CS553: Cloud Computing

Report

Programming Assignment-2

Problem:
We have to sort the input file generated by tool (example –gensort) we have to generate 2 datasets 10GB Dataset and 100GB dataset, which is in the format of Key value pair. The sorting application should read a large file and sort it in place. File size has to be greater than memory size.

Methodology:
We have to sort this file using 3 different methodologies. Hadoop, Spark and Shared-Memory.

Runtime environment settings:
Runtime environment has been setup with operating system Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Installed Hadoop File System, Spark, Java and Ant. Steps to install required things are mentioned in respective experiment.

Virtual Cluster (1-node):

1. **For Hadoop:** As this is 1-node cluster, I took one Amazon EC2 c3.large instance of Ubuntu OS and Added storage – of type EBS general purpose (SSD) volume of size 100 GB.

2. **For Spark:** I installed Spark setup on my local machine. Then I used AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY to setup cluster of 16 nodes. To request 16 nodes virtual cluster, we used below command
   ./spark-ec2 -k abhijeetkedari -i /home/ubuntu/abhijeetkedari.pem -s 1 -t c3.large --spot-price=0.16 launch spark_abhijeet

   This command creates a virtual cluster of *1* node of type *c3.large spot* with price for bid equals to 0.16 and we specified an name for all slave node and master node as *spark_abhijeet*.

3. **For Shared-Memory:** I took one Amazon EC2 c3.large instance of Ubuntu OS and Added storage – of type EBS general purpose (SSD) volume of size 200 GB

Virtual Cluster (16-nodes):

1. **For Hadoop:** Initially we created a Master node by taking one Amazon EC2 c3.large instance of Ubuntu OS and then we updated all required .xml and .sh file and saved image. From that saved image we created a 16 slave instances and updated their slaves and hosts files accordingly. To create cluster of 16 instances.

2. **For Spark:** I installed Spark setup on my local machine. Then I used AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY to setup cluster of 16 nodes. To request 16 nodes virtual cluster, we used below command

./spark-ec2 -k abhijeetkedari -i /home/ubuntu/abhijeetkedari.pem -s 16 -t c3.large --spot-price=0.16 launch spark_abhijeet

This command creates a virtual cluster of *16* nodes each of type *c3.large spot* with price for bid equals to *0.16* and we specified an name for all slave node and master node as *spark_abhijeet*

Please include any difficulties you might have incurred in your setup of the virtual cluster.
For setting up virtual cluster I faced following challenges:

1. For Hadoop 1 Node, there was no big challenge to setup a virtual cluster as it was single node and we just added an extra EBS volume for large data storage.
2. For Hadoop 16 nodes we have challenge for setting up clusters like we have to update hosts and slave file for each and every slave along with master, so that was difficult part as we have to update 2 files and on each slave with different value (master and slave DNS) so while doing it manually initially I have problem of typo or missed any entries causes cluster to no getting up and respond.
3. For spark it was much easier compared to Hadoop to set up 1 node and 16 node cluster. As spark has feature to launch the required number of nodes in cluster directly.
4. For Shared-memory as I have to work on 1-node cluster only it was not that challenging except EBS part that we need to mount to cluster

Description of what OS you used (Linux distribution, kernel), what ANT version, what Java version, what Hadoop version, what Spark version, and what MPI version you used.

- OS - Ubuntu Server 14.04 LTS (HVM) - Linux 3.13.0-24-generic x86_64 kernel
- Ant version - 1.9.4
- Java version - java-7-openjdk-amd64
- Hadoop version - Hadoop2.7.2
- Spark version - Spark 1.6.0.

Discuss the installation steps you took to setup your virtual cluster
Shared-Memory sort–

- Shared-memory sort is an application which sorts a file; which is larger than memory size.
- To improve the performance of our system we have used multi-threaded approach.
- File IO operation cost highest in such large processing task, so to minimize the IO operations we read data in terms of chunks and performed operations on them (Computation in CPU is much faster than IO operations)
- As our file size is greater than memory I have used External Sort strategy. Divide-sort-Merge sort.
- **Steps:**
    1. For shared memory we required to setup virtual cluster of 1 node only.
    2. So, I took one Amazon EC2 c3.large instance of Ubuntu OS and Added storage – of type EBS general purpose (SSD) volume of size 200 GB
    3. Installed required software on that instance like ssh, Java, Ant.
        - sudo apt-get install ssh
        - sudo apt-get install openjdk-7-jdk –y
        - sudo apt-get install ant
    4. Mounted external disk (EBS) created while requesting instance.
        - sudo mkfs.ext4 /dev/xvdf
        - sudo mke2fs -F -t ext4 /dev/xvdf
        - sudo mkdir /data

