

Department of Computer Engineering
Academic Term: July-November 2023

Rubrics for Lab Experiments

Class : B.E. Computer
Semester : VIII

Subject Name: *Distributed Computing*
Subject Code : *CSC801*

Practical No:	
Title:	
Date of Performance:	
Roll No:	
Name of the Student:	

Evaluation:

Performance Indicator	Below average	Average	Good	Excellent	Marks
On time Submission (2)	Not submitted(0)	Submitted after deadline (1)	Early or on time submission(2)	---	
Test cases and output (4)	Incorrect output (1)	The expected output is verified only a for few test cases (2)	The expected output is Verified for all test cases but is not presentable (3)	Expected output is obtained for all test cases. Presentable and easy to follow (4)	
Coding efficiency (2)	The code is not structured at all (0)	The code is structured but not efficient (1)	The code is structured and efficient. (2)	-	
Knowledge(2)	Basic concepts not clear (0)	Understood the basic concepts (1)	Could explain the concept with suitable example (1.5)	Could relate the theory with real world application(2)	
Total					

Signature of the Teacher :

Lab 10

Aim: To Study HDFS and MapReduce

Lab Outcome:

Describe the concepts of distributed File Systems with some case studies **Theory:**

Hadoop:

With growing data velocity, the data size easily outgrows the storage limit of a machine. A solution would be to store the data across a network of machines. Such filesystems are called *distributed filesystems*. Since data is stored across a network all the complications of a network come in.

This is where Hadoop comes in. It provides one of the most reliable filesystems. HDFS (Hadoop Distributed File System) is a unique design that provides storage for *extremely large files* with streaming data access pattern and it runs on *commodity hardware*.

Let's elaborate the terms:

- **Extremely large files:** Here we are talking about the data in range of petabytes (1000 TB).
- **Streaming Data Access Pattern:** HDFS is designed on principle of *write-once and read-many-times*. Once data is written large portions of dataset can be processed any number times.
- **Commodity hardware:** Hardware that is inexpensive and easily available in the market. This is one of feature which specially distinguishes HDFS from other file system.

Nodes: Master-slave nodes typically forms the HDFS cluster.

1. NameNode(MasterNode):

- Manages all the slave nodes and assign work to them.
- It executes filesystem namespace operations like opening, closing, renaming files and directories.
- It should be deployed on reliable hardware which has the high config. not on commodity hardware.

2. DataNode(SlaveNode):

- Actual worker nodes, who do the actual work like reading, writing, processing etc.
- They also perform creation, deletion, and replication upon instruction from the master.
- They can be deployed on commodity hardware.

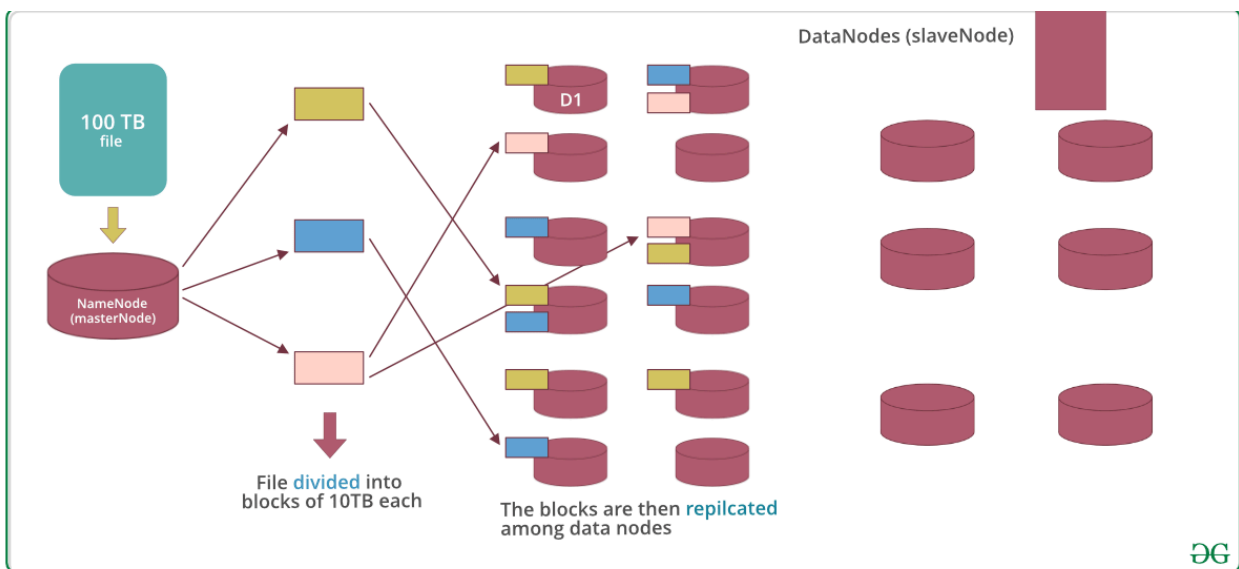
HDFS daemons: Daemons are the processes running in background. • **Namenodes:**

- Run on the master node.
- Store metadata (data about data) like file path, the number of blocks, block Ids. etc.
- Require high amount of RAM.
- Store meta-data in RAM for fast retrieval i.e to reduce seek time. Though a persistent copy of it is kept on disk.

• **DataNodes:**

- Run on slave nodes.
- Require high memory as data is actually stored here.

Data storage in HDFS: Now let us see how the data is stored in a distributed manner.



Assuming that 100TB file is inserted, then master node(namenode) will first divide the file into blocks of 10TB (default size is 128 MB in Hadoop 2.x and above). Then these blocks are stored across different datanodes(slavenode). Datanodes(slavenode) replicate the blocks among themselves and the information of what blocks they contain is sent to the master. Default replication factor is 3 means for each block 3 replicas are created (including itself). In `hdfs.site.xml` we can increase or decrease the replication factor i.e we can edit its configuration here.

Terms related to HDFS: • HeartBeat: It is the signal that datanode continuously sends to namenode. If namenode doesn't receive heartbeat from a datanode then it will consider it dead. • Balancing: If a datanode is crashed the blocks present on it will be gone too and the blocks will be under-replicated compared to the remaining blocks. Here master node(namenode) will give a signal to datanodes containing replicas of those lost blocks to replicate so that overall distribution of blocks is balanced. • Replication: It is done by datanode.

Features: • Distributed data storage. • Blocks reduce seek time. • The data is highly available as the same block is present at multiple datanodes. • Even if multiple datanodes are down we can still do our work, thus making it highly reliable. • High fault tolerance. MapReduce: MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner. MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job. MapReduce Architecture explained in detail: • One map task is created for each split which then executes map function for each record in the split.

- It is always beneficial to have multiple splits because the time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better to load balanced since we are processing the splits in parallel. • However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time. • For most jobs, it is better to make a split size equal to the size of an HDFS block (which is 64 MB, by default). • Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS. • Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation. • Map output is intermediate output which is processed by reduce tasks to produce the final output. • Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill. • In the event of node failure, before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output. • Reduce task doesn't work on the concept of data locality. An output of every map task is fed

to the reduce task. Map output is transferred to the machine where reduce task is running. • On this machine, the output is merged and then passed to the user-defined reduce function. • Unlike the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output

How MapReduce Organizes Work? Hadoop divides the job into tasks. There are two types of tasks: • Map tasks (Splits & Mapping) • Reduce tasks (Shuffling, Reducing) The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called a • Jobtracker: Acts like a master (responsible for complete execution of submitted job) • Multiple Task Trackers: Acts like slaves, each of them performing the job For every job submitted for execution in the system, there is one Jobtracker that resides on Namenode and there are multiple tasktrackers which reside on Datanode.

