10.100.55.92	ubuntu1	node1
VM/server 2	JT	10.100.55.93	ubuntu2	node2
VM/server 3	SNN	10.100.55.94	ubuntu3	node3

VM/server 4	Slave1 (DN+TT)	10.100.55.95	ubuntu4	node4
VM/server 5	Slave2 (DN+TT)	10.100.55.96	ubuntu5	node5

It is also possible if you have two machines and want to run few VMs in both machines. We assign one VM each to NN, JT, SNN and two VMs for slaves.

You can experiment with two VM's for the system with low configuration. If that be the case, then assign NN, JT, SNN in VM1 and slave to VM2.

Make sure all the nodes to meet the system requirements given in section 3.1 and have same username**.** I assume user name "**itadmin**" for all the nodes. The term "node" means here either physical machine or VM.

Step 1: set domain name in all the nodes in the cluster

Go to /etc/hosts to set DNS. Therefore, you can use domain name instead of typing IP address every time. Comment 127.0.*.1 with # and enter IP, domain name, and hostname of a node with tab space one in a line (refer *Table 3-2*).

```
$ sudo vi /etc/hosts
        #127.0.0.1
        #127.0.1.1
        10.100.55.92    node1  ubuntu1
        10.100.55.93    node2  ubuntu2
        10.100.55.94    node3  ubuntu3
        10.100.55.95    node4  ubuntu4
        10.100.55.96    node5  ubuntu5
```

disable firewall in all machines in the Hadoop cluster

```
$ sudo ufw disable
$ sudo ufw status
```

Step 2: set passwordless communication

Create SSH key in NN and send to all other nodes. Similarly, create SSH key in JT and send to all other nodes. You need not create any key in slaves and SNN. In case, if you host NN and JT in same node, enough doing once. Ex: create key in node1 (NN) as follows

```
$ ssh-keygen  or ssh-keygen -t rsa                    // leave empty for name, pwd…
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

add key to ~/.ssh/authorized_keys file in other nodes

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub username@IP

$ ssh-copy-id -i ~/.ssh/id_rsa.pub itadmin@node2
$ ssh-copy-id -i ~/.ssh/id_rsa.pub itadmin@node3
$ ssh-copy-id -i ~/.ssh/id_rsa.pub itadmin@node4
$ ssh-copy-id -i ~/.ssh/id_rsa.pub itadmin@node5
$ ssh node1
```

first attempt might ask username and pwd. From very next try, it connects without username and password. Use exit command to come out of remote connection.

```
$ exit
$ ssh node2          //try ssh for other nodes to make sure passwordless connection
$ exit
```

Similarly, create key in node2 (JT), send to all other nodes (1,3,4,5), and verify the passwordless connection.

Step 3 to Step 8 must be done in all nodes in the cluster

Hadoop does not have a single, global location for configuration information. Instead, each Hadoop node in the cluster has its own set of configuration files, and it is up to administrators to ensure that they are kept in sync across the system.

Step 3: download Hadoop & unpack the tar file

- if you have Hadoop file in windows, use winscp to copy from windows to ubuntu or
- you can download Hadoop tarball release from apache software foundation links.

  https://archive.apache.org/dist/hadoop/core/

- else download using Linux command

  ```
  $ wget https://archive.apache.org/dist/hadoop/core/hadoop-1.2.1/hadoop-1.2.1.tar.gz
  $ tar -zxvf hadoop-1.2.1.tar.gz
  ```

Step 4: move hadoop file to /usr/local/hadoop

```
$ sudo cp -r hadoop-1.2.1 /usr/local/hadoop
$ ls /usr/local/hadoop
```

Step 5: setup environment variable to Hadoop

```
$ vi ~/.bashrc
        export HADOOP_HOME=/usr/local/hadoop
        export PATH=$PATH:$HADOOP_HOME/bin
$ source .bashrc
$ $PATH
```

Step 6: configure Hadoop environment variable (verify java bin path in your machine)

```
$ sudo vi /usr/local/hadoop/conf/hadoop-env.sh
        export JAVA_HOME= /usr/lib/jvm/java-8-openjdk-amd64
        export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
$ hadoop version
```