    - sudo mount /dev/xvdf /data
5. Created a large input file for Shared-memory application.
    - ./gensort -a 100000000 /data/SharedMemoryCode/input
6. Started our application and stored intermediate result and final output in EBS Volume.
    - cd /data/SharedMemoryCode/
    - ant
7. Validated our result with the help of valsort.
    - ./valsort /data/SharedMemoryCode/output998

Hadoop Sort step:

- We have performed this sort on 1 Node and 16 Node (multi node) so we have different approach for these.
- **Steps for 1 node:**
    1. In 1 node cluster we have to sort 10 GB of dataset.
    2. Initially I took one instance of Amazon EC2 c3.large instance of Ubuntu OS and Added storage – of type EBS general purpose (SSD) volume of size 200 GB.
    3. **Install Hadoop** framework on ec2 instance
       Sudo wget *http://apache.mirrors.pair.com/hadoop/common/stable2/hadoop-2.7.2.tar.gz*

    4. Then **Install ssh**
       sudo apt-get install ssh

    5. Then **Install java**
       sudo apt-get install openjdk-7-jdk

    6. Update all configurations file ***core-site.xml, hadoop-env.sh, hdfs-site.xml, yarn-site.xml, mapred-site.xml***

       core-site.xml –
        - This file address Hadoop daemon where the NameNode in cluster along with the IP and port.
        - Includes I/O settings that are common to HDFS and MapReduce.

       hadoop-env.sh –
        - This file used to mention environment variables which affects JDK used by **Hadoop Daemon** (bin/hadoop).
        - Example - $JAVA_HOME, heap size (HADOOP_HEAP).

       hdfs-site.xml –
        - Hdfc-site.sh file conatins the configuration settings for HDFS Daemon, the Name node, The Secondary Name node and data node.
        - Hdfs-site.xml file specify default block replication and permission checking on HDFS

       mapred-site.xml –
        - This file contain configuration settings about job tracker and task tracker.
        - *mapreduce.job.tracker* parameter is hostname and port pair on which job tracker listen for RPC communication.

7. Update the **hosts** and **slave** file to setup 1 node virtual cluster.
   hosts –
   - private ip and public dns of instance.

   slave –
   - public dns of master.

8. **RSA Key generation**
   - *eval "$(ssh-agent)"*
     ssh-keygen -t rsa
     ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@ec2-54-172-180-220.compute-1.amazonaws.com
     chmod 600 ~/.ssh/authorized_keys

9. **Mounting of ESB to instance**
   - *sudo mkfs.ext4 /dev/xvdf*
     sudo mke2fs -F -t ext4 /dev/xvdf
     sudo mkdir /data
     sudo mount /dev/xvdf /data

10. **Format NameNode**
    - *bin/hadoop namenode –format*

11. **Start dfs and yarn**
    - ./start-dfs.sh
    - ./start-yarn.sh
    - Jps - this command you can used to check status of services.

- **Steps for 16 nodes:**
  1. Repeat steps from 2 to 6
  2. Update the **hosts** and **slave** file to setup 16 node virtual cluster.
     In Master node
     hosts –
     - private ip and public dns of instance of master and all the slaves.

     slave –
     - public dns of master and all the slaves.

     In Slave nodes
     hosts –
     - private_ip and public_dns of master and that slave.

     slave –
     - public dns of master and that slave.

  3. **RSA Key generation**
     - RSA key generation has to perform on each and every slave in cluster
     - *eval "$(ssh-agent)"*
       ssh-keygen -t rsa
       ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@ec2-54-172-180-220.compute-1.amazonaws.com

chmod 600 ~/.ssh/authorized_keys

4. **Mounting of ESB to instance**
    - *sudo mkfs.ext4 /dev/xvdf*
      sudo mke2fs -F -t ext4 /dev/xvdf
      sudo mkdir /data
      sudo mount /dev/xvdf /data

5. **Format NameNode**
    - *bin/hadoop namenode –format*

6. **Start dfs and yarn**
    - ./start-dfs.sh
    - ./start-yarn.sh
    - Jps - this command you can used to check status of services.