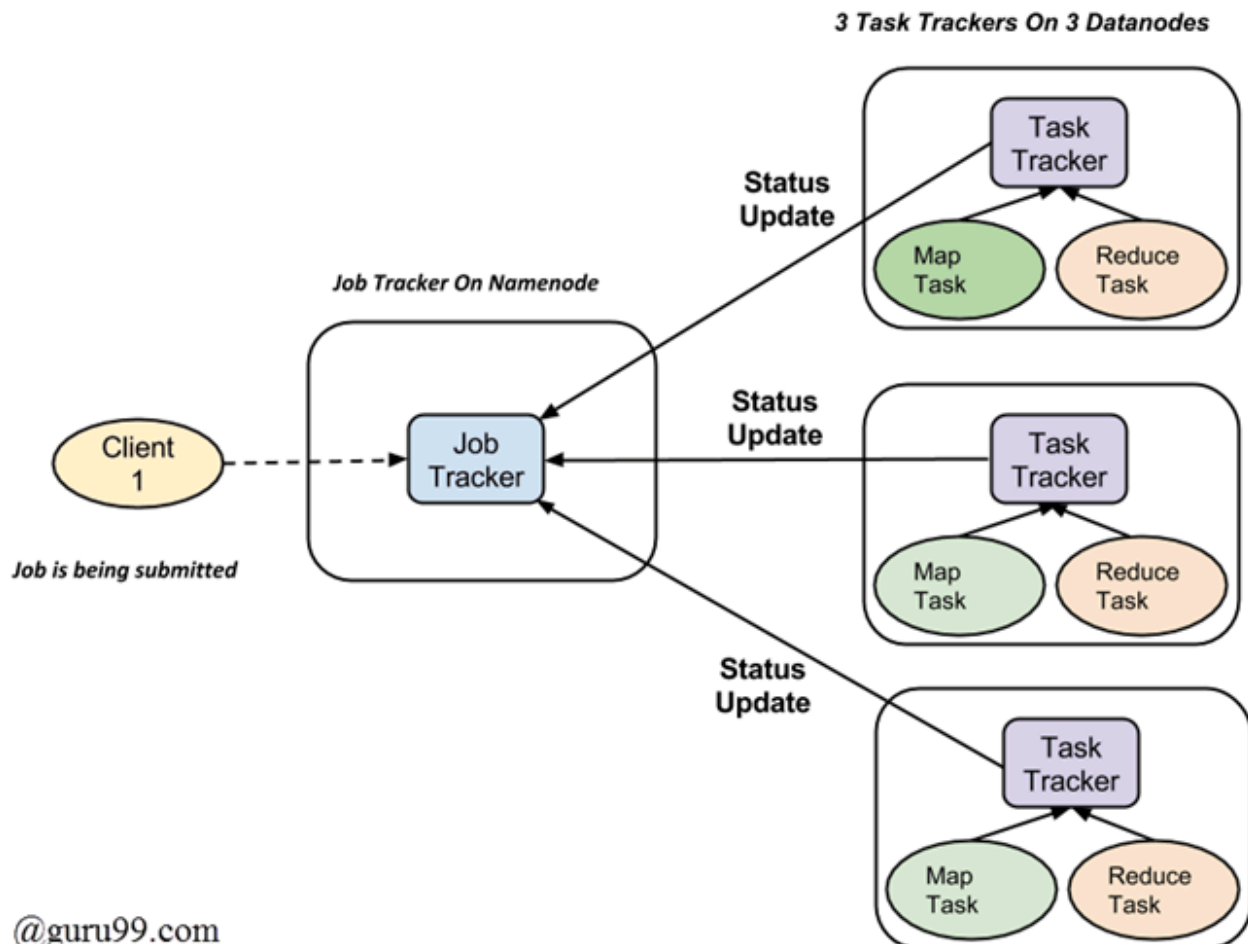


Fig. How Hadoop Mapreduce Works • A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster. • It is the

responsibility of job tracker to coordinate the activity by scheduling tasks to run on different data nodes. • Execution of individual task is then to look after by task tracker, which resides on every data node executing part of the job. • Task tracker's responsibility is to send the progress report to the job tracker. • In addition, task tracker periodically sends 'heartbeat' signal to the Jobtracker so as to notify him of the current state of the system. • Thus job tracker keeps track of the overall progress of each job. In the event of task failure, the job tracker can reschedule it on a different task tracker.

Implementation of Hadoop Cluster and HDFS:

To demonstrate the working and configuration of HDFS, a cluster consisting of 1 Master and 2 Slaves is configured. The machine nodes are independent AWS ec2 instances. For ease of operation a tool called MobaXterm is used for remote login (SSH) into the instances.

After successful configuration, the daemons are started in both master and slave nodes. Subsequently, our HDFS can be tested by storing files.

Step 1: Setting up Remote Machines

Three ec2 instances having Ubuntu as local OS were created using AWS Free Tier account.

Instance1 -> Master Private IP: 172.31.87.194

Instance3 -> Slave1 Private IP: 172.31.92.80

Instance4 -> Slave2 Private IP: 172.31.28.27

Instances (3) Info Refresh Connect Instance state ▼ Actions ▼ Launch instances ▼

Find instance by attribute or tag (case-sensitive)

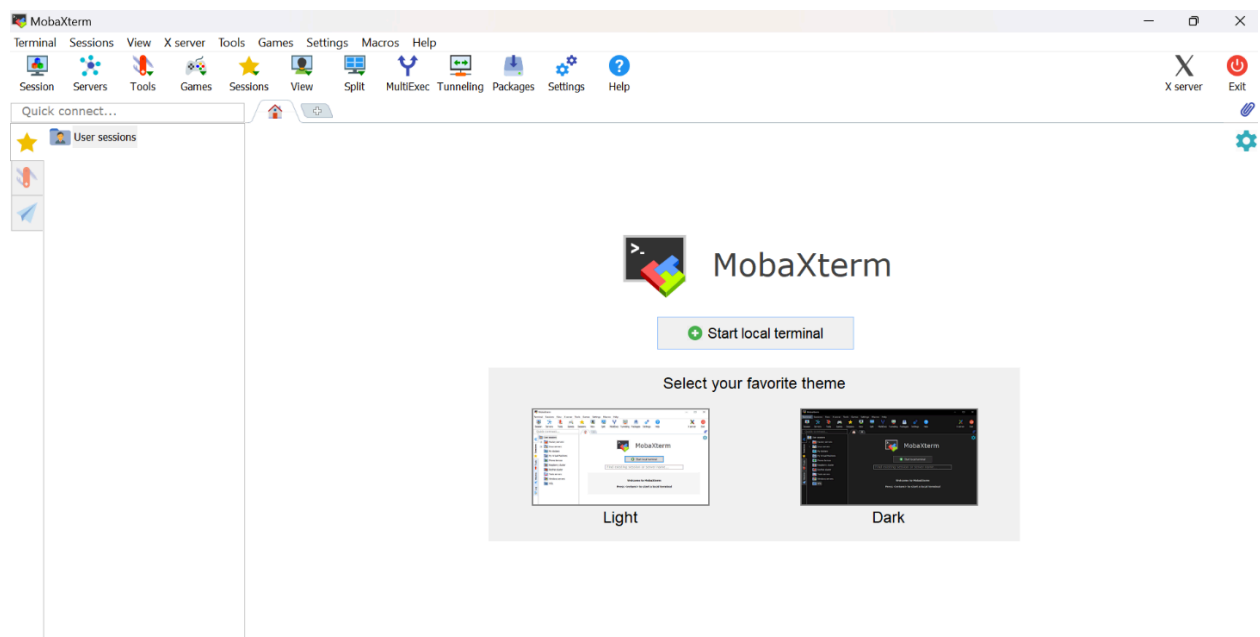
Instance state = running X Clear filters

