CS 553 Cloud Computing
Programming Assignment 2
Final Report

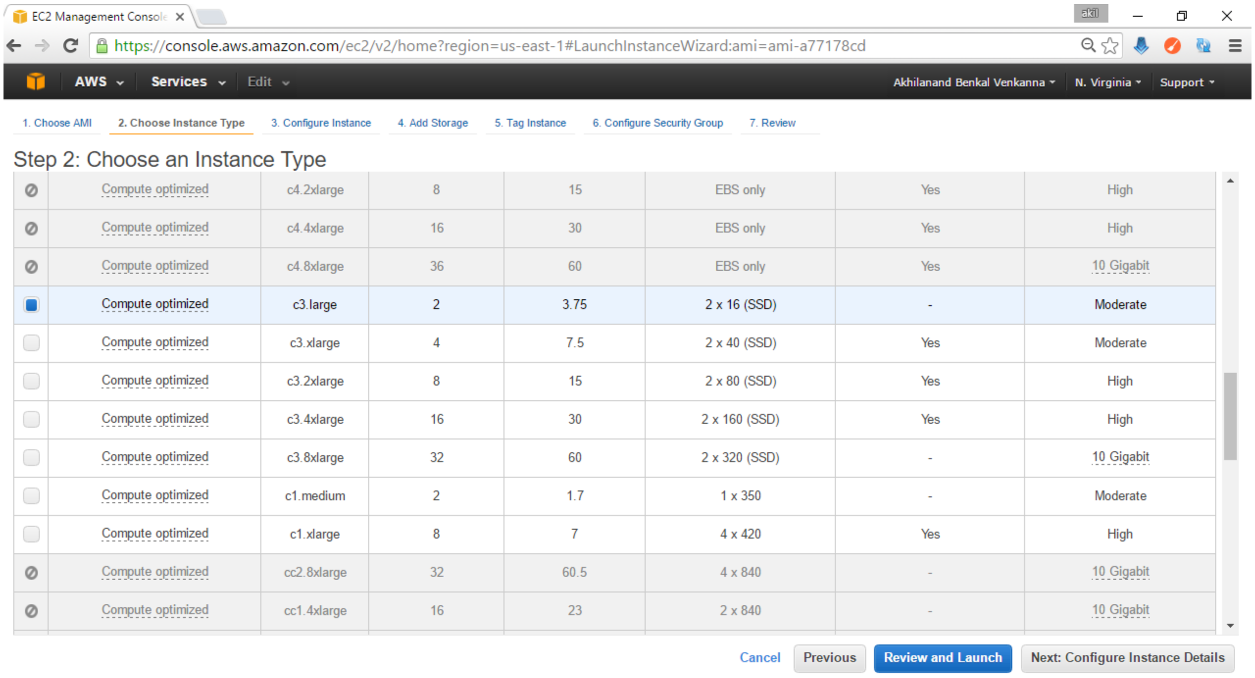
Akhilanand BV(A20358137)

The main aspects of this Assignment are:

- Setting up a virtual cluster of 1 and 16 nodes using C3 large instances of AMAZON WEB SERVICES(AWS)
- Performing Shared Memory Sort(Datasets greater than RAM size)
- Implement the Hadoop Sort application
- Implement the Spark Sort application
- Measuring the performance of the above 3 methods.

Creating a Virtual Cluster using AMAZON WEB SERVICES(AWS):

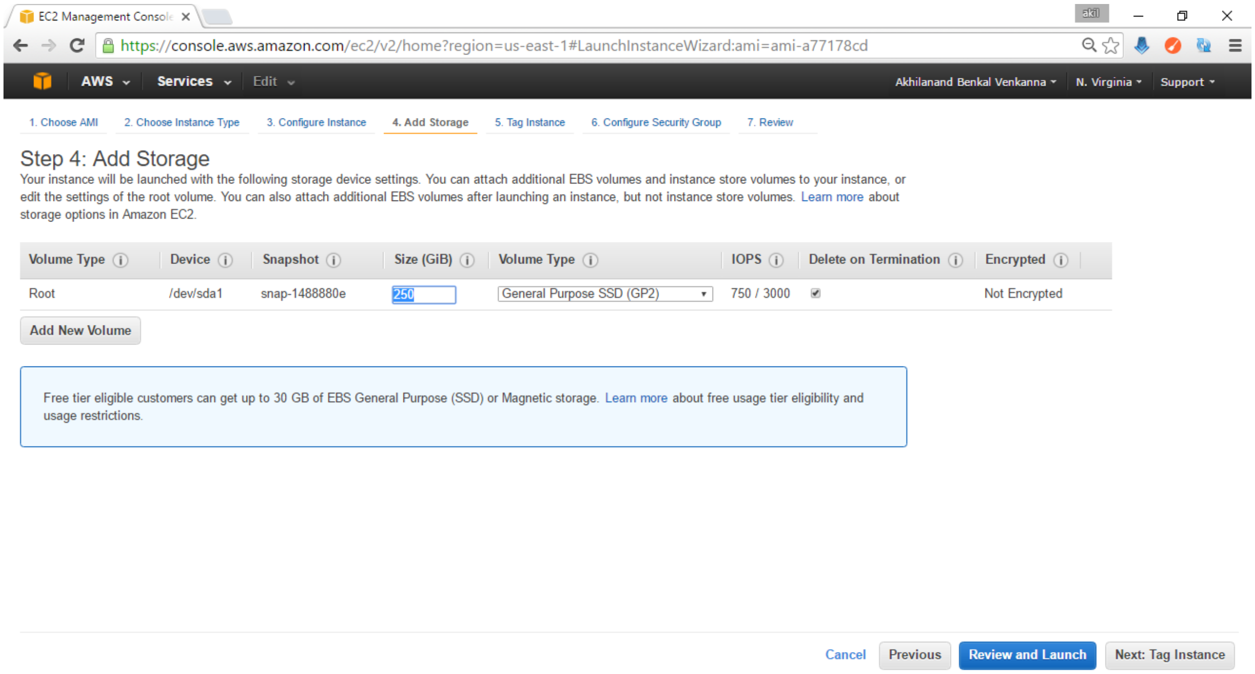
- Login to https://aws.amazon.com and select EC2 instances from the Network and services section.
- We select launch instance .



Step 2: Choose an Instance Type

⊘	Compute optimized	c4.2xlarge	8	15	EBS only	Yes	High
⊘	Compute optimized	c4.4xlarge	16	30	EBS only	Yes	High
⊘	Compute optimized	c4.8xlarge	36	60	EBS only	Yes	10 Gigabit
■	Compute optimized	c3.large	2	3.75	2 x 16 (SSD)	-	Moderate
□	Compute optimized	c3.xlarge	4	7.5	2 x 40 (SSD)	Yes	Moderate
□	Compute optimized	c3.2xlarge	8	15	2 x 80 (SSD)	Yes	High
□	Compute optimized	c3.4xlarge	16	30	2 x 160 (SSD)	Yes	High
□	Compute optimized	c3.8xlarge	32	60	2 x 320 (SSD)	-	10 Gigabit
□	Compute optimized	c1.medium	2	1.7	1 x 350	-	Moderate
□	Compute optimized	c1.xlarge	8	7	4 x 420	Yes	High
⊘	Compute optimized	cc2.8xlarge	32	60.5	4 x 840	-	10 Gigabit
⊘	Compute optimized	cc1.4xlarge	16	23	2 x 840	-	10 Gigabit

- For our experiment we have used C3.Large instances but there will be a pool of different type of instances which we can choose from as per our project requirements .
- We can take Spot instances or OnDemand instances , for our experiment we have selected Spot instances.

