PA-2 REPORT

CS-553 - CLOUD COMPUTING
FALL 2017

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

Contents

Problem 2
Solution 2
Run Time Environment Settings 2
Installation 3
Java 3
Shared Memory 3
Hadoop 3
Spark 4
Challenges 5
External Sorting 5
Hadoop 5
Spark 5
Performance Evaluation 5
Overview on CloudSort Benchmark 11
Comparison Results with Sort Benchmark 11
Execution Screenshots 12
Shared Memory 12
128GB Dataset 12
1TB Dataset 14
Hadoop Single Node 16
128GB Dataset 16
1TB Dataset 19
Hadoop 8-Node Cluster 23
1TB Dataset 23
Spark Single Node Cluster 27
128GB Dataset 27
1TB Dataset 32
Spark 8 Node Cluster 34

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

Problem

- The problem is to sort a large dataset of varying size 128GB and 1TB on different distributed environment as well as on standalone platform. Here, distributed environment refers to Hadoop and Spark Framework. For Standalone Execution, External Sorting Approach is required to adapt. Running platform on which program should be executed, is Amazon AWS. For TeraSort of 128GB, it is required to execute on i3.large instance and for execution of 1TB TeraSort, it is required to run on i3.4xlarge instance from amazon aws.

Solution

- To generate a large dataset, gensort is used which generated records having first 10-byte as key and rest as value. After sorting is done, valsort is used to check whether all records are in order or not.

- For Shared Memory, sorting is done with help of merge-sort algorithm. According to available free memory from JVM, dataset is divided into several chunks of fixed size. Each of this chunk is kept in memory and sorting has been performed on in-memory chunks. First 10 bytes of each line is considered as key and sorted upon that key. Multithreading is achieved on above steps where each thread works on unique chunk to divide and sort. Once sorting is achieved, all the chunks are merged and sorted.

- For Hadoop Framework, we have used MapReduce for sorting large dataset. During Map Phase, each mapper generates a key of first 10 bytes of records. The result of map phase is sorted collections of keys in ascending order. Here no intermediate runs from map step are been stored and hence reduce phase does no work. This is disadvantage of Hadoop framework. We executed TeraSort on both single node as well as multiple nodes.

- For Spark Framework, we must put input data in hdfs directory as spark takes input from hdfs. Each record from the input file is read and a key of first 10 byte is generated. Using SortByKey function, each of the records are sorted in ascending order. Once results are sorted, all the records are stored in hdfs output directory specified. Spark is faster than Hadoop because it performs execution in-memory.

Run Time Environment Settings

➔First, we must mount extra space as intermediate shuffle data creates a lot of temporary files. This is achieved by raiding external memory using raid0. When mount_generate_raid.sh script is executed, additional 2 volumes of EBS and one SSD is mounted at mount point defined. Here, EBS volumes are pre-configured while creating instances.

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

Installation

- Deployed on UBUNTU

Java

- Executing below commands will install **java version 1.8.0_151.**

  sudo -E apt-get purge oracle-java8-installer -y
  sudo -E apt-get install oracle-java8-installer –y

Shared Memory

- 128GB: Number of Threads = 2 as Total VCPUs are 2 for i3.large instance. Max Heap Size for JVM is 12GB as total 128GB memory and Min Tmp File Size will be 512MB size.
- 1TB: Number of Threads = 16 as Total VCPUs are 16 for i3.4xlarge instance. Max Heap Size for JVM is 120GB as total 128GB memory and Min Tmp File Size will be 481MB size.