<input type="checkbox"/>	Name ▲	Instance ID	Instance state ▼	Instance type ▼	Status check	Alarm status	Availability zone
<input type="checkbox"/>	instance1	i-036807135ca998acd	Stopped	t2.micro	2/2 checks passed	No alarms	us-east-1
<input type="checkbox"/>	instance3	i-015ed27bf316ae264	Stopping	t2.micro	2/2 checks passed	No alarms	us-east-1
<input type="checkbox"/>	instance4	i-0b1506b6fc8f234d8	Stopping	t2.micro	2/2 checks passed	No alarms	us-east-1

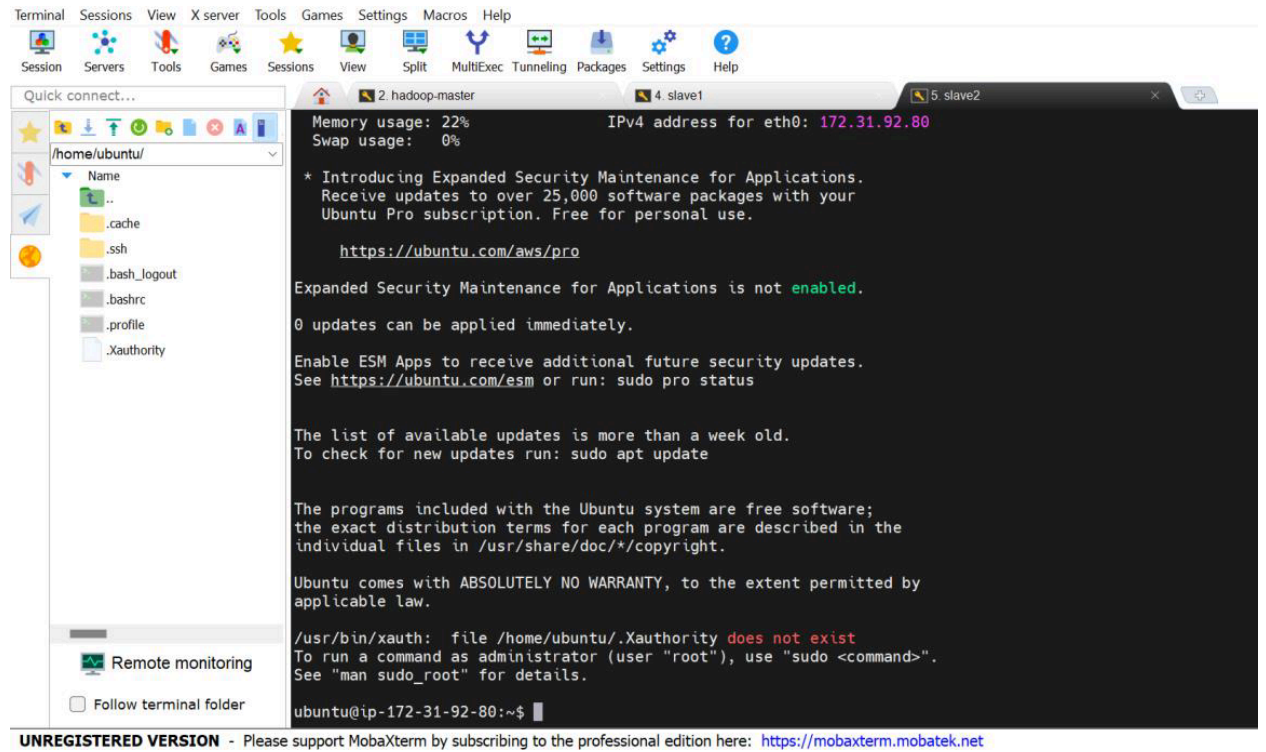
Select an instance Settings Close

Step 2: Setting up SSH Client

MobaXterm is used as an SSH Client to login to remote machines



After connecting to all our nodes, the environment looks like this. SSH connections to our remote machines are lined up as tabs



Step 3: Configuration of Hadoop Cluster

Step 3.1: Download Hadoop and JDK in all three machines

The step is repeated for all the 3 nodes

Download jdk-8


```

ubuntu@ip-172-31-87-194:~$ sudo apt install openjdk-8-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  adwaita-icon-theme alsa-topology-conf alsa-ucm-conf at-spi2-core ca-certificates-java dconf-gsettings-backend
  dconf-service fontconfig fontconfig-config fonts-dejavu-core fonts-dejavu-extra gsettings-desktop-schemas
  gtk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common libasound2 libasound2-data libasynclibs0
  libatk-bridge2.0-0 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0 libavahi-client3
  libavahi-common-data libavahi-common3 libcairo-gobject2 libcairo2 libcups2 libdatrie1 libdbus-1-3 libdrm-amdgpu1
  libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libflac8 libfontconfig1 libfontenc1 libgail-common libgail18
  libgdk-pixbuf-2.0-0 libgdk-pixbuf2.0-bin libgdk-pixbuf2.0-common libgif7 libgl1 libgl1-amd-glx libgl1-mesa-dri
  libgl1-mesa-glx libglapi-mesa libglvnd0 libglx-mesa0 libglx0 libgraphite2-3 libgtk2.0-0 libgtk2.0-bin libgtk2.0-common
  libharfbuzz0b libice-dev libice6 libjbig0 libjpeg-turbo8 libjpeg8 liblcms2-2 libllvm15 libogg0 libopus0 libpango-1.0-0
  libpangocairo-1.0-0 libpangoft2-1.0-0 libpciaccess0 libpcsc-lite libpixmap-1-0 libpthread-stubs0-dev libpulse0 librsync2-2
  librsync2-common librsyncs-config librsyncs5 libsm-dev libsm6 libsndfile1 libthai-data libthai0 libtiff5 libvorbis0a
  libvorbisenc2 libwebp7 libx11-dev libx11-xcb1 libxau-dev libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0 libxcb-present0
  libxcb-render0 libxcb-shape0 libxcb-shm0 libxcb-sync1 libxcb-xf86-dri3-0 libxcb-xfixes0 libxcb1-dev libxcomposite1 libxcursor1 libxdamage1
  libxdmcp-dev libxfixes3 libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxpm4 libxrandr2 libxrender1 libxshmfence1
  libxt-dev libxt6 libxtst6 libxv1 libxxf86dga1 libxxf86vm1 openjdk-8-jdk-headless openjdk-8-jre openjdk-8-jre-headless
  session-migration ubuntu-mono x11-common x11-utils x11proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  default-jre libasound2-plugins alsa-utils cups-common gvfs libice-doc liblcms2-utils opus-tools pcscd pulseaudio
  librsync2-doc libsm-doc libx11-doc libxcb-doc libxt-doc openjdk-8-demo openjdk-8-source visualvm libnss-mdns
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei fonts-wqy-zenhei fonts-indic mesa-utils
The following NEW packages will be installed:
  adwaita-icon-theme alsa-topology-conf alsa-ucm-conf at-spi2-core ca-certificates-java dconf-gsettings-backend
  dconf-service fontconfig fontconfig-config fonts-dejavu-core fonts-dejavu-extra gsettings-desktop-schemas
  gtk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common libasound2 libasound2-data libasynclibs0
  libatk-bridge2.0-0 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0 libavahi-client3
  libavahi-common-data libavahi-common3 libcairo-gobject2 libcairo2 libcups2 libdatrie1 libdbus-1-3 libdrm-amdgpu1
  libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libflac8 libfontconfig1 libfontenc1 libgail-common libgail18
  libgdk-pixbuf-2.0-0 libgdk-pixbuf2.0-bin libgdk-pixbuf2.0-common libgif7 libgl1 libgl1-amd-glx libgl1-mesa-dri