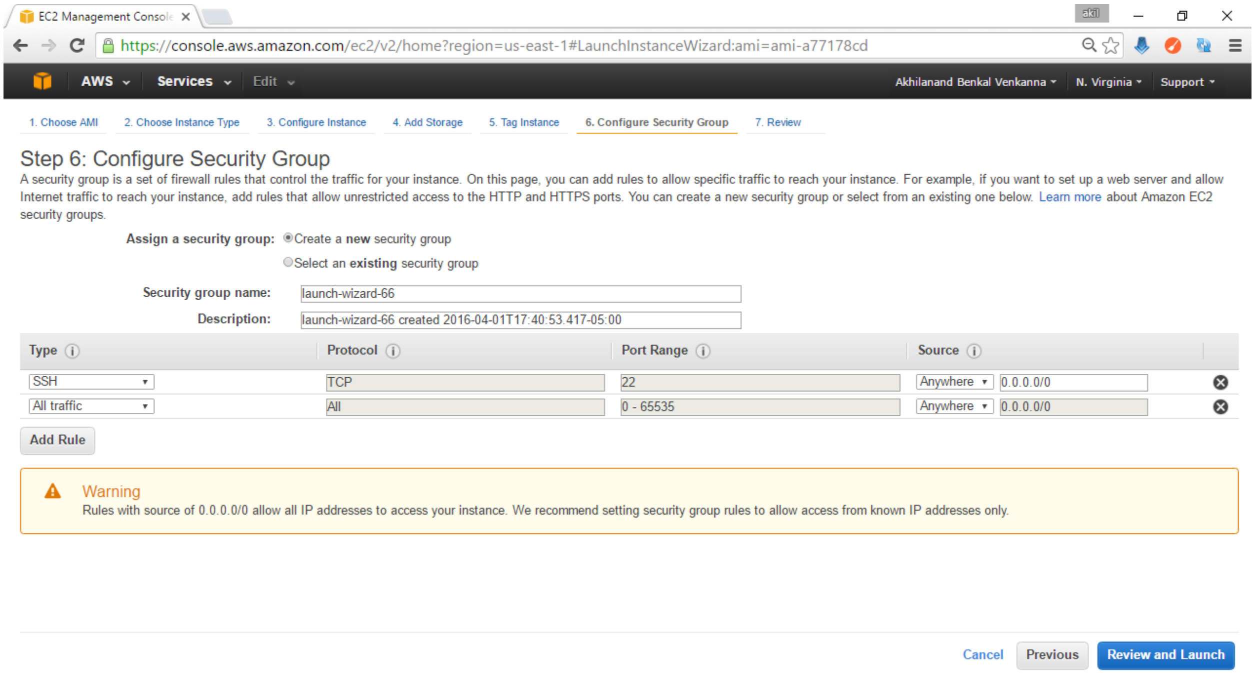


Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/sda1	snap-1488880e	250	General Purpose SSD (GP2)	750 / 3000	☑	Not Encrypted

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. Learn more about free usage tier eligibility and usage restrictions.



Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

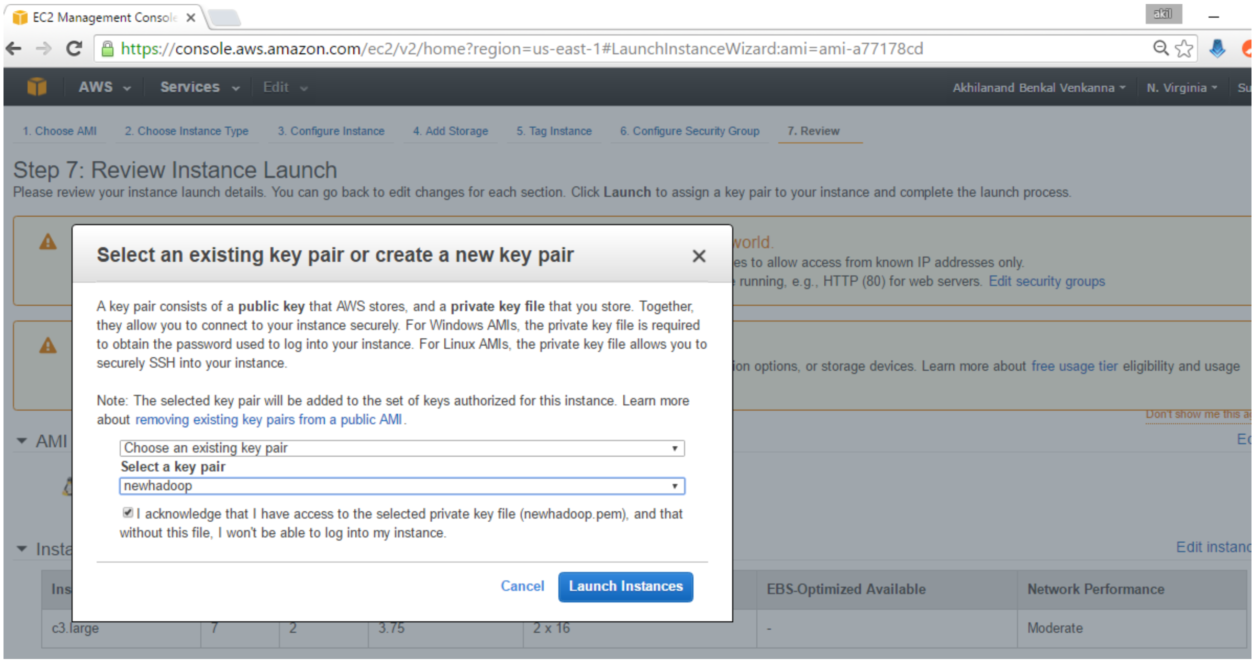
Assign a security group: ● Create a **new** security group
○ Select an **existing** security group

Security group name: launch-wizard-66

Description: launch-wizard-66 created 2016-04-01T17:40:53.417-05:00

Type	Protocol	Port Range	Source	
SSH	TCP	22	Anywhere 0.0.0.0/0	⊗
All traffic	All	0 - 65535	Anywhere 0.0.0.0/0	⊗

⚠ Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.


EC2 Management Console
https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:ami=ami-a77178cd
AWS
Services
Edit
Akhilanand Benkal Venkanna
N. Virginia
1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Tag Instance
6. Configure Security Group
7. Review

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

world.
es to allow access from known IP addresses only.
running, e.g., HTTP (80) for web servers. Edit security groups

ion options, or storage devices. Learn more about free usage tier eligibility and usage

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

Choose an existing key pair

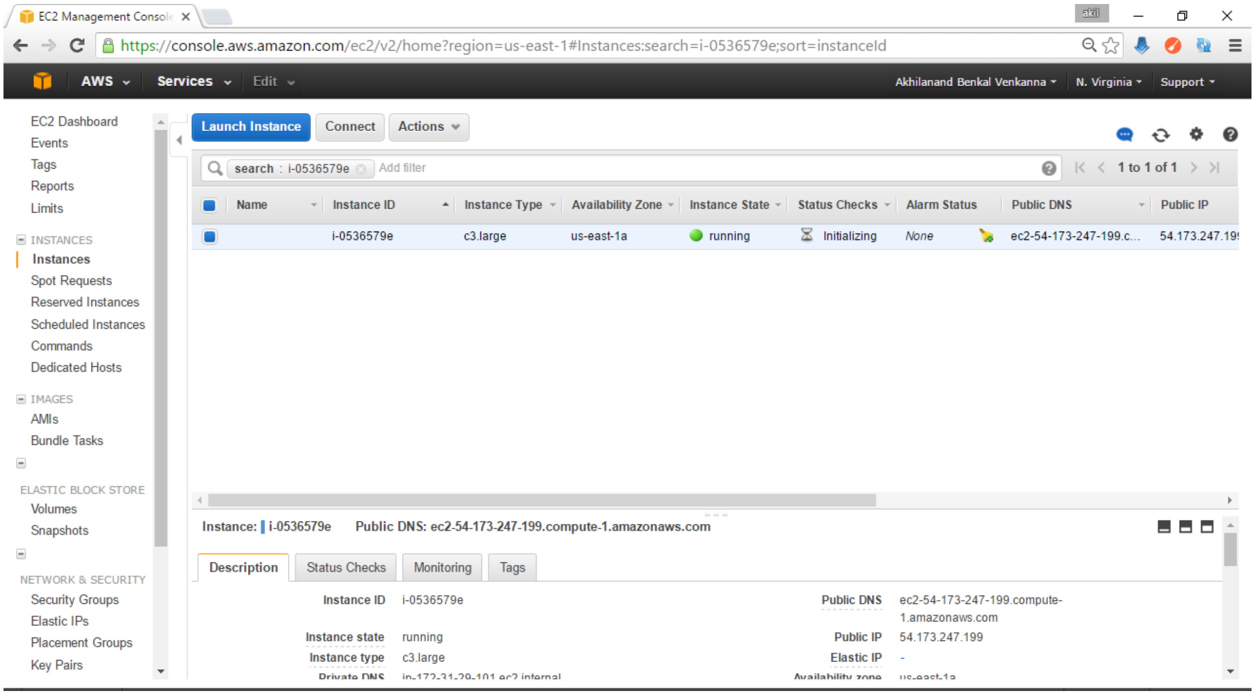
Select a key pair

newhadoop

☑ I acknowledge that I have access to the selected private key file (newhadoop.pem), and that without this file, I won't be able to log into my instance.

