CS553 Cloud Computing
Programming Assignment 2

Performance Evaluation

submitted by:
Chiranjeevi Ankamreddy
A20359837

The programming assignment carries out Sort application implemented in 3 different ways: Java, Hadoop, Spark and MPI.Here, sorting application reads a large file and sort it in place where program should be able to sort larger than memory files. And we have to create two different datasets, a small and a large dataset. we have generated 10 Gb and 100Gb dataset since storing 100GB dataset on Amazon S3.

I've Evaluated sorting application on these four ways:

1.Hadoop

2.Spark

3.Shared Memory

4.Mpi

All the experiments have been performed using 10GB and 100GB datasets on 1 and 16 nodes. **Instance Type – C3.xlarge - Zone – US East (N Virginia)**

Hadoop Version – 2.7.2

Spark Version – 1.6.1

Java Version - 1.7.0_95

Operation System –ubuntu 14.04

Mpi version- Mpich-3.2

1.Hadoop

In Hadoop,we need to Install Hadoop including the HDFS distributed file system and have to configure Hadoop to run the job tracker and filesystem metadata service on separate nodes, leaving 16 nodes available to run workers for map and reduce tasks. I have set up to run Hadoop cluster. I have to turn off data replication on the HDFS file system to ensure you get the most performance out of our system. In Hadoop Sort application, I evaluated its performance on 1 node and 16 nodes, did strong scaling experiments from 1 node to 16 nodes.

1.Hadoop cluster Installation and execution

Master Node:

1. Initially download the Pem form Amazon AWS ,and chiru.pem and Hadoop xml Scripts files at /home/ec2-user/ location.

2.Add permission to chiru .pem file using following command

chmod 400 chiru.pem

3.Generate the key for agent Session by following command.

eval `ssh-agent -s`

ssh-add chiru.pem

4.Install the Haddop by following command,
-Install Hadoop
And execute the script Hadoopinstall.sh and RAID 0 the underlying Disk.

5.Set the all the environment variables in .bashrc file

export CONF=/usr/local/hadoop/etc/hadoop

export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.95.x86_64 export HADOOP_INSTALL=/usr/local/hadoop export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin export HADOOP_COMMON_HOME=$HADOOP_INSTALL export HADOOP_MAPRED_HOME=$HADOOP_INSTALL export YARN_HOME=$HADOOP_INSTALL export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HDFS_HOME=$HADOOP_INSTALL export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar

6.Now save the Bashrc and execute the fllowing command,

~/.bashrc

7.now,Update the following xml files at path /usr/local/hadoop/etc/hadoop with same configuaration as provided in hadoopsetup folder.

- mapred-site.xml
- core-site.xml hdfs-site.xml
- yarn-site.xml

8.Create AMI of Master Node and 1 slave for 1 node,and create 16 slaves instances using AMI of master node.

9.Update the slaves file of Master at location /usr/local/hadoop/etc/hadoop and invoke the IP's of all 16 running slaves.

10.Copy slaves file to following path /home/ec2-user

11.now,configure_slaves.sh on slave nodes and execute configure_slaves.sh on slaves nodes using following commands from master node. And copy all files on slaves node.

scp -i chiru.pem -r /home/ec2-user/slaves slavesip:/home/ec2-user

scp -i chiru.pem -r /home/ec2-user/configure_slaves.sh slavveip:/home/ec2-user

12. Running configure_slaves ssh -i chiru.pem slavesip '/home/ec2-user/configure_slaves.sh'

13.On the Mater node execute the following command to format name node

hadoop namenode -format

14.now Execute the script c on Master Node.

15.Use Jps command and you will see the following running proces on the Master Node

- NameNode
- Secondary NameNode
- Resource Manager

16.Please check the cluster report using following command
Hadoop dfsadmin -report and you will see 16 live Data nodes.
17. create a imput file under new directory.

hadoop fs -mkdir input

18.Add input.txt file in input folder

hadoop fs -put input.txt input

19.Now Compile java program and run it.

hadoop hadoop com.sun.tools.javac.Main HadoopSort.java

20.Creating jar from .class file

jar cf HadoopSort.jar *.class

21.Now,Execute jar hadoop jar HadoopSort.jar Sample argument1 argument2 The following are the

Now,I've to modify these configuration files as you go to a multi-node run on hadoop 2.7.2 cluster.

1) conf/master

2) conf/slaves

3) conf/core-site.xml

4) conf/hdfs-site.xml

5) conf/mapred-site.xml

Following is the description of each file , changed the properties during Installation.

->Yarn-site.xml

The property yarn.nodemanager.aux-services notifies the Nodemanager that there is auxilary services called mapreduce_shuffle which the Nodemanager needs to implement. We can also set the amount of physical memory, that can be allocated for containers using yarn.nodemanager.resource.memory-mb property and set the address of the schedular interface and other configurations related to resource manager.

From 1 node to 16 , we will need to update the slave file and provide the appropriate memoy for map task and reduce task in mapred-site.xml using mapreduce.map.memory.mb and mapreduce.reduce.memory.mb properties respectively .

```
<property>
    <name>yarn.nodemanager.resource.memory-mb</name>
    <value>2808</value>
</property
```

2.Slaves

The slaves file contains IP address of each DataNode . As we were working with 16 nodes , the slaves file contained 16 different Ips of Data nodes.

3.Core-site.xml -
The core-site.xml informs the hadoop daemon on which machine Namenode is running in the cluster. I have used the property **hadoop.temp.dir** to inform hadoop framework about where to store the intermediate temporary results i.e. in cluster setup I have used the mounted raid 0 disk to store the intermediate results by providing appropriate configuration in core-site.xml.

```
<property>
   <name>hadoop.temp.dir</name>
            <value>/mnt/raid/temporary_dir</value>
</property>
```

4.hdfs-site.xml -

The hdfs-site.xml contains the configuration setting for hdfs daemons for Namenode, Datanode and Secondary Namenode. We can manipulate the no of replication of blocks created for the input file by

using dfs.replication, by defaults the replication factor is 3.Inorder to get best performance, replication factor as 40 given. I have used the mounted raid 0 disk for storing data blocks in Namenode and Datanode

```
<configuration>
 <property>
    <name>dfs.replication</name>
    <value>40</value>
  </property>
  <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:///mnt/raid/temporary_dir/namenode</value>
</property>
<property>
   <name>dfs.datanode.data.dir</name>
   <value>file:///mnt/raid/temporary_dir/datanode</value>
</property>
</configuration>
```