Step 7: edit configuration files

core-site.xml - specify IP of a node that is going to run NN

$ sudo vi /usr/local/hadoop/conf/core-site.xml

```
<configuration>
   <property>
         <name>fs.default.name</name>
         <value>hdfs://node1:10001</value>
   </property>
   <property>
         <name>hadoop.tmp.dir</name>
         <value>/usr/local/hadoop/tmp</value>
   </property>
</configuration>
```

mapped-site.xml - to specify IP address of node that is going to run JT

$ sudo vi /usr/local/hadoop/conf/mapred-site.xml

```
<configuration>
       <property>
             <name>mapred.job.tracker</name>
             <value>node2:10002</value>
       </property>
</configuration>
```

hdfs-site.xml - to configure HDFS components

$ sudo vi /usr/local/hadoop/conf/hdfs-site.xml

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

masters - to specify SNN IP

$ sudo vi /usr/local/hadoop/conf/masters

```
node3
```

slaves - to specify node that runs TT and DN

$ sudo vi /usr/local/hadoop/conf/slaves

```
node4
node5
```

Step 8: change ownership to username and grant access to Hadoop services

$ sudo chown -R itadmin /usr/local/hadoop
$ sudo chmod -R 777 /usr/local/hadoop

give permission to read/write .ssh

$ chmod 600 ~/.ssh/authorized_keys
$ chmod 700 ~/.ssh

Step 9: create HDFS namespace (in node1)

Once all nodes configured successfully then format HDFS to create namespace.

$ hadoop namenode -format

/usr/local/hadoop/tmp directory is created after this command in node1 (NN)

Step 10: launch Hadoop daemons: start NN first and then JT.

Start HDFS using the following command in node1 (NN). It starts the NN in node1, SNN in node3 and DN in node4, node5.

```
$ start-dfs.sh
$ jps        // run this in NN, SNN, DN to ensure appropriate Hadoop services running.
```

Start MR in node2 (JT). It starts JT in node2 and TT in node4, node5.

```
$ start-mapred.sh
$ jps                                    // in JT and TT to ensure services running.
```

To stop services, follow the order

```
$ stop-mapred.sh                         // MR should be stopped first in node2
$ stop-dfs.sh                            // then HDFS is stopped next in node1
```

Now, follow the section **3.2.2.3, 3.2.2.4** to run sample wordcount job and validate the output via WUI. Launching commands for HDFS and MR can be done from any node in the cluster as all node has the configuration of master and slaves. Ultimately, you will get the same result.

However, test with small and large dataset to observe the running time that Hadoop takes. Hadoop takes more time for small files than normal local data processing. Because, there are lot of intermediate steps (magical phase to happen) that take considerable amount of time. However, with large dataset you can observe Hadoop out performing local file system processing.

3.3.2 Frequently used Hadoop commands

Hadoop commands are very similar to Linux commands. You need to remotely login to any one of the nodes in the Hadoop cluster to issue commands.

$ hadoop - lists all possible subcommands such as namenode, secondarynamenode, datanode, namenode -format, dfsadmin, mradmin, fsck, fs, jobtracker, tasktracker, historyserver, balancer, job, queue, version, jar…

$ hadoop fs - command to interact with HDFS. sub commands are: put, remove, cat, ls, copy to local, move to local, mkdir, chown…

$ hadoop fsck - command to query about file system related information like block reports.

$ hadoop dfsadmin - meant for HDFS cluster administrative commands and used to get statistics, refresh nodes, set balancer bandwidth.

$ hadoop job - helps to get job related information like counters and see/kill running jobs

$ hadoop jar - runs job

$ hadoop jobtracker - runs commands to interact with JT node

$ hadoop namenode - runs commands to interact with NN

$ hadoop datanode - runs commands on DN

HDFS and MR commands

FileSystem shell: it is an interface between user and HDFS.