```

support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

Download Hadoop

```

ubuntu@ip-172-31-87-194:~$ wget https://dldcn.apache.org/hadoop/common/hadoop-2.10.2/hadoop-2.10.2.tar.gz
--2023-04-05 19:57:56-- https://dldcn.apache.org/hadoop/common/hadoop-2.10.2/hadoop-2.10.2.tar.gz
Resolving dldcn.apache.org (dldcn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dldcn.apache.org (dldcn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 414624228 (395M) [application/x-gzip]
Saving to: 'hadoop-2.10.2.tar.gz'

hadoop-2.10.2.tar.gz      100%[=====>] 395.42M  88.8MB/s   in 4.2s

2023-04-05 19:58:00 (95.0 MB/s) - 'hadoop-2.10.2.tar.gz' saved [414624228/414624228]

ubuntu@ip-172-31-87-194:~$

```

Extracting Hadoop

```

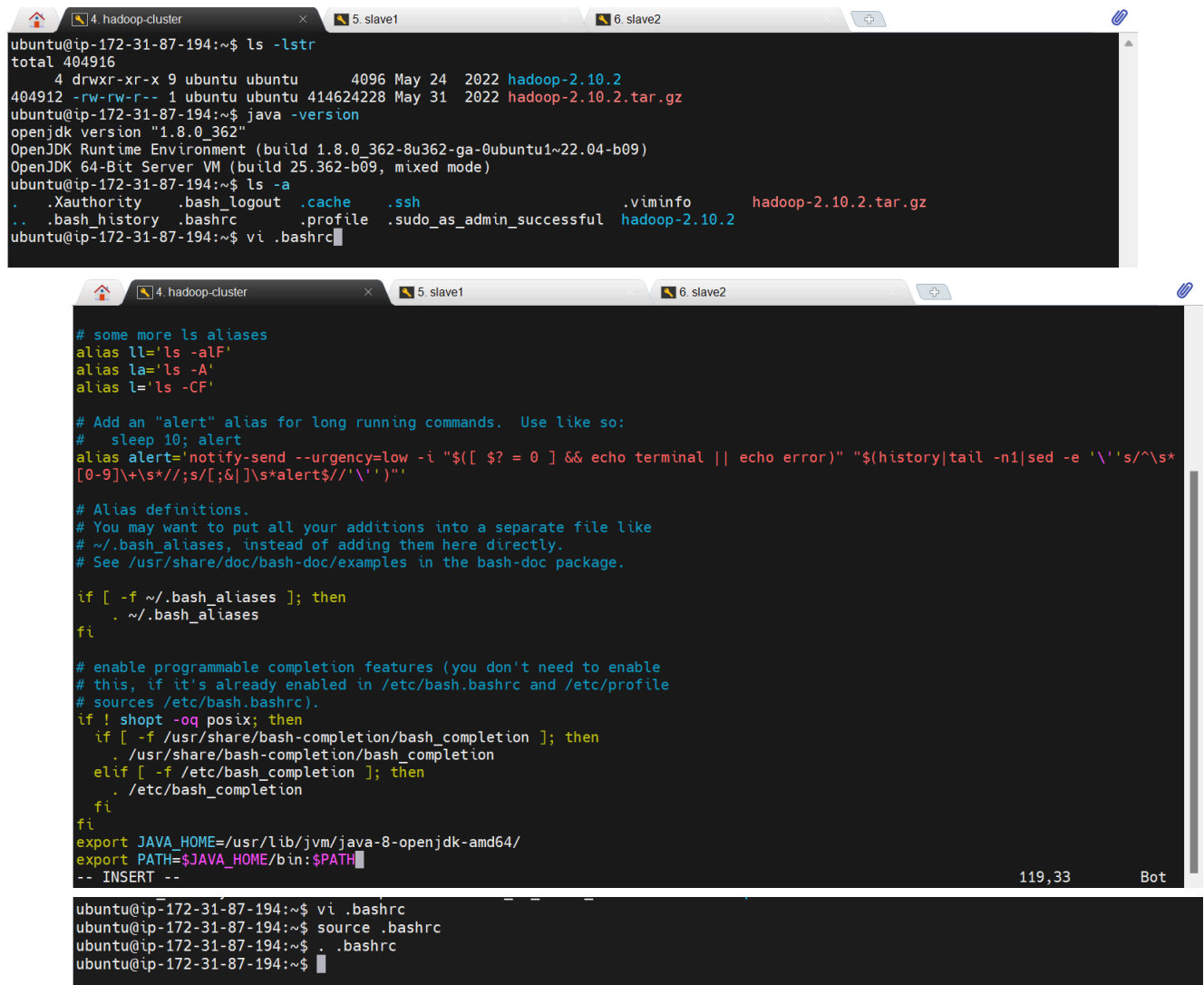
ubuntu@ip-172-31-87-194:~$ ls -la
.  ..  .Xauthority  .bash_logout  .bashrc  .cache  .profile  .ssh  .sudo_as_admin_successful  hadoop-2.10.2.tar.gz
ubuntu@ip-172-31-87-194:~$ tar -zxvf hadoop-2.10.2.tar.gz
hadoop-2.10.2/
hadoop-2.10.2/NOTICE.txt
hadoop-2.10.2/sbin/
hadoop-2.10.2/sbin/start-yarn.cmd
hadoop-2.10.2/sbin/start-balancer.sh
hadoop-2.10.2/sbin/slaves.sh
hadoop-2.10.2/sbin/start-dfs.sh

```

Step 3.2: Environment Setup for java

This step is repeated for all three nodes

Adding JDK Path in .bashrc file and executing it



The image shows two terminal windows from a hadoop-cluster. The first window shows the output of 'ls -lstr' and 'java -version', confirming the JDK installation. The second window shows the contents of the '.bashrc' file, which includes aliases, an 'alert' alias, and the addition of the JDK path to the PATH variable. The terminal output for the second window is as follows:

```
ubuntu@ip-172-31-87-194:~$ ls -lstr
total 404916
 4 drwxr-xr-x 9 ubuntu ubuntu    4096 May 24  2022 hadoop-2.10.2
404912 -rw-rw-r-- 1 ubuntu ubuntu 414624228 May 31  2022 hadoop-2.10.2.tar.gz
ubuntu@ip-172-31-87-194:~$ java -version
openjdk version "1.8.0_362"
OpenJDK Runtime Environment (build 1.8.0_362-8u362-ga-0ubuntu1~22.04-b09)
OpenJDK 64-Bit Server VM (build 25.362-b09, mixed mode)
ubuntu@ip-172-31-87-194:~$ ls -a
.  .Xauthority  .bash_logout  .cache  .ssh  .viminfo  hadoop-2.10.2.tar.gz
.. .bash_history .bashrc      .profile .sudo_as_admin_successful hadoop-2.10.2
ubuntu@ip-172-31-87-194:~$ vi .bashrc
```

```
# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]} && echo terminal || echo error" "$(history|tail -n1|sed -e '\''s/^s*[0-9]\+:\s*//;s/;/&]\s*alert$//'\''")"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
. ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
if [ -f /usr/share/bash-completion/bash_completion ]; then
. /usr/share/bash-completion/bash_completion
elif [ -f /etc/bash_completion ]; then
. /etc/bash_completion
fi
fi
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
export PATH=$JAVA_HOME/bin:$PATH
-- INSERT --
```

```
ubuntu@ip-172-31-87-194:~$ vi .bashrc
ubuntu@ip-172-31-87-194:~$ source .bashrc
ubuntu@ip-172-31-87-194:~$ . .bashrc
ubuntu@ip-172-31-87-194:~$
```