5.Mapred-site.xml

The mapred-site.xml contains the configuration settings for MapReduce framework using property mapreduce.framework.name. We have to configure jobtracker's address as Name node using property mapreduce.jobtracker.address . We can also configure the amount of memory to request from the scheduler for each map task and each reduce task using properties mapreduce.map.memory.mb and mapreduce.reduce.memory.mb repectively. I have used the following configuration to request from scheduler for performing map task and reduce task.

```
<property>
     <name>mapreduce.map.memory.mb</name>
     <value>2560</value>
</property>
<property>
     <name>mapreduce.reduce.memory.mb</name>
     <value>5120</value>
</property>
```

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

A: Master node have two key functional pieces that make up Hadoop: storing lots of data (HDFS), and running parallel computations on all that data (Map Reduce). Slave Nodes make up the vast majority of machines and do all the dirty work of storing the data and running the computations. Each slave runs both a Data Node and Task Tracker daemon that communicate with and receive instructions from their master nodes.

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?

A: In Hadoop cluster,we need to set uniqe ports since communication happens through unique ports(In a cluster).because if we we use same port number for each user,it will not be able to start Hadoop cluster successfully and failure communication will be happen.Port Binding error will be appeared when cluster started.

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

A: we can change the number of mappers and reducers from the configuration file i.e mapred-site.xml file.and change the both he property names i.e mapreduce.job.maps (Mapper)and mapreduce.job.reduces(Reducer)

<name> mapreduce.job.maps </name>

<value>2</value>

<name> mapreduce.job.maps </name>

<value>40</value>

2. Spark

In spark, we need to Install Linux distribution in a virtual machine on Amazon EC2 and setup virtual cluster of 1 node and 17 nodes(1Master and 16 slaves) on Amazon EC2 using c3.large instance. Moreover, Add an additional EBS volume while creating your instance to load the dataset for16 multi-node cluster. And configure it to run the HDFS file system in order to be able to handle the 100GB dataset. We should turn off data replication on the HDFS file system to ensure you get the most performance out of my system. Also configured RDD storage in such a way that it do not create replicas of 100GB data; this will also give you the best performance at the cost of resilience in case of failures. In Spark Sort application, I evaluated its performance on 1 node and 16 nodes, did strong scaling experiments from 1 node to 16 nodes.

1.Spark cluster Installation and execution

1 Node:

Here,we need to Setup virtual cluster of 1 node on Amazon EC2 using c3.large instance and sort the 10GB of Dataset.

1. Initially we need to download the package of Spark and extract it by following Commands:

```
wget www-eu.apache.org/dist/spark/spark-1.6.0/spark-1.6.0-bin-
hadoop2.6.tgz
tar -xzvf spark-1.6.0-bin-hadoop2.6.tgz

2.Now navigate the Ec2 folder by,

cd spark-1.6.0-bin-hadoop2.6
cd ec2

3.Get the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY form AWS Security
Credentials. And export it.

export AWS_ACCESS_KEY_ID=AKIAJQOIC55IK7WES4HQ
export AWS_SECRET_ACCESS_KEY=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

4.Add permission for chiru.pem file by,
chmod 400 Parth.pem

And execute the command for buils the cluster for 1 node.

./spark-ec2 -k chiru -i /home/chiru/Downloads/Spark/chiru.pem -s 1 -t
c3.large --spot-price=0.03 launch spark_workers

5.Now Login into the cluster(master node) by following command.


./spark-ec2 -k chiru -i /home/chiru/Downloads/Spark/chiru.pem  login
spark_workers

And the export the data form Local storage by,
scp -i chiru.pem /home/chiru/Downloads/Spark/valsort
root@54.172.122.246:/root
scp -i chiru.pem /home/chiru/Downloads/Spark/gensort
root@54.172.122.246:/root
scp -i chiru.pem /home/chiru/Downloads/Spark/sort.scala
root@54.172.122.246:/root


6.Send the external disk Esb by,


sudo mkfs.ext4 /dev/xvdh
sudo mke2fs -F  -t  ext4 /dev/xvdh
sudo mkdir /ebs_store
sudo mount /dev/xvdh /ebs_store
sudo chmod 777 /ebs_store

7.Navigate to Hdfs directory and add permission for gensort to create
dataset.
```

```
cd /root/bin/ephemeral-hdfs/
bin/hadoop fs -mkdir /sparkdata

cd...
chmod 777 gensort

8.Now create Dataset of 10Gb by,

./gensort -a 1000000000 /ebs_store/sparkdaata
Inorder To check the Generted Data,enter the following Command.
ssh -i chiru.pem root@54.172.122.246
cd /ebs_store
ls

9.Move to hdfs data set.

cd /root/ephemeral-hdfs/bin

./hadoop fs -Ddfs.replication=1 -put /ebs_store/sparkdaata /sparkdata
ls

10.Execute the scala code to sort the 10Gb Data set by,

Spark-shell -i /root/sort.scala

And notedown the Readings.

11.Copy the Expected output from Hdfs store by,


bin/hadoop dfs -getmerge /sparkdata/output /ebs_store/expectedoutput
```

Validation of Output:

12.Add the permission for valsort.

Chmod 777 valsort

13.Run the Following command to validate the output file.

./valsort expectedoutput

14.Take the first 10 lines of outputfile by,

Head -10 expectedoutput

15.Take the flast 10 lines of outputfile by,

tail -10 expectedoutput

16 Nodes:

In 16 Nodes,I need to virtual cluster of 1 node and 17 nodes(1Master and 16 slaves) on Amazon EC2 using c3.large instance. Moreover, Add an additional EBS volume while creating your instance to load the dataset for16 multi-node cluster. And configure it to run the HDFS file system in order to be able to handle the 100GB dataset

1. Initially we need to download the package of Spark and extract it by following Commands:

```
wget www-eu.apache.org/dist/spark/spark-1.6.0/spark-1.6.0-bin-
hadoop2.6.tgz
tar -xzvf spark-1.6.0-bin-hadoop2.6.tgz

2.Now navigate the Ec2 folder by,

cd spark-1.6.0-bin-hadoop2.6
cd ec2

3.Get the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY form AWS Security
Credentials. And export it.

export AWS_ACCESS_KEY_ID=AKIAJQOIC55IK7WES4HQ
export AWS_SECRET_ACCESS_KEY=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

4.Add permission for chiru.pem file by,
chmod 400 Parth.pem
```

