6. Hadoop 2.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 other than Linux such as Windows, MAC, Solaris… Hadoop YARN (RM, NM) 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

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.

Pseudo-distributed mode: Hadoop daemons run on separate JVM in a machine. It simulates the cluster environment running Hadoop services on different JVMs.

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

Fully distributed or cluster mode: Hadoop daemons run in different physical machines.

Virtual cluster mode: Hadoop daemons run in different VM's.

We have discussed enough about single and multi-node implementation for Hadoop 1.x. The following sections will not comprise much detail on concepts and commands used. So, please refer Hadoop 1.x before trying Hadoop 2.x to understand more about installation.

6.1 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
```

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. My physical machine IP address is 10.100.55.92

```
$ ifconfig -a                       // check for available interfaces

eth0      Link encap:Ethernet  HWaddr 24:be:05:18:ba:13
          inet6 addr: fe80::26be:5ff:fe18:ba13/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          collisions:0 txqueuelen:1000
          Interrupt:20 Memory:f7100000-f7120000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX bytes:481369929 (481.3 MB)bytes:4869929 (481.3 MB)
```

```
$ 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 try Ubuntu VM on windows, then you can use

- WINSCP to transfer files from windows (host) to Ubuntu VM. WINSCP is a file transfer protocol with nice GUI.
- PUTTY to remote login from windows (host) 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 in gzipped tar file from apache Hadoop release page.

If you want to install Hadoop on Windows machines, then install Cygwin and SSH server. It provides Unix-like environment on windows. Step-by-step instructions are given in the following link http://opensourceforu.com/2015/03/getting-started-with-hadoop-on-windows/.

6.2 Single node setup

6.2.1 Standalone or local mode

Step 1: download Hadoop 2.7.0 & unpack the tar file

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

```
$ wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.0/hadoop-2.7.0.tar.gz
```

```
$ tar -zxvf hadoop-2.7.0.tar.gz
```

Step 2: move to /usr/local/hadoop

```
$ sudo cp -r hadoop-2.7.0 /usr/local/hadoop
```

Step 3: setup environment variable for Hadoop

```
$ vi .bashrc
    export HADOOP_HOME=/usr/local/hadoop
    export PATH=$PATH:$HADOOP_HOME/bin
    export PATH=$PATH:$HADOOP_HOME/sbin
    export HADOOP_HDFS_HOME=$HADOOP_HOME
    export HADOOP_MAPRED_HOME=$HADOOP_HOME
    export HADOOP_COMMON_HOME=$HADOOP_HOME
    export YARN_HOME=$HADOOP_HOME
    export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
    export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
    export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
```

```
$ source .bashrc
$ $PATH
```

Step 4: edit Hadoop configuration file

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

```
$ yarn version
```

Step 5: grant privileges

Give full access rights to the Hadoop services to read and write at /usr/local/hadoop. username in my machine is **itadmin**.

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

Step 6: running a sample job

You need not start or stop any services to work on this mode. MR uses local file system, not HDFS.

Create a file input.txt in local file system and launch wordcount job with output directory called "result" (it is created on local file system, not on HDFS). Once job is submitted, LocalJobRunner will take care of the job execution as no MR is running.

```
$ yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar  wordcount input.txt result
$ ls result
        part-r-00000
        _SUCCESS
$ cat result/part-r-00000
```

6.2.2 Pseudo distributed mode

assume IP of working node (physical machine/VM) is **10.100.55.92** and username is **itadmin**

Step 1: generate ssh key for password less communication

```
$ ssh-keygen  or ssh-keygen -t rsa                    // leave empty for name, pwd…
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
$ chmod 700 ~/.ssh
```

Step 2: download Hadoop2.7.0 & unpack the tar file

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

```
$ wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.0/hadoop-2.7.0.tar.gz
$ tar -zxvf hadoop-2.7.0.tar.gz
```

Step 3: move to /usr/local/hadoop

```
$ sudo cp -r hadoop-2.7.0 /usr/local/hadoop
```

Step 4: edit .bashrc file to setup environment variable for Hadoop

```
$ vi .bashrc
        export HADOOP_HOME=/usr/local/hadoop
        export PATH=$PATH:$HADOOP_HOME/bin
        export PATH=$PATH:$HADOOP_HOME/sbin
        export HADOOP_HDFS_HOME=$HADOOP_HOME
        export HADOOP_MAPRED_HOME=$HADOOP_HOME
        export HADOOP_COMMON_HOME=$HADOOP_HOME
        export YARN_HOME=$HADOOP_HOME
        export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
        export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
```

```
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
```

```
$ source .bashrc
$ $PATH
```

Step 5: configure Hadoop environment variable

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

Step 6: edit configuration files

To setup NN

fs.defaultFS - 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, spilling map output, to store blocks in DN's…

```
$ sudo vi /usr/local/hadoop/etc/hadoop/core-site.xml
<configuration>
      <property>
              <name>fs.defaultFS</name>
              <value>hdfs://10.100.55.92:10001</value>
      </property>
      <property>
              <name>hadoop.tmp.dir</name>
              <value>/usr/local/hadoop/tmp</value>
      </property>
</configuration>
```

To setup YARN

```
$ sudo vi /usr/local/hadoop/etc/hadoop/yarn-site.xml
<configuration>
     <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>
 </configuration>
```

auxiliary services: in MR1, there is TT to send intermediate data to reducer. But, in MR2, NM doesn't do that (because it is YARN component). So, we have separate auxiliary service that must be set in yarn-site.xml.

yarn.nodemanager.aux-services - containers don't know what is executing inside since YARN may run any data processing tools job. So, map tasks running inside the container don't know how to perform shuffle. Therefore, this property specifies explicitly shuffle should be taken care by MR framework itself.

yarn.nodemanager.aux-services.mapreduce.shuffle.class – denotes the library class that handles shuffle and sort.