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

- **Master node** in Hadoop cluster is node which host various storage as well as processing management services.
- Master is critical and if it fails whole system can go down, so there are normally 3 copies present. Where the key services Active NameNode, Standby NameNode, and Resource Manager each one with their server.
- Master responsibilities are like opening, closing and renaming files. Mapping of block to DataNodes
- **Slaves node(s)** executes task upon instructions from the master and handle the data motion between map and reduce.
- Slave node(s) perform operations like read and write whenever they get request from master, operations performs are like block creation, deletions and replications.

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?

- We need to set the unique port number in configuration files because if we use same port number in multiple configuration file you will end up with getting error messages.
- System cannot listen to different types of requests on same port. So conflicting error messages will get generated
- Below is the example of one of error you might face when you use incorrect port number.
- INFO ipc.Client: Retrying connect to server: localhost/127.0.0.1:9000. Already tried 0 time(s).
- ERROR org.apache.hadoop.hdfs.server.namenode.FSNamesystem: FSNamesystem initialization failed.

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

- We can set/change the number of mapper and reducers in *mapred-site.xml.*
- We can either use default number of mapper and or reducer or we can mention maximum number of mapper and reducer.

- Below is the example for setting up 8 mappers and 8 reducers which can be maximum mapper and reducer will get set by task tracker.
- *<property>*
  <name>mapreduce.tasktracker.map.tasks.maximum</name>
  <value>8</value>
  <description>maximum number of mapper: task tracker</description>
  </property>
  <property>
  <name>mapreduce.tasktracker.reduce.tasks.maximum</name>
  <value>8</value>
  <description>maximum number of reduce: task tracker </description>
  </property>

Spark Sort step:

- We have performed this sort on 1 Node and 16 Node (multi node) so we have different approach for these.
- **Steps for 1 node:**
  1. In 1 node cluster we have to sort 10 GB of dataset.
  2. **Install Spark** framework on local machin
     wget www-eu.apache.org/dist/spark/spark-1.6.0/spark-1.6.0-bin-hadoop2.6.tgz
  3. **Export Keys**
     - *export AWS_ACCESS_KEY_ID= XXXXXXXXXXXXXXXXXXXXXXXXXXXX*
     - *export AWS_SECRET_ACCESS_KEY= XXXXXXXXXXXXXXXXXXXXXXXX*
  4. **Initialize the 1 nodes**
     - **./spark-ec2 -k abhijeetkedari -i /home/abhijeet/Downloads/CS553/Assignment2/Spark/abhijeetkedari.pem -s 1 -t c3.large --spot-price=0.03 launch spark_instance**
  5. **Login to master**
     - **./spark-ec2 -k abhijeetkedari -i /home/abhijeet/Downloads/CS553/Assignment2/Spark/abhijeetkedari.pem login spark_instance**
  6. **Mounting of ESB to instance**
     - *sudo mkfs.ext4 /dev/xvdp*
       sudo mke2fs -F -t ext4 /dev/xvdp
       sudo mkdir /data
       sudo mount /dev/xvdp /data

- **Steps for 16 nodes:**
  1. Repeat step from 1 to 3.
  2. **Initialize the 16 nodes**
     - **./spark-ec2 -k abhijeetkedari -i /home/abhijeet/Downloads/CS553/Assignment2/Spark/abhijeetkedari.pem -s 16 -t c3.large --spot-price=0.03 launch spark_instance**
  3. **Login to master**
     - **./spark-ec2 -k abhijeetkedari -i /home/abhijeet/Downloads/CS553/Assignment2/Spark/abhijeetkedari.pem login spark_instance**
  4. **Mounting of ESB to instance**