And execute the command for buils the cluster for 1 node.

```
./spark-ec2 -k chiru -i /home/chiru/Downloads/Spark/chiru.pem -s 16 -t
c3.large --spot-price=0.03 launch spark_workers

5.Now Login into the cluster(master node) by following command.


./spark-ec2 -k chiru -i /home/chiru/Downloads/Spark/chiru.pem  login
spark_workers

And the export the data form Local storage by,
scp -i chiru.pem /home/chiru/Downloads/Spark/valsort
root@54.172.122.246:/root
scp -i chiru.pem /home/chiru/Downloads/Spark/gensort
root@54.172.122.246:/root
scp -i chiru.pem /home/chiru/Downloads/Spark/sort.scala
root@54.172.122.246:/root


6.Send the external disk Esb by,


sudo mkfs.ext4 /dev/xvdh
sudo mke2fs -F  -t  ext4 /dev/xvdh
```

```
sudo mkdir /ebs_store
sudo mount /dev/xvdh /ebs_store
sudo chmod 777 /ebs_store

7.Navigate to Hdfs directory and add permission for gensort to create
dataset.


cd /root/bin/ephemeral-hdfs/
bin/hadoop fs -mkdir /sparkdata

cd...
chmod 777 gensort

8.Now create Dataset of 100Gb by,

./gensort -a 10000000000 /ebs_store/sparkdaata
Inorder To check the Generted Data,enter the following Command.
ssh -i chiru.pem root@54.172.122.246
cd /ebs_store
ls

9.Move to hdfs data set.

cd /root/ephemeral-hdfs/bin

./hadoop fs -Ddfs.replication=1 -put /ebs_store/sparkdaata /sparkdata
ls

10.Execute the scala code to sort the 100Gb Data set by,

Spark-shell -i /root/sort.scala

And notedown the Readings.

11.Copy the Expected output from Hdfs store by,


bin/hadoop dfs -getmerge /sparkdata/output /ebs_store/expectedoutput
```

Validation of Output:

12.Add the permission for valsort.

Chmod 777 valsort

13.Run the Following command to validate the output file.

./valsort expectedoutput

14.Take the first 10 lines of outputfile by,

Head -10 expectedoutput

15.Take the flast 10 lines of outputfile by,

tail -10 expectedoutput

2.Shared Memory

I've Implemented the Shared-Memory Sort application using Java and measured its performance on 1 node on a c3.large instance (this will be called Share-Memory Sort); I made Shared-Memory Sort is multi-threaded (1-2-4-8 threads) to take advantage of multiple cores and measured the time to sort on the 10GB dataset.

Sort is an application which sorts a file-resident dataset,it should be able to sort datasets that are larger than memory. I've developed java code ,it runs on any machine. But I did performance evaluation must be done on Amazon EC2 on a "c3.large" instance. And Measured the time to execute the Sort application on the 10GB dataset produced with the Gensort on 1 node.,Varied the number of threads from 1 to 8,evaluated the best performance. I've Saved the first 10 lines of output from the Shared-Memory Sort application, as well as the last 10 lines of output, for each dataset, in Sort-shared-memory-10GB.txt.

Java Installation and performance evaluation on Amazon EC2 on a "c3.large" instance:

1.Initiallyy Downnload the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY form AWS Security Credentials. And export it.

```
export AWS_ACCESS_KEY_ID=AKIAJQOIC55IK7WES4HQ
export AWS_SECRET_ACCESS_KEY=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
```

2.Navigate the Directory and add the permission to the chiru.pem.

cd Sharedmemory/

chiru@chiru:~/Downloads/Sharedmemory$ chmod 400 chiru.pem

3.Now connect to the Instance by,

chiru@chiru:~/Downloads/Sharedmemory$ ssh -i /home/chiru/Downloads/Sharedmemory/chiruoregon.pem ubuntu@52.33.58.29

4.And send the Gensort,valsort and javacode to the intstance by following Commands.

scp -i chiru.pem /home/chiru/Downloads/Sharedmemory/valsort ubuntu@52.33.58.29:/home/ubuntu

scp -i chiru.pem /home/chiru/Downloads/Sharedmemory/gensort ubuntu@52.33.58.29:/home/ubuntu

scp -i chiru.pem /home/chiru/Downloads/Sharedmemory/SharedMemory.java ubuntu@52.33.58.29:/home/ubuntu

5. Send the external disk Esb by,

sudo mkfs.ext4 /dev/xvdf

sudo mke2fs -F -t ext4 /dev/xvdf

sudo mkdir /shared_store

sudo mount /dev/xvdf /shared_store

sudo chmod 777 /shared_store

6.Create Gensort of 10gb by

chmod 777 gensort

./gensort -a 100000000 /shared_store/input

7.Now install jdk thencompile the Code and run it by,

Javac Sharememory.java

Java Sharedmemory

And we storing outputfile in output.txt

Validation of Output:

8.Add the permission for valsort.

Chmod 777 valsort

9.Run the Following command to validate the output file.

./valsort output.txt

10.Take the first 10 lines of outputfile by,

Head -10 output.txt

11.Take the flast 10 lines of outputfile by,

tail -10 output.txt

4. Mpi

I've mplemented sorting across with MPI (MPI Sort). Here,sorting application reads a large file and sort it in place where program should be able to sort larger than memory files.Using gensort,I've generated a 10GB and 100GB dataset. For MPI Sort, I have to setup MPI on your virtual cluster(Star Cluster) of 16 nodes, and implemented MPI Sort. And Measured the performance of the MPI Sort at 1 node and 16 node scales.

Mpi Virtual cluster Installation and execution:

1.For MPI,we need to install and configure starcluster to run the sort on Amazon ec2.

Firstly,we need to download the starcluster package by,

git clone git://github.com/jtriley/StarCluster.git

2.now,Navigate to Starcluster and run the distribute_setup.py.