To setup MR related properties

$ sudo cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml

mapreduce.framework.name – specifies resource management component as YARN.

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

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

To setup NN, SNN, DN configuration (optional)

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

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

To setup SNN: SNN is automatically up and run with NN, however, you can setup.

To setup slave (DN and NN)

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

```
10.100.55.92
```

Step 7: change ownership, access mode and create Namespace.

$ sudo chown -R itadmin /usr/local/hadoop
$ sudo chmod -R 777 /usr/local/hadoop
$ hdfs namenode -format // HDFS namespace is created in /usr/local/hadoop/tmp/dfs/name

$ ls /usr/local/hadoop/tmp/dfs/name/current // you see something as follows

```
fsimage_0000000000000000000  fsimage_0000000000000000000.md5
seen_txid  VERSION
```

Step 8: start HDFS/YARN

To start/stop dfs and yarn

$ start-dfs.sh // location for DN and SNN to store are created

$ ls /usr/local/hadoop/tmp/dfs/

```
name data secondarynamenode
```

$ jps

```
29557 DataNode
```

```
29753 SecondaryNameNode
29403 NameNode
```

$ start-yarn.sh

$ ls /usr/local/hadoop/tmp/ // location for RM and NM to store are created

```
dfs nm-local-dir
```

$ jps

```
30259 NodeManager
29557 DataNode
29753 SecondaryNameNode
29403 NameNode
29949 ResourceManager
```

$ stop-yarn.sh
$ stop-dfs.sh

After following any one of the ways, to make sure services running:

$ hdfs dfsadmin -report // to check HDFS
$ hdfs fsck / -blocks -files -locations
$ yarn node -list // to check YARN

Step 9: running word count program

$ hdfs dfs -mkdir /input
$ vi input.txt
$ hdfs dfs -copyFromLocal input.txt /input
$ hdfs dfs -ls /input
$ yarn jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar wordcount /input/input.txt /output
$ hdfs dfs -cat /output/part-r-00000
$ hdoop fs -ls file:/// // lists out local file system root
$ hdoop fs -ls hdfs://192.168.1.20:10001/ // lists out HDFS meta-dat

To kill hanging tasks/applications

$ yarn application -list // lists currently running applications
$ yarn application -kill jobid

You can use linux commands to terminate jobs

$ jps // to see process id
$ sudo kill -9 processid // this will kill process and thus job also

Validating output

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

Step 10: access Hadoop services using the below URI

1. Web UI for Hadoop NN: http://10.100.55.92:50070/ (*Figure 6-1*)

Go to utilities → browse file system to see the input and output files.

