Performance Report

Sort on Hadoop/Spark

Pavankumar Shetty
CWID-A20354961
CS-553 Spring 2016
Main Campus

INTRODUCTION

This programming assignment covers the Sort application implemented in 3 different ways: **Java (Shared Memory), Hadoop, and Spark**.
The sorting application needed to read a large file (**10GB on single node and 100GB on 16 nodes**) and sort it in place.
As part of this experiment, created the datasets, 10GB dataset and 100GB dataset using Gensort when instances were in running state rather than storing the dataset on Amazon S3. All the experiments were performed on **c3.large** instances.

(i) Shared memory sort:

Sort is an application which sorts a file-resident dataset; As per the requirement, our sort function should be able to sort datasets that are larger than memory of our instances. Sort was performed in multi-threaded environment on 1 virtual node. Time to execute the Sort application on the 10GB dataset produced with the Gensort on 1 node. The multi-threaded experiment was conducted by varying the number of threads from 1 to 8 to check the performance.

Implementation - Data Structures and Algorithms used:

External Merge Sort

It is used as the data being sorted doesn't fit into the main memory of a c3.large instance RAM size. So they should be put locally to the disk, which has slower access rate. Hence breaking down the main file as intermediate files to fit in the memory sounds to be a better approach. So I used a sort and merge-sort strategy.
Sorting phase involves reading the chunks of data, sort it, and write it out to a temporary file.

In the next phase, some small chunk of data is taken from every temporary files and put it in the memory and sorted and the least key is put to the final output file. This is repeated till every chunk of read from the temporary files.

Execution and Environment setup:

ubuntu@ip-172-31-2-233:~$
ubuntu@ip-172-31-2-233:~$ **echo "Shared Memory Sort of 10GB data"**

Shared Memory Sort of 10GB data

*** Place the java file "SharedMemoryOneThread.java" in the instance, which has java JRE and JDK installed.

*** Compile the java file using javac command,
ubuntu@ip-172-31-2-233:~$ **javac SharedMemoryOneThread.java**
ubuntu@ip-172-31-2-233:~$

*** Generate the 10GB unsorted data using the gensort command

ubuntu@ip-172-31-2-233:~$ **./64/gensort -a 100000000 10GBUnsorted**

*** Execute the java file with 2 parameters input and output filenames.

ubuntu@ip-172-31-2-233:~$ **java SharedMemoryOneThread 10GBUnsorted 10GBSorted**

*** Convert the output file into dos format to run the valsort

ubuntu@ip-172-31-2-233:~$ **unix2dos 10GBSorted**
unix2dos: converting file 10GBSorted to DOS format ...

*** Run the valsort on the output file

ubuntu@ip-172-31-2-233:~$ **./64/valsort 10GBSorted**
Records: 100000000
Checksum: 2faf0ab746e89a8
Duplicate keys: 0
SUCCESS - all records are in order

*** Get the top 10 lines of sorted data

ubuntu@ip-172-31-2-233:~$ **head -10 10GBSorted**

```
 "O!uve 00000000000000000000000001228D4 77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
 PMd32= 00000000000000000000000003440CC1 FFFFEEEE6666CCCCBBBB999933335555DDDDDDDD777788886666
 ^3CO], 0000000000000000000000000158C5C5 5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
 !&S3/]] 00000000000000000000000002145D78 8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
 !,=U#,9 00000000000000000000000019072E3 33332222FFFFBBBB0000FFFFAAAA666655553333DDDD3333CCCC
 !0f[ITd 000000000000000000000000003CAAB4B 9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
 !f6Suy2 000000000000000000000000003ABFD84 EEEE555555556666AAAA5555BBBBDDDD00001111666660000DDDD
 #%NIpq. 000000000000000000000000003B36FB9 11110000333344441111666666666AAAAAAAA00001111CCCCEEEE
 #'^cl'~ 00000000000000000000000002EDC5C8 8888AAAA11114444FFFF77773333EEEE44440000FFFF99999999
 $"-'Q)] 00000000000000000000000005F1265D CCCC6666EEEE22220000DDDDAAAA88886666BBBB00006666AAAA
```

*** Get the last 10 lines of sorted data

ubuntu@ip-172-31-2-233:~$ **tail -10 10GBSorted**

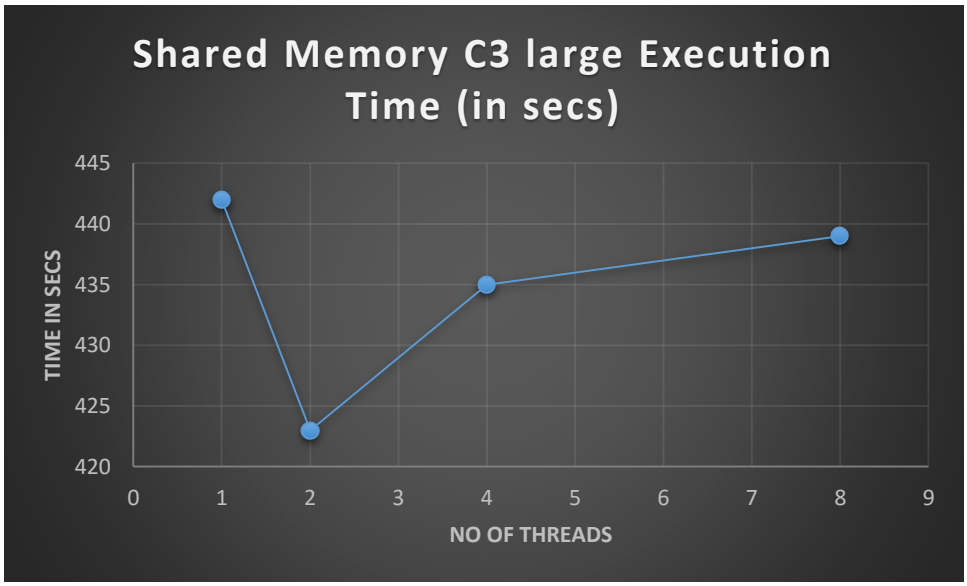
```
~~~uq2k#=U 00000000000000000000000002C06745 99991111DDDD222211110000FFFFEEEEFFFF33337777CCCC2222
~~~v/0&Qnm 00000000000000000000000004709701 CCCC88883333FFFF0000000000099991111FFFF777744446666
~~~yKOl:gE 00000000000000000000000002048B4F CCCC1111444488882222666666BBBB888855557777EEEEBBBB0000
~~~yK^H.il 00000000000000000000000000463D004 44440000FFFF3333999944447777DDDDFFFFAAAA11118888DDDD
~~~yL;C'XE 00000000000000000000000005B0D211 2222EEEE33330000222221111CCCCFFFF555577774444BBBB6666
~~~zbA_ Tt 00000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~zeO^FEg 00000000000000000000000001E06130 4444CCCCBBBB999922228888555588888CCCCFFFF000011111111
~~~}GxjWHI 0000000000000000000000000CA1345 777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC2222
~~~}P;]g0g 00000000000000000000000040DA3E4 4444FFFF444466663333EEEE88888888DDDDEEEE44442222DDDD
~~~}kU|K<p 00000000000000000000000005E4A0AA 0000666655551111BBBB88889999AAAA55550000333355557777
```

MULTI THREADING SHARED MEMORY SORT

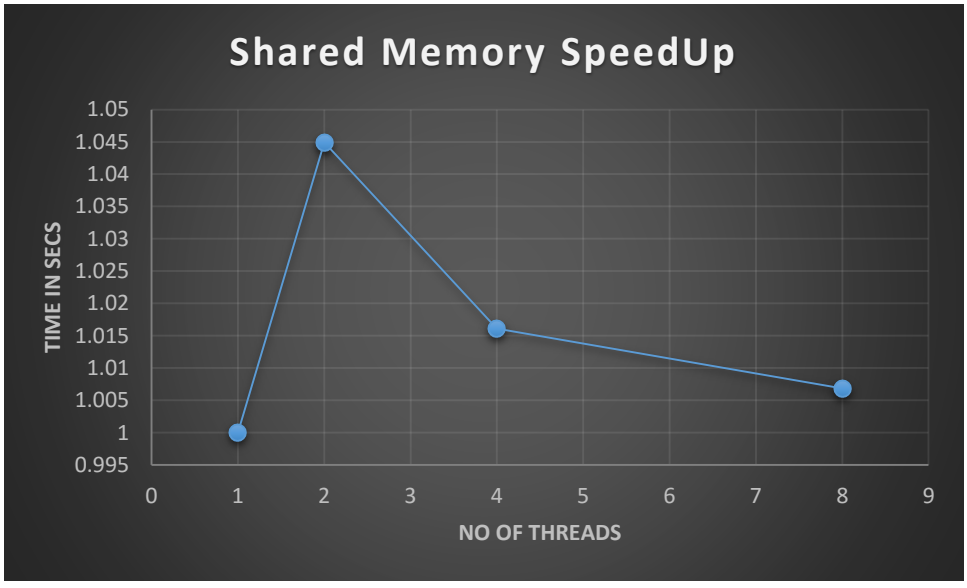
As a separate experiment, ran the shared memory with 1, 2, 4 and 8 threads on 10GB dataset to determine the performance in multi-threaded environment.

No of threads	Execution Time	Serial time Execution	SpeedUp Factor
1	442	442	1
2	423	442	1.044917258
4	435	442	1.016091954
8	439	442	1.006833713

Execution Chart:



Speedup Line Chart:



(ii) Hadoop sort:

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers. It is designed to scale up from single servers to thousands of machines (and in our experiment, we have formed a cluster of 16 nodes), each offering local computation and storage.
A MapReduce job usually splits the input data-set into independent splits which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps and are fed as input to the reduce tasks. Every job is stored in a Hadoop Distributed File-System (combines together the file systems on all data nodes to make them into one large file system). The Hadoop framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks using JobTracker and TaskTracker.

1. Single Node Sorting

MapReduce is done on the dataset on single node.
Instance type: c3.large
Hadoop version: Hadoop 2.7.2
Execution and Environment setup:

Open up the c3.large instance with java installed AMI.

Send the files, "HadoopPshetty4New.pem" file, 64 folder(which has Gensort and Valsort) and connect to it from local system and follow the following steps to setup and run the sort program on hadoop.

// add the pem file to the instance
eval `ssh-agent -s`
chmod 600 HadoopPshetty4New.pem

ssh-add HadoopPshetty4New.pem

// Create a jar for your mapper reducer sorting java file and place it in bin of hadoop folder.
// before creating jar, add this below path in ./bashrc
export HADOOP_CLASSPATH=/usr/lib/jvm/java-1.7.0-openjdk-amd64/lib/tools.jar

vim HadoopSortSingleNode.java
./hadoop com.sun.tools.javac.Main HadoopSortSingleNode.java
jar cf HadoopSortSingleNode.jar HadoopSortSingleNode*.class

// Change the configuration files in **~/hadoop-2.7.2/etc/hadoop$**

1. vi **core-site.xml**

// Add the below configuration in this file.
// fs.defaultFS - NameNode URL and port should be specified here
// hadoop.tmp.dir - HDFS will be configured in this path.

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://ec2-54-175-61-81.compute-1.amazonaws.com:9000</value>
    </property>
        <property>
        <name>hadoop.tmp.dir</name>
        <value>/mnt/raid</value>
    </property> <!-- Comment this property if not using the raid -->
</configuration>
```