$ cd StarCluster

$ sudo python distribute_setup.py

3.Install the cluster by,

$ sudo python setup.py install

4.Now,select the option to configure the cluster by,

$ starcluster help

Select 2

5.Generate the key by folloeing command and check the key in the config file.

starcluster createkey mykey -o ~/.ssh/mykey.rsa

In Config file,

```
# match your key name e.g.:
[key mykey]
KEY_LOCATION=~/.ssh/mykey.rsa
```

6.Now,open the Config file by

chiru@chiru:~/Downloads/MPI/StarCluster$ chmod 777 /home/chiru/.starcluster/config

chiru@chiru:~/Downloads/MPI/StarCluster$ /home/chiru/.starcluster/config

gedit config

7.After opening the config file,we need to enter the al fileds related to amazon ec2.

```
AWS_ACCESS_KEY_ID = AKIAJQOIC55IK7WES4HQ
AWS_SECRET_ACCESS_KEY =ANS0tafXPZg+0jK7wvB0Pj++drU9nt8iOpze/MDI
# replace this with your account number
AWS_USER_ID=104216196862
# Uncomment to specify a different Amazon AWS region  (OPTIONAL)
# (defaults to us-east-1 if not specified)
# NOTE: AMIs have to be migrated!
#AWS_REGION_NAME = us-east-1c
#AWS_REGION_HOST = ec2.us-east-1c.amazonaws.com
```

8.Now Enter the Size of the cluster in congig file.

```
# number of ec2 instances to launch
```

For 1 node

CLUSTER_SIZE = 1

For 16 nodes

CLUSTER_SIZE = 16

9.Now compile the Mpi Sorting code by,

mpicc Mpi_sort.c -o Mpi_sort

10.Run the code by following command

mpirun -np 2 ./Mpi_sort

11.Generate the Gensort of 10gb for 1 node and 100GB for 16 nodes.

chmod 777 gensort

./gensort -a 100000000 /Star/input

12.validate the valsort on output.txt

./valsort output.txt

Now,Take down the first 10 lines and last 10 lines of output.

Performance :

1.Shared Memory

I've Implemented the Shared-Memory Sort application using Java and measured its performance on 1 node on a c3.large instance .I made Shared-Memory Sort is multi-threaded (1-2-4-8 threads) to take advantage of multiple cores and measured the time to sort on the 10GB dataset.

Sort is an application which sorts a file-resident dataset,it should be able to sort datasets that are larger than memory. I've developed java code ,it runs on any machine. But I did performance evaluation must be done on Amazon EC2 on a "c3.large" instance. And Measured the time to execute the Sort application on the 10GB dataset produced with the Gensort on 1 node. Varied the number of threads from 1 to 8,evaluated the best performance.and Performed valsort on Output file.

For One thread,

The time taken to perform sorting on 10 GB dataset using 1 thread : 11.94 minutes
For Two threads,

The time taken to perform sorting on 10 GB dataset using 2 threads:10.28 miniutes

For Four threads,

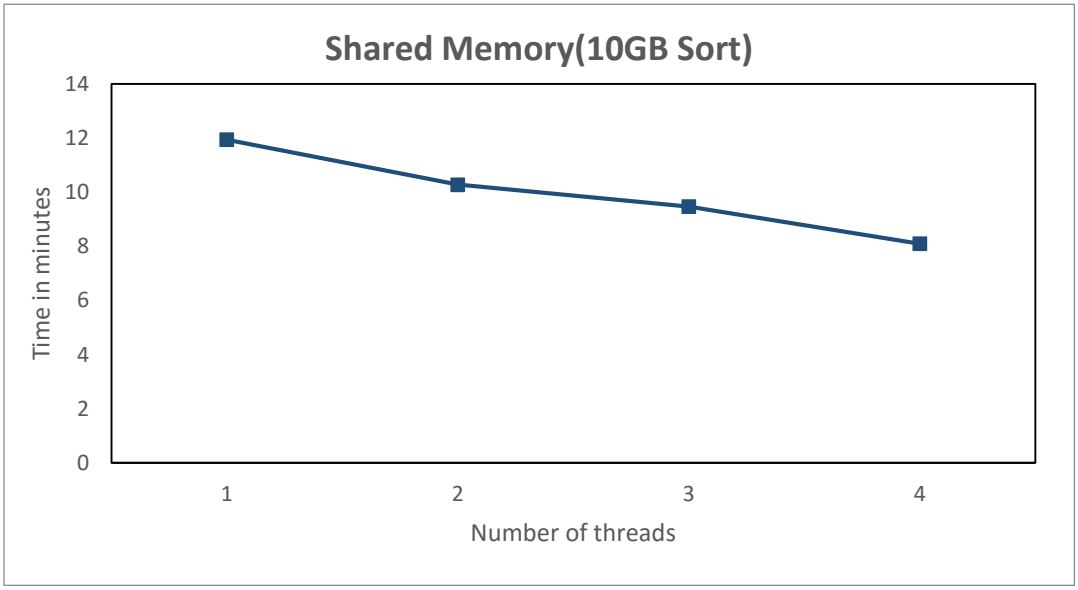
The time taken to perform sorting on 10 GB dataset using 4 threads:9.47 miniutes

For Eight threads,

The time taken to perform sorting on 10 GB dataset using 8 threads:8.1 miniutes

	Time taken (Minutes)			
Number of Threads	1 Thread	2 Threads	4 Threads	8 Threads
10 GB	11.94	10.28	9.47	8.1

Graph:



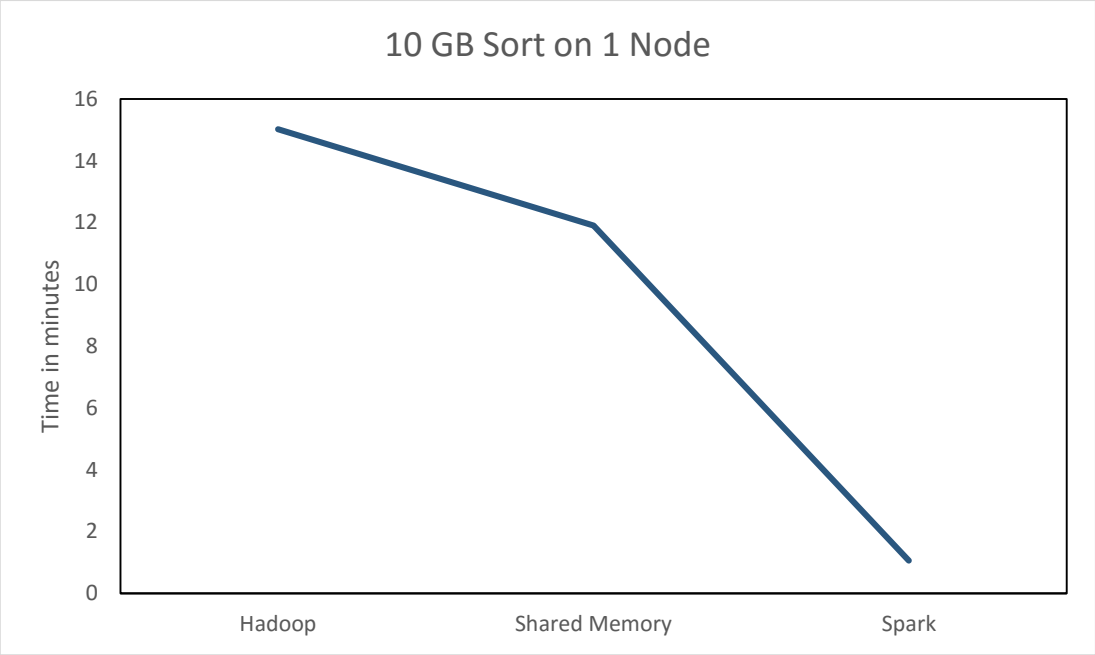
We need to Compare the performance of the three versions of Sort i.e Shared-Memory, Hadoop, and Spark on 1 node scale and compare the Shared-Memory performance of 1 node to Hadoop and Spark Sort at 16 node scales .