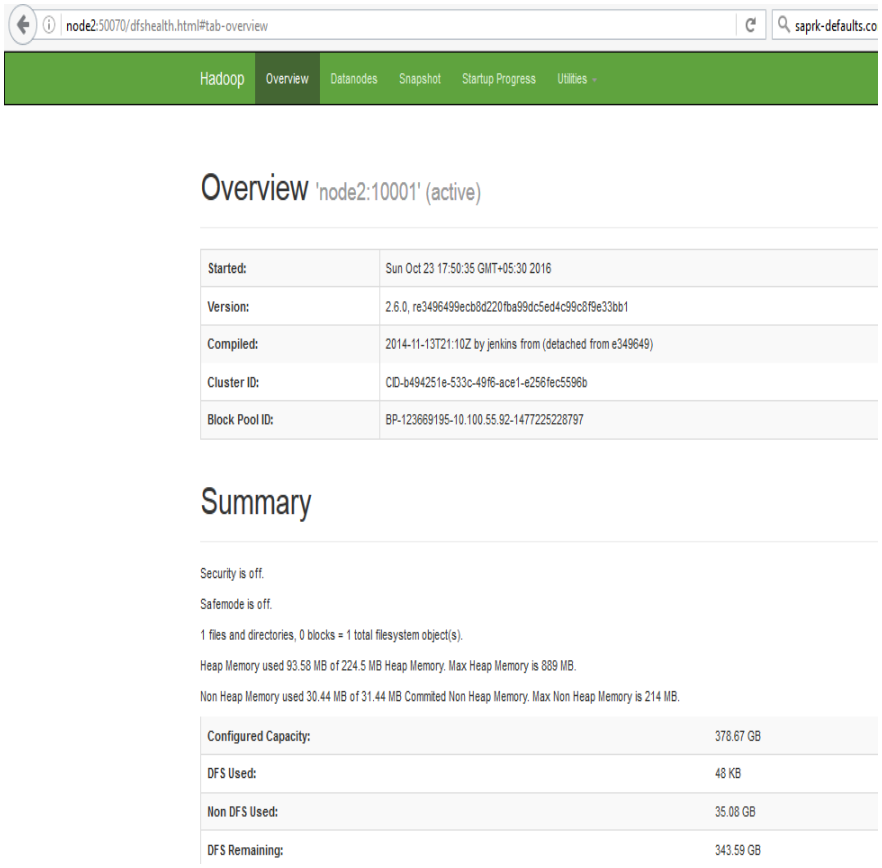

Figure 6-1: Namenode WUI

2. Web UI for application details: http://10.100.55.92:8088/ (*Figure 6-2*)

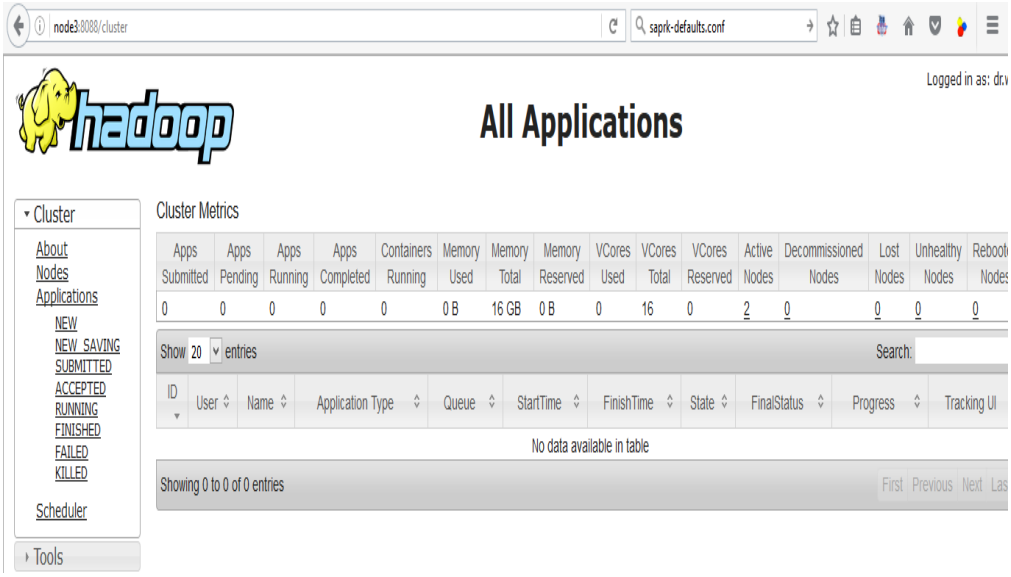

Figure 6-2 WUI for applications

6.3 Multi-node setup: cluster or fully distributed mode deployment

Follow the requirements given in section 6.1 in all nodes before beginning the installation. Consider five physical machines or VMs as given in *Table 6-1*. 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. Username must be same for all the nodes in the cluster. I assume username in my machine "**itadmin**". I represent physical machine or VM generically as node.

Table 6-1: Multi-node implementation

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

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

goto /etc/hosts for DNS setting. Therefore, you can use domain name instead of typing IP address every time. Comment 127.0.*.1 using # and enter one IP, domain name, and hostname of a node with tab space every line.

```
$ 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 password less communication

Create key in NN and send to all other nodes, and create key in RM and send to all other nodes. you need not create key in slaves. Refer Hadoop 1.x multi-node installation simultaneously.

Generate key in node1

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

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

Send keys from node1 to all other nodes in the cluster: the syntax is

```
$ 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
```

Similarly, generate key in node2 and send to all other nodes in the cluster.

Step 3 to Step 7 must be done in all nodes (masters and slaves) in the cluster

Step 3: download Hadoop2.7.0 and move into hadoop folder

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

```
$ wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.0/hadoop-2.7.0.tar.gz
$ tar -zxvf hadoop-2.7.0.tar.gz
$ sudo cp -r hadoop-2.7.0 /usr/local/hadoop
```

Step 4: configure environment variable to Hadoop

```
$ vi  .bashrc
        export HADOOP_HOME=/usr/local/hadoop
        export PATH=$PATH:$HADOOP_HOME/bin
        export PATH=$PATH:$HADOOP_HOME/sbin
        export HADOOP_HDFS_HOME=$HADOOP_HOME
        export HADOOP_MAPRED_HOME=$HADOOP_HOME
        export HADOOP_COMMON_HOME=$HADOOP_HOME
        export YARN_HOME=$HADOOP_HOME
        export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
        export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
        export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"

$ source .bashrc
$ $PATH
```

Step 5: edit Hadoop configuration file

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

$ yarn version
```

Step 6: configure Hadoop services

To setup NN

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

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

To setup YARN

$ sudo vi /usr/local/hadoop/etc/hadoop/yarn-site.xml

```
<configuration>
      <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>
            <name>yarn.resourcemanager.hostname</name>
            <value>ubuntu2</value>
      </property>
      <property>
            <name>yarn.resourcemanager.address</name>
            <value>node2:8030</value>
      </property>
      <property>
            <name>yarn.resourcemanager.resource-
      tracker.address</name>
            <value>node2:8031</value>
      </property>
      <property>
            <name>yarn.resourcemanager.scheduler.address</name>
            <value>node2:8032</value>
      </property>
      <property>
            <name>yarn.resourcemanager.admin.address</name>
            <value>node2:8033</value>
      </property>
      <property>
           <name>yarn.resourcemanager.webapp.address</name>
           <value>node2:8088</value>
      </property>
      <property>
            <name>yarn.nodemanager.disk-health-checker.min-healthy-
      disks</name>
            <value>0.0</value>
      </property>
```

```
        <property>
              <name>yarn.nodemanager.disk-health-checker.max-disk-
        utilization-per-disk-percentage</name>
              <value>98.5</value>
        </property>
</configuration>
```

To setup MR related properties

$ sudo cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml

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

```
<configuration>
   <property>
          <name>mapreduce.framework.name</name>
          <value>yarn</value>
   </property>
   <property>
          <name>mapreduce.jobhistory.address</name>
          <value>node2:10020</value>
   </property>
   <property>
          <name>mapreduce.jobhistory.webapp.address</name>
          <value>node2:19888</value>
   </property>
   <property>
          <name>mapreduce.jobhistory.intermediate-done-dir</name>
          <value>/usr/local/hadoop/tmp</value>
   </property>
   <property>
          <name>mapreduce.jobhistory.done-dir</name>
          <value>/usr/local/hadoop/tmp</value>
   </property>
</configuration>
```