Ins					EBS-Optimized Available	Network Performance
c3.large	7	2	3.75	2 x 16	-	Moderate

- We add the Storage requirements next and later configure the security groups .
- We add All TCP, All UDP and All ICMP for our experiments.
- Once the cluster launches we can access our instances by giving the Public DNS IP available in the website.



search : i-0536579e

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP
	i-0536579e	c3.large	us-east-1a	running	Initializing	*None*	ec2-54-173-247-199.c...	54.173.247.19

Instance: i-0536579e **Public DNS: ec2-54-173-247-199.compute-1.amazonaws.com**

Instance ID	i-0536579e	**Public DNS**	ec2-54-173-247-199.compute-1.amazonaws.com
Instance state	running	**Public IP**	54.173.247.199
Instance type	c3.large	**Elastic IP**	-
Private DNS	ip-172-31-29-101.ec2.internal	**Availability zone**	us-east-1a

Shared-Memory Sort:

System Configuration:

JAVA VERSION:1.7

INSTANCE TYPE: c3. Large (10GB) UBUNTU

vCPU=2

Memory=3.75GB

SSD Storage=2*16GB

We have implemented the External Sort algorithm in JAVA to achieve the shared memory sort. External Sort works well for the data being sorted does not fit in the main memory ie RAM and run on the slower external memory.

Algorithm:

We split the input data into multiple blocks which would fit in the internal memory .We then sort the smaller chunks by using any sorting algorithm O(n log n), I have used Merge Sort and perform in-memory sort and write it back to disk. Once all the chunks are sorted , we then sort these locally sorted chunks and write to a larger sorted block on the HDD or SSD. Generally all of the above steps are performed in multiple phases .

We perform Multiway merge in our program for the Merge phase of our program.We utilize priority queues to find the smallest element and store them into main memory in a heap tree.

For our experiment, we split the 100GB data into 200 chunks to perform the Shared memory sort.

Generation of Data:

We have used ***gensort*** provided by Ordinal.com. Gensort program can be used to generate input records of given size and of particular type which we need by setting the parameters while running the gensort command.

```
Syntax:
gensort [-a] [-c] [-bSTARTING_REC_NUM] [-tN] NUM_RECS FILE_NAME[,opts]
```

-a : This will generate ascii records , without this flag binary records will be generated that contain the highest density of randomness in the 10-byte key.

-c : It will calculate the sum of the crc32 checksums of each of the generated records and send it to standard error.

-bN : It will set the beginning record generated to N and by default the first record generated is record 0.

-s: It will generate input records with skewed keys and if it used with -a option, then skewed ascii records are generated.

-tN : We can use this to make use of N internal program threads to generate the records.

For our experiment:

```
$cd
$wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz
$tar xvf gensort-linux-1.5.tar.gz
$cd 64/
$chmod 777 gensort
$chmod 777 valsort
$./gensort -a 1000000000 100gbinput.txt //For generation of 100gb input
$./gensort -a 100000000 10gbinput.txt//For generation of 10gb input
```

Once the input file is generated we then run our java code

```
$javac SharedMemorySort.java
```

```
$java SharedMemortSort
```

Once our output file is generated , we should perform Valsort on it to validate that our Data has been sorted properly.

We need to convert our generated file to DOS format as valsort works on DOS environment.

For this,

```
$sudo yum install unix2dos
```

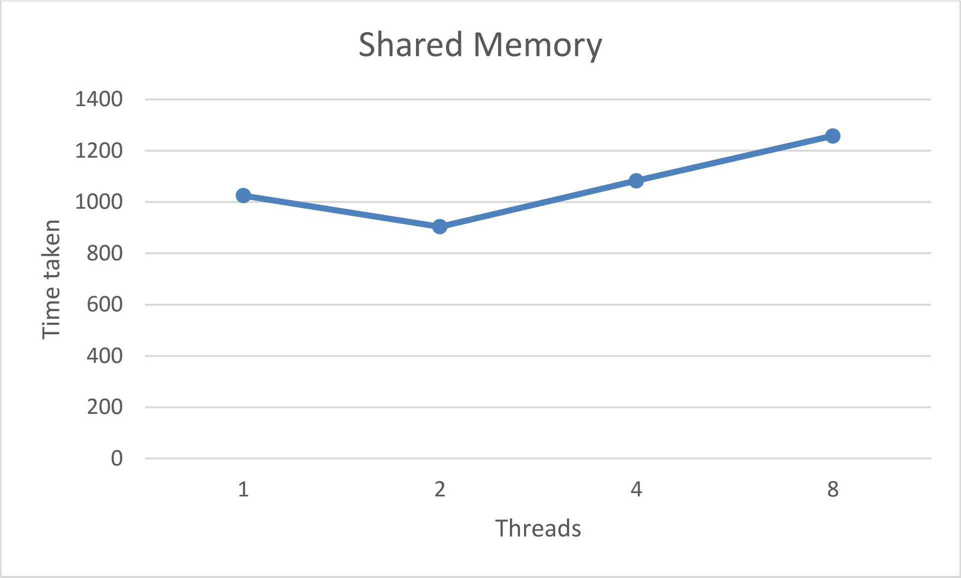
```
$sudo unix2dos sharedmemoryoutput
```

```
$./valsort sharedmemoryoutput
```

OUTPUT:

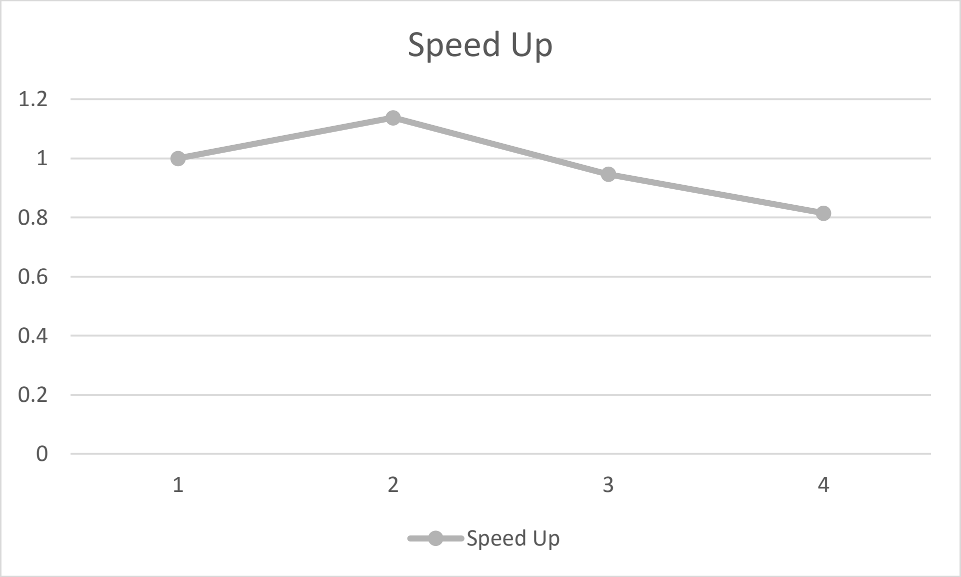
10 GB DATASET ON c3.large instance:

Number of Threads	Execution Time
1	1025
2	904
4	1083
8	1258



Speed Up

Number of Threads	Execution Time	Speed Up
1	1025	1
2	904	1.138
4	1083	0.9464
8	1258	0.8147



Inference :

After performing the experiments on 1, 2 , 4 and 8 threads , we observe that we get best performance on 2 threads. But thetime taken increases for 1 threads and hinders the performance . We get almost similar performance for 4 and 8 threads.