Execution line Charts & Throughput Chart for 1 node Experiments :

1) Comparision of sorting 10GB data on a **single** node. The following table shows the time required by different sorting techniques in minutes to sort 10GB .

	Shared Memory Execution Time (Minutes)	Hadoop Execution Time (Minutes)	Spark Execution Time (Minutes)
10GB	11.9	15.2	1.06

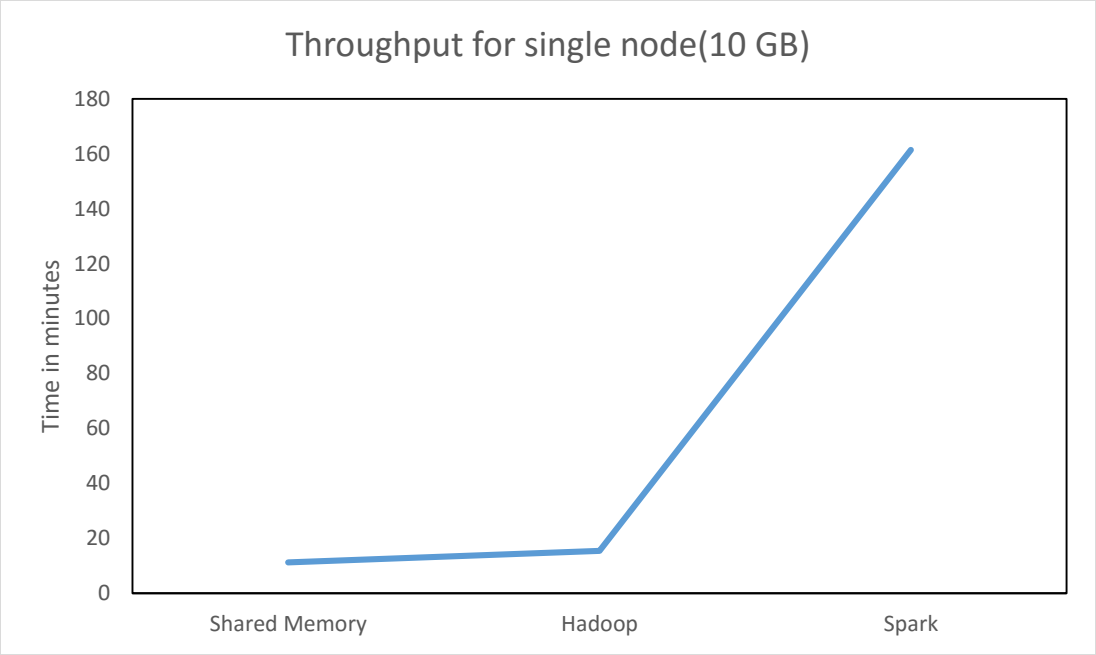
Graph:



2) Comparison of throughput obtained by sorting 10GB dataset on a **single** node. The following table shows the **throughput** obtained in MB/second for sorting 10 GB of data on a single node.

	Shared Memory Throughput (MB/second)	Hadoop Throughput (MB/second)	Spark Throughput (MB/second)
10GB	15.51	11.28	161.4

Graph:



Evaluation for 1 node Experiments :

From the above graphs Sorting on single node using 10GB datasets , Spark Sort works faster than Hadoop and Shared Memory. The reason for this is while working on single node there are loads of framework overhead while working with Shared memory and Hadoop.In Shared Memory 10GB Sort will be fairly simple considering only a single node as there will no additional overheads and the program can function smoothly. But in the spark ,it proceses data in memory.Spark jobs works better than Hadoop and shared memory when data fits in memory.

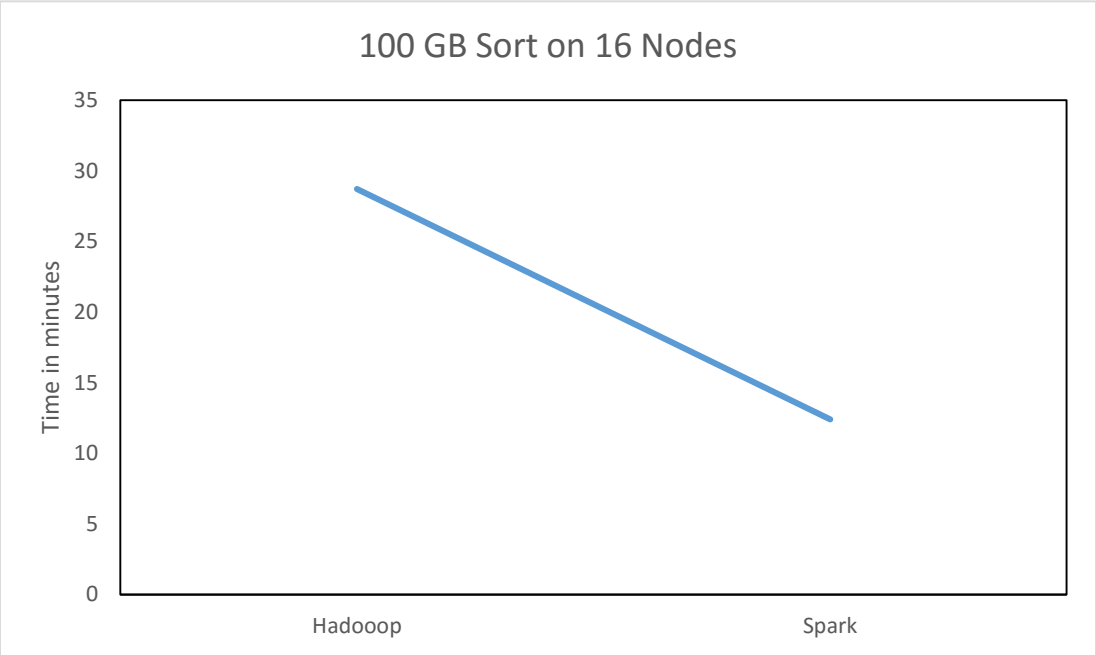
As the time required by Spark is relatively low it will obtain high amount of throughtput as compared to hadoop and Shared memory while working with 10 GB of dataset .