To setup NN, SNN, DN configuration (optional)

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

```
<configuration>
      <property>
         <name>dfs.replication</name>
         <value>3</value>
      </property>
      <property>
            <name>dfs.http.address</name>
            <value>node1:50070</value>
       </property>
</configuration>
```

To setup SNN: it is up and running automatically with NN

To setup slave (DN and NN)

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

```
node3
node4
node5
```

Step 7: change ownership (username) and access mode

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

Provide permission for remote access

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

Step 8: format NN to create new namespace in node1

```
$ hdfs namenode -format                    // HDFS starts with root (/) file system.
```

Step 9: start HDFS/YARN Manually

In NN (node1)

```
$ start-dfs.sh
$ jps
$ hdfs dfsadmin -report                    // check storage status, and working
$ stop-dfs.sh                              // to stop HDFS
```

In RM (node2)

```
$ start-yarn.sh
$ mr-jobhistory-daemon.sh start historyserver
$ jps
$ yarn node -list                          // check YARN status, to check NN has started
$ stop-yarn.sh                             // to stop YARN
```

Step 10: running word count program

```
$ hdfs dfs -mkdir /input
$ vi input.txt
$ hdfs dfs -copyFromLocal input.txt /input
$ hdfs dfs -ls /input
$ yarn jar  /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-
examples-2.7.0.jar wordcount /input/input.txt /output
$ hdfs dfs -cat /output/part-r-00000
```

To kill hanging tasks/applications

```
$ yarn application -list
$ yarn application -kill jobid
```

You can use linux commands to kill running container

```
$ jps                                      // to see process id for running tasks
$ sudo kill -9 processid                   // this will kill task
```

To validate counters

```
$ mapred job -counter jobid counter_group counter_name

$ mapred job -counter job_1492929051122_0002 org.apache.hadoop.mapreduce.
TaskCounter REDUCE_INPUT_GROUPS
```

Access the Hadoop components using web URL.

1. Web UI for Hadoop NN: http://node1:50070/
2. Web UI for application details: http://node2:8088/
3. Web UI for job history if it is setup then http://node2:19888/
4. Web UI for NM http://node3:8042/

Lunching commands for HDFS and MR can be done from any node in the cluster as every node has the configuration of all master and slaves. Ultimately, you will get the same result. You can launch MR job anywhere from the cluster as a job client. Because, in all nodes we have set the address of RM. Once job is received it is handed over to appropriate RM.

Test with small data and large data to observe the running time that Hadoop takes. Hadoop takes more latency for small files than normal local file system processing. Because, there are lot of intermediate steps to take place. However, with large dataset you can observe that Hadoop outperforms with less latency than local processing. Moreover, latency of same job also may vary due to other processes running in the system.

6.4 Node commission/decommission from Hadoop cluster

As an administrator of a Hadoop cluster, you will need to add or remove nodes from time to time. Ex: to grow the storage and computing capacity in a cluster, you commission new nodes. Sometimes, it is necessary to decommission a node if it is failing more often or its performance is noticeably slow.

6.4.1 Commissioning nodes

It is a potential security risk to allow any machine to connect to the NN as a slave. Because, the machine may gain access to data that is not authorized to see. Such nodes are not under your control and may stop at any time potentially causing data loss. Let's add two nodes in our Hadoop cluster. New node's IP addresses are: 10.100.55.97 and 10.100.55.98

1. Send SSH key from NN and RM to the new nodes, and complete the requirements given in section 6.1. Hadoop installation steps.

2. Add nodes IP in the slaves file in both NN and RM for future cluster wide operations.

```
$ sudo vi /usr/local/hadoop/etc/hadoop/slaves
    10.100.55.97
    10.100.55.98
```

3. Start DN and NM at 10.100.55.97 and 10.100.55.98

```
$ hadoop-daemon.sh start datanode
$ yarn-daemon.sh start nodemanager
$ jps
```

4. Update NN with the new set of permitted DNs using the following command.

```
$ hdfs dfsadmin -refreshNodes          // in NN
```

5. Update the RM with the new set of permitted NMs using the following command.

```
$ yarn rmadmin -refreshNodes           // in RM
```

6. Check WUI of NN and RM to see the increment in number of nodes and resources.

7. If too many slave nodes added, then it is better to balance the cluster by moving blocks.

```
$ start-balancer.sh                    // in NN
```

6.4.2 Decommissioning nodes

Some slaves may perform very slow and you expect slaves to die shortly. So, it is better to decommission from the cluster with right procedure, so as to preserve data blocks. The way to decommission DNs is to inform the NN that you wish to take out of circulation few nodes, so that it can replicate the blocks to other DNs before the DNs are shut down.

Even if you shut down a NM that is running MR tasks, the MRAppMaster will notice the failure and reschedule the tasks on other nodes. The decommissioning process is controlled by an "exclude" file that contains list of IP address to remove. To remove nodes from the cluster:

1. Stop DN and NM services in the node you wanted to remove from cluster.

```
$ hadoop-daemon.sh stop datanode
$ yarn-daemon.sh stop nodemanager
$ sudo poweroff
```

2. Remove nodes from slaves file in NN and RM.

3. Update NN with the new set of permitted DNs.

```
$ hdfs dfsadmin -refreshNodes
```

4. Update RM with the new set of permitted NMs.

```
$ yarn rmadmin -refreshNodes
```

5. Check WUI of NN and RM to see the decrement in number of nodes and resources.

6. If too many slave nodes removed, then it is better to balance the cluster.

```
$ start-balancer.sh
```

6.5 Gateway node (client node)

In production clusters, Hadoop nodes (NN, RM, slaves…) should not be used as clients to submit job and upload big data. Because, it overloads the functionality of that particular node and causes performance down, especially, NM cannot launch more number of containers.