2. vi **slaves**

// Add the localhost in the file, if not already added.
localhost

3. Vi **hadoop-env.sh**
// hadoop-env.sh has environment variable settings which is internally used by Hadoop.
// I have uncommented the java path and configured the java path.

 # The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

4. vi **hdfs-site.xml**
hdfs-site.xml has all the configuration or permission settings of HDFS, namenode path, and datanode paths of your local file systems.
// Made the replication factor of the data stored in HDFS as 1, to get the most performance out of the system and also to reduce the data over the instances. Data is not replicated here, as the default replication is 3 in Hadoop.
// Set the permission as false to permit all the users to use the HDFS. i.e, disabling the permission check.
// Added the namenode path and Data node path, while configuring the Multi node cluster.

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

```
    </property>
</configuration>
```

5. vi **mapred-site.xml**

// mapred-site.xml is used to specify which MapReduce framework we want to use, how many mappers and reducers we have to increase.
// Set the mappers and reducers to 4 as c3.large has 2 virtual cores.
// Mappers and reducers can also be set using command line
// **–D Mapred.reduce.task=4 and –D Mapred.map.task=4**

```
<configuration>
<property>
        <name>mapred.job.tracker</name>
        <value>hdfs://ec2-54-175-61-81.compute-
1.amazonaws.com:8021</value>
</property>

<!-- Increase the per task JVM memory of mapper and reducer, if heap
error comes in between -->
<property>
        <name>mapreduce.map.java.opts</name>
        <value>-Xmx2342m</value>
</property>
<property>
        <name>mapreduce.reduce.java.opts</name>
        <value>-Xmx4684m</value>
</property>

<!-- Increase the memory of mapper and reducer, if memory not
sufficient error comes in between -->
<property>
        <name>mapreduce.map.memory.mb</name>
```

```
        <value>2928</value>
</property>
<property>
        <name>mapreduce.reduce.memory.mb</name>
        <value>5856</value>
</property>

<!-- Set the maximum limit of mapper and reducer -->
<property>
   <name>mapred.tasktracker.map.tasks.maximum</name>
   <value>4</value>
</property>
<property>
   <name>mapred.tasktracker.reduce.tasks.maximum</name>
   <value>4</value>
</property>

<!-- Set the number of mapper and reducer to be used in the program-->
<property>
        <name>mapred.map.tasks</name>
        <value>4</value>
</property>
<property>
        <name>mapred.reduce.tasks</name>
        <value>4</value>
</property>
</configuration>
```

// Done with configuring the things.

<!-- OPTIONAL

// Used to turn off the safe mode of namenode.

./hdfs dfsadmin -safemode leave

// To format the namenode
sudo bin/hadoop namenode -format
-->

// To delete temp files
rm -Rf /tmp/hadoop-ubuntu/

// To grant permissions
sudo chmod 777 /mnt/raid/

// To format the namenode
cd ../bin/
sudo ./hdfs namenode -format

// start the dfs
cd ../sbin/
~/hadoop-2.7.2/sbin$ ./start-dfs.sh
jps

// Make a director on HDFS
cd ../bin/
~/hadoop-2.7.2/bin$ ./hdfs dfs -mkdir -p /user/pshetty4/inputfolder

// Creating RAID on disk

sudo apt-get install mdadm
sudo umount -l /mnt
sudo mdadm --create --force --verbose /dev/md0 --level=0 --
name=MY_RAID --raid-devices=3 /dev/xvdb /dev/xvdc /dev/xvdd

// Optional if mdadm says Device busy

sudo mdadm --stop /dev/md0

sudo mkfs.ext4 -L MY_RAID /dev/md0
sudo mkdir -p /mnt/raid
sudo mount LABEL=MY_RAID /mnt/raid

// Now check the mounting status using lsblk command.
lsblk

// Create 10GBUnsorted data
sudo ~/64/./gensort -a 100000000 /mnt/raid/10GBUnsorted
OR
// Without raid
sudo ~/64/./gensort -a 100000000 ~/10GBUnsorted

// PUT THE 10GBUnsorted on HDFS
~/hadoop-2.7.2/bin$ ./hadoop fs -put /mnt/raid/10GBUnsorted /user/pshetty4/inputfolder/
OR
// Without raid
~/hadoop-2.7.2/bin$ ./hadoop fs -put ~/10GBUnsorted /user/pshetty4/inputfolder/