Step 3.3: SSH Setup Between Master and Slaves

```
ubuntu@ip-172-31-87-194:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.

Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa): Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:3sQIMog7W/u/Q6U2g8LZ6Bqg2BchRtRB7fEvfZ8FsRM ubuntu@ip-172-31-87-194
The key's randomart image is:
+----[RSA 3072]-----+
| .oooo |
| o.. o  E |
| . + +..o  + |
| o . +..oo  + |
| +..+.. oSoo  o |
| +=.o.*..oo . . |
| =.o..o o.... . o |
| ..o .      o |
| .. ..oo     |
+----[SHA256]-----+
ubuntu@ip-172-31-87-194:~$

ubuntu@ip-172-31-87-194:~$ cd .ssh
ubuntu@ip-172-31-87-194:~/.ssh$ ls -lstr
total 12
4 -rw----- 1 ubuntu ubuntu 395 Apr  5 18:43 authorized_keys
4 -rw-r--r-- 1 ubuntu ubuntu 577 Apr  6 03:09 id_rsa.pub
4 -rw----- 1 ubuntu ubuntu 2610 Apr  6 03:09 id_rsa
ubuntu@ip-172-31-87-194:~/.ssh$

ubuntu@ip-172-31-87-194:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQC5RS2e0E4CDHj06JYcZ08/NiLVfYBHfmSrzhWeZbdwMv4Ecd5/keB8J5CHMXpLW6Nx1Qg0v6w89x/LtUjstEosha
dPWYw087KtHsaVE2/c/IoAE9YcgJMyvcmLSDiQ4PfCIQK59bnp/5DYgigPHGE+tF6gA8Tiga3PYgpmozEGkgLbIGaMY8q8hwdm2fYImT+RuAayFvKLamfy13SFqC8D
Z+k1aDc9MYNb6Yua8E0dKau1hG7RN3faNZ5LthjxCNEWxxesODLJ+Rm/kXNRyrhlg6qmIvRvnxuyj23WNCuDuGyNLeaYBvgdaC+kC1f7Ujro9RZPvZgqlsFkJh6g9v
wcZJEcpY2kfH/RdDU4A6gA6KfcuJoy0VRy5PzqYZK4yW93erKIWgXWLA/SvdJkR3WDemE34lsbD0G6IzCCE86iTEfh2fosIPcZENxdd8gGymLJEgJ8CwFVS6N0Ck5G
eF2wJ6NkrYfc1fSsgt/n/8vmJGKa/eU3TDmiFtLHeu8QDP0= ubuntu@ip-172-31-87-194
ubuntu@ip-172-31-87-194:~/.ssh$
```

- Generate Key Pairs on each node using: `ssh-keygen -t rsa`. The generated pairs are stored in `id_rsa.pub` in `.ssh` folder
- Key Pairs from all the nodes are copied together and pasted in `authorized_keys` file present in `.ssh` folder at every node.

Key Pair generation at slave nodes

```
4. hadoop-cluster 5. slave1 6. slave2
ubuntu@ip-172-31-85-217:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.

Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa): Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:4qvY5XlWHEUK32v9bXLDMLlEXF3rMe4vsdvDGB3BsQ0 ubuntu@ip-172-31-85-217
The key's randomart image is:
+----[RSA 3072]-----+
|  . .+... o |
|  o + ....+ |
|  + . .+=. |
|  . . o .+oo |
|  . S o ..oo |
|  o o . E=o |
|  o + o**+ |
|  .. + . . =Bo |
|  o+.. .o+ |
+----[SHA256]-----+
ubuntu@ip-172-31-85-217:~$
```

```
4. hadoop-cluster 5. slave1 6. slave2
ubuntu@ip-172-31-92-80:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.

Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa): Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:m4hFtD2hRzb+2sa+Xe1W4jJN/V6W+VWtIdFIhGDfoFk ubuntu@ip-172-31-92-80
The key's randomart image is:
+----[RSA 3072]-----+
|  . =o.Eoo |
|  . B.o=+ o |
|  + =o . + . |
|  . . o . o o |
|  . S . . =. |
|  o . * . o.B |
|  . . + + . +B |
|  o .o.o o* |
|  o .o o+ |
+----[SHA256]-----+
ubuntu@ip-172-31-92-80:~$
```

Final authorized_keys file: This file needs to be maintained at every node

```
4. hadoop-master 11. slave1 6. slave2
ubuntu@ip-172-31-87-194:~$ cd .ssh
ubuntu@ip-172-31-87-194:~/.ssh$ la -a
. . . authorized_keys id_rsa id_rsa.pub
ubuntu@ip-172-31-87-194:~/.ssh$ vi authorized_keys
ubuntu@ip-172-31-87-194:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCVt/R8WVLxg5J0Uryh0cJ0j0DAiY0A6NGCzOZ8s6TPPZCAwKyE3JCIOyvS/fDn+tIheFv3r0A7srkaRMfv8
0Zg8/lmapJumQqIH7yDcdqjG1ZMV8Wycg4hK4F7KK33MiRa0cqy34uiImQDmZdAb3B96TX8YumXYSMzB8fCQxiF0ihXEBUTW8tICLqcdPSuhLZeafJmprejFT
gIPnxIPruWUmDoHvUmrH7TW1UNo0Y7sLmPAvgastnS/E3VaYGuyxXlVQqGBFCyrIovRQ4utQiTxihC7j83e0IUr4/VPgvtEsML1+h0XtA4JVnMMK0p2RXQtX
SUBWXLmjx6E7njSLj instance1-key
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQC5RS2e0E4CDHj06JYcZQ8/NiLvfYBHfmsRzhWeZbdwNv4Ecd5/keB8J5CHMXpLW6NxlQg0v6w89x/ltUjst
EoshadPWyw087KtHsaVE2/c/IoAE9YcgJMyvcmLSdiQ4PfcIQK59bpn/5DYgigPHGE+tF6gA8Tlqa3PYgpmozEGkgLbIGaMY8q8hwdm2fYImT+RuAayFvKLam
fy13SFqC8DZ+K1aDc9MYNb6YuaE0dKau1hG7RN3faNZ5LthjzCNEWxxes0D1j+Rm/kXNryrhlg6qmIvRvnxyj23wNCuDWGyNLeaYBvgdaC+kC1f7UjroRZ
PvZgqlsFkjh6g9vwcZJECpy2kfH/RdDU4A6gA6KfcuJoy0VRy5PzqYZK4yW93erKIWgXWLA/SvdJkR3WdemE34lsbD0G6IzCCE86iTEfh2fosIPcZEnxdd8gG
ymLJEgJ8CwFVS6N0Ck5GeF2wJ6NkrYfcLfSsgt/n/8vmJGKa/eU3TDmIFtLHeu8QDP0= ubuntu@ip-172-31-87-194
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgCvtX6LHSIfku+WbAdQ6f5bkyJ0VYHZIRV43byeont3AqL5unasytNwn9IiP7VJVMJEdUsoppFcT8LhJZLaB
9Qw0nR4pzTdnzhrkKMjM/FA4ym+jg4tlzx0pEhTr+NGZnmIFWUvneuXfL/n8wdkSudjCW+2Q5Mhh3x9QIQIvft9eTfMs0zgX7yPuPQ2tMLUveJbDosX+qE
dxUouQbfx+G+kQ+uDBgNzGeQ0/aRUiH2rNshb0MNCgmknwzmlNp0w//zzzQ65M7YHhSOKN0bJ/fIANhwLqk9DD5TAfXUtuVUNDtCpAUatFfmvqBHEGgJczA29V
XbzaR0/m6E8gwsWL7QWlriji7Lup3PegGy2ri0LTPxaCgU9zCICxFz6achdEGbvoU+jWab2PN2mo0gZz8UsJ5LYckb2hKC81X/76XW/ZdQkj0obK/L10JAqaX
f2GylqBbJUEIEcnP1g9jnjG3lBIDDSme8FB0s30N1r+xFj65CosVJnodor6VTaSLJE8= ubuntu@ip-172-31-28-27
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgCp9MLicln9I110u9ybskEkamLRCCyQzQw0HoUtxxBmEHB/RPHW9zq8rt3w+sLuyUzWlhXU32sV6NNcLQgX
bUY2YRtBuB/eXlSWfQ+ZlpkdsPhYhC48usEvHCKip6BYgG0d4BsU5ntAfZjiE3JJEWraRkkPmAExMHND/ellh+urYl9YlMLhavl+/pMT07yaJsimvzEt1XZ3jb
tdH6Kc/W0nDb2I9t24cmv4LHFB+FTVb+dMwvZrELgpYBPDyeLz7fsG6ZTTwtM9TdPt4sD9ZTuRbTvff8SNDSPiZd+v/9noHX2ccvLEj/3DAlyD6ho2+m7P7W
7xa4ipLDRME9/j8gkiq5cDbt57saXcD02kgJobakRYw2kZY4TIk1GvURMMhwhLH2sC4urIq3anI9+419ntFNotL11qWUCJiRIWfXK2Lk88rBJrraYlZk5x+sJ
Zjgdg81oZH73vt58S1XDoC7xYgNqxDHZZtofcGdyNG3qlJ9+2LHESs/KIiCqAdP9eE= ubuntu@ip-172-31-92-80
ubuntu@ip-172-31-87-194:~/.ssh$
```