Hadoop

- All the installation steps for Hadoop are included in hadoop_install.sh
- Below commands will install **hadoop version 2.7.1.**

  sudo wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.1/hadoop-2.7.1.tar.gz
  sudo tar xzf hadoop-2.7.1.tar.gz

- Also, we are required to set environment variables for hadoop as below:

  export HADOOP_HOME=/usr/local/hadoop
  export HADOOP_MAPRED_HOME=/usr/local/hadoop
  export HADOOP_COMMON_HOME=/usr/local/hadoop
  export HADOOP_HDFS_HOME=/usr/local/hadoop
  export YARN_HOME=/usr/local/hadoop
  export HADOOP_COMMON_LIB_NATIVE_DIR=/usr/local/hadoop/lib/native
  export JAVA_HOME=/usr/
  export
  PATH=$PATH:/usr/local/hadoop/sbin:/usr/local/hadoop/bin:/usr/bin:/bin:/opt/spark/bin

- Changes required in configuration files for Single Node Cluster
  1) core-site.xml: Add public DNS of the node.

2) hdfs-site.xml: Add block size and point dir of namenode and data node to mount point.
3) mapred-site.xml: Add number of reducers and provide temporary directory for mapred.
4) yarn-site.xml: Add disk-health-checker-max-disk-utilization property with 98.5% value to avoid having unhealthy node issue.

- Additional changes in configuration files for Multi Node Cluster
  1) Generate RSA public key and add it to authorized keys on master node.

     ssh-keygen -t rsa -P ""
     cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

  2) Send key to all slave node to create password less ssh.
  3) slaves **–** Add own public DNS if slave else add private IP of master node followed by DNS of all slaves.
  4) masters- Add public DNS if master.
  5) hdfs-site.xml- Check if replication factor is 1 i.e. replication turned off.

Spark

- All the installation steps for Hadoop are included in install_spark.sh
- Below command will download **spark-2.2.0**
  wget https://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-hadoop2.7.tgz

- Once spark tar is downloaded, untar spark.
  sudo tar zxvf spark-2.2.0-bin-hadoop2.7.tgz -C /opt

- Set environment variable to bashrc file
  export SPARK_HOME=/opt/spark
  source ~/.bashrc

- We have added one property in file which is at location /opt/spark/conf/spark-defaults.conf.template. After adding below line, all intermediate temp will be stored in file specified below.
  spark.local.dir /mnt/raid/spark_temp

- Also, we have to update one more file which is at location /opt/spark/conf/ log4j.properties.template. Find log4j.rootCategory=INFO file and replace INFO by ERROR. This will avoid creating unnecessary temp logs except error logs.

Challenges

External Sorting

- Due to less heap size of JVM, there was always an error of "Memory Out of Bound Exception" while running for Big Data-Set. Hence, we increased heap size by specifying Xmx value during execution.
- Also, our execution generates large number of small tmp files. So, we must push the limit of generating number of files to 65536 to avoid error of "Too many files open".

Hadoop

- There was always issue of "unhealthy node" due to excessive amount of temp file generate. It required a considerable amount of external memory to be raided and configured in such a way that all the temp files must be stored at mount point.

Spark

- There was memory issue due to excessive shuffle data generated. So, we changed log4j.properties files in such a way that all error logs are only stored instead of all the logs.