Execution line Charts & Throughput Chart for 16 node Experiments :

1. Comparision of sorting 100GB dataset over a **cluster of 16** nodes. The following table shows the time required by different sorting techniques in minutes to sort 100GB of data over a virtual cluster of 16 nodes.

	Hadoop Execution Time (Minutes)	Spark Execution Time (Minutes)
100GB	28.7	12.4

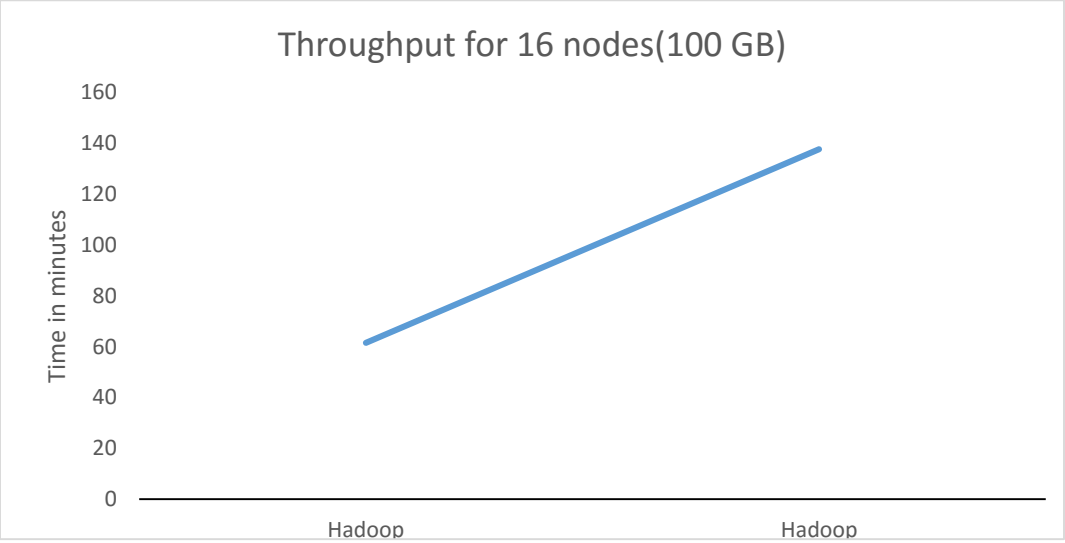
Graph:



2) Comparison of throughput obtained by sorting 100GB dataset over a cluster of **16** nodes. The following table shows the **throughput** obtained in MB/second for sorting 100GB of data on a single node.

	Hadoop Throughput(MB/second)	Spark Throughput (MB/second)
100GB	61.46	137.63

Throughput Charts for 16 Nodes :



Evaluation for 16 nodes Experiments :

From the above graphs ,when we are working with 16 nodes spark works faster as compared to Hadoop. The reasons why spark runs much faster as compared to hadoop :Spark reduces the number or read/write to disc as compared to hadoop, While working with hadoop MapReduce pushes the data back to disk after processing it ,Spark Stores intermediate data in memory ,Spark uses the concept of resilient distributed dataset which allows it to transparently stored in memory and reducing disk read/write.

From the graphs of throughput we can observed that spark obtains throughput of nearly 150 MB/s where as hadoop obtains throughput of around 195 MB/s . While runnining Spark and hadoop with large datasets we can neglect the time consumed due to framework overheads.

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?

A: From the above graphs and observation we can conclude that for **1 node** scale Spark Experiments works much better than Shared Memory and Hadoop.

For working with 16 nodes Spark works better than hadoop as most of the computations are performed in memory , even data transfer from one node to another nodes happens through main memory which makes spark to run faster.

For working with 100 nodes and 1000 nodes Spark works best , which is based on functional programming language scala which is better suited for distributed environment. The inmemory batch processing and reduction in disk read/write makes spark about 10 to 100 times faster than Map Reduce .

Compare your results with those from the Sort Benchmark specifically the winners in 2013 and 2014 who used Hadoop and Spark.

A: The winners in 2013 and 2014 who used Hadoop and spark :

Winners 2014 : Apache Spark

100 TB in 1,406 seconds

207 Amazon EC2 i2.8xlarge nodes x

(32 vCores - 2.5Ghz Intel Xeon E5-2670 v2,

244GB memory, 8x800 GB SSD)

Winner 2013: Hadoop

102.5 TB in 4,328 seconds

2100 es x

(2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks)

From the 2013 and 2014 winners : apache spark outperforms hadoop when performed experiment for sorting around 100 TB. The number of nodes and total memory used across all the nodes on spark is

much less than that used for hadoop. And in memory computations and less disk read write gives competitive edge to spark over hadoop. When compared to my results , it gives me competitive edge on spark over hadoop on performing experiment on 100GB with 16 nodes.

what can you learn from the CloudSort benchmark ?

A:The benchmark suggests to perform the sorting for 100 GB of data using minmum cost on any cloud platform .**and** The following are the reasons for using cloud platfotm for sorting

Accessibility :As we all are familiar public clouds like Amazon are acessible to every one with no up front cost.

Affordability : Running a 1 TB is very cheap and managable on Amazon EC2 which may cost around 80$.

Auditability : Audbility becomes extremely easily while working on public clouds , the auditors can re run the experiments and verify the result,

The sorting should be performed on public cloud for sorting fixed amount of data with relatively low cost , working on a public cloud will help new innovation in IO intensive tasks as most current public clouds offers poor IO intensive workloads.

I have faced Challenges Faced while working on Experiments

-Understand the hadoop and Spark architecture in details to obtain the optimal result was one of challenge I faced while working on the assignment.