Step 3.4: Configuration of Hadoop Files. Files present in path:
hadoop-2.10.2/etc/Hadoop

```
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2/etc/hadoop$ vi slaves
```

Configuration of slaves files (Only at Master Node): Add Private IPs of Slaves
in slaves file present in master node

```
172.31.28.27
172.31.92.80
~
~
~
~
~
~
~
```

Configuration of env files (Needs to be repeated for every node): Adding JDK
path to each file

mapred-env.sh

```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# export JAVA_HOME=/home/y/libexec/jdk1.6.0/

# when HADOOP_JOB_HISTORYSERVER_HEAPSIZE is not defined, set it.
if [ "$HADOOP_JOB_HISTORYSERVER_HEAPSIZE" = "" ];then
  export HADOOP_JOB_HISTORYSERVER_HEAPSIZE=1000
fi

export HADOOP_MAPRED_ROOT_LOGGER=INFO,RFA
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/

#export HADOOP_JOB_HISTORYSERVER_OPTS=
#export HADOOP_MAPRED_LOG_DIR="" # Where log files are stored. $HADOOP_MAPRED_HOME/logs by default.
#export HADOOP_JHS_LOGGER=INFO,RFA # Hadoop JobSummary logger.
#export HADOOP_MAPRED_PID_DIR= # The pid files are stored. /tmp by default.
#export HADOOP_MAPRED_IDENT_STRING= #A string representing this instance of hadoop. $USER by default
#export HADOOP_MAPRED_NICENESS= #The scheduling priority for daemons. Defaults to 0.
~
"mapred-env.sh" 31L, 1559B
```

yarn-env.sh

```
5. hadoop-master 6. slave1 7. slave2
fi
# restore ordinary behaviour
unset IFS

YARN_OPTS="$YARN_OPTS -Dhadoop.log.dir=$YARN_LOG_DIR"
YARN_OPTS="$YARN_OPTS -Dyarn.log.dir=$YARN_LOG_DIR"
YARN_OPTS="$YARN_OPTS -Dhadoop.log.file=$YARN_LOGFILE"
YARN_OPTS="$YARN_OPTS -Dyarn.log.file=$YARN_LOGFILE"
YARN_OPTS="$YARN_OPTS -Dyarn.home.dir=$YARN_COMMON_HOME"
YARN_OPTS="$YARN_OPTS -Dyarn.id.str=$YARN_IDENT_STRING"
YARN_OPTS="$YARN_OPTS -Dhadoop.root.logger=${YARN_ROOT_LOGGER:-INFO,console}"
YARN_OPTS="$YARN_OPTS -Dyarn.root.logger=${YARN_ROOT_LOGGER:-INFO,console}"
if [ "$JAVA_LIBRARY_PATH" != "x" ]; then
    YARN_OPTS="$YARN_OPTS -Djava.library.path=$JAVA_LIBRARY_PATH"
fi
YARN_OPTS="$YARN_OPTS -Dyarn.policy.file=$YARN_POLICYFILE"

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
```

hadoop-env.sh

```
5. hadoop-master 6. slave1 7. slave2
# HDFS Mover specific parameters
###
# Specify the JVM options to be used when starting the HDFS Mover.
# These options will be appended to the options specified as HADOOP_OPTS
# and therefore may override any similar flags set in HADOOP_OPTS
#
# export HADOOP_MOVER_OPTS=""
###
# Router-based HDFS Federation specific parameters
# Specify the JVM options to be used when starting the RBF Routers.
# These options will be appended to the options specified as HADOOP_OPTS
# and therefore may override any similar flags set in HADOOP_OPTS
#
# export HADOOP_DFSROUTER_OPTS=""
###
###
# Advanced Users Only!
###
# The directory where pid files are stored. /tmp by default.
# NOTE: this should be set to a directory that can only be written to by
# the user that will run the hadoop daemons. Otherwise there is the
# potential for a symlink attack.
export HADOOP_PID_DIR=${HADOOP_PID_DIR}
export HADOOP_SECURE_DN_PID_DIR=${HADOOP_PID_DIR}

# A string representing this instance of hadoop. $USER by default.
export HADOOP_IDENT_STRING=$USER

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
"hadoop-env.sh" 119L, 5022B 119,17 Bot
```

Configuration of XML Files:

core-site.xml: Same for master as slaves

The IP address provided is the private IP address of the Master Node

```

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

<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://172.31.87.194:50000</value>
</property>
</configuration>
~
~

```

yarn-site.xml: Same for master and slaves

The IP address provided is the private IP address of the Master Node

```

<property>
<name>yarn.nodemanager.aux-services</name> <value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
<description>The hostname of the RM.</description>
<name>yarn.resourcemanager.hostname</name>
<value>172.31.87.194</value>
</property>
<property>
<description>The address of the applications manager interface in the RM.</description>
<name>yarn.resourcemanager.address</name>
<value>172.31.87.194:8032</value>
</property>

```

hdfs-site.xml:

Master Node:

```

<configuration>
<property>
<name>dfs.namenode.name.dir</name>
<value>/home/ubuntu/hadoop2-dir/namenode-dir</value>
</property>
</configuration>
~

```

Slave Nodes:

```
<configuration>

<property>
<name>dfs.datanode.data.dir</name>
<value>/home/ubuntu/hadoop2-dir/datanode-dir</value>
</property>

</configuration>
```

mapred-site.xml: Same for master and slaves

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

Step 4: Adding inbound rules in security groups of instances

Master Instance:

Inbound rules Info						
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info	
sgr-01dbf782726765778	Custom TCP ▼	TCP	8032	Custom ▼ Q 0.0.0.0/0 ✕		Delete
sgr-066505b604cc38522	Custom TCP ▼	TCP	50090	Custom ▼ Q 0.0.0.0/0 ✕		Delete
sgr-0094cc3175570dd00	Custom TCP ▼	TCP	8020	Custom ▼ Q 0.0.0.0/0 ✕		Delete
sgr-0f44d937cec22fc9d	Custom TCP ▼	TCP	9000	Custom ▼ Q 0.0.0.0/0 ✕		Delete
sgr-030038630b08ee42f	SSH ▼	TCP	22	Custom ▼ Q 0.0.0.0/0 ✕		Delete

Slave Instances:

Inbound rules Info						
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info	
sgr-0e0dbf3bdf561bd2f	Custom TCP ▼	TCP	50020	Custom ▼ Q 0.0.0.0/0 ✕		Delete
sgr-011c222e5ff2d18de	SSH ▼	TCP	22	Custom ▼ Q 0.0.0.0/0 ✕		Delete
sgr-0d64e55cafa8c996e	Custom TCP ▼	TCP	50075	Custom ▼ Q 0.0.0.0/0 ✕		Delete
sgr-021af2ed049d97cce	Custom TCP ▼	TCP	50010	Custom ▼ Q 0.0.0.0/0 ✕		Delete