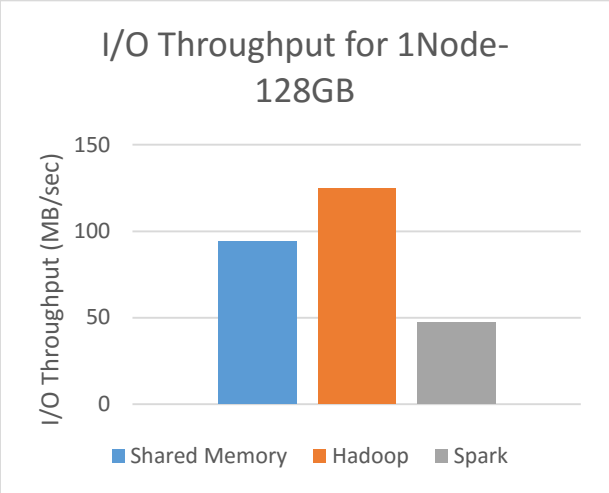
Performance Evaluation

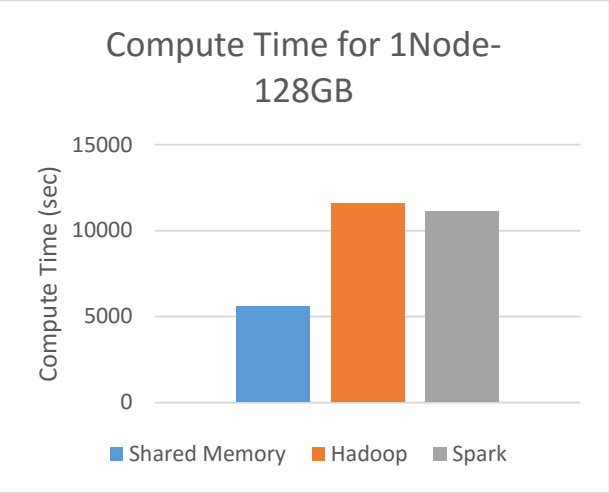
➔Below section covers performance results of TeraSort on 1TB and 128 GB Dataset for shared memory, Hadoop, and spark. We analyzed and compared results of Shared Memory, Hadoop, and Spark Configuration for 128 GB and 1TB Dataset. Also, we compared results of 1 node cluster vs 8-node cluster.

- **Comparison Results and Running Time Factors for 128 GB Dataset**

1) Instance-Type: i3.large (2 VCPUs, 15.25 GB RAM, 475 SSD)
2) EBS Volume Added: 600GB
3) No of Threads used for Shared Memory Computation: 2
4) Number of Reducers for Hadoop: 2
5) Block Size: 1GB
6) Comparison:

	Shared Memory	Hadoop	Spark
Compute Time(Seconds)	5580.11	11580.18	11109.9
Data Read(GB)	256	639	322
Data Write(GB)	256	769	194
I/O Throughput(MB/Sec)	93.95657075	124.505146	47.559744





7) Conclusion:
→From above graph, it can be inferred that shared memory takes much less time as compared to Hadoop and spark. The reason behind such behavior is that number of I/Os for shared memory is less as compared to Hadoop and spark as it writes to raid 0 level instead of HDFS and hence enhancing performance much faster for shared memory. Also, throughput for Hadoop is higher than spark and shared memory. But still Hadoop takes much I/Os to disk which degrades its performance. On the other hand, spark takes much less I/Os to disk as compared to Hadoop. Hence, performance for spark is better than Hadoop as it performs in-memory computation. So, we can conclude that disk is bottleneck for spark as it is only efficient with in-memory computation. Also, number of threads considered for shared memory to make it more efficient, is 2 which is equal to cores of instance

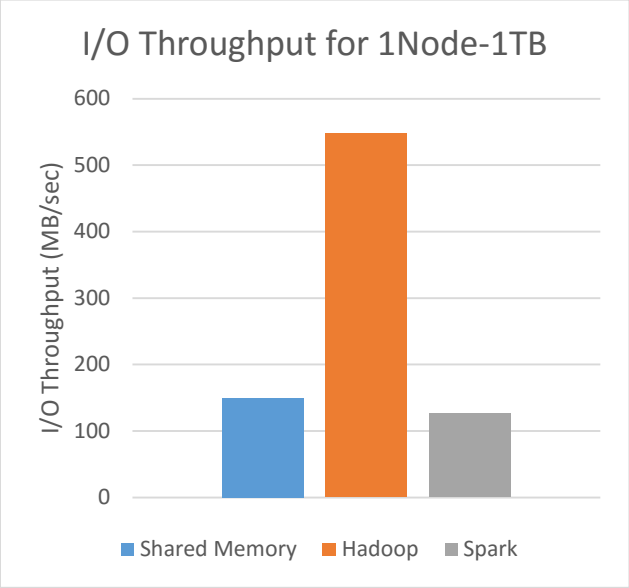
➢ Comparison Results and Running Time Factors for 1 TB Dataset