Setup virtual cluster of 16 nodes on Amazon EC2

We first ,Log in into the Amazon AWS service website (http://aws.amazon.com/).
Select EC2 from the network and services section.
We then choose the instance type for our assignment it is c3.large
Next step is to configure the instance type either we can choose the instance a spot instance or on Demand instance. To get the instances in a lesser price we have chosen spot instances.

To connect to a node:

Since mine is a windows machine , I have used putty and WinSCP to connect and work around the instances .

Putty Connection and WinSCP:

We need to provide the IP address of our instances and give the private key file(.ppk) to login.

Once we are connected:

- We have to install java on the virtual node.
- $sudo apt-get install openjdk-7-jreheadless
- Install javac
- $sudo apt-get install openjdk-7-jdk
- To check the version of JAVA installed use,
  $java –version.

Once our Master Node is setup , we can take the image of this and launch our cluster of 16 slave nodes

Hadoop:

System Configuration:

JAVA VERSION:1.7

INSTANCE TYPE: c3. Large (10GB) UBUNTU

vCPU=2

Memory=3.75GB

SSD Storage=2*16GB

Installation:

We first ,Log in into the Amazon AWS service website (http://aws.amazon.com/).
Select EC2 from the network and services section.
We then choose the instance type for our assignment it is c3.large
Next step is to configure the instance type either we can choose the instance a spot instance or on Demand instance. To get the instances in a lesser price we have chosen spot instances.

To connect to a node:

Since I am working on windows machine , I have used putty and WinSCP to connect and work around the instances .

Putty Connection and WinSCP:

We need to provide the IP address of our instances and give the private key file(.ppk) to login.

Once we are connected:

- We have to install java on the virtual node.
- $sudo apt-get install openjdk-7-jreheadless
- Install javac
- $sudo apt-get install openjdk-7-jdk
- To check the version of JAVA installed use,
  $java –version.

To install Hadoop

- We need do download the Hadoop tar file from the Apache website
- For this, $Wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.4.1/hadoop-2.4.1.tar.gz
- $mkdir Hadoop
- We have unzip it $ tar xvzf hadoop-2.5.1.tar.gz

We need to configure few things, for our cluster to run properly

1. ~/.bashrc
2. Hadoop/conf/Hadoop-env.sh
3. Hadoop / conf /core-site.xml
4. Hadoop / conf /yarn-site.xml
5. Hadoop / conf /hdfs-site.xml
6. Hadoop / conf /mapred-site.xml.template

Configuration Files:

1) BASHRC:
It is a shell script that Bash runs whenever it is started interactively. The .bashrc file itself contains a series of configurations for the terminal session.
Here we update the JAVA home for our instances so that it can load JAVA from there.We add HADOOP_CLASSPATH to perform jar related options of Hadoop without which we will get errors like class not found.

2)HADOOP_ENV.sh
In order to develop Hadoop programs in java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing **JAVA_HOME** value with the location of java in your system.

3)core-site.xml

It contains information about he port numbers that the Hadoop instances use , size of Read/Write buffers ,memory allocated for the file system, memory limit for storing the data.

4)hdfs-site.xml

It contains the information of performance related things such as the number of nodes the data should be replicated, namenode path, and datanode paths the local file systems.

5)**yarn-site.xml**

It is used to configure yarn into Hadoop. YARN is Yet Another Resource Manager.

6) **mapred-site.xml**

There will not be a file called mapred-site.xml rather there will be a file called **mapred-site,xml.template** . We need to copy using:

```
$Cp mapred-site,xml.template  mapred-site,xml
```

We can control the number of mappers and reducers we need.To control shuffle phase timeouts.to decide number of threads a shuffle should have and lot more things.

1. ~/.bashrc:

```
export CONF=/home/ubuntu/hadoop-2.7.2/etc/hadoop

export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

export PATH=$PATH:$/home/ubuntu/hadoop-2.7.2/bin

export HADOOP_CLASSPATH=/usr/lib/jvm/java-7-openjdk-amd64/lib/tools.jar

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
```

For the changes to take place immediately without a restart , run source ~/.bashrc

2./ conf /hadoop-env.sh: This file contains some environment variable settings used by Hadoop

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
```

3. ./ conf /core-site.xml:

```
<configuration>

<property>
<name>fs.default.name</name>
<value>hdfs:<ec2-52-90-66-50.compute-1.amazonaws.com>:9000 </value>
</property>

<property>
<name>hadoop.tmp.dir</name>
<value>/mnt/raid</value>#For Raided volumes
</property>
<name>hadoop.tmp.dir</name>
<value>/home/ubuntu/Hadoop/hadoop_tmp/tmp</value>
</property>
</configuration>
```

4) / conf /hdfs -site.xml

```
<configuration>

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

<property>
<name>dfs.permissions</name>
<value>false</value>
</property>
```

5)

/conf /mapred-site.xml:
By default, the /usr/local/hadoop/etc/hadoop/ contains the /usr/local/hadoop/etc/hadoop/mapred-site.xml.template file which has to be renamed/copied with the name mapred-site.xml:

```
$cp ~ Hadoop/etc/hadoop/mapred-site.xml.template ~ Hadoop /etc/hadoop/mapred-site.xml
```

```
<configuration>

<property>
 <name>mapreduce.jobtracker.address</name>
 <value>hdfs:// ec2-52-90-66-50.compute-1.amazonaws.com:8021</value>
 <description>The host and port that the MapReduce job tracker ru
ns at.If "local",then jobs are run in-process as a single map and reduce task.
 </description>
</property>

<property>
 <name>mapreduce.framework.name</name>
 <value>yarn</value>
 <description>The framework for running map reduce jobs</description>
</property>

</configuration>
```

6)
yarn-site.xml

```
<configuration>
<property>
 <name>yarn.nodemanager.aux.services</name>
 <value>mapreduce_shuffle</value>
</property>

<property>
 <name>yarn.resourcemanager.scheduler.address</name>
 <value>ec2-52-90-66-50.compute-1.amazonaws.com:8030</value>
</property>

<property>
 <name>yarn.resourcemanager.address</name>
 <value>ec2-52-90-66-50.compute-1.amazonaws.com:8032</value>
</property>


<property>
 <name>yarn.resourcemanager.webapp.address</name>
 <value>ec2-52-90-66-50.compute-1.amazonaws.com:8088</value>
</property>
```

```
<property>
 <name>yarn.resourcemanager.resource-tracker.address</name>
 <value>ec2-52-90-66-50.compute-1.amazonaws.com:8031</value>
</property>


<property>
 <name>yarn.resourcemanager.admin.address</name>
 <value>ec2-52-90-66-50.compute-1.amazonaws.com:8033</value>
</property>

</configuration>
```

7)vi slaves

```
ec2-52-90-66-50.compute-1.amazonaws.com
```

Once our Master Node is setup , we can take the image of this and launch our cluster of 16 slave nodes.After all of our slaves are launched , we need to edit 2 more files on the MASTER node

1. conf/hosts
2. conf/slaves

In the MASTER node , update with all the IP address of the slaves ie 16 nodes to the conf/slaves file

```
$vi ~/Hadoop/conf/slaves
```

In conf/hosts Add all the IP's of the the slaves in the file including the MASTER IP.