Step 5: Formatting of Hadoop Cluster and Starting of the daemons

Format Hadoop Cluster: Command is executed only at the Master Node

```
ubuntu@ip-172-31-87-194:~$ ls
hadoop-2.10.2  hadoop-2.10.2.tar.gz
ubuntu@ip-172-31-87-194:~$ cd hadoop-2.10.2
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$ bin/hadoop namenode -format
```

```
23/04/06 08:34:37 INFO namenode.FSDirectory: ACLs enabled? false
23/04/06 08:34:37 INFO namenode.FSDirectory: XAttrs enabled? true
23/04/06 08:34:37 INFO namenode.NameNode: Caching file names occurring more than 10 times
23/04/06 08:34:37 INFO snapshot.SnapshotManager: Loaded config captureOpenFiles: false skipCaptureAccessTimeOnlyChange: false
23/04/06 08:34:37 INFO util.GSet: Computing capacity for map cachedBlocks
23/04/06 08:34:37 INFO util.GSet: VM type = 64-bit
23/04/06 08:34:37 INFO util.GSet: 0.25% max memory 966.7 MB = 2.4 MB
23/04/06 08:34:37 INFO util.GSet: capacity = 2^18 = 262144 entries
23/04/06 08:34:37 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
23/04/06 08:34:37 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
23/04/06 08:34:37 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
23/04/06 08:34:37 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
23/04/06 08:34:37 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
23/04/06 08:34:37 INFO util.GSet: Computing capacity for map NameNodeRetryCache
23/04/06 08:34:37 INFO util.GSet: VM type = 64-bit
23/04/06 08:34:37 INFO util.GSet: 0.0299999999329447746% max memory 966.7 MB = 297.0 KB
23/04/06 08:34:37 INFO util.GSet: capacity = 2^15 = 32768 entries
23/04/06 08:34:37 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1635550088-172.31.87.194-1680770077648
23/04/06 08:34:37 INFO common.Storage: Storage directory /home/ubuntu/hadoop2-dir/namenode-dir has been successfully formatted.
23/04/06 08:34:37 INFO namenode.FSImageFormatProtobuf: Saving image file /home/ubuntu/hadoop2-dir/namenode-dir/current/fsimage.ckpt.00000000000000000000 using no compression
23/04/06 08:34:37 INFO namenode.FSImageFormatProtobuf: Image file /home/ubuntu/hadoop2-dir/namenode-dir/current/fsimage.ckpt.00000000000000000000 of size 325 bytes saved in 0 seconds.
23/04/06 08:34:37 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
23/04/06 08:34:37 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid = 0 when meet shutdown.
23/04/06 08:34:37 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at ip-172-31-87-194.ec2.internal/172.31.87.194
*****/
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$
```

Starting of daemons: Command Executed only at Master Node

```
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$ jps
9492 Jps
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$ sbin/start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [ip-172-31-87-194.ec2.internal]
ip-172-31-87-194.ec2.internal: starting namenode, logging to /home/ubuntu/hadoop-2.10.2/logs/hadoop-ubuntu-namenode-ip-172-31-87-194.out
172.31.92.80: starting datanode, logging to /home/ubuntu/hadoop-2.10.2/logs/hadoop-ubuntu-datanode-ip-172-31-92-80.out
172.31.28.27: starting datanode, logging to /home/ubuntu/hadoop-2.10.2/logs/hadoop-ubuntu-datanode-ip-172-31-28-27.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/ubuntu/hadoop-2.10.2/logs/hadoop-ubuntu-secondarynamenode-ip-172-31-87-194.out
starting yarn daemons
starting resourcemanager, logging to /home/ubuntu/hadoop-2.10.2/logs/yarn-ubuntu-resourcemanager-ip-172-31-87-194.out
172.31.28.27: starting nodemanager, logging to /home/ubuntu/hadoop-2.10.2/logs/yarn-ubuntu-nodemanager-ip-172-31-28-27.out
172.31.92.80: starting nodemanager, logging to /home/ubuntu/hadoop-2.10.2/logs/yarn-ubuntu-nodemanager-ip-172-31-92-80.out
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$
```

Daemons running at the Master:

```
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$ jps
10225 Jps
9974 ResourceManager
9848 SecondaryNameNode
9643 NameNode
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$
```

Daemons running at slave 1:

```
4. hadoop-master 11. slave1 6. slave2
ubuntu@ip-172-31-28-27:~$ jps
12248 Jps
ubuntu@ip-172-31-28-27:~$ jps
12452 NodeManager
12329 DataNode
12558 Jps
ubuntu@ip-172-31-28-27:~$
```

Daemons running at slave 2:

```
4. hadoop-master 11. slave1 6. slave2
ubuntu@ip-172-31-92-80:~$ jps
12015 Jps
ubuntu@ip-172-31-92-80:~$ jps
12325 Jps
12220 NodeManager
12095 DataNode
ubuntu@ip-172-31-92-80:~$
```

Step 6: Launching the Namenode Web Interface

The Web UI Provided by Apache, provides the following information about the cluster:

Non Heap Memory used 49.74 MB of 50.94 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	15.15 GB
DFS Used:	48 KB (0%)
Non DFS Used:	7.72 GB
DFS Remaining:	7.4 GB (48.84%)
Block Pool Used:	48 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	2 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	Fri Apr 07 13:21:05 +0530 2023
Last Checkpoint Time	Fri Apr 07 12:57:09 +0530 2023

Step 7: Testing HDFS by storing files

Creating a Test File

```
ubuntu@ip-172-31-87-194:~$ ls
hadoop-2.10.2  hadoop-2.10.2.tar.gz  hadoop2-dir
ubuntu@ip-172-31-87-194:~$ vi testfile1
```

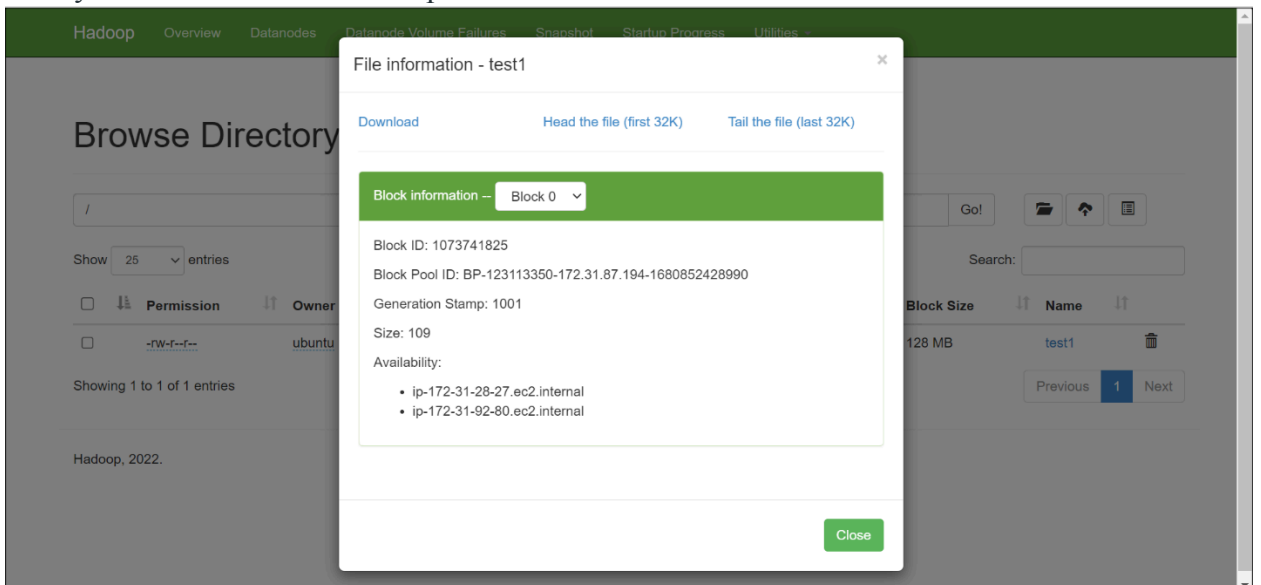
```
This is a test file. This will be stored in HDFS having 2 slave nodes. The default replication factor is 3.
~
~
~
~
~
~
```