//Remove the 10GB from raid locally
~/hadoop-2.7.2/bin$ rm -r /mnt/raid/10GBUnsorted
OR
// Without raid
~/hadoop-2.7.2/bin$ rm -r ~/10GBUnsorted

// Run the job on HDFS
~/hadoop-2.7.2/bin$ ./hadoop jar HadoopSortSingleNode.jar HadoopSortSingleNode /user/pshetty4/inputfolder 10GBSorted

// Check the file existence on HDFS
./hadoop fs -ls

// Get the sorted file to raid locally
./hadoop fs -get 10GBSorted /mnt/raid/

OR

// Without raid
./hadoop fs -get 10GBSorted ~/

// Convert it to dos to run valsort
unix2dos part-r-00000

// run the valsort on file
~/64/./valsort part-r-00000

ubuntu@ip-172-31-4-13:~/10GBSorted$ ~/64/./valsort part-r-00000
Records: 100000000
Checksum: 2faf41a78498914
Duplicate keys: 0
SUCCESS - all records are in order

// Get the first ten records
head -10 part-r-00000

ubuntu@ip-172-31-4-13:~/10GBSorted$
ubuntu@ip-172-31-4-13:~/10GBSorted$ head -10 part-r-00000

```
  "O!uve      0000000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
  PMd32=      0000000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555DDDDDDDD777788886666
  ^3CO],      00000000000000000000000000158C5C5  5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
 !&S3/]]      00000000000000000000000000002145D78  8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
 !,=U#,9      00000000000000000000000000019072E3  33332222FFFFBBBB0000FFFFAAAA666655553333DDDD3333CCCC
 !0f[ITd      000000000000000000000000003CAAB4B  9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
```

```
!f6Suy2      0000000000000000000000003ABFD84  EEEE555555556666AAAA5555BBBBDDDD0000111166660000DDDD
#%NIpq.      00000000000000000000000003B36FB9  1111000033334444111166666666AAAAAAAA00001111CCCCEEEE
#'^cl'~      000000000000000000000000002EDC5C8  8888AAAA11114444FFFF77773333EEEE44440000FFFF99999999
$"-'Q)]      00000000000000000000000005F1265D  CCCC6666EEEE22220000DDDDAAAA88886666BBBB00006666AAAA
```

// Get the last ten records

tail -10 part-r-00000

ubuntu@ip-172-31-4-13:~/10GBSorted$

ubuntu@ip-172-31-4-13:~/10GBSorted$ tail -10 part-r-00000

```
~~~uq2k#=U       000000000000000000000000002C06745  99991111DDDD222211110000FFFFEEEEFFFF33337777CCCC2222
~~~v/0&Qnm       00000000000000000000000004709701  CCCC88883333FFFF000000000000099991111FFFF777744446666
~~~yKOl:gE       000000000000000000000000002048B4F  CCCC1111444488882222666 6BBBB888855557777EEEEBBBB0000
~~~yK^H.il       00000000000000000000000000463D004  44440000FFFF3333999944447777DDDDFFFFAAAA11118888DDDD
~~~yL;C'XE       000000000000000000000000005B0D211  2222EEEE3333000022221111CCCCFFFF555577774444BBBB6666
~~~zbA_ Tt       00000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~zeO^FEg       000000000000000000000000001E06130  4444CCCCBBBB999922228888555588888CCCCFFFF000011111111
~~~}GxjWHI       0000000000000000000000000CA1345  777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC2222
~~~}P;]g0g       00000000000000000000000040DA3E4  4444FFFF444466663333EEEE88888888DDDDEEEE44442222DDDD
~~~}kU|K<p       000000000000000000000000005E4A0AA  0000666655551111BBBB88889999AAAA55550000333355557777
```

2. Multi Node Sorting

Map and Reduce is done on the dataset on single node.

Instance type: c3.large

Type and no of slave nodes: 16 & c3.large

Type and no of slave nodes: 1 & c3.large

Hadoop version: Hadoop 2.7.2

Execution and Environment setup:

Open all instances with Hadoop installed AMI's and place the public dns and private ipaddress to configure the Multi-Node Cluster

Note: Configure one datanode initially with the configuration listed below and take a AMI in amazon Ec2 console and launch 16 more instances and list all those in slaves. This will ease the work of setting up the cluster.

Change the permissions of the pem key for SSH to data nodes.

my Linux terminal$ **sudo chmod 600 ~/.ssh/HadoopPshetty4New.pem**

Now let's SSH into the one of the local ec2 machine with the following SSH command template:

my Linux terminal$ **ssh -i ~/.ssh/HadoopPshetty4New.pem ubuntu@ec2_instance_public_dns**
Setting up a config file in the ~/.ssh directory to ease of connecting to instance.
my Linux terminal$ **~/.ssh/config** file.

//Create one, if it does not exist. Config file is also submitted along with this submission. Config includes all the ipaddress of the 1 master and 16 slave instances.

// Send this config file and pem file to the namenode.

my Linux terminal$ **scp /home/ubuntu/.ssh/HadoopPshetty4New.pem ~/.ssh/config namenode:~/.ssh**

The authenticity of host 'ec2-52-91-102-135.compute-1.amazonaws.com (172.31.4.185)' can't be established.
ECDSA key fingerprint is 30:c0:45:72:05:df:8c:3e:1f:46:14:89:44:f9:2c:7d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-52-91-102-135.compute-1.amazonaws.com,172.31.4.185' (ECDSA) to the list of known hosts.
HadoopPshetty4New.pem
100% 1692 1.7KB/s 00:00

config
100% 2095 2.1KB/s 00:00

---

my Linux terminal$ **ssh namenode**

// connects to the master node.

On the NameNode, create the public/private rsa keypair using
~/.ssh/id_rsa.pub,
and add it to it's authorized_keys

namenode$ **ssh-keygen -f ~/.ssh/id_rsa -t rsa -P ""**
namenode$ **cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys**

Copy the public/private rsa keypair to each DataNode's
~/.ssh/authorized_keys
to enable the passwordless SSH capabilities from the NameNode to any DataNode.

namenode$ **cat ~/.ssh/id_rsa.pub | ssh datanode1 'cat >> ~/.ssh/authorized_keys'**
namenode$ **cat ~/.ssh/id_rsa.pub | ssh datanode2 'cat >> ~/.ssh/authorized_keys'**

Try doing ssh to datanode,
namenode$ **ssh datanode1**

% NameNode and DataNode COMMON configurations %

Open /home/ubuntu/hadoop-2.7.2/etc/hadoop for all the configuration files.

For convienence, this line "/home/ubuntu/hadoop-2.7.2/etc/hadoop " has been used as $HADOOP_CONF_DIR.

$HADOOP_CONF_DIR :- /home/ubuntu/hadoop-2.7.2/etc/hadoop

sudo vim $HADOOP_CONF_DIR/hadoop-env.sh

// Uncomment and add this below line

 # The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

sudo vim $HADOOP_CONF_DIR/core-site.xml

// change the localhost to namenode_public_dns in all datanodes

```
<configuration>
 <property>
  <name>fs.defaultFS</name>
  <value>hdfs://namenode_public_dns:9000</value>
 </property>
</configuration>
```

sudo vim $HADOOP_CONF_DIR/yarn-site.xml

// add these configuration lines in it

```
<configuration>
<!-- Site specific YARN configuration properties -->
 <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>namenode_public_dns</value>
 </property>
</configuration>
```

// copy the mapred-site.xml from the template

```
sudo cp $HADOOP_CONF_DIR/mapred-site.xml.template
$HADOOP_CONF_DIR/mapred-site.xml
sudo vim mapred-site.xml
```