Therefore, we need to use one or more gateway nodes other than Hadoop nodes for load balancing the user inputs and set away activities from Hadoop nodes in production cluster. No hadoop daemons are run in Gateway node. But, it should have all Hadoop binaries of same version as Hadoop nodes contain. Gateway nodes must be network mounted, having RAID configured. Because, client should have queue to send or receive data between users and Hadoop cluster.

6.6 Hadoop Development Tools (HDT)

6.6.1 Writing MRv2 job 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. Good understanding in java utilities, java collection is required for writing good MR programs. Follow the steps given below:

1. create input.txt file.
2. load input.txt onto HDFS.
3. make sure JDK in Hadoop 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 the MR application.
9. review output.

Moving big data onto HDFS

Create a text file input.txt.

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

Upload input.txt onto HDFS.

```
$ hdfs dfs -mkdir /data
$ hdfs dfs -copyFromLocal input.txt /data
$ hdfs dfs -ls /data
```

MRv2 is backward compatible. So, MRv1 jobs also can be run on YARN. There is no much change in new MR API than old MR API. Only change you can see is programming construct of MR job.

Writing MR jobs: I used Eclipse in windows and moved jar to Hadoop cluster.

Install latest Java, download latest Eclipse on Windows. Set path to Java and start Eclipse. We are going to create a MR job in Eclipse and move to Hadoop cluster. I assume that Hadoop-2.7.0 single node or multi-node setup is up and running. Make sure JDK of Hadoop cluser and 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.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
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.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WC{

	public static class UDFmapper extends 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, Context
		context)throws IOException, InterruptedException {

			String line = value.toString();
			StringTokenizer itr = new StringTokenizer(line);
			while (itr.hasMoreTokens()) {
				word.set(itr.nextToken());
				context.write(word, one);
			}
		}
	}

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

		private IntWritable result = new IntWritable();
		public void reduce(Text key, Iterable<IntWritable> values,
		Context context) throws IOException, InterruptedException {

			int sum = 0;
			for (IntWritable val : values) {
				sum += val.get();
			}
			result.set(sum);
			context.write(key, result);
		}
	}

	public static void main(String[] args) throws Exception {

			if(args.length !=2) {
				System.err.println("Usage:<inputpath>
				<outputpath>");
				System.exit(-1);
			}
			Configuration conf = new Configuration();
			Job job = Job.getInstance(conf, "word count");
			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]));

			System.exit(job.waitForCompletion(true)? 0:1);
	}
}
```

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-2.7.0\share\hadoop\common\lib → select all jars from this file.
hadoop-2.7.0\share\hadoop\common\ → select hadoop-common-2.7.0.jar
hadoop-2.7.0\share\hadoop\hdfs\ → select hadoop-hdfs-2.7.0.jar
hadoop-2.7.0\share\hadoop\mapreduce\ → select hadoop-mapreduce-client-common-2.7.0.jar and hadoop-mapreduce-client-core-2.7.0.jar
hadoop-2.7.0\share\hadoop\yarn\ → select hadoop-yarn-client-2.7.0.jar and hadoop-yarn-common-2.7.0.jar

4. create jar file

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

```
$ yarn jar jarname.jar  driver_class_name  /location_filename   /output_directory
$ yarn jar Job.jar WC  /data/input.txt   /output
```

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

NN_IP:50070	to see all status and NN information.
RM_IP:8088	to see running job information.
RM_IP:19888	to see job history information.

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

```
$ hdfs dfs -ls /output
        _SUCCESS                    // an empty file, just says job is done
        part-r-00000                // which contains job output

$ hdfs dfs -cat /output/part-r-00000
```

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