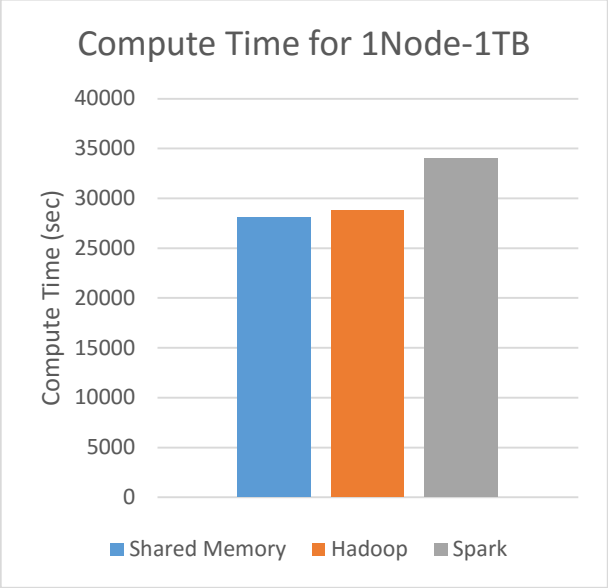
1) Instance-Type: i3.4xlarge (16 VCPUs, 122 GB RAM, 2 x 1900 SSD)
2) EBS Volume Added: 3TB
3) No of Threads used for Shared Memory Computation: 16
4) Number of Reducers for Hadoop: 8
5) Block Size: 1GB
6) Comparison:

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

	Shared Memory	Hadoop	Spark
Compute Time(Second)	28059.6	28800	33987
Data Read(GB)	2048	7182.9	2613.8
Data Write(GB)	2048	8227.28	1589.9
I/O Throughput(MB/Sec)	149.478396	547.9175111	126.654



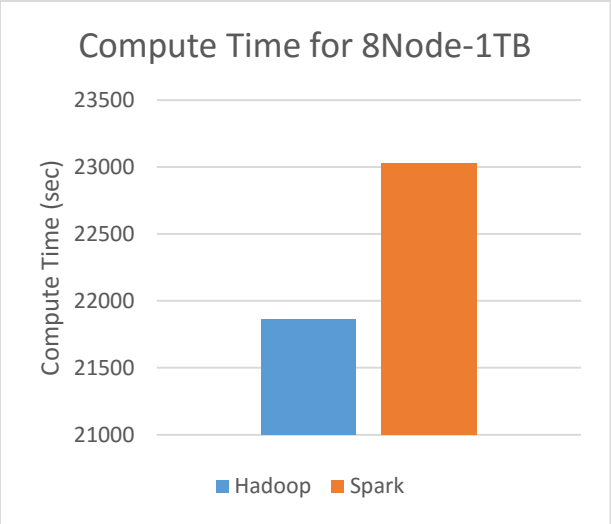


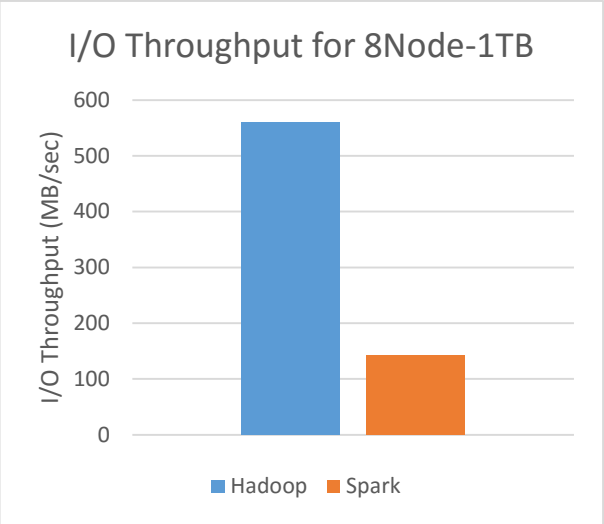
7) Conclusion:

→ From above graph, it can be inferred that shared memory takes much less time as compared to Hadoop and spark. The reason behind such behavior is that number of I/Os for shared memory is less as compared to Hadoop and spark as it writes to raid 0 level instead of HDFS and hence enhancing performance much faster for shared memory. Also, throughput for Hadoop is higher than spark and shared memory. But still Hadoop takes much I/Os to disk which degrades its performance. On the other hand, spark takes much less I/Os to disk as compared to Hadoop. Hence, performance for spark is better than Hadoop as it performs in-memory computation. Also, number of threads considered for shared memory to make it more efficient, is 16 which is equal to number of cores of instance. Hence **Shared Memory** gives best performance for 1 Node Cluster.