-Setting up 16 nodes for hadoop and spark with best optimal configurations was very challenging which I managed spending severeal days and trying out different properties that worked best for the experiments.

-Understand how to work with hdfs system was also challenging

Compare MPI Sort with Hadoop and Spark at 1 node and 16 node Scales:

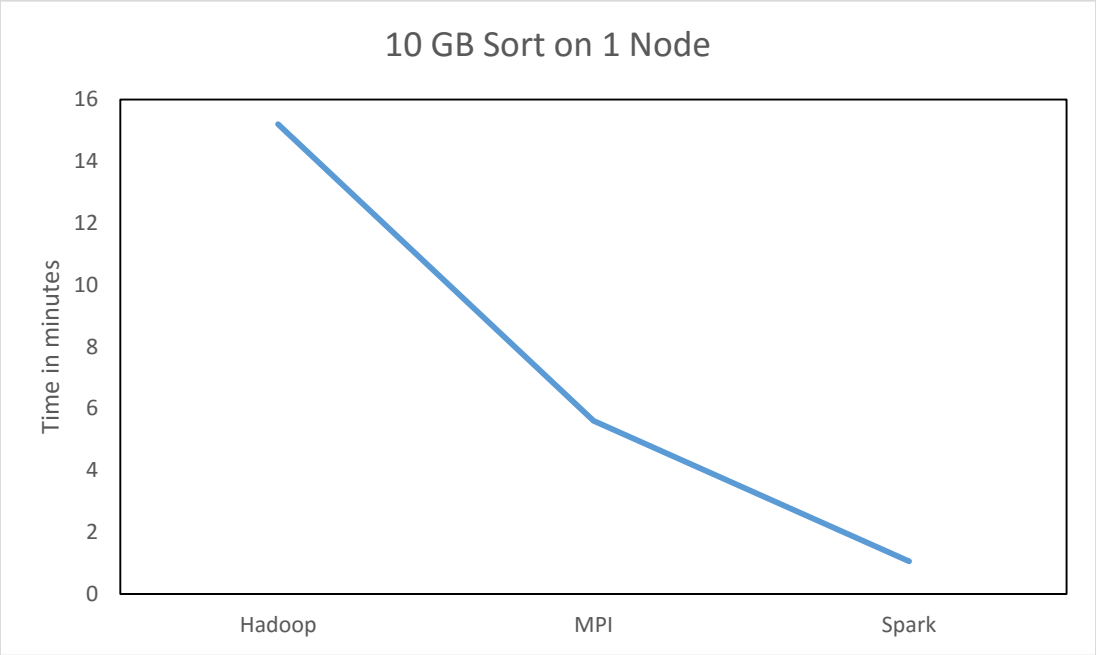
We have to Compare the performance of the three versions of Sort i.e MPI, Hadoop, and Spark on 1 node scale and 16 node scales.

Execution Line Charts & Throughput Chart for 1 node Experiments

1. Comparision of sorting 10GB data on a **single** node. The following table shows the time required by different sorting techniques in minutes to sort 10GB .

	MPI Execution Time (Minutes)	Hadoop Execution Time (Minutes)	Spark Execution Time (Minutes)
10GB	5.6	15.2	1.06

Graph:



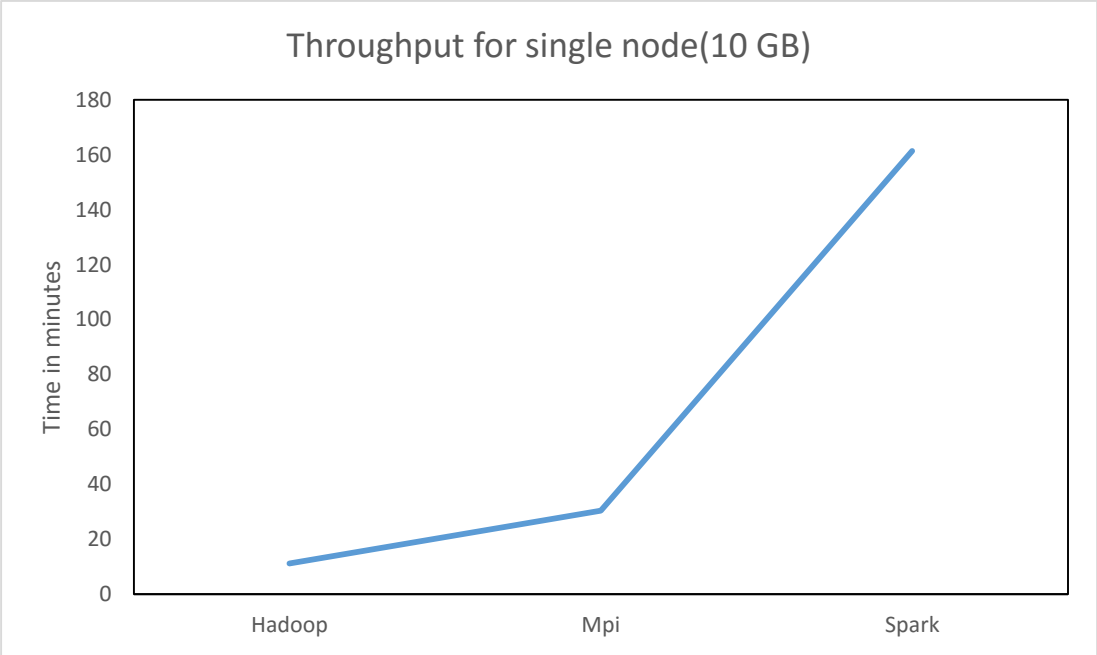
Execution Time Charts for 1 node

2) Comparison of throughput obtained by sorting 10GB dataset on a **single** node. The following table shows the **throughput** obtained in MB/second for sorting 10 GB of data on a single node.

	MPI Throughp(MB/second)	Hadoop Throughput (MB/second)	Spark Throughput (MB/second)
10GB	30.47	11.22	161

Throughput Charts for Single Node

Graph:



Evaluation for 1 node Experiments:

From the above graphs Sorting on single node using 10GB datasets , Spark Sort works faster than Hadoop and MPI. The reason for this is while working on single node there are loads of framework overhead while working with Mpi and Hadoop.In Shared Memory 10GB Sort will be fairly simple considering only a single node as there will no additional overheads and the program can function smoothly. But in the spark ,it proceses data in memory.Spark jobs works better than Hadoop and MPI when data fits in memory.