```
$ hdfs dfs -cat /output/part-*
$ hdfs dfs -getmerge /output_dir_in_hdfs   file_name_local
$ hdfs dfs -getmerge /output  result.txt
```

Try running your MR program first with small amount of data 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.

6.6.2 Launching MR job from Eclipse (on Ubuntu) onto Hadoop using HDT

It is painful to code MR job in Eclipse, convert to jar, move into Hadoop node, and launch job. How would it be to launch MR job directly from Eclipse onto Hadoop cluster? Hadoop Development Tool (HDT) helps us to launch MR job directly from Eclipse onto Hadoop cluster. Download hadoop-eclipse-plugin jar for your Hadoop version. Assume we have single node Hadoop2.7.0 environment and use Eclipse on Ubuntu ubuntu node (can be hadoop node also).

IP: 10.100.55.92 username: itadmin

- Download latest Eclipse.
- Download hadoop-eclipse connector "hadoop-eclipse-plugin-2.6.0.jar"
- Copy hadoop-eclipse-plugin-2.6.0.jar driver to Eclipse → plugins folder.
- Launch Eclipse → create project and write WC program (see couple of pages back).
- In eclipse, goto windows → open perspective → other perspective → other → select mapreduce.
- Now, you will see MR elephant symbol as shown in *Figure 6-3*.

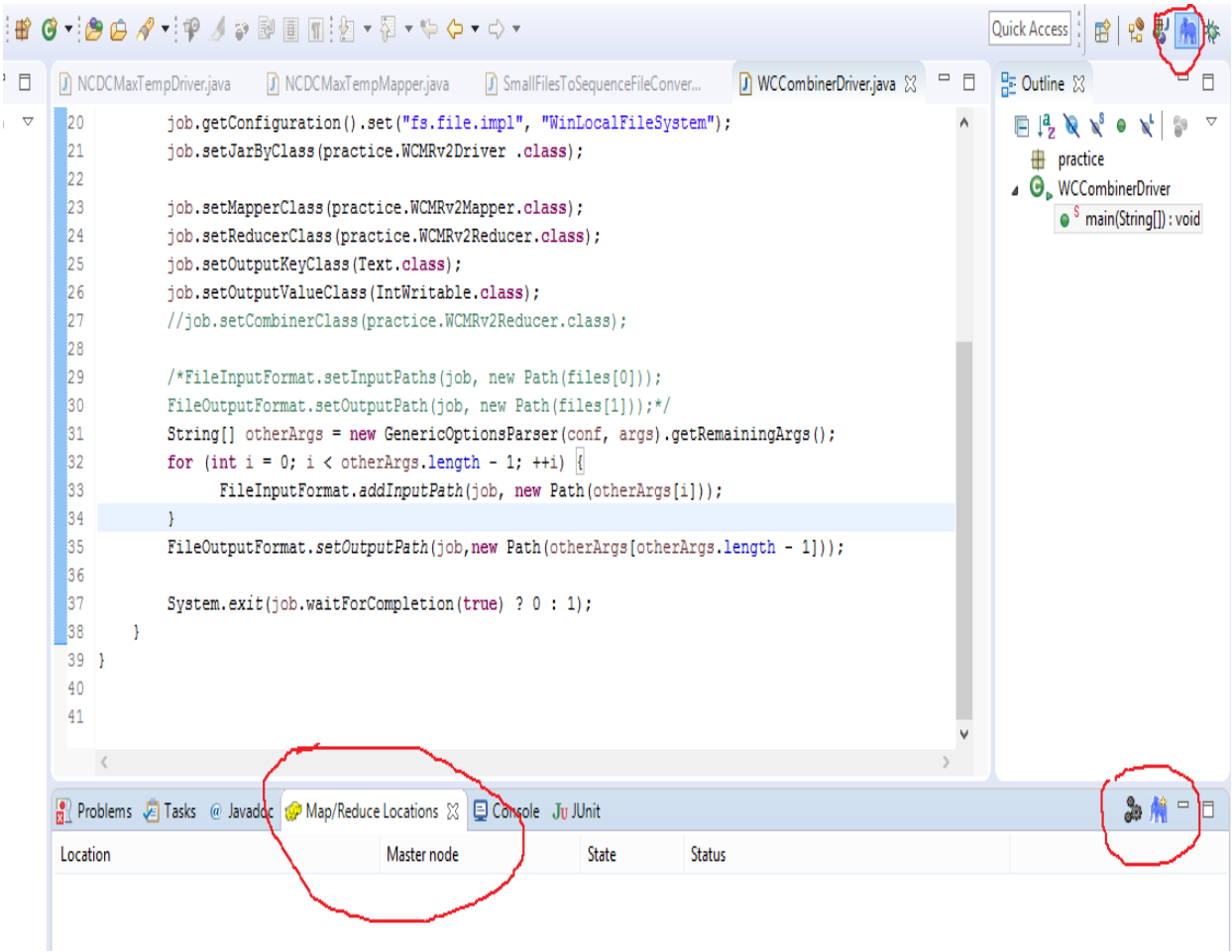

Figure 6-3: Configure HDT in Eclipse

- select MR blue logo on the right bottom corner to add our Hadoop environment. You will get as shown in *Figure 6-4*.

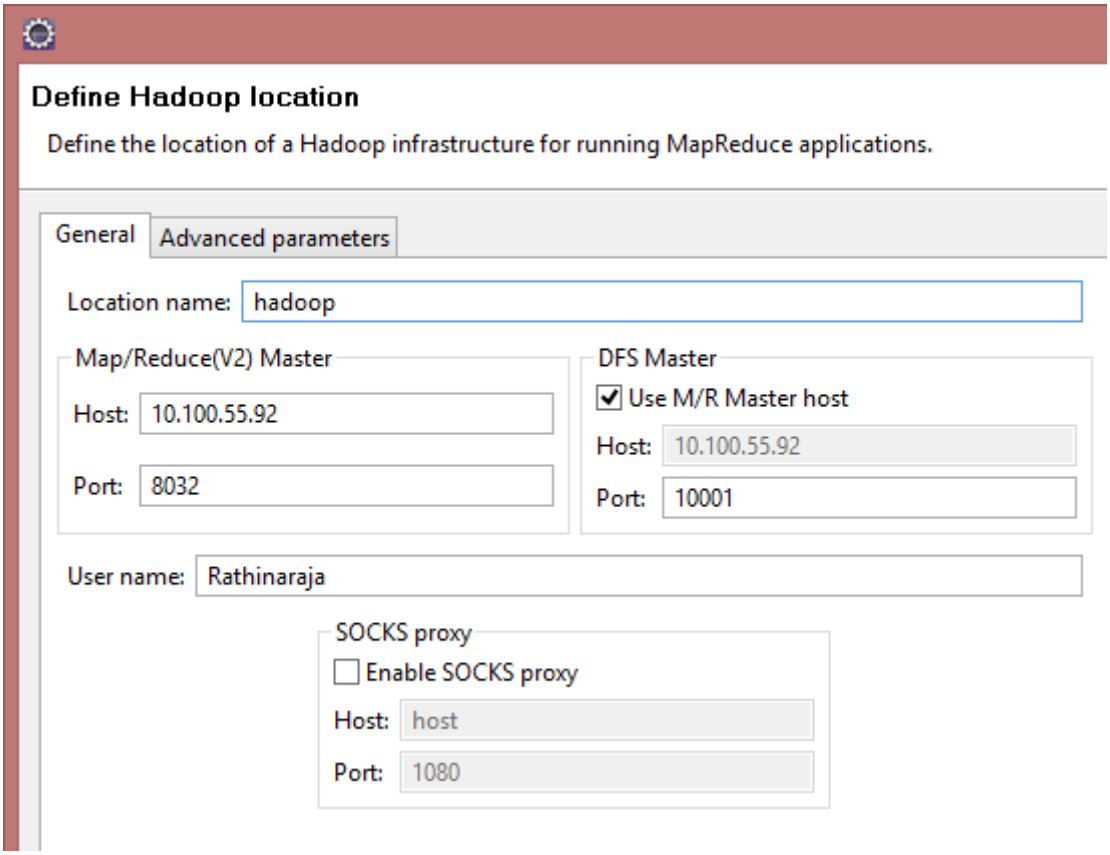

Figure 6-4: Enter NN and RM IP address and Port Number

- edit only the following fields as specified in core-site.xml and yarn-site.xml
  location name: hdfs
  MRv2 master host: 10.100.55.92 port: 8032 (default por number)
  DFS master: 10.100.55.92 port:10001 (given in core-site.xml)
- goto advanced parameters and change only the following property
  - yarn.resourcemanager.address:8032 (it is a very last property)

If Hadoop is setup on single node, then give same IP to both MRv2 and HDFS. If you have multi-node setup then type respective IP.

- Now, in Eclipse project section, you can see HDFS files are being displayed.
- In MR driver code, only change you have to include is specifying the file path with HDFS URI. Ex: assume already you have uploaded input.txt onto HDFS.