```
$vi ~/Hadoop/conf/hosts
```

Once all our configuration is done.We need format our namenode by using :

```
$cd hadoop/bin
$./hdfs namenode -format
```

Next step is to start DFS and YARN

```
$cd Hadoop/sbin
$./start-dfs.sh
$./start-yarn.sh
```

To verify all our effort has paid off , we need to check the list of things started.

```
$jps
```

It will list
Namenode
Datanode
SecondaryNameNode
ResourceManager

Once Hadoop is installed , we will be able to access HDFS

Few commands which I used during the experiment:

To create a directory in hdfs

```
~/hadoop/bin$ ./hdfs dfs -mkdir /user
~/hadoop/bin$ ./hdfs dfs -mkdir /user/akhil
~/hadoop/bin$ ./hdfs dfs -mkdir /user/akhil/input
```

To check files available:

```
~/hadoop/bin$ ./hadoop fs -ls /
```

To copy files from local filesystem to hdfs

```
~/hadoop/bin$./hadoop fs -put /home/ubuntu/64/10gbinput.txt  /
```

To remove a directory

```
~/hadoop/bin$./hadoop fs -rm -r /user
```

Hadoop Sort:

10 GB on single node:

Create the 10GB input file by using Gensort

```
./gensort -a 10gbinput.txt
```

Move it to HDFS :

```
~/Hadoop/bin/ hadoop fs -put ~/64/10gbinput.txt /
```

To create JAR,

Go to the location of Code

```
$/home/ubuntu/hadoop/bin/hadoop com.sun.tools.javac.Main MainSort.java
$jar cf MainSort.jar MainSort*.class
```

To run the JAR file,

```
bin/hadoop jar /home/ubuntu/code/MainSort MainSort /10gbinput.txt  /output
```

Get the file HDFS to local

```
~/Hadoop/bin/ hadoop fs -get /output ~/64/
```

Validate the result by performing Valsort on it

```
$unix2dos part-00000
$./valsort part-00000
```

100GB on 16-node:

Create the 100GB input file by using Gensort

```
./gensort -a 100gbinput.txt
```

Move it to HDFS :

```
~/Hadoop/bin/ hadoop fs -put ~/64/100gbinput.txt /
```

To create JAR,

Go to the location of Code

```
$/home/ubuntu/hadoop/bin/hadoop com.sun.tools.javac.Main MainSort.java
$jar cf MainSort.jar MainSort*.class
```

To run the JAR file,

```
bin/hadoop jar /home/ubuntu/code/MainSort MainSort /10gbinput.txt  /output
```

Get the file HDFS to local

```
~/Hadoop/bin/ hadoop fs -get /output ~/64/
```

Validate the result by performing Valsort on it