```
<configuration>
 <property>
  <name>mapreduce.jobtracker.address</name>
  <value>namenode_public_dns:54311</value>
 </property>
 <property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
 </property>
</configuration>
```

% NameNode specific configurations %

// adding all the hosts ipaddresses to /etc/hosts

sudo vi /etc/hosts

ec2-52-91-102-135.compute-1.amazonaws.com ip-172.31.4.185
.
.
ec2-52-87-151-144.compute-1.amazonaws.com ip-172.31.1.99

// Open up the hdfs-site.xml
sudo vim $HADOOP_CONF_DIR/hdfs-site.xml

<property>
<name>dfs.namenode.name.dir</name>
<value>file:///home/ubuntu/hadoop2.7.2/hadoop_data/hdfs/namenode
</value>
</property>

Add a masters file to config directory and add namenode hostname to it.

sudo touch $HADOOP_CONF_DIR/masters

sudo vim $HADOOP_CONF_DIR/masters

Open up the slave file and add all the datanodes hostname to it.

sudo vim $HADOOP_CONF_DIR/slaves

%% Data Node configurations %%

// Open up the hdfs-site.xml

sudo vim $HADOOP_CONF_DIR/hdfs-site.xml

<property>
<name>dfs.namenode.name.dir</name>
<value>file:///home/ubuntu/hadoop2.7.2/hadoop_data/hdfs/datanode
</value>
</property>

%% Starting the cluster %%

Format the name node first and then launch all dfs and yarn, either separately using **start-dfs.sh** and **start-yarn.sh**

namenode$ **~/hadoop-2.7.2/bin/hdfs namenode -format**
namenode$ **~/hadoop-2.7.2/sbin/start-dfs.sh**
namenode$ **~/hadoop-2.7.2/sbin/start-yarn.sh**
OR
namenode$ **~/hadoop-2.7.2/bin/hdfs namenode -format**
namenode$ **~/hadoop-2.7.2/sbin/start-all.sh**

namenode$ **~/hadoop-2.7.2/sbin/jps**
8176 SecondaryNameNode
8623 Jps
7910 NameNode
8334 ResourceManager

Try in any of the datanode whether it is connected or not.
datanode$ **jps**
1580 NodeManager
1464 DataNode
1724 Jps

%% **Running the application** %%

// Generate 100GB unsorted data using gensort
sudo ~/64/./gensort -a 1000 ~/hadoop-2.7.2/bin/100GBUnsorted

// make your own directory in HDFS
./hdfs dfs -mkdir -p /user/pshetty4/inputfolder

// Put the data in HDFS
./hadoop fs -put 100GBUnsorted /user/pshetty4/inputfolder/

// Remove the local 100GBUnsorted file once uploaded to HDFS
rm 100GBUnsorted

// Run the jar
./hadoop jar HadoopSortSingleNode.jar HadoopSortSingleNode /user/pshetty4/inputfolder 100GBSorted

// Check the hdfs for file
./hadoop fs -ls

// get the sorted file from hdfs locally
./hadoop fs -get 100GBSorted .

// Traverse till the output file and convert the file to dos to run valsort
cd 100GBSorted/
unix2dos part-r-00000
~/64/./valsort part-r-00000

ubuntu@ip-172-31-4-185:~/hadoop-2.7.2/bin/100GBSorted$

~/64/./valsort part-r-00000

// Output

Records: 1000000000

Checksum: 1dcd7024ab4bf409

Duplicate keys: 0

SUCCESS - all records are in order

// Get the top ten sorted keys

head -10 part-r-00000

ubuntu@ip-172-31-4-185:~/hadoop-2.7.2/bin/100GBSorted$ head -10 part-r-00000

```
  !4+ABv      0000000000000000000000017F7E829  EEEE333344441111222288833334444666633332222DDDDEEEE
  "O!uve      00000000000000000000000001228D4  777788880000222244444DDDDDDDEEEE00000000CCCC7777DDDD
  %!$sU(      0000000000000000000000002E6C821C  22223333777744445555111119999CCCC4444EEEEFFFF11115555
  &5rX|X      00000000000000000000000399BC288  5555CCCCBBBB99999999DDDD111100001111EEEE7777DDDD9999
  'ic%So     000000000000000000000000031F06B7D  EEEEBBBBAAAA8888DDDDDDDD77772222444411116666444AAAA
  *0G1Io      00000000000000000000000003B5E85A1  1111AAAA9999CC CCBBBB111199991111333399991111AAAA6666
  ,(GhT_     000000000000000000000000002D0172DC  1111CCCC1111DDDDCCCCEEEE9999CCCC8888CCCCFFFF55555555
  0*vYm3      00000000000000000000000026D61578  DDDD7777AAAAEEEEEEEE6666AAAA2222CCCC5555555522229999
  2C>)8d      00000000000000000000000026C79E66  444400001111CCCC6666BBBB555577776666CCCC2222AAAABBBB
  PMd32=      000000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555DDDDDDDD777788886666
```

// Get the last ten sorted keys

tail -10 part-r-00000

ubuntu@ip-172-31-4-185:~/hadoop-2.7.2/bin/100GBSorted$ tail -10 part-r-00000