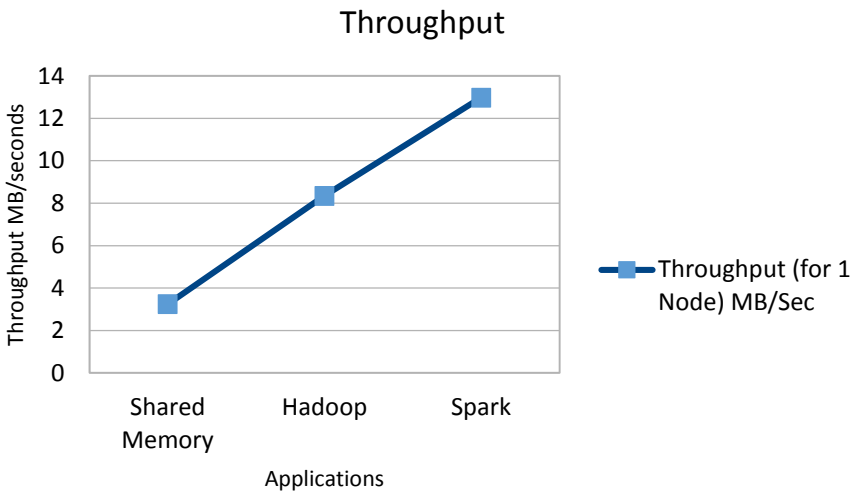
- *sudo mkfs.ext4 /dev/xvdp*
  sudo mke2fs -F -t ext4 /dev/xvdp
  sudo mkdir /data
  sudo mount /dev/xvdp /data

- **Gensort:**
  1. To generate the file of different size I have used gensort tool
     ./gensort -a <Bytes> /data/input
  2. Here <Bytes> represents the number of bytes you required to generate, accordingly each line get specified number of bytes and it get generated in form of Key-value pairs.
- **Valsort:**
  1. To validate the result generated by all the application/system I have used valsort tool
     ./valsort /data/part-r-00000
  2. Here *part-r-00000* represents the output file
- **Head and tail:**
  1. To get the first 10 and last 10 recods from output file to validate the record order I have used below commands
     head -10 part-r-00000 >>/data/head
     tail -10 part-r-00000 >>/data/tail

Performance:

Compare the performance of the three versions of Sort (Shared-Memory, Hadoop, and Spark) on 1 node scale and explain your observations;

System	Throughput (for 1 Node) MB/Sec
Shared Memory	3.245397134
Hadoop	8.347245409
Spark	12.98424751

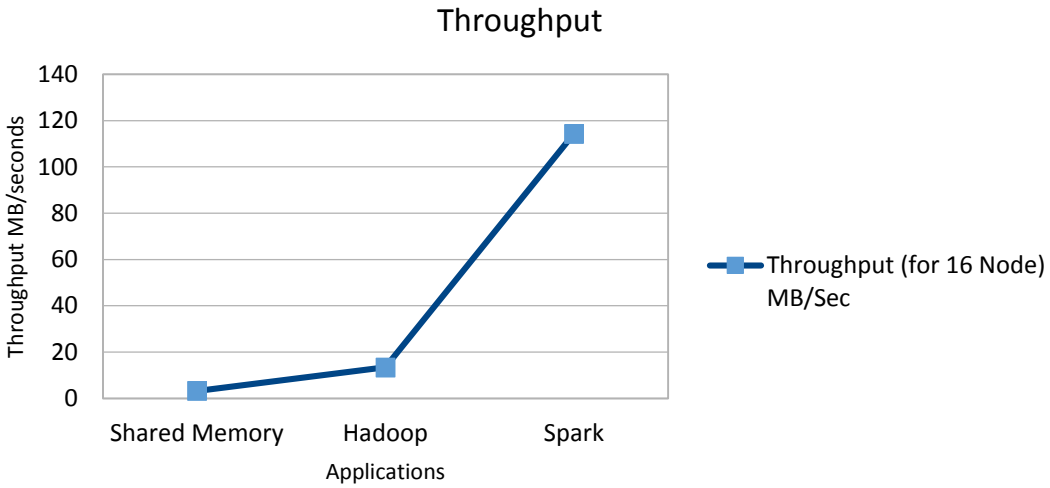


Observation –

- Here I have compared all system performance on 1 node with 10 GB dataset
- Shared-memory has lowest performance compared to Hadoop and spark.
- Because in first part of sorting records in Shared-memory I have not used the memory to its capacity also I have kept the fixed size of chunk to read every time from disk.
- In second part of sorting records I have read a records line by line so it is very time consuming process.
- In comparing between Hadoop and spark, spark has better performance because spark process data in-memory, whereas Hadoop revert back to disk with map / reduce action.
- This difference is lower compared to single node shared-memory and multi node Hadoop and multi node spark.