➢ Comparison Results for 8 Node Cluster

	Hadoop	Spark
Compute_Time(Second)	21867	23030.7
Data Read(GB)	5933.69	2113.7
Data Write(GB)	6056.47	1089
I/O Throughput(MB/Sec)	561.4818603	142.3997013





Conclusion:

→From above graph, it can be clearly inferred that compute time for Hadoop is much less than spark. Though spark takes much time for computation, only 10% of total time was taken to sort the data and remaining time was taken to write data to disk. This concludes that even though spark has much computation time, it is still faster than Hadoop for sorting operation. Hence **Spark** gives best performance for 8 Nodes.

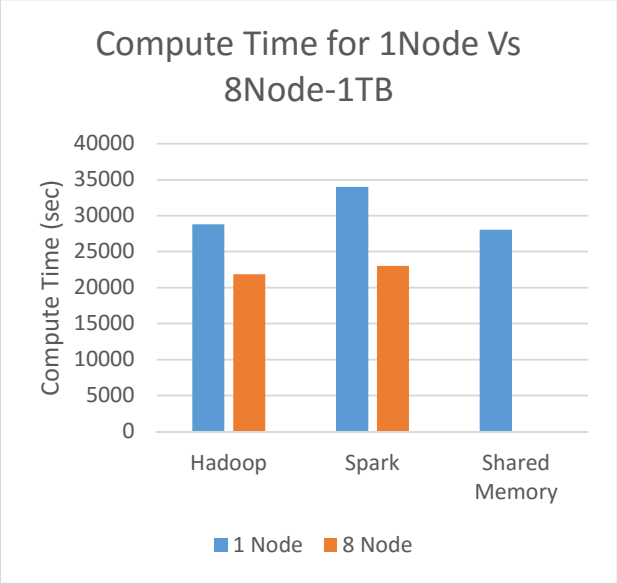
➢ Comparison Results for 8 Node Cluster vs 1 Node Cluster

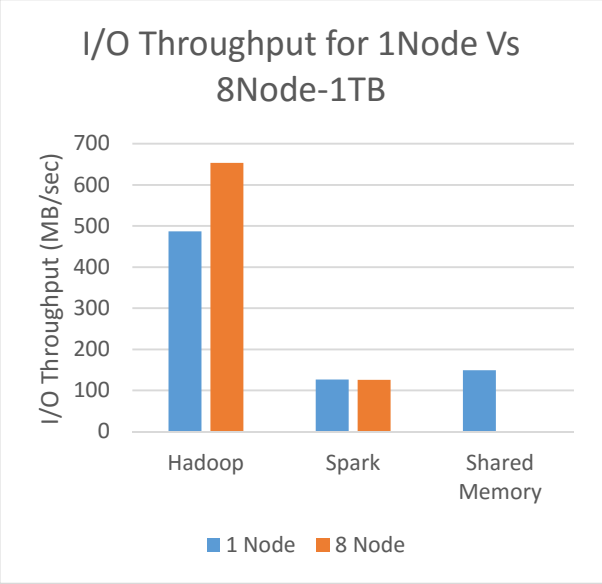
	Hadoop	Spark	Shared Memory
1 Node	28800	33987	28059.6
8 Node	21867	23030.7	N/A