```
~~~~#iay1X      000000000000000000000000025D35EDF  6666AAAA555599997777000022223333888 8FFFF999922220000
~~~~+@p){@      00000000000000000000000085426F4  77773333555511111111000 0CCCC55559999AAAA7777DDDDDDDD
~~~~,R^_?n      00000000000000000000000001034E347  11111111999900001111 8888AAAA55554444EEEE999933338888
~~~~.Ey`^)     000000000000000000000000016F0E66B  CCCC6666DDDD2222DDDD111188889999EEEEEEEEEEEEBBBB4444
~~~~4!kA7x      00000000000000000000000001F1A1E26  EEEE777711117777BBBB1111EEEE88884444DDDDDDDDEE EEBBBB
~~~~8Ii/!@      00000000000000000000000001F05932F  11119999BBBB444477770000 11114444CCCCAAAA6666DDDD0000
~~~~<I'5>F     000000000000000000000000008CB2293  88883333BBBB1111666699998888555588888888222288 88CCCC
~~~~G-)m^)      00000000000000000000000013397F73  DDDDFFFFBBBBCCCCFFFF44446666AAAA111133333333AAAACCCC
~~~~c+I&cP      00000000000000000000000074BDF64  888800005555000DDDD22227777AAAA000033332222AAAADDDD
~~~~hb&5X*      000000000000000000000000032C0E06B  7777BBBBBBBB9999EEEEAAAAAAAA0000CCCCDD DD4444BBBB4444
```

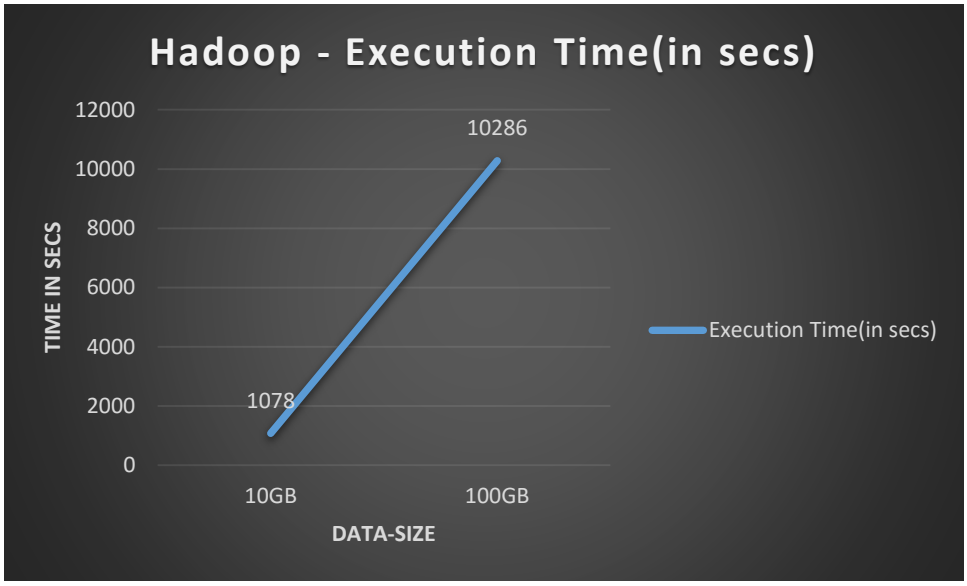
Implementation:

- Hadoop provides **Mapper** and **Reducer** interfaces and the same has been implemented as MapReduce program in java. It provides map and reduce methods. In map, it transforms the input records into intermediate splits and the output of this phase is given to input of reducer phase. In reducer phase, it reduces the splits and the data is sorted with sort and shuffle phases and the output is written to the file system.
- Followed the **environmental setup** that has been explained above. (Launching the instances, configuring it to run, making executable jar file of sort functionality)
- Created the 10GB dataset on single node and 100GB dataset on Multi node mode using **Gensort** and put it in the HDFS and ran the jar to sort the data across all the nodes.
- Once the experiment is completed, we can get the output file from the HDFS locally and perform **Valsort** to check the sort correctness.

Note: The experiment has been done again by changing the number of mappers and reducers to 4 using **mapred-site.xml** in conf directory **mapred.map.tasks** and **mapred.reduce.tasks**. Decrease in time taken to process the sorting was noticed when mappers and reducers were increased.

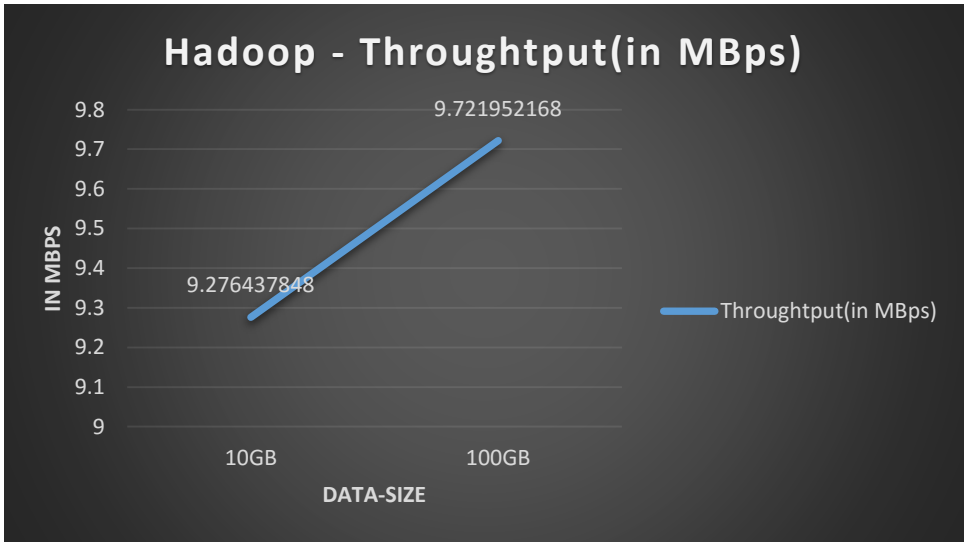
Execution Chart of Hadoop on 10GB and 100GB Dataset:

Data-size	Execution Time(in secs)
10GB	**1078**
100GB	**10286**



Throughput of Hadoop:

Data-size	Throughtput(in MBps)
10GB	**9.276437848**
100GB	**9.721952168**



HADOOP Q & A's

1) What is a Master node? What is a Slaves node?

Ans: Name node and the secondary name node in Hadoop architecture are termed as Master nodes.

Data nodes are termed as Slaves node.

- The **Name node**, the central controller of HDFS, holds all the HDFS **metadata** for the multi node cluster. It knows only what blocks makes up a file and where those blocks are located in HDFS. It has **ResourceManager** service, which is the heart of YARN, checks the scheduling of application tasks and management of the Hadoop cluster's resources.
- The **Data node,** are where Hadoop data is actually stored and where data processing takes place**.** It has **NodeManager** service, coordinates the resources for an individual slave node and reports back to the Resource Manager.

2) Why do we need to set unique available ports to those configuration files on a shared environment? What errors or side-effects will show if we use same port number for each user?

Ans: Hadoop uses a lot of ports for its internal and external communications. Usually, you can only have one application (or services) listening on a single port at one time, that's the reason why ports exist, so that it allows multiple services in Hadoop architecture to share the network without any conflicts.

Errors/Side effects:

The port is in use OR there is an instance of the service already running OR Address already in use OR Ports are not available.

3) How can we change the number of mappers and reducers from the configuration file?

We can change it using **mapred-site.xml** which is available in conf directory of Hadoop,

```
<property>
        <name>mapred.map.tasks</name>
        <value>4</value>
</property>
<property>
        <name>mapred.reduce.tasks</name>
        <value>4</value>
</property>
```

// This can be set also in command prompt using **"–D Mapred.reduce.task=4**" and **"–D Mapred.map.task=4**"

(iii) Spark sort:

The Apache Spark software library is a fast engine that allows for the distributed processing of large data sets across clusters of computers. It can run the programs up to 100x faster than Hadoop. It runs on top of Hadoop also and can access many data sources like HDFS, Cassandra, etc. It provides a powerful tool, **Spark Shell** (which runs on JVM) to analyze data and can run Scala, Python languages. It uses hadoop's HDFS and YARN to run jobs on big data. **Resilient Distributed Datasets (RDDs)** is created by referencing a dataset in HDFS.

1. Single Node Sorting

Instance type: c3.large
Hadoop version: Spark 1.6.0
Execution and Environment setup:

** Copy the Spark folder into the instance,
scp -i spark_node.pem /spark_folder ubuntu@spark_node_public_dns

** connect to the instance,
ssh spark_node_public_dns

** Add the pem file
eval `ssh-agent -s`
chmod 600 SparkPshetty4pem.pem
ssh-add SparkPshetty4pem

** Copy the **SortTheKeySingleNode.py** file to the bin.

** Using gensort, create 10GB of unsorted data,

./gensort -a 100000000 10GBUnsorted

** Submit the python file using spark-submit to run the job.
ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin$ **./spark-submit SortTheKeySingleNode.py**

** Once the job is completed, traverse till the output file convert it to dos format.
ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin$ **cd 10GBSorted/**
ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin/10GBSorted$ **unix2dos part-00000**
ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin/10GBSorted$ **~/64/valsort part-00000**
// Output
Records: 359637
Checksum: 2bf1116d3c69f
Duplicate keys: 0
SUCCESS - all records are in order

ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin$ **cd 10GBSorted/**

ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin/10GBSorted$ **unix2dos part-00297**

ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin/10GBSorted$ **~/64/valsort part-00297**

// Output

Records: 379445

Checksum: 2e4c88aa07dbf

Duplicate keys: 0

SUCCESS - all records are in order

** Finally get the first 10 and last 10 lines of the output file

ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin/10GBSorted$ **head 10 part-00000**

head: cannot open ‘10’ for reading: No such file or directory

==> part-00000 <==

```
  "O!uve  00000000000000000000000001228D4  777788880000222244444DDDDDDDDEEEE00000000CCCC7777DDDD
  PMd32=  00000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555DDDDDDDD777788886666
  ^3CO],  00000000000000000000000000158C5C5  5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
  !&S3/]]  00000000000000000000000002145D78  8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
  !,=U#,9  00000000000000000000000019072E3  33332222FFFFBBBB0000FFFFAAAA666655553333DDDD3333CCCC
  !0f[ITd  00000000000000000000000003CAAB4B  9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
  !f6Suy2  00000000000000000000000003ABFD84  EEEE555555556666AAAA5555BBBBDDDD00001111666660000DDDD
  #%NIpq.  00000000000000000000000003B36FB9  11110000333344441111666666666AAAAAAAA00001111CCCCEEEE
  #'^cI'~  00000000000000000000000002EDC5C8  8888AAAA11114444FFFF77773333EEEE44440000FFFF99999999
  $"-'Q)]  00000000000000000000000005F1265D  CCCC6666EEEE22220000DDDDAAAA88886666BBBB00006666AAAA
```

ubuntu@ip-172-31-3-205:~/spark-1.6.0-bin-hadoop2.6/bin/10GBSorted$ **tail 10 part-00000**

tail: cannot open ‘10’ for reading: No such file or directory

==> part-00297 <==