Storing it in the HDFS:

```
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$ bin/hadoop dfs -put /home/ubuntu/testfile1 /test1
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

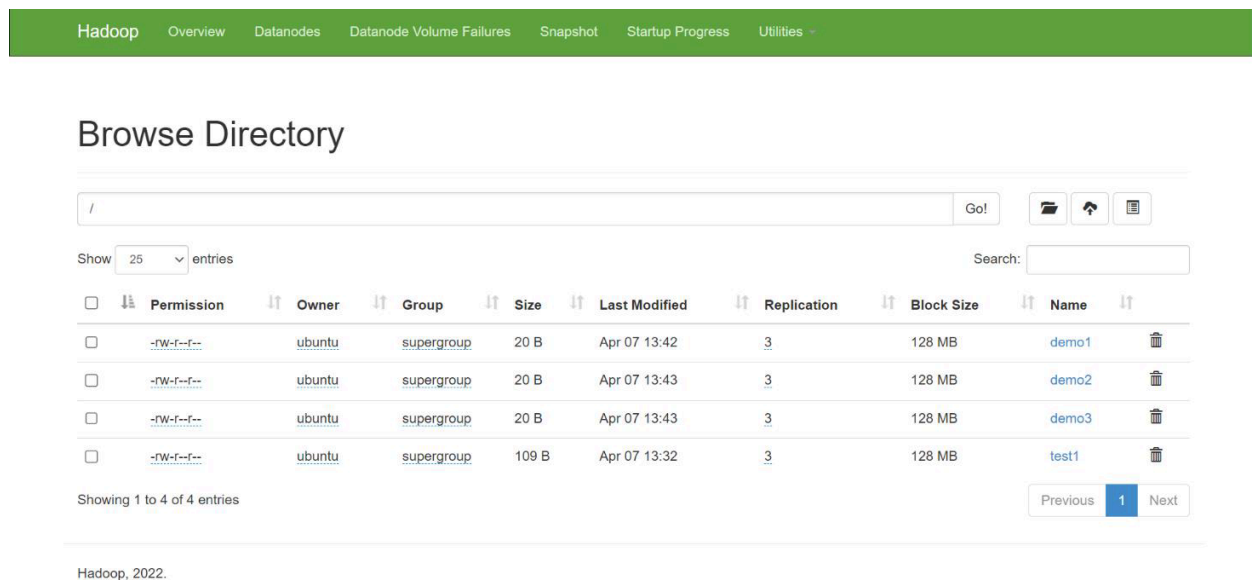
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2$
```

Successfully Stored in the DFS with replication factor of 3



Storing more files:

```
2. hadoop-master 3. slave1 4. slave2
ubuntu@ip-172-31-87-194:~/hadoop-2.10.2/logs$ cd
ubuntu@ip-172-31-87-194:~$ ls
hadoop-2.10.2  hadoop-2.10.2.tar.gz  hadoop2-dir  testfile1
ubuntu@ip-172-31-87-194:~$ vi demofile1
ubuntu@ip-172-31-87-194:~$ vi demofile2
ubuntu@ip-172-31-87-194:~$ vi demofile3
ubuntu@ip-172-31-87-194:~$ vi demofile3
ubuntu@ip-172-31-87-194:~$
```



Conclusion:

In conclusion, our experiment studying MapReduce and Hadoop Distributed File System (HDFS) has shown that these technologies are powerful tools for managing and processing large-scale data sets in a distributed computing environment.

The implementation of Hadoop Distributed File System (HDFS) with one master and two slave nodes on Amazon Web Services (AWS) EC2 instances was successful. The experiment showed that HDFS can be deployed and managed on the cloud, allowing for the storage and processing of large amounts of data in a distributed manner.

References:

- Data Engineering. (2022, April 12). Hadoop Multi Node Cluster Setup

[Video]. YouTube. https://www.youtube.com/watch?v=_iP2Em-5Abw

- Kumar, M. R. N. (2022, September 17). Hadoop-3.3.1 Installation guide for Ubuntu - Dev Genius [Video]. Medium.
<https://blog.devgenius.io/install-configure-and-setup-hadoop-in-ubuntu-a3cdd6305a0e>
- Gaurav Sharma. (2022, October 20). AWS Tutorials - 10 - Create First EC2 Instance | EC2 Instance Creation in AWS [Video]. YouTube.
<https://www.youtube.com/watch?v=f-T4xWUZWSk>

Postlab Questions:

1. What are the differences between traditional file systems and HDFS?

Feature	Traditional File Systems	HDFS
Architecture	Typically single-node	Distributed across multiple nodes
Data storage	Data stored in a single server or disk	Data stored across a cluster of machines
Fault tolerance	Generally less fault-tolerant	Built-in fault tolerance through replication
Scalability	Limited scalability, especially for large datasets	Highly scalable, can handle petabytes of data
Access patterns	Optimized for random reads and writes	Optimized for streaming reads and writes
Consistency	Strong consistency, typically immediate updates	Eventual consistency, eventual updates

Metadata management	Centralized metadata management	Distributed metadata management
Data locality	Limited data locality, data may need to be moved to computation nodes	Maximizes data locality, computation happens where data resides
Processing framework	Not inherently integrated with big data processing frameworks	Integrated with Hadoop ecosystem for distributed processing
Use cases	General-purpose file storage for single servers or small clusters	Specifically designed for storing and processing large datasets in a distributed environment, commonly used in big data applications

2. Enlist key features and components of HDFS.

1. Distributed Storage: HDFS distributes data across multiple nodes in a cluster, enabling scalability and fault tolerance.
2. High Fault Tolerance: Data replication across multiple nodes ensures high availability and fault tolerance. If a node fails, data can be retrieved from replicas stored on other nodes.
3. Scalability: HDFS is designed to scale horizontally, allowing it to handle massive amounts of data by adding more nodes to the cluster.
4. Data Locality: HDFS optimizes data processing by moving computation to where the data resides, minimizing data movement across the network.

5. Streaming Data Access: HDFS is optimized for streaming data access rather than random reads and writes, making it suitable for large-scale data processing.
6. Block-based Storage: Data is stored in large blocks (typically 128MB or 256MB), which improves throughput and reduces the overhead of managing a large number of small files.
7. Metadata Management: HDFS maintains metadata about files and directories in a centralized metadata server called the NameNode.
8. Data Replication: HDFS replicates data across multiple nodes to ensure fault tolerance and data durability. The replication factor is configurable, typically set to three.
9. Rack Awareness: HDFS is aware of the physical network topology, allowing it to place replicas across multiple racks for improved fault tolerance and data locality.
10. Checksums: HDFS uses checksums to detect and handle data corruption, ensuring data integrity during storage and transmission.
11. Command Line Interface (CLI): HDFS provides a command-line interface for users to interact with the file system, similar to traditional file systems.
12. Web User Interface (UI): HDFS includes a web-based UI that provides information about the cluster, file system, and data nodes.
13. Hadoop Ecosystem Integration: HDFS is a core component of the Apache Hadoop ecosystem, seamlessly integrating with other Hadoop projects such as MapReduce, HBase, Hive, and Spark for distributed data processing and analysis.