	Hadoop	Spark	Shared Memory
1 Node	487.03	126.524	149.47
8 Node	653.77	125.98	N/A

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



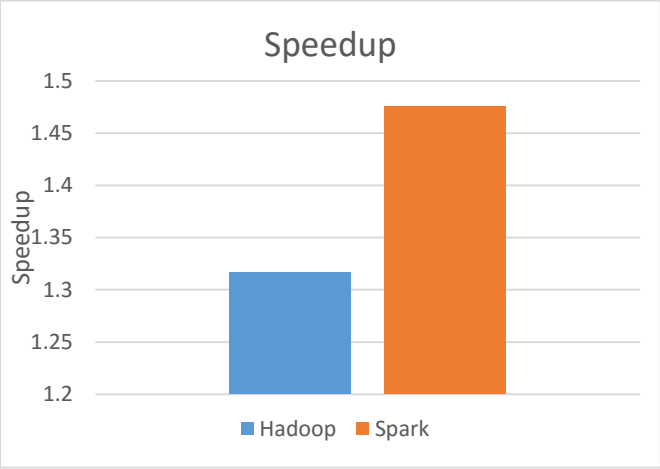


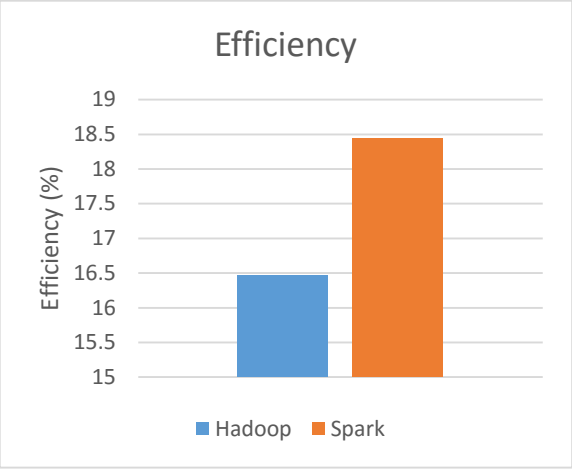
Conclusion:

→Above graph shows comparison results of compute time and throughput between 1 Node and 8 Node for 1 TB dataset of shared memory, Hadoop and spark configuration. It is clearly inferred that compute time for each configuration is less for 8 node cluster than single node cluster. Hence, for each configuration 8 nodes computation gives better performance as workload is divided among nodes in cluster.

→Also, as you scale up on number of nodes to 100, spark will give best performance. The reason for such behavior is, spark computes data in-memory and Hadoop always compute on intermediate data stored on disk. So as nodes increases, available memory for spark to do computation increases. Hence it computes much faster than Hadoop and shared memory. The same behavior will work if cluster scales upto 1000 nodes.

- **Speedup and Efficiency**





→Speedup = Compute Time for 1 Node / Compute Time for 8 Node
→Efficiency = Speedup / number of nodes

Conclusion:
→Above graphs justifies all our above argument for comparison true as speedup and efficiency for spark is more than Hadoop. Hence spark gives better performance than Hadoop.

Experiment (instance/dataset)	**Shared Memory** **TeraSort**	**Hadoop** **TeraSort**	**Spark** **TeraSort**	**MPI** **TeraSort**
Compute Time (sec) [1xi3.large 128GB]	5580.11	11580.18	11109.9	N/A
Data Read (GB) [1xi3.large 128GB]	256	639	322	N/A
Data Write (GB) [1xi3.large 128GB]	256	769	194	N/A
I/O Throughput (MB/sec) [1xi3.large 128GB]	93.95	124.50	47.55	N/A
Compute Time (sec) [1xi3.4xlarge 1TB]	28059.6	28800	33987	N/A
Data Read (GB) [1xi3.4xlarge 1TB]	2048	7182.9		N/A
Data Write (GB) [1xi3.4xlarge 1TB]	2048	8227.28	1589.9	N/A
I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]	149.47	547.91	126.654	N/A
Compute Time (sec) [8xi3.large 1TB]	N/A	21867	23030.7	N/A
Data Read (GB) [8xi3.large 1TB]	N/A	5933.69	2113.7	N/A
Data Write (GB) [8xi3.large 1TB]	N/A	6056.47	1089	N/A
I/O Throughput (MB/sec) [8xi3.large 1TB]	N/A	561.481	142.399	N/A
Speedup (weak scale)	N/A	1.31	1.47	N/A
Efficiency (weak scale)	N/A	16.46%	18.44%	N/A