Compare the Shared-Memory performance of 1 node to Hadoop and Spark Sort at 16 node scales and explain your observations

System	Throughput (for 16 Node) MB/Sec
Shared Memory	3.245397134
Hadoop	13.41874274
Spark	114.3422974



Observation -

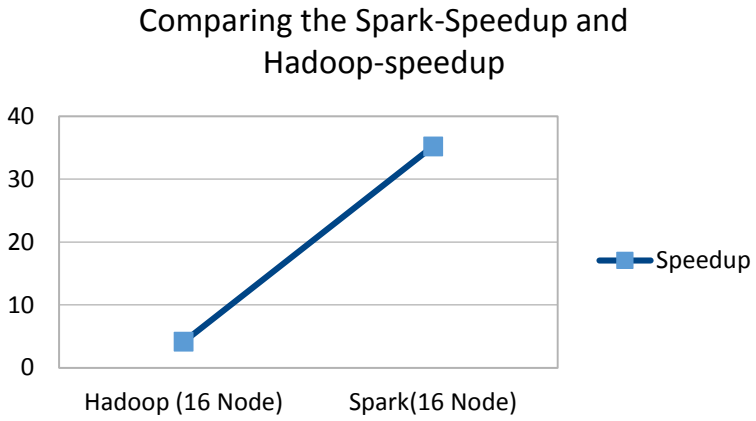
- Here I have compared throughput of Shared-memory at 1 node (10 GB) with Hadoop and Spark system at 16 nodes with 100 GB dataset.
- Shared-memory has lowest performance compared to Hadoop and spark.
- Because in first part of sorting records in Shared-memory I have not used the memory to its capacity also I have kept the fixed size of chunk to read every time from disk.
- In second part of sorting records I have read a records line by line so it is very time consuming process.
- In comparing between Hadoop and spark, spark has better performance because spark process data in-memory, whereas Hadoop revert back to disk with map / reduce action.
- This difference is higher compared to single node shared-memory, Hadoop and spark.
- As mentioned spark work in memory and unlike the Hadoop do disk operations with increased in number of nodes Hadoop performing more/same number of operation as single node but with multiple node in-memory has been increased and spark performance has significantly improved.

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

1. **one speedup (speedup-shared-memory) should be relative to the Shared-memory Sort performance (note that you might get a speedup less than 1);**

System	Hadoop (16 Node)	Spark(16 Node)
Speedup	4.134699755	35.23214345

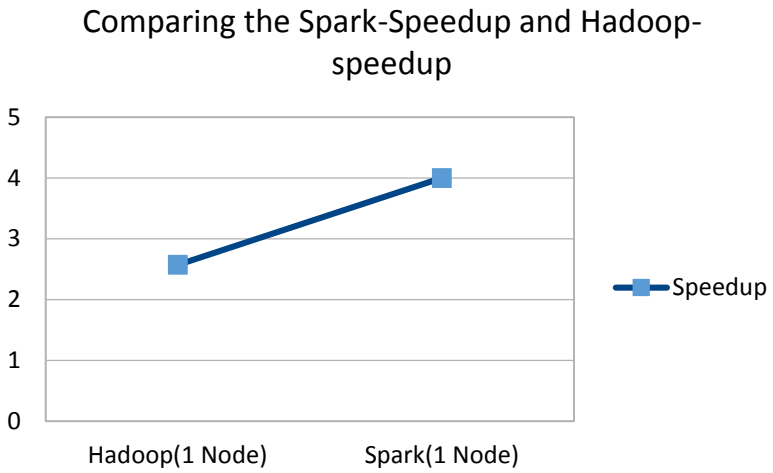


Observation –

- As mentioned in first table I have throughput for all system, then I have computed the speedup factor for Hadoop and spark relative to shared memory, which is represented in second table.
- According to speedup calculated we can say that Spark has higher speedup factor than hadoop relative to Shared-memory. This is mainly because of RDD.
- Spark do operations in-memory, also Iterative computations on the same block of data, Spark maintains the upper hand.
- When considering single-pass ETL tasks like data integration or transformation, MapReduce is the better option.