```
~~~uq2k#=U  00000000000000000000000002C06745  99991111DDDD222211110000FFFFEEEEFFFF33337777CCCC2222
~~~v/0&Qnm  00000000000000000000000004709701  CCCC88883333FFFF000000000000099991111FFFF777744446666
~~~yKOl:gE  00000000000000000000000002048B4F  CCCC1111444488882222666 6BBBB88885 5557777EEEEBBBB0000
~~~yK^H.il  00000000000000000000000000463D004  44440000FFFF3333999944447777DDDDFFFFAAAA11118888DDDD
~~~yL;C'XE  00000000000000000000000005B0D211  2222EEEE3333000022221111CCCCFFFF555577774444BBBB6666
~~~zbA_ Tt  00000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~zeO^FEg  00000000000000000000000001E06130  4444CCCCBBBB999922228888555558888CCCCFFFF000011111111
```

~~~}GxjWHI 00000000000000000000000000CA1345 777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC22 22
~~~}P;]g0g 00000000000000000000000040DA3E4 4444FFFF444466663333EEEE88888888DDDDEEEE44442222DDDD
~~~}kU|K<p 00000000000000000000000005E4A0AA 0000666655551111BBBB88889999AAAA55550000333355557777

2. Multi Node Sorting

Instance type: c3.large
Hadoop version: Spark 1.6.0
Execution and Environment setup:
*** Once single node is done, create a AMI of it without volume and open up a new instance with this AMI as it already has spark installed and open the ec2 folder of spark directory. Copy the pem file which should be used to launch cluster.

localTerminal$ scp -i SparkPshetty416Nodes.pem SparkPshetty416Nodes.pem ubuntu@ip-172-31-49-133/~

ubuntu@ip-172-31-49-133:~/spark-1.6.0-bin-hadoop2.6$ **cd spark/ec2**

*** Export the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY to the ec2 folder.

ubuntu@ip-172-31-49-133:~/spark-1.6.0-bin-hadoop2.6/ec2$ **export AWS_ACCESS_KEY_ID=******************
ubuntu@ip-172-31-49-133:~/spark-1.6.0-bin-hadoop2.6/ec2$ **export AWS_SECRET_ACCESS_KEY=****************

*** Change the mode for SSH to work between master and workers

ubuntu@ip-172-31-49-133:~/spark-1.6.0-bin-hadoop2.6/ec2$ **chmod 400 SparkPshetty416Nodes.pem**
~~~

*** Using the AMI which has spark and hadoop installed, Launch a 16 nodes cluster using "spark-ec2" command.
********** ./spark-ec2 -k <keypair>
-i <key-file> -s <num-slaves> --ami <ami-id>
launch <cluster-name>,
where <keypair> is the name of your EC2 key pair,
<key-file> is the private key file for your key pair,
<num-slaves> is the number of slave nodes to launch
<cluster-name> is the name to give to your cluster
<ami-id> name generated in Amazon EC2 webconsole. **********

ubuntu@ip-172-31-49-133:~/spark-1.6.0-bin-hadoop2.6/ec2$ **./spark-ec2 -k SparkPshetty416Nodes -i SparkPshetty416Nodes.pem -s 16 --instance-type=c3.large --region=us-east-1 --ami= ami-940ee4f4 --spot-price=0.035 --hadoop-major-version=yarn launch 16NodesSpark100GB**

*** Now connect to the master node using the pem file and with root as an user, as it is amazon AMI.
Before this copy the gensort and valsort and python sort file to the master instance.

My Linux Terminal$ **scp -i ~/SparkPshetty416Nodes.pem -r 64/ root@ec2-54-200-18-218.compute-1.amazonaws.com:~**
My Linux Terminal$ **ssh -i ~/SparkPshetty416Nodes.pem root@ ec2-54-200-18-218.compute-1.amazonaws.com**

*** Create the 100 GB unsorted file using gensort

root@ip-172-31-31-55$ **~/64/gensort -a 1000000000 100GBUnsorted**

*** Go to hadoop directory and put the unsorted file into HDFS.

root@ip-172-31-31-55~]$ **cd ephemeral-hdfs/bin/**
root@ip-172-31-31-55 bin]$ **./hadoop fs -put ~/100GBUnsorted /**

*** Submit the job using SortTheKeySingleNode.py file.

root@ip-172-31-31-55~]$ **cd ../../spark/bin/**
root@ip-172-31-31-55 bin]$ **./spark-submit ~/SortTheKeySingleNode.py**

*** Check the file existence on HDFS
root@ip-172-31-31-55~]$ **cd ../../ephemeral-hdfs/bin/**
root@ip-172-31-31-55 bin]$ **./hadoop fs –ls /**

All the output files (**744** output files) will be in the directory
/100GBSorted

*** Get the sorted file to raid locally
root@ip-172-31-31-55 bin]$ **./hadoop fs -get /100GBSorted ~/**

*** Convert it to dos to run valsort
root@ip-172-31-31-55 bin]$ **unix2dos ~/part-r-00000**
root@ip-172-31-31-55 bin]$ **unix2dos ~/part-r-00744**

*** run the valsort on file
~/64/./valsort part-r-00000

*** Get the first ten records
head -10 part-r-00000

```
!4+ABv 00000000000000000000000017F7E829 EEEE33334444111122228888333344446666633332222DDDDEEEE
"O!uve 00000000000000000000000001228D4 77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
%!$sU( 0000000000000000000000002E6C821C 22223333777744445555111119999CCCC4444EEEEFFFF11115555
&5rX|X 000000000000000000000000399BC288 5555CCCCBBBB99999999DDDD111100001111EEEE7777DDDD9999
'ic%So 00000000000000000000000031F06B7D EEEEBBBBAAAA8888DDDDDDDD77772222444411116666444AAA A
*0G1Io 00000000000000000000000003B5E85A1 1111AAAA9999CCCCBBBB11119999111133339999111 1AAAA6666
,(GhT_ 00000000000000000000000002D0172DC 1111CCCC1111DDDDCCCCEEEE9999CCCC8888CCCCFFFF55555555
```

```
0*vYm3  00000000000000000000000026D61578  DDDD7777AAAAEEEEEEEE6666AAAA2222CCCC5555555522229999
2C>)8d  00000000000000000000000026C79E66  444400001111CCCC6666BBBB555577776666CCCC2222AAAABBBB
PMd32=  00000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555DDDDDDDD777788886666
```

// Get the last ten records

tail -10 part-r-00744