Uniform Resource Identifiers are used to locate resources like files… All the file system shell commands take URIs as arguments. Ex:

- HDFS: hdfs://NN_IP:port#/file/location
- HAR: Hadoop Archive har:///location/file.har
- Local file system: file:///location/file/name

Every command returns exit code either 0 for success or -1 for error.

```
$ hadoop fs                          // fs indicates we are working with HDFS
                                     and lists all possible commands in fs
$ hadoop fs -help                    // lists short summary of all commands
$ hadoop fs -help commandName        // displays short summary of commands
$ hadoop fs -help put
```

1. to list meta-data from root (/) and create directory in HDFS

```
$ hadoop fs -mkdir /dir1             // to create directory in HDFS root
$ hadoop fs -ls /                    // to list meta-data
      /usr/local/hadoop/tmp/
      /dir1

$ hadoop fs -mkdir dir2              //directory is created in HDFS username
$ hadoop fs -ls /                    //displays meta-data of hadoop root dir
      /dir1
      /user/itadmin/dir2
      /usr/local/hadoop/tmp/
$ hadoop fs -ls                      //by default HDFS stores in /user/username
      /user/itadmin/dir2               in single node installation.

$ hadoop fs -ls hdfs://node1:10001/  // in case of multi node to see files in root

$ hadoop fs -ls -R /                 // to show all hidden files and sub
                                        directories recursively.

$ hdoop fs -ls file:///              // lists out local root file system
```

2. to copy/move files from local file system to HDFS

```
$ hadoop fs -put  local_source  /hdfs_destination
$ hadoop fs -copyFromLocal  local_source  /hdfs_destination
$ hadoop fs -moveFromLocal  local_source  /hdfs_destination
```

If source is file, then destination can be a file or directory
If source is directory, then destination must be a directory

put vs copyFromLocal: put can copy more than one input file at a time, and read directly from stdin.

3. to copy/move files from HDFS to local file system

```
$ hadoop fs -get  /hdfs_source  local_destination
$ hadoop fs -copyToLocal  /hdfs_source  local_destination
$ hadoop fs -moveToLocal  /hdfs_source  local_destination
```

4. to display a file from HDFS

```
$ hadoop fs -cat /filename
```

5. to delete file, directory from HDFS

```
$ hadoop fs -rm  /filename
$ hadoop fs -rmr /directory
```

6. copy and move a file from one location to another location in HDFS itself

```
$ hadoop fs -cp /hdfs_source /hdfs_destination
$ hadoop fs -mv  /hdfs_source /hdfs_destination
```

7. other file system operations

```
$ hadoop dfsadmin -report                          // displays block report
$ hadoop fsck /hdfs_directory -files -blocks -locations
$ hadoop fs -du /<file/directory name>             // displays the size of files in bytes
$ hadoop fs -cat /hdfs_file | head -n 5            // displays top five lines
$ hadoop fs -setrep number /filename               // replication by default 3
```

8. mapreduce commands

```
$ hadoop job -list                                 // lists currently running applications
$ hadoop job -kill <jobid>
$ hadoop job -list all                             // returns history of jobs
```

9. to transfer file from one node to another in Linux, make sure target machine location has write permission of that current user.

```
$ scp -r directory username@IP:~/
```

10. working with jars

```
$ hadoop jar /usr/local/hadoop/hadoop-examples-1.2.1.jar  //lists available jobs
$ jar tvf name.jar                                        // lists the contents of jar
```

You can create empty file on HDFS, but can't edit any files in HDFS. You must create in local file system and move onto HDFS. Therefore, vi, gedit… don't work with HDFS.

```
$ hadoop fs -touchz /file.txt
$ hadoop fs -ls /
```

Check the following link for more commands
https://hadoop.apache.org/docs/r1.2.1/commands_manual.html

3.4 Writing user defined MR application using Eclipse

So far, we have been executing predefined jobs that are coming with Hadoop distribution. Now, let's create a job for wordcount using Eclipse and run in Hadoop environment. Good understanding in java utilities, collections is required for writing good MR programs. Steps are

1. create input.txt file.
2. load input.txt onto HDFS.
3. make sure JDK on cluster and Eclipse matches.
4. write MR job.
5. add Hadoop dependencies with your project (we usually define in pom.xml) in Eclipse.
6. create jar file.
7. copy jar file to gateway node.
8. run MR job/application.
9. review output.