```
$unix2dos part-00000
$./valsort part-00000
```

Implementation:

- Hadoop MapReduce has been implemented in java.
- Launch all the instances and perform necessary configuration as mentioned above
- Add the storage depending on file size to be sorted
- Generate the dataset using gensort
- Transfer the generated data to HDFS
- Create the JAR file and run the jar file by specifying the input and output path location

- **MapReduce** is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.

- In MAP phase the input is divided into input splits for processing by map tasks running in parallel across the Hadoop cluster.

- In REDUCE phase the results from map phase are used as input to a set of parallel reduce tasks. The reduce tasks consolidate the data into final results.

- In between Map and Reduce phase , there are shuffle and sort stage.

- Shuffle is the process of transfering data from the mappers to the reducers, it is necessary for the reducers, since otherwise, they wouldn't be able to have any input .

- Sort starts a new reduce task, when the next key in the sorted input data is different than the previous

- Once the output is generated move it back to the local file system .

- Perform a valsort on the output data to validate the results.

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

- The master node is the one which stores a lot of data and runs parallel computations on data, it controls all the slaves. Name Node and Secondary name node in Hadoop architecture are called as Master nodes .It divides the work between the slaves.
  It holds all of the HDFS metadata in the cluster and is the controller. It decides the splitting and distributing of data and resources to it.

- Slave node is the Data node in Hadoop architecture. It is where the data that needs to be computed is stored .Its execution depends on the master node as and when to execute.It just stores the compute data and computes the data.

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?

   The reason why we need to set unique available ports is that there would be multiple users and non unique ports for each user might cause conflicts on the network side .It can be a port address translation issue when IP address does a port based internal and external transfers.
   It can also lead to insecure operations and interference of operations might happen as ports are different.

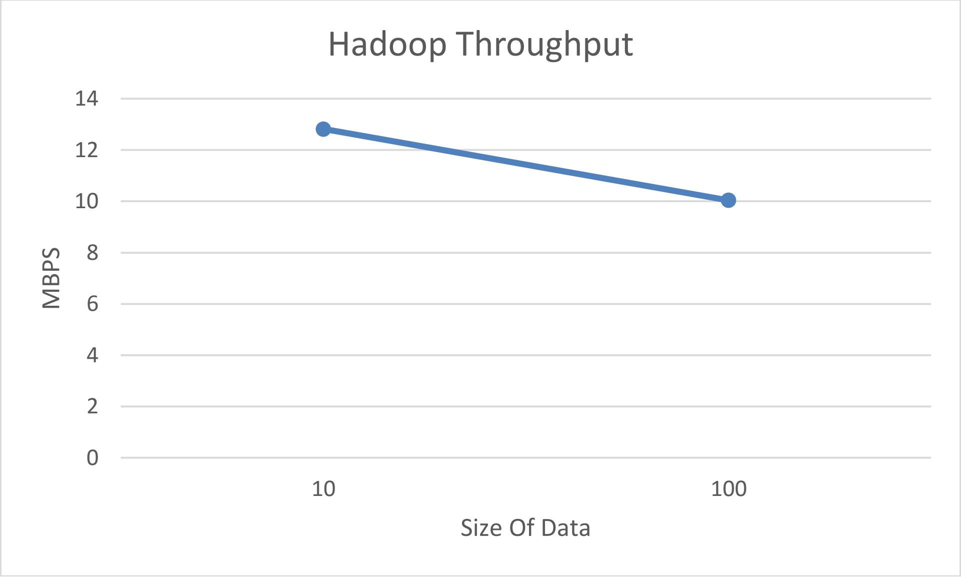
3) How can we change the number of mappers and reducers from the configuration file?

- This can be done by editing the mapred-site.xml available in the Hadoop/conf directory.
  ```
  <property>
  <name>mapred.map.tasks<\name>
  <value>20<\value>
  <\property>
  <property>
  <name>mapred.reduce.tasks<\name>
  <value>4<\value>
  <\property>
  ```
- The number of Mappers and Reducers can also be changed by:
  ```
  $ -D Mapred.map.task=20
  $ -D Mapred.red.task=4
  ```

<u>Performance:</u>

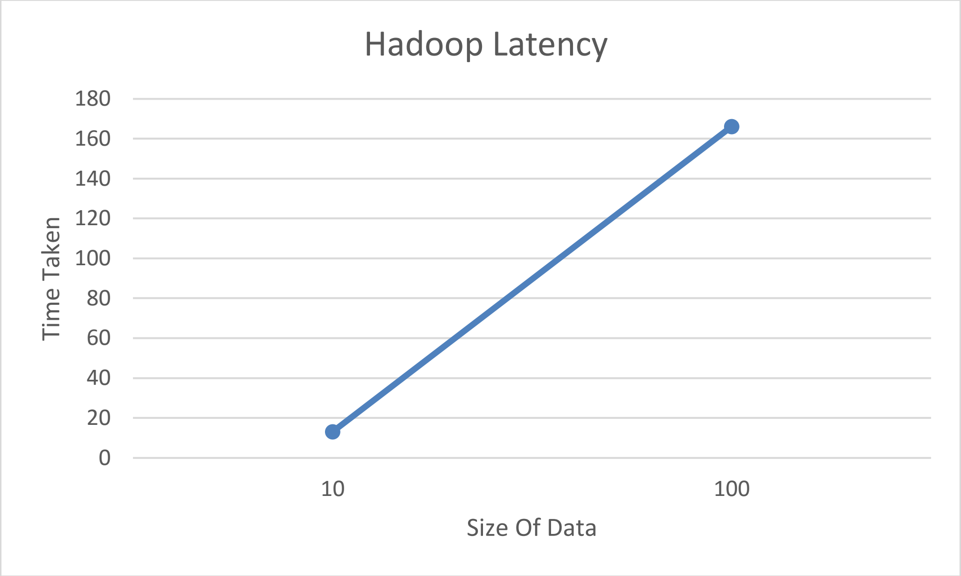
<u>Throughput:</u>

Size Of Data(in GB)	Throughput(MBPS)
10	12.82
100	10.04



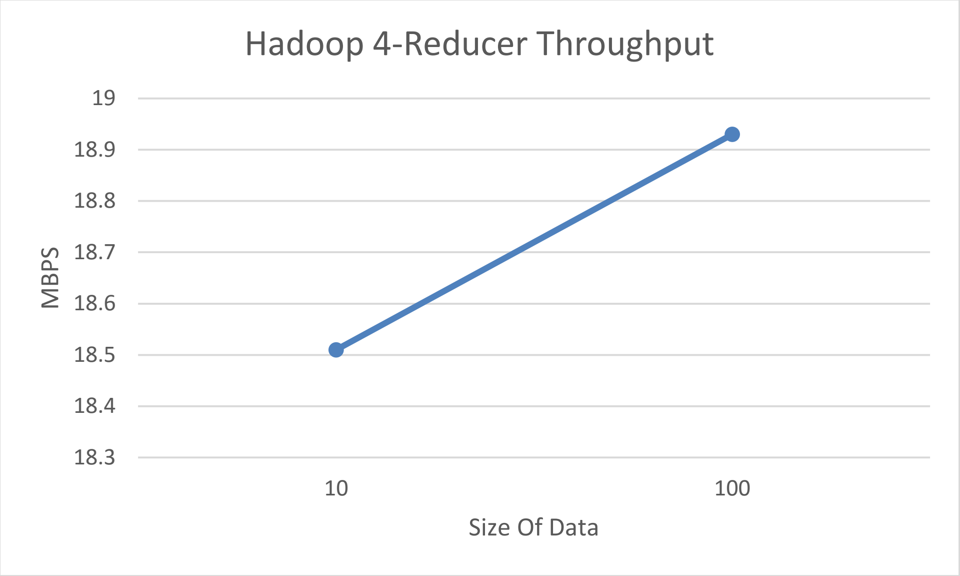
Latency:

Size Of Data(in GB)	Execution Time(in Minutes)
10	13
100	166

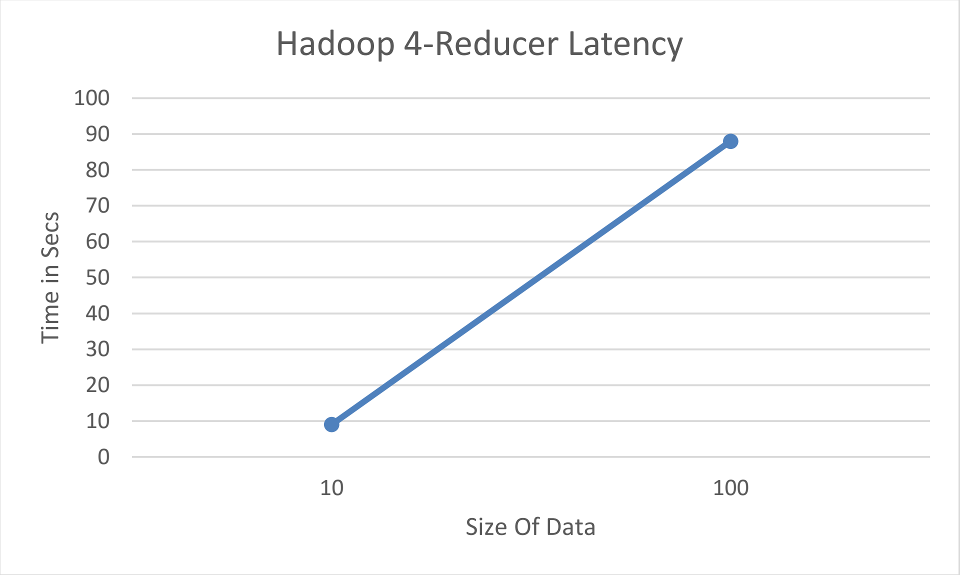


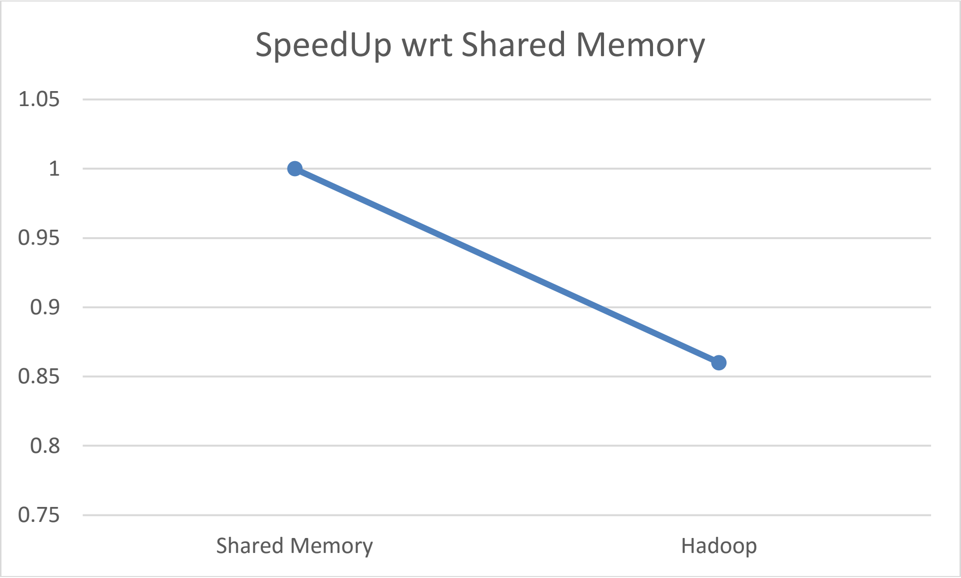
Throughput:

Size Of Data(in GB)	Throughput(MBPS)
10	18.51
100	18.93


Hadoop 4-Reducer Throughput
19
18.9
18.8
18.7
18.6
18.5
18.4
18.3
MBPS
10
100
Size Of Data

Latency:


Hadoop 4-Reducer Latency
100
90
80
70
60
50
40
30
20
10
0
Time in Secs
10
100
Size Of Data



Paradigms	SpeedUp wrt Shared Memory	Time
Shared Memory	1	15.06
Hadoop	0.86	13

Conclusion:

Best results are achieved when the number of receivers are increased . We can see the gain in performance when the number of reducers are increased to 4.

We achieve better results on Hadoop when we effectively utilize Reducers and Mappers. This is the core of the Cloud architecture design.

SPARK:

System Configuration:

JAVA VERSION:1.7

INSTANCE TYPE: c3. Large (10GB) UBUNTU

vCPU=2

Memory=3.75GB

SSD Storage=2*16GB

Before launching spark we need export few things

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
```

```
$chmod 400 sparkforsydney.pem
```

These are available in the AWS website and are unique to each user.

- Download spark binary from http://apache.claz.org/spark/spark-1.4.1/spark-1.4.1-bin-hadoop2.6.tgz
- Check if java is installed if not than you can download it using the following command.
  $**sudo apt-get install default-jdk**
- Set environment variable JAVA_HOME to where java is installed.
  $export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
- Go to Spark /sbin and run
  ./start-master.sh

To launch a spark cd spark/ec2

```
$ ~/spark/ec2$ ./spark-ec2 --key-pair=sparkforsydney --identity-file=/home/ubuntu/ sparkforsydney.pem
--instance-type=c3.large -s 16 --region=us-east-1  --spot-price=0.019 --hadoop-major-version=yarn
launch spark16nodecluster
```

Implementation:

- We have implemented the Spark sort in python.