```
FileInputFormat.addInputPath(job, new
                  Path("hdfs://192.168.1.21:10001/input.txt"));
FileOutputFormat.setOutputPath(job, new
                  Path("hdfs://192.168.1.21:10001/output"));
```

- Run program.
- You can see the output file in Eclipse itself. Try refreshing or reconnecting HDFS if output files not displayed.

6.6.3 Running MR job in Eclipse itself (no Hadoop environment) on Windows or Ubuntu

Let's Execute MR job in Eclipse itself to test our MR job easier and faster (need not setup Hadoop at all). If MR job is executed in Eclipse, then it is submitted to LocalJobRunner (no HDFS and MR daemons running to handle MR job). It is very similar to setting mapred.job.tracker is "local" for Hadoop standalone mode.

MR framework itself applies MR algorithm to execute local job. So, with Eclipse you can launch MR job on small amount of input data to ensure the job runs without exception. It is a kind of unit testing. If you want check the HDFS and MR concepts, then you must at least use pseudo-distributed mode. The steps to run Hadoop 1.2.1 are:

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

Step 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 WCMRv1AllinOne 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(WCMRv1AllinOne.class);
    job.setJarByClass(WCMRv1AllinOne.class);

    job.setMapperClass(UDFmapper.class);
    job.setReducerClass(UDFreducer.class);
    job.setNumReduceTasks(0);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    //FileInputFormat.setInputPaths(job,new              Path("D:/Eclipse
Project/SamplePrograms/input.txt"));
    //FileOutputFormat.setOutputPath(job,new             Path("D:/Eclipse
Project/SamplePrograms/output"));
    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 MRv1.WCMRv1AllinOne(), args);
    System.exit(exitCode);
}
}
```

Step 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

Step 4: Passing input and output locations as arguments

When you execute MR job in Eclipse itself, you have to pass input and output locations as arguments to job as we give in command line in cluster mode while launching job. Therefore,

Goto run → run configurations → java applications → select java program → select arguments → enter input file location and output location. Ex:

"D:/Eclipse Project/SamplePrograms/input.txt" "D:/Eclipse Project/SamplePrograms/output"

Step 5: click run button. Check the output location.

This procedure just works fine in Ubuntu. But, when you run Eclipse on Windows, you have to create one more java program as follows.

Right click on MapReduce project → new → class → name: WinLocalFileSystem → write the following code

```
import java.io.IOException;
import org.apache.hadoop.fs.LocalFileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.permission.FsPermission;

public class WinLocalFileSystem extends LocalFileSystem {
	public WinLocalFileSystem() {
		super();
		System.err.println("Patch for HADOOP-7682: "+"Instantiating
		workaround file system");
	}
	@Override
	public boolean mkdirs(Path path, FsPermission permission)
	throws IOException {
		boolean result=super.mkdirs(path);
		this.setPermission(path,permission);
		return result;
	}
	@Override
	public void setPermission(Path path, FsPermission permission)
	throws IOException {
		try {
			super.setPermission(path,permission);
		}
		catch (IOException e) {
			System.err.println("Patch for HADOOP-7682: "+
			"Ignoring IOException setting persmission for path
			\""+path+ "\": "+e.getMessage());
		}
	}
}
```

Add the following line in main() of WC.java right after job object creation.

```
	JobConf job = new JobConf(WCMRv1AllinOne.class);
	job.getConfiguration().set("fs.file.impl", "WinLocalFileSystem");
```

now, run the job. It just works and check the output file location.

6.7 HDFSv2 and YARN commands

FileSystem shell: it is an interface between user and HDFS.
URI: are used to locate resources like files… All the FS shell commands take URIs as arguments.