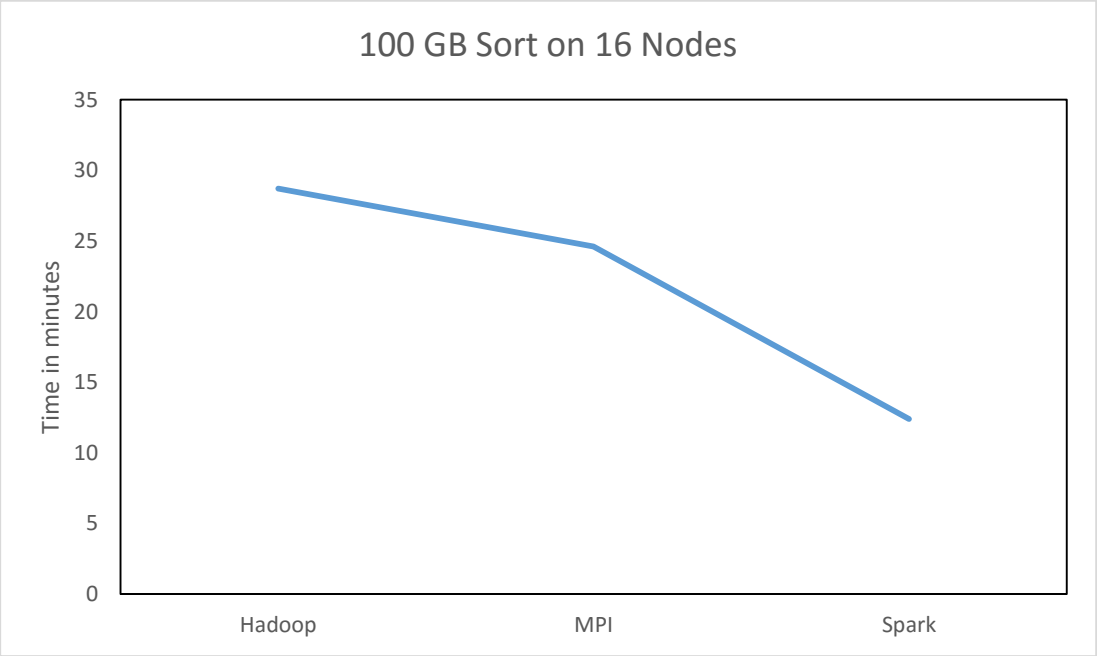
As the time required by Spark is relatively low it will obtain high amount of throughtput as compared to hadoop and Mpi while working with 10 GB of dataset .

Execution Line Charts & Throughput Chart for 16 node Experiments:

1.Comparision of sorting 100GB dataset over a **cluster of 16** nodes. The following table shows the time required by different sorting techniques in minutes to sort 100GB of data over a virtual cluster of 16 nodes.

	MPI ExecutionTime (Miniutes)	Hadoop Execution Time (Minutes)	Spark Execution Time (Minutes)
100GB	24.6	28.7	12.4

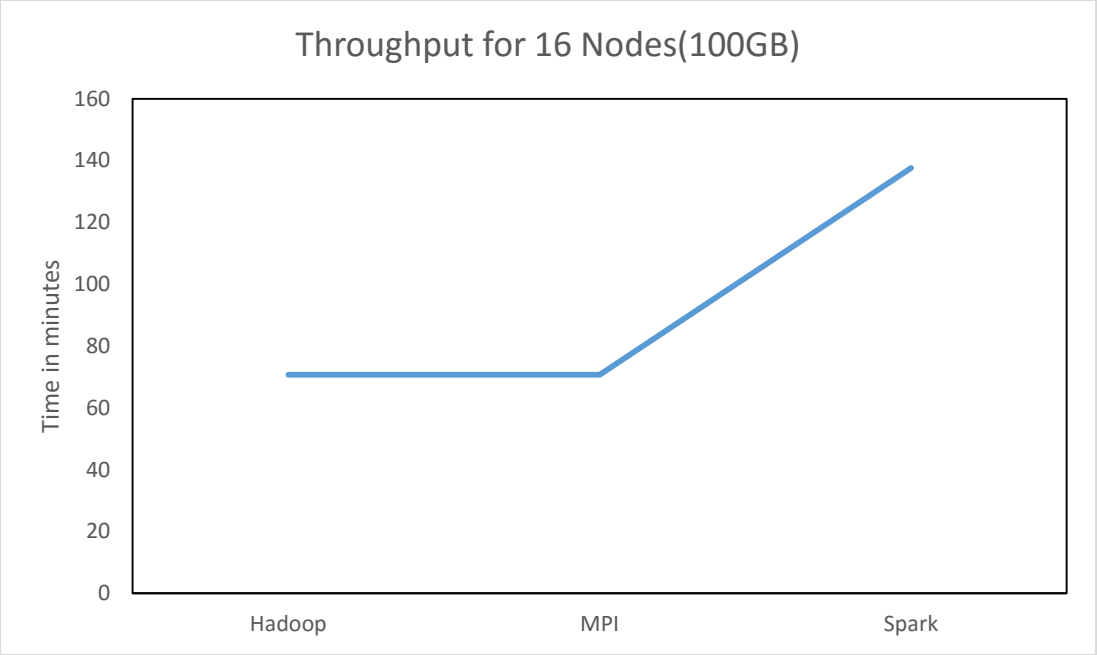
Graph :



2) Comparision of throughput obtained by sorting 100GB dataset over a cluster of **16** nodes. The following table shows the **throughput** obtained in MB/second for sorting 100GB of data on a single node.

	MPI Throughput (MB/second)	Hadoop Throughput (MB/second)	Spark Throughput (MB/second)
100GB	70.7	61.46	137.63

Graph:



Evaluation of 16 node experiments:

From the above graphs ,when we are working with 16 nodes spark works faster as compared to Hadoop. The reasons why spark runs much faster as compared to hadoop :Spark reduces the number or read/write to disc as compared to hadoop, While working with hadoop MapReduce pushes the data back to disk after processing it ,Spark Stores intermediate data in memory ,Spark uses the concept of resilient distributed dataset which allows it to transparently stored in memory and reducing disk read/write.

From the graphs of throughput we can observed that spark obtains throughput of nearly 150 MB/s where as hadoop obtains throughput of around 195 MB/s . While runnining Spark and hadoop with large datasets we can neglect the time consumed due to framework overheads.