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

Overview on CloudSort Benchmark

➔ Cloudsort benchmark is a new total-cost ownership (TCO) benchmark which is based on implementing efficient external sort along with achieving minimum cost for cloud platforms. This benchmark mainly focuses on achieving desired cost for cloud environments having intensive IO workloads. Traditionally proposed TCO were not able to cover costs dimensions such as management and maintenance cost. But, after introduction of IAAS, major vendors were capable of offering on-demand compute, network and storage at desired scale with amortized pricing covering overall costs of demanded resources. Cloudsort explains efficiency of external sort from a TCO perspective. It focuses of finding minimum cost for sorting fixed number of records on any public cloud. It was aiming for transition to public cloud and was more efficient because of its extensive IO computation capabilities. Hence external sort was considered as base for estimating TCO as external sort has highly intensive IO workloads. There are many several reason explained about why public cloud was taken into consideration. Most of advantages for considering public cloud was based on accessibility and affordability of public cloud for huge enterprises. Also, many ground rules were proposed which clearly states nature of public cloud to be considered. Hence, this benchmark not only focuses on building platforms with efficient usage of resources, but also find the most efficient sort implementations from TCO perspective.

Comparison Results with Sort Benchmark

	Spark	*Hadoop*
Size of Dataset	100TB	102.5TB
# of Nodes	207	2100
Memory	244GB	64GB
SSD	8x800GB	12x3TB
Processor	32 vCores - 2.5Ghz Intel Xeon E5-2670 v2	2 2.3Ghz hexcore Xeon E5-2630
Author	Databricks	Yahoo! Inc
Computation Rate	4.27TB/min	1.42TB/min

→Above table states sorting configuration and results on 100TB data for different framework Spark and Hadoop. It is clearly inferred that Spark benchmark was performed while considering more memory as spark computes in-memory, whereas hadoop benchmark was computed while keeping disk in mind. Also, computation rate for spark is more compared to hadoop. Hence, all our above arguments of supporting spark rather than hadoop for computation was true.

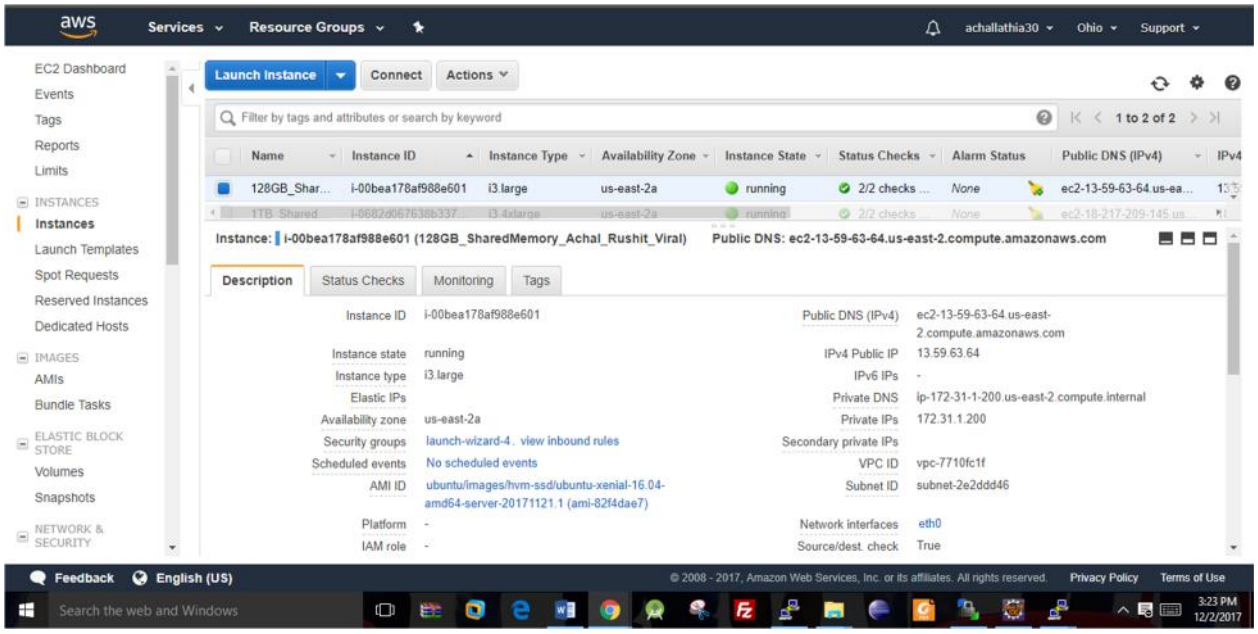
Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