- 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.

```
$ hdfs dfs                                  //dfs indicates we are working with HDFS
$ hdfs dfs -help                            //lists short summary of commands in fs
$ hdfs dfs -help commandName                //displays summary of specific command
$ hdfs dfs -help put
```

1. create user directory and list meta-data (username of my machine is itadmin)

In HDFSv2, we have to create home directory (/user/itadmin) manually as soon as HDFS is up and running unlike HDFSv1.

```
$ hdfs dfs -mkdir /user/itadmin
$ hdfs dfs -chown itadmin:itadmin /user/itadmin   // providing ownership
$ hdfs dfsadmin -setSpaceQuota 1t /user/itadmin // to set 1TB limit on given user dir
$ hdfs dfs -mkdir dir
$ hdfs dfs -ls
        dir
$ hdfs dfs -mkdir /dir1                     // to create directory in HDFS root
$ hdfs dfs -ls /                            // displays meta-data of hadoop root dir
        /user
        /dir1
$ hdfs dfs -ls hdfs://node1:10001/          // in case of multi node.
$ hdfs dfs -ls -R /                         // shows hidden files and sub directories.
$ hdfs dfs -ls file:///                     // lists out local file system root
```

You can create an empty file on HDFS, but can't edit any files in HDFS. You must create in local file system and send it to HDFS. Therefore, vi, gedit will not work with HDFS.

```
$ hadoop fs -touchz /file.txt                       // file size is zero
$ hadoop fs -ls /
```

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

```
$ hdfs dfs -put  local_source /hdfs_destination
$ hdfs dfs -copyFromLocal  local_source /hdfs_destination
$ hdfs dfs -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, and read directly from stdin.

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

```
$ hdfs dfs -get  /hdfs_source  local_destination/filename
$ hdfs dfs -copyToLocal  /hdfs_source  local_destination
$ hdfs dfs -moveToLocal  /hdfs_source  local_destination
```

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

```
$ hdfs dfs -cp /hdfs_source /hdfs_destination
$ hdfs dfs -mv  /hdfs_source /hdfs_destination
```

5. to display a file from HDFS

```
$ hdfs dfs -cat file:///filename
$ hdfs dfs -cat hdfs://NN_IP:port#/file1.txt   hdfs://NN_IP:port#/filename
```

6. to delete file, directory from HDFS

```
$ hdfs dfs -rm  /filename
$ hdfs dfs -rmr /directory
```

7. appendToFile (many files are concatenated while uploading onto HDFS into one file)

```
$ vi file1.txt
        hi how are you
$ vi file2.txt
        how do you do
$ hdfs dfs -appendToFile file1.txt file2.txt /filename
$ hdfs dfs -cat /filename
```

8. checksum: Returns the checksum information of a file.

```
$ hdfs dfs -checksum hdfs://NN_IP:port#/filename

$ hdfs dfs -checksum /filename
        MD5-of-0MD5-of-512CRC32C
        000002000000000000000000cfdfdc2afa26c94447e349865436d7e2
```

9. chmod

```
$ hdfs dfs -chmod 777 /filename
```

10. chown

```
$ hdfs dfs -chown username /filename
```

11. to display HDFS used and unused space in bytes.

```
$ hdfs dfs -du /
        33              /input.txt
        726663168       /largedeck.txt
        137             /new.txt
$  hdfs dfs -df /
        Filesystem              Size            Used    Available       Use%
        hdfs://node1:10001  237560135680  675840  93917169664   0%
```

12. to get file path

```
$ hdfs dfs -find / -name input.txt -print
```

13. tail displays last 1 KB from a file to stdout (console).

```
$ hdfs dfs -tail /file_name
```

14. to display top or bottom n number of lines of a file in HDFS

```
$ hdfs dfs -cat /output/part-r-00000 | head -n 5      // displays top five lines
$ hdfs dfs -cat /output/part-* | tail -n 5            // displays bottom five lines
```

15. counting number of lines and number of files in HDFS.

```
$ hdfs dfs -count -q -h /

$ hdfs dfs -ls / | wc -l              // to display number of files in a HDFS directory.
$ hdfs dfs -cat /input.txt | wc -l    // to display number of lines/records in log.txt file
```

16. text takes a source file and outputs the file in text format.

```
$ hdfs dfs -text /file_name
```

17. to list top 5 biggest/smallest size files on HDFS sizewise.

```
$ hdfs dfs -du / |sort -g -r| head -n 5
$ hdfs dfs -du / |sort -g -r| tail -n 5
```

18. other file system operations

```
$ hdfs dfsadmin -report                              // displays block report
$ hdfs fsck /hdfs_directory -files -blocks -locations
$ hdfs dfs -setrep number /filename                  // replication by default 3
```

19. mapreduce commands

```
$ yarn                                               // shows sub commands
$ yarn application
$ yarn application -list                             //lists running applications
$ yarn application -kill <application_id>
$ yarn application -list all                         // returns history of jobs
```

20. working with jars

```
$ yarn jar /usr/local/hadoop/hadoop-examples-1.2.1.jar  // displays the jobs in the jar
$ jar tvf name.jar                                      // to list what is inside jar
```

21. 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:~/
```

22. to execute commands in remote node without logging into remote node

```
$ ssh username@IP "ps -ef | grep -i namenode"
```

23. to find NN machine in the cluster

```
$ hdfs getconf -namenodes            // finds the NN's hostname from fs.default.name
$ hdfs getconf -secondarynamenodes
```

Check the following link for more commands

https://hadoop.apache.org/docs/r2.7.3/hadoop-yarn/hadoop-yarn-site/YarnCommands.html