One speedup (speedup – shared- memory) should be relative to the Shared-memory Sort performance

System	Hadoop(1 Node)	Spark(1 Node)
Speedup	2.572025876	4.000819306

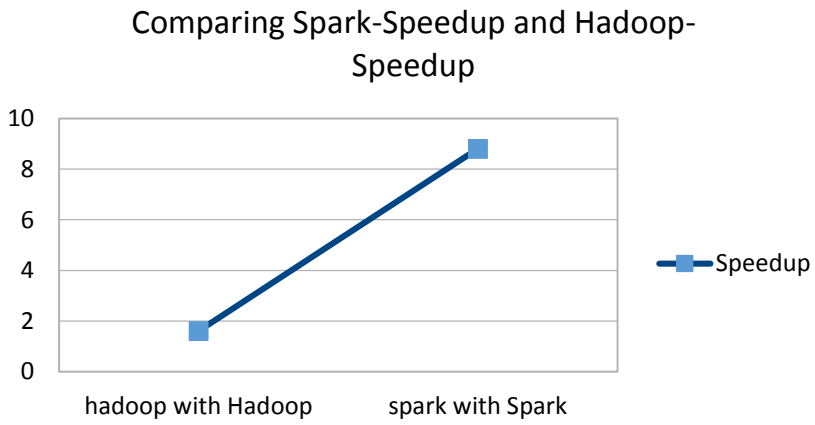


Observation –

- As mentioned in first table I have throughput for all system, then I have computed the speedup factor for Hadoop and spark relative to shared memory, which is represented in second table.
- According to speedup calculated we can say that Spark has higher speedup factor than Hadoop relative to Shared-memory.

2. **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).**

System	Hadoop with Hadoop	Spark with Spark
Speedup	1.60756538	8.806232113



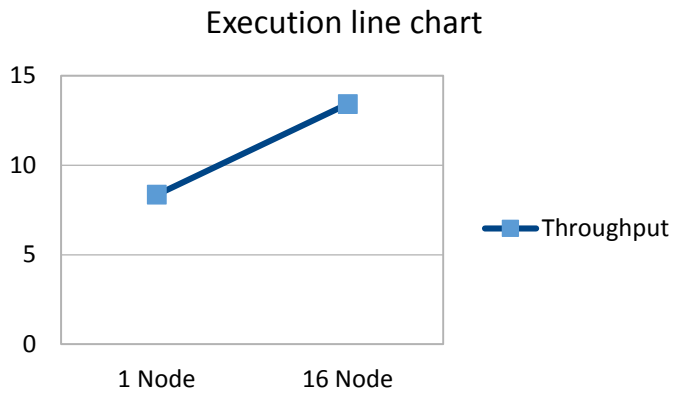
Observations -

- As mentioned in both table, I have mentioned throughput for Hadoop and Spark system for 16 nodes and 1 node, then I have computed the speedup factor for Hadoop and spark relative to Spark or Hadoop performance respectively, which is represented in third table.

- Apache Spark performs better where there's enough memory to fit all data e.g. using dedicated clusters.
- Hadoop MapReduce was created for use when memory space is limited, or when you need it to run alongside other services.
- According to speedup calculated we can say that Spark has higher speedup factor than Hadoop.

Hadoop –

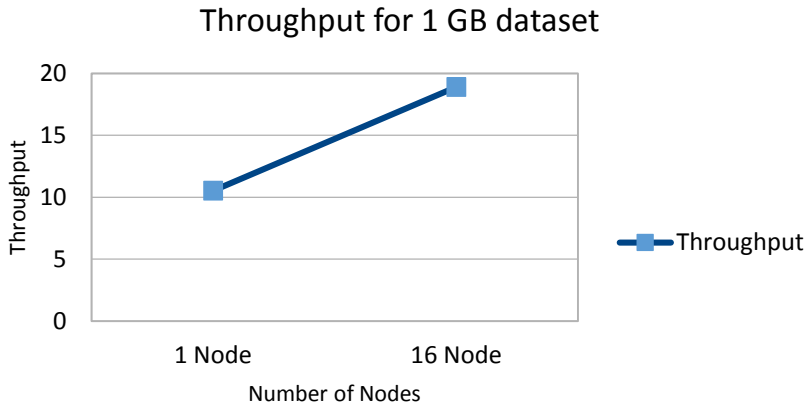
Nodes	Throughput
1 Node (10 GB)	8.34724541
16 Node (100 GB)	13.4187427