```
~~~~#iay1X  00000000000000000000000025D35EDF  6666AAAA555599997777000022223333888 8FFFF999922220000
~~~~+@p){@  00000000000000000000000085426F4  77773333555511111111000 0CCCC55559999AAAA7777DDDDDDDD
~~~~,R^_?n  00000000000000000000000001034E347  11111111999900001111888 8AAAA55554444EEEE999933338888
~~~~.Ey`^)  00000000000000000000000016F0E66B  CCCC6666DDDD2222DDDD111188889999EEEEEEEEEEEEBBBB4444
~~~~4!kA7x  00000000000000000000000001F1A1E26  EEEE777711117777BBBB1111EEEE88884444DDDDDDDDEEEEBBBB
~~~~8Ii/!@  00000000000000000000000001F05932F  11119999BBBB444477770000111144444CCCCAAAA6666DDDD0000
~~~~<I'5>F  00000000000000000000000008CB2293  88883333BBBB1111666699998888555588888888222288888CCCC
~~~~G-)m^)  00000000000000000000000013397F73  DDDDFFFFBBBBCCCCFFFF44446666AAAA111133333333AAAACCCC
~~~~c+I&cP  00000000000000000000000074BDF64  8888000055550000DDDD22227777AAAA000033332222AAAADDDD
~~~~hb&5X*  00000000000000000000000032C0E06B  7777BBBBBBBB9999EEEEAAAAAAAA0000CCCCDDDD4444BBBB4444
```

*** TO stop the cluster after the job is run use destroy.

./spark-ec2 destroy 16NodesSpark100GB

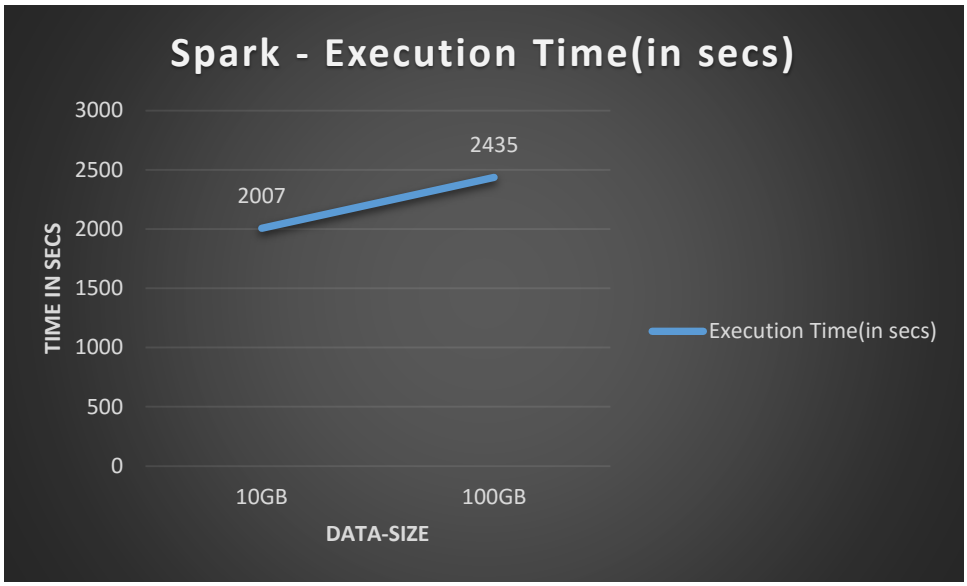
Implementation:

- Python library has been used to implement.
- Followed the environmental setup that has been explained above. (Launching the instances/Clusters with proper Disk size, connecting it to master node)
- Created the 10GB dataset on single node and 100GB dataset on Multi node mode using Gensort and put it in the HDFS and ran the "SortTheKeySingleNode" python file to sort the data across all the nodes. The input is divided into individual key value RDD pairs with first 10 bytes as key and the next 90 bytes as value. Used **map()** do the same. And then with this result, used **sortbykey()** function to sort the data and using **saveAsTextFile()** function stored it to the HDFS.
- Once the experiment is completed, we can get the output file from the HDFS locally and perform Valsort to check the sort correctness.
- I have used c3.large instance for this and made major changes of SPARK_EXECUTOR_INSTANCES to 4, SPARK_EXECUTOR_MEMORY to 2GB, SPARK_WORKER_MEMORY to 2,

SPARK_EXECUTOR_INSTANCES to 4 in **spark-env.sh**, to get the best performance out of the run.

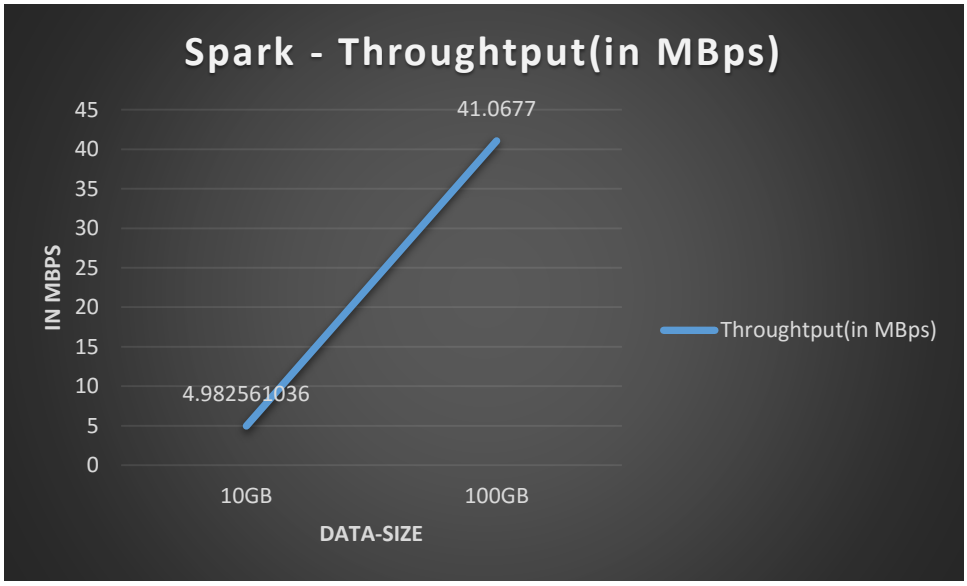
Execution Chart:

Data-size	Execution Time(in secs)
10GB	**2007**
100GB	**2435**



Throughput Chart:

Data-size	Throughtput(in MBps)
10GB	**4.982561036**
100GB	**41.0677**



Performance related Observations

1. Compare the performance of the three versions of Sort (Shared-Memory, Hadoop, and Spark) on 1 node scale and explain your observations. Compare the Shared-Memory performance of 1 node to Hadoop and Spark Sort at 16 node scales and explain your observations.