3.4.1 Move big data onto HDFS

Create a text file input.txt.

```
$ vi input.txt
    hi how are you?
```

Upload input.txt onto HDFS.

```
$ hadoop fs -mkdir /data
$ hadoop fs -copyFromLocal input.txt /data
$ hadoop fs -ls /data
```

3.4.2 Write MR job: I used Eclipse in windows and moved jar to Hadoop cluster.

Install latest Java, download latest Eclipse on Windows. We are going to create a MR job in Eclipse and move to Hadoop cluster. I assume that Hadoop-1.2.1 single node or multi-node setup is up and running. Make sure JDK of Hadoop and JDK in Eclipse matches.

1. File → new → java project → project name: MapReduce → Finish

2. Right click on project → new → class → name: WC → write the following code

```
import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
```

```
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WC extends Configured implements Tool{
	public static class UDFmapper extends MapReduceBase implements
	Mapper<LongWritable, Text, Text, IntWritable> {

		private final static IntWritable one = new IntWritable(1);
		private Text word = new Text();

		public void map(LongWritable key, Text value,
		OutputCollector<Text, IntWritable> output, Reporter reporter)
		throws IOException {

			String line = value.toString();
			StringTokenizer tokenizer = new StringTokenizer(line);
			while (tokenizer.hasMoreTokens()) {
				word.set(tokenizer.nextToken());
				output.collect(word, one);
			}
		}
	}

	public static class UDFreducer extends MapReduceBase implements
	Reducer<Text, IntWritable, Text, IntWritable> {

		public void reduce(Text key, Iterator<IntWritable> values,
		OutputCollector<Text, IntWritable> output, Reporter reporter)
		throws IOException {
			int sum = 0;
			while (values.hasNext()) {
				sum += values.next().get();
			}
			output.collect(key, new IntWritable(sum));
		}
	}

	public int run(String[] args) throws Exception {

		if(args.length <2) {
			System.err.println("Usage:<inputpath> <outputpath>");
			System.exit(-1);
		}

		JobConf job = new JobConf(WC.class);
		job.setJarByClass(WC.class);

		job.setMapperClass(UDFmapper.class);
		job.setReducerClass(UDFreducer.class);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(IntWritable.class);

		FileInputFormat.addInputPath(job, new Path(args[0]));
		FileOutputFormat.setOutputPath(job, new Path(args[1]));

		JobClient.runJob(job);
		return 0;
	}
	public static void main(String[] args) throws Exception {

		int exitCode = ToolRunner.run(new WC(), args);
		System.exit(exitCode);
```

```
        }
}
```

3. Add supporting jars to MR program from the following locations:

Right click on project → select properties → java build path → libraries → add external jar →

hadoop-1.2.1\lib → select all jars.
hadoop-1.2.1\ → select hadoop-client-1.2.1.jar and hadoop-core-1.2.1.jar

4. create JAR

Right click on your project → export → java → JAR → select project → specify location → finish. Give "Job" as JAR name.

5. move Job.jar into any one of the nodes in Hadoop cluster using Winscp software.

6. Run the job: syntax is

```
$ hadoop jar jarname.jar  driver_class_name  /input_file_location  /output_directory
$ hadoop jar Job.jar WC  /data/input.txt   /output
```

Note: every time you run, you have to give different output directory in HDFS. Head to WUI to see job progress, statistics and other HDFS information.

NN_IP:50070 to see HDFS details
JT_IP:50030 to see JT details

The **output** is displayed in HDFS along with two other files in HDFS output folder

```
_SUCCESS                    // an empty file, just says job is done
_logs                       // log about the job
part-r-00000                // which contains our output

$ hadoop fs -cat /output/part-00000          // to display output
```

If there are many parts (files) in output directory, then we need to view one by one. However, to view entire output parts as single file, use the following command.

```
$ hadoop fs -cat /output/part-*
$ hadoop fs -getmerge /output_dir_in_hdfs   file_name_local
$ hadoop fs -getmerge /output  result.txt
$ cat result.txt
```

Note: You can load data onto HDFS and launch job from any node in the cluster as a job client. Because, we have set mapred.xml and core-site.xml in all nodes. Therefore, job request is handed over to JT.

Try running your MR program with small amount of data first in local mode set up, so that, it will be easy for you to detect any errors and verify the logic of your program. Then, go for launching in multi-node cluster with huge data.