Observation –

- With increase in number of nodes task get divided well and throughput of system increases

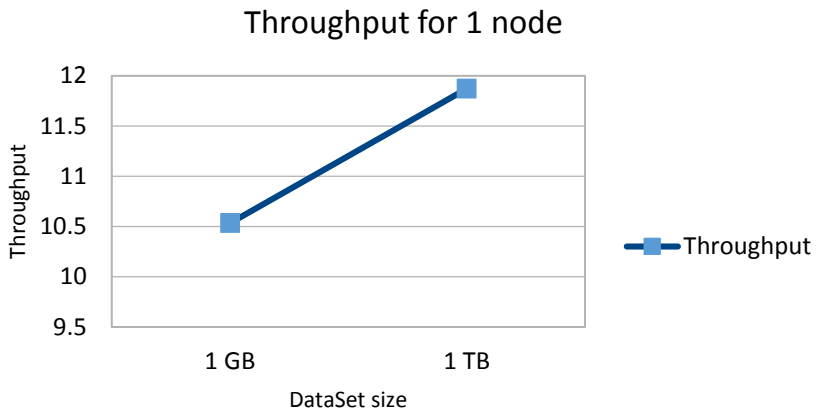
Number of Nodes	Throughput
1 Node (1 GB)	10.53263537
16 Node (1 GB)	18.90680834



Observation –

- With increase in number of nodes task get divided well and throughput of system increases

Dataset	Throughput
1 GB	10.53263537
1 TB	11.87052209

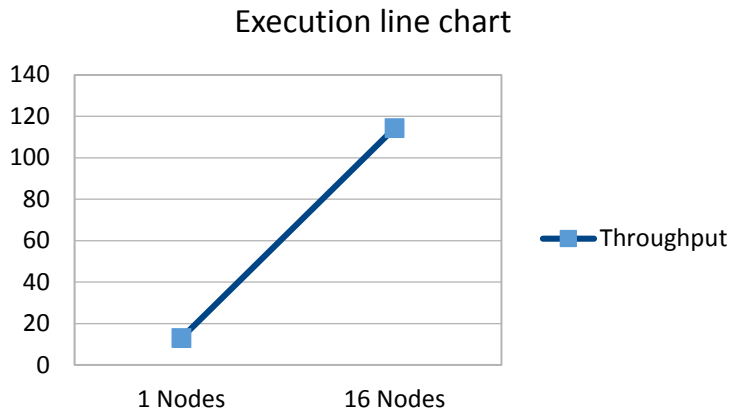


Observation –

- With increase in number of nodes task get divided well and throughput of system increases