Answer:

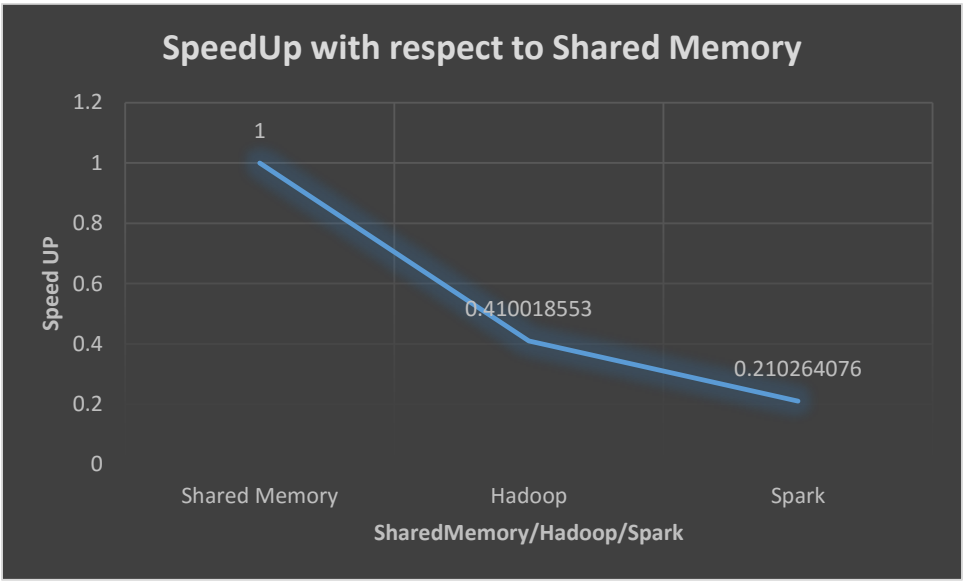
Time taken chart of all 3 sort(in secs):

Time taken to sort in secs

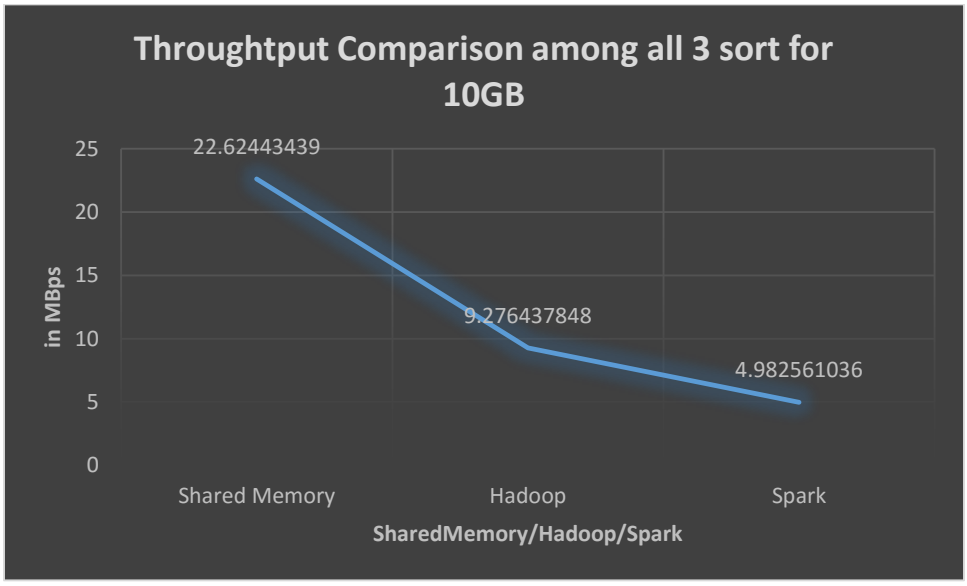
Data-Size	Shared Memory	Hadoop	Spark
10GB	422	1078	2007
100GB	NA	10286	2435

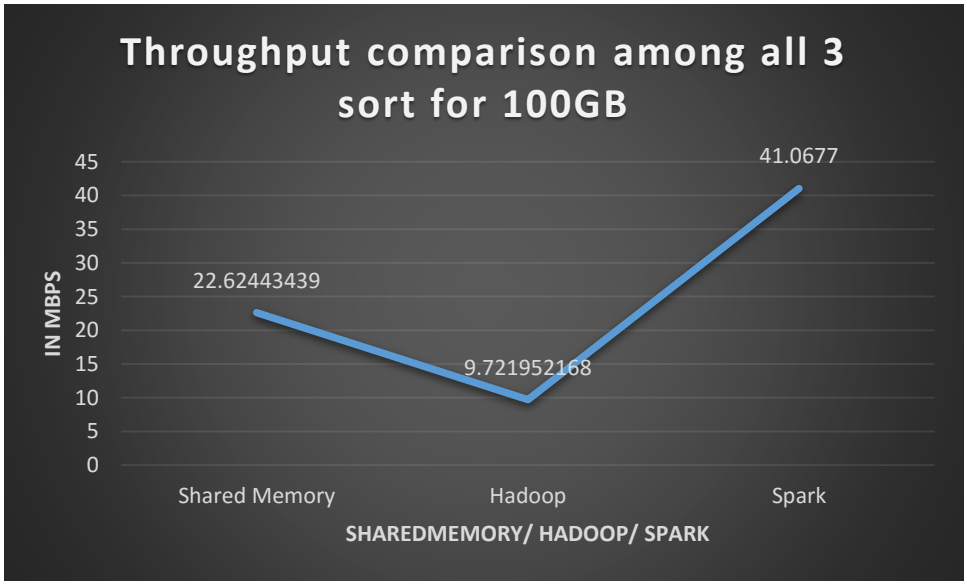
Speedup chart of Hadoop and Spark with respect to Shared Memory, 10GB Dataset:

FrameWork	SpeedUp with respect to Shared Memory
Shared Memory	1
Hadoop	**0.410018553**
Spark	**0.210264076**



Throughput comparison among all 3 sort versions for 10 GB and 100GB:





From the above graphs and the value, we can infer that SharedMemory using External Sort performs well compared to Hadoop and Spark. But when taken into consideration about handling big data, Hadoop performs better on single node than Spark.

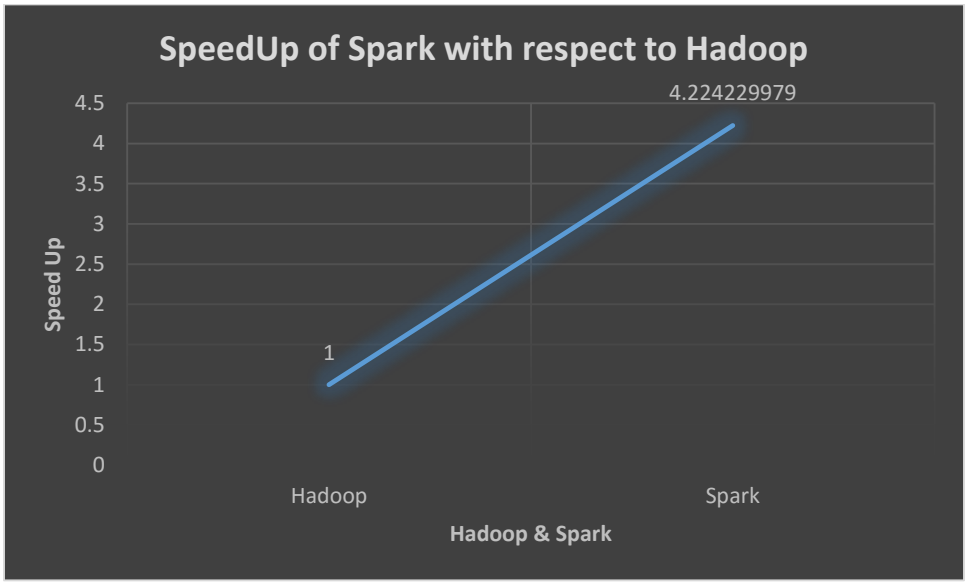
2. Draw an execution line chart and a speed up line chart for 1 node and 16 node cases, for Java, Hadoop, and Spark Sort. For Spark and Hadoop, compute two different speedups, using different base cases; one speedup (speedup-shared-memory) should be relative to the Shared-memory Sort performance (note that you might get a speedup less than 1); the second speedup (speedup-spark & speedup-hadoop) should be relative to the Spark or Hadoop performance at 1 node scale respectively (should be a number greater than 1). **What conclusions can you draw**? Which seems to be **best at 1 node scale**? How about **16 nodes**? Can you predict which would be best at **100 node scale**? How about **1000 node scales**?

Answer:

Speedup chart of Spark with respect to Hadoop, 100GB Dataset:

Hadoop in secs	Spark in secs
10286	2435

FrameWork	SpeedUp of Spark with respect to Hadoop
Hadoop	1
Spark	4.224229979



Best at one node scale: Hadoop

Best at 16 node scale: Scala

Best at 100 node scale: Scala

Best at 100 node scale: Scala

References:

http://www.dummies.com/how-to/content/master-nodes-in-hadoop-clusters.html
http://www.dummies.com/how-to/content/slave-nodes-in-hadoop-clusters.html
http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/
http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds
https://asadrazajaffree.wordpress.com/2014/05/14/multi-node-hadoop-cluster/

https://cloudcelebrity.wordpress.com/2013/08/14/12-key-steps-to-keep-your-hadoop-cluster-running-strong-and-performing-optimum/
http://www.ashishsharma.me/2011/08/external-merge-sort.html
http://lemire.me/blog/2010/04/06/external-memory-sorting-in-java-the-first-release/
http://spark.apache.org/docs/latest/index.html
https://hadoop.apache.org/docs/r1.2.1/single_node_setup.html
http://spark.apache.org/docs/latest/programming-guide.html
http://hadoop.apache.org/