- Launch the cluster and perform all necessary implementation stated above .

- Add storage depending on file size to be sorted

- Generate the dataset using gensort

- Transfer the generated data to HDFS

- In my code when the file is read from HDFS and is divided into key value RDD pairs as the 1st 10 bytes to be key and next 90 to be the value

- We have used the sortbykey() function which sorts the data on the basis of keys .

- Once the output is generated , it is stored as a tuple and converted to text file by using the SaveAsTextFile function.

- We have used coalesce function to avoid full shuffle as the for bigger datasets too many splits would happen and keep the data on minimum number of nodes.

- The output is sent into the HDFS which can be accessed using get command

- This output value is checked using ./valsort in gensort

We can tune the performance of the application by changing the below parameters in spark-env.sh

Options read in YARN client mode

SPARK_EXECUTOR_INSTANCES: It is the number of executors to start It is 2 by default

SPARK_EXECUTOR_CORES: It is the number of cores for the executors .By Default it is 1.

SPARK_EXECUTOR_MEMORY: It is the memory allocations per each Executor and is by default 1GB

Options for the daemons used in the standalone deploy mode

SPARK_WORKER_CORES:It is used to set the number of cores to use on the machine

SPARK_WORKER_MEMORY: It is used to set how much total memory workers have to give executors

SPARK_WORKER_INSTANCES: It is used to set the number of worker processes per node

For our c3 large instances , given the resources availaibility ,I have set

```
SPARK_EXECUTOR_INSTANCES=4
SPARK_EXECUTOR_CORES=1
SPARK_EXECUTOR_MEMORY=2
```

1 - Node 10GB:

```
$tar xvf gensort-linux-1.5.tar.gz
$Cd 64/
$./gensort -a 100000000 10gbinputfile.txt
```

We need move the locally generated data to HDFS so that Hadoop can access it.

```
$ bin/hdfs dfs -ls /
$ bin/hdfs dfs -put ~/64/10gbinput.txt  /
```

1. $ cd
2. $cd spark/bin
3. Chmod 777 spark-submit
4. ./spark-submit /home/Ubuntu/spark/code.py
   //Input(HDFS) and output(local) are specified in the program

Once the ouput is generated in HDFS we need to “get ” it back to local drive so that we can validate it.

```
$bin/hadoop fs -get /output ~/64/
```

```
Since valsort works on DOS , we need convert it .
$sudo unix2dos part-r-00000
$cd /64/
$./valsort part-r-00000
```

Since we need to get the first 10 and last 10 lines of our sorted output,

```
$ head part-r-00000
$ tail part-r-00000
```

<u>16 – node 100GB</u>

```
~/spark/ec2$ ./spark-ec2 --key-pair=sparkforsydney --identity-
file=/home/ubuntu/ sparkforsydney.pem --instance-type=c3.large –s=16 -
-region= ap-southeast-2  --spot-price=0.019 --hadoop-major-version=yarn
launch spark16node
```

```
Generate Input data:
$Cd 64/
$./gensort -a 1000000000 100gbinputfile.txt
```

We need move the locally generated data to HDFS so that Hadoop can access it.

```
$ bin/hdfs dfs -ls /
$ bin/hdfs dfs -put ~/64/100gbinput.txt  /
```

1. $cd
2. $cd spark/bin
3. Chmod 777 spark-submit
4. ./spark-submit /home/Ubuntu/spark/code.py
   //Input(HDFS) and output(local) are specified in the program
   Once the ouput is generated in HDFS we need to "get " it back to local drive so that we can validate it.

```
$bin/hadoop fs -get /output ~/64/
```

Since valsort works on DOS , we need convert it .

```
$sudo unix2dos part-r-00000
$cd /64/
$./valsort part-r-00000
```

Since we need to get the first 10 and last 10 lines of our sorted output,

```
$ head part-r-00000
$ tail part-r-00000
```

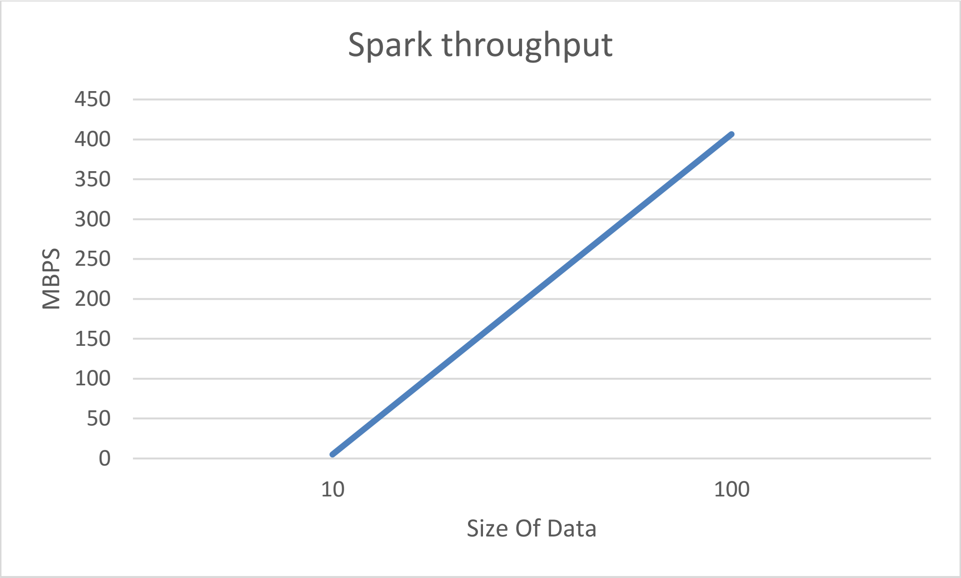
```
$./spark-ec2 destroy spark-cluster
```

This will destroy the cluster.

Performance:

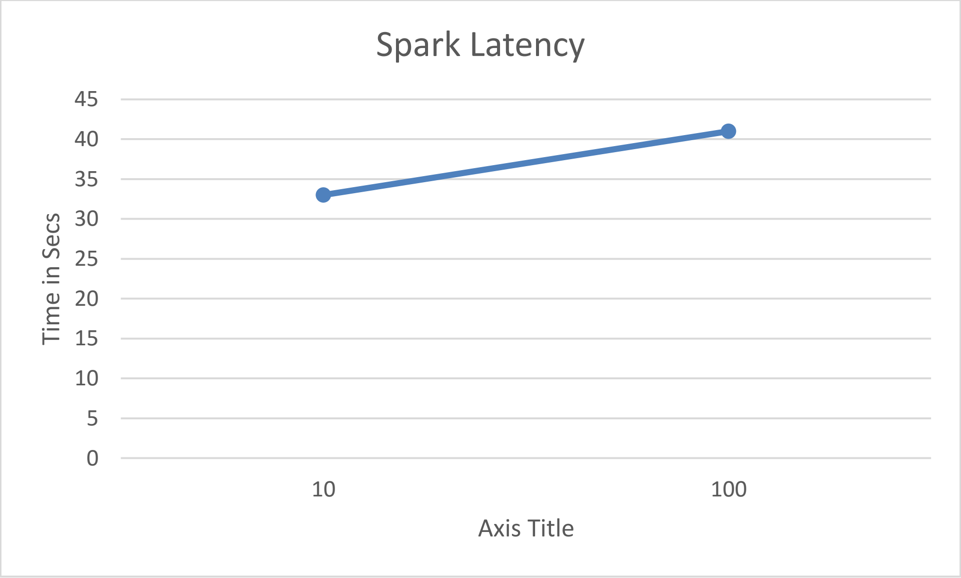
Throughput:

Size Of Data(in GB)	Throughput(MBPS)
10	5.05
100	406.5



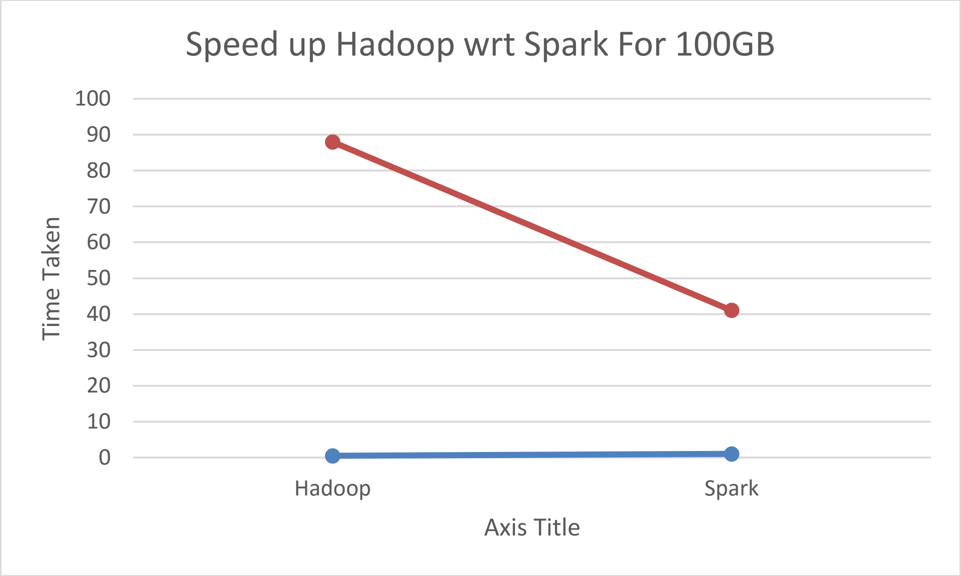
Latency:

Size Of Data(in GB)	Execution Time(in Minutes)
10	33
100	41



Speed up Hadoop wrt Spark For 100GB

Paradigms	SpeedUp wrt Spark	Time
Spark	1	41
Hadoop	0.46	88



Paradigms	SpeedUp wrt Shared Memory	Time
Shared Memory	1	15.06
Spark	0.45	33

Conclusion:

After doing the above experiments we can infer that Spark has better performance as the size of the dataset increase .

The performance wont be optimum for lower datasets.

We can enhance the performance of our program by effectively utilizing the resources available . These things can be effectively configuring **SPARK-ENV.sh.**

In my experiment, I could see the drastic change in performance when we changed

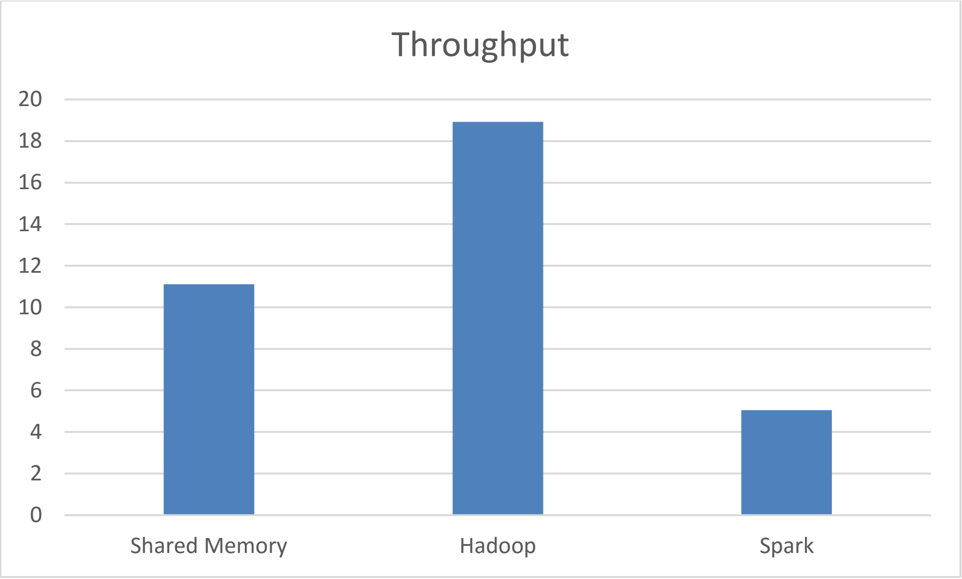
SPARK_EXECUTOR_INSTANCES to 4.

Comaparison of 3 methods used

Throughput for 1 node:

For 10GB

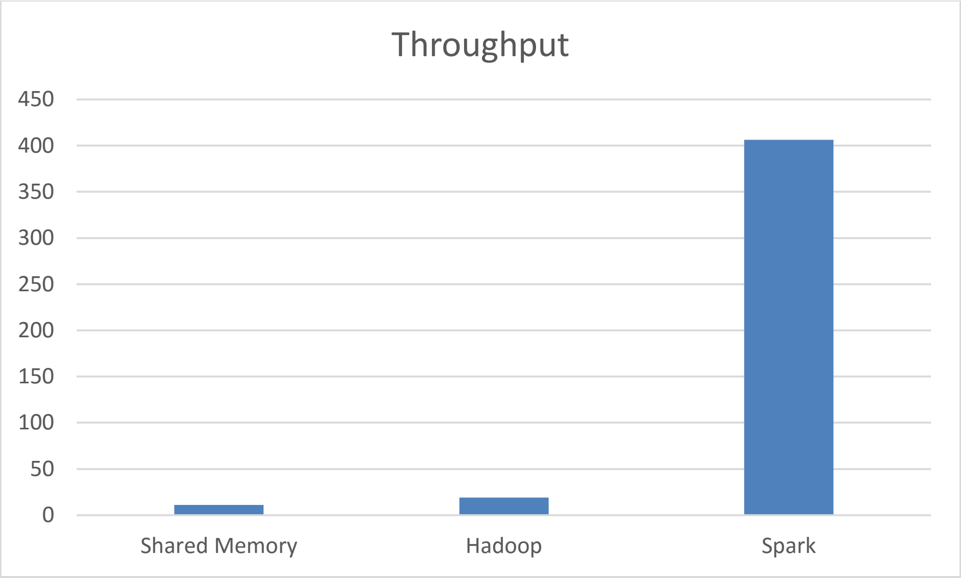
Paradigm	Throughput
Shared Memory	11.11
Hadoop	12.82
Spark	5.05



For 16 node :

For 100 GB

Paradigm	Throughput
Shared Memory	11.11
Hadoop	18.93
Spark	406.5



For 1 Node scale:

Hadoop seems to be the best at **1 node scale**

For 16 Node cases:

Spark is the **best** choice for this.

For 100 node scale :

Spark is best suitable for this.

For 1000 Node scale:

Spark works well with big numbers as well when comapared with Hadoop.

According experiments conducted , it shows that Hadoop doesn’t withstand for nodes larger than >4000. Resource allocation becomes the bottleneck.Hence SPARK is best suitable for this case.

The winners of the Sort benchmarking have exponential performance when compared to our results.

We hardly achieve throughput in MBPS but they have achieved in the order of TB per Minute.

Eventhough the resources used are of higher order those algorithms would give better performance any time .

Cloudsort Benchmark :

It emphasizes on total cost involved in sorting certain amount of public cloud resources taking into consideration IAAS as there are many service providers .

It highlights the missing feature in sort benchmarks which is total-cost-of-ownership . It also stresses on cloud platforms tbeing poorly

provisioned for IO-intensive workloads.

It plants the idea of the cheapest way to sort a fixed

number of records on the public cloud

References:

http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Weiss/L17-ExternalSortEX2.html

http://www.apache.org/

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

http://insightdataengineering.com/blog/hadoopdevops/

http://paxcel.net/blog/how-to-setup-apache-spark-standalone-cluster-on-multiple-machine/

https://github.com/lemire/externalsortinginjava/blob/master/src/main/java/com/google/code/externalsorting/ExternalSort.java