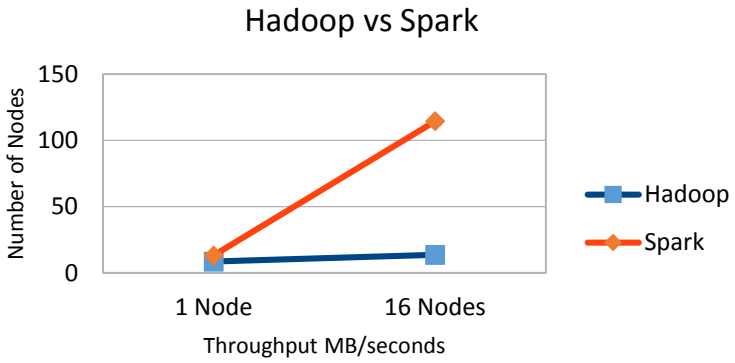
Spark -

Nodes	Throughput
1 Nodes	12.9842475
16 Nodes	114.342297



Hadoop vs Spark –

Nodes	Hadoop	Spark
1 Node	8.34724541	12.9842475
16 Nodes	13.4187427	114.342297

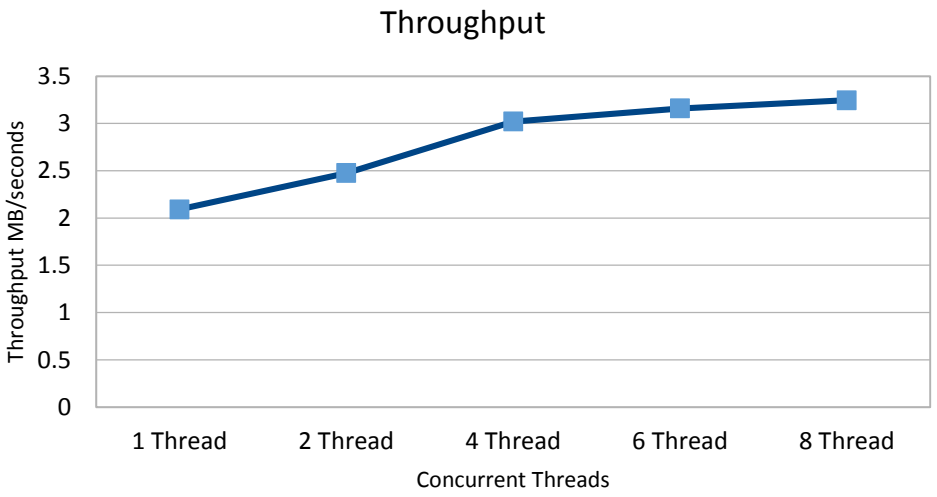


Observation-

- In compared to Hadoop and Spark Hadoop perform same as Spark at number of nodes 1 but if we increase the number of nodes to 16 Spark performance increases significantly .

Shared memory –

	Throughput				
Nodes	1 Thread	2 Thread	4 Thread	6 Thread	8 Thread
1	2.09083595	2.47518442	3.02084321	3.15921329	3.24539713



Observation –

- As we keep increase in threads throughput keeps increasing
- But after a certain point like 6 or 8 threads it has start keeping same performance.
- If we keep increase in number of threads performance actually will start degrading as thread management become more overhead for operating system.

Conclusion –

- Hadoop MapReduce use to write back to disk with every action of map and reduce.
- Spark has advantage of processing data in-memory so spark has better performance compared to Hadoop.
- Though Spark has higher processing speed than Hadoop, this comes with the cost of larger memory size.
- So Spark required higher memory/cache allocation to load data and process it. This limits to large data and start degrading the performance.
- There is termination of process on completion of job in Hadoop.
- Spark always perform better If there are iterative computation on the same block of data.
- Whenever there is Single-pass ETL tasks example Transformation, Hadoop MapReduce perform better.
- Apache Spark performs better where there's enough memory to fit all data e.g. using dedicated clusters. Hadoop MapReduce was created for use when memory space is limited, or when you need it to run alongside other services.

Which seems to be best at 1 node scale?

- Spark is performing better than Hadoop and shared-memory on 1 node scale.
- Spark has highest throughput value that is 12.98424
- Spark is performing slightly better than Hadoop as it has only one node to take advantage of in-memory.

How about 16 nodes?

- Spark is performing better than Hadoop and shared-memory on 16 node scale.
- Spark has highest throughput value that is 114.3422
- Spark is performing far better than Hadoop as it has many nodes to take advantage of in-memory.

Can you predict which would be best at 100 node scale?

- Spark would be best at 100 node scale.
- As we keep increase in nodes we keep increase total in-memory for cluster and Spark perform in-memory operation so it will increase throughput of overall system.
- Spark performs better where there's enough memory to fit all data Hadoop was created for use when memory space is limited, or when you need it to run alongside other services.

How about 1000 node scales?

- At 1000 node scales Spark would be a way best compared to Hadoop.
- As we keep increase in nodes we keep increase total in-memory for cluster.
- Apache Spark performs better where there's enough memory to fit all data e.g. using dedicated clusters. Hadoop MapReduce was created for use when memory space is limited, or when you need it to run alongside other services.

Compare your results with those from the Sort Benchmark [9], specifically the winners in 2013 and 2014 who used Hadoop and Spark

In 2014 - Winner Apache Spark

In 2013 - Winner Hadoop

- From result I can observe that in 2013 Hadoop took 2100 nodes for 102.5 TB and time taken was 4328 seconds
- And in 2014 Apache Spark took 207 nodes where data was almost same size of 100 TB and time taken was 1406 seconds.
- So from these observations even we cannot say it by sure Apache Spark perform better as we are not sure about the underlying hardware and environment for these two, but according to results we can say that Spark is performing better and in my experiment I have got the same trends. Spark has better performance over Hadoop.

What can you learn from the CloudSort benchmark

- CloudSort benchmark helps in drive innovation in public cloud for supporting IO intensive task.
- In future this will also help in searching out the most efficient sort implementation.