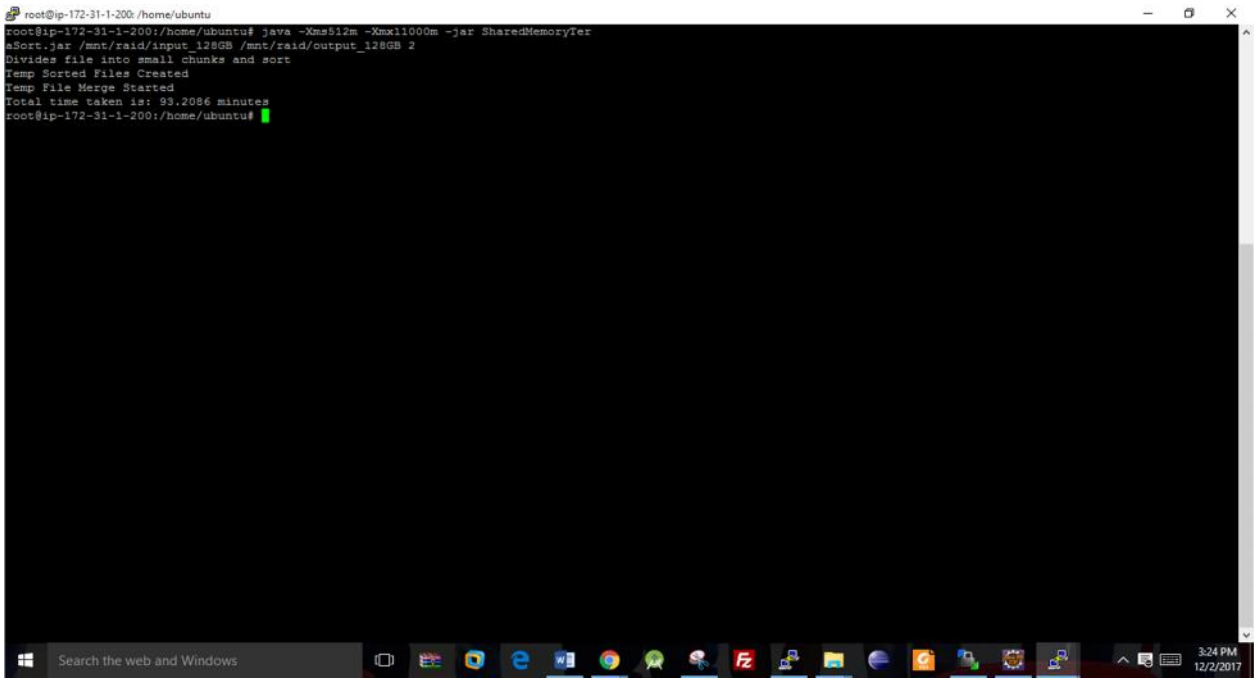
Execution Screenshots

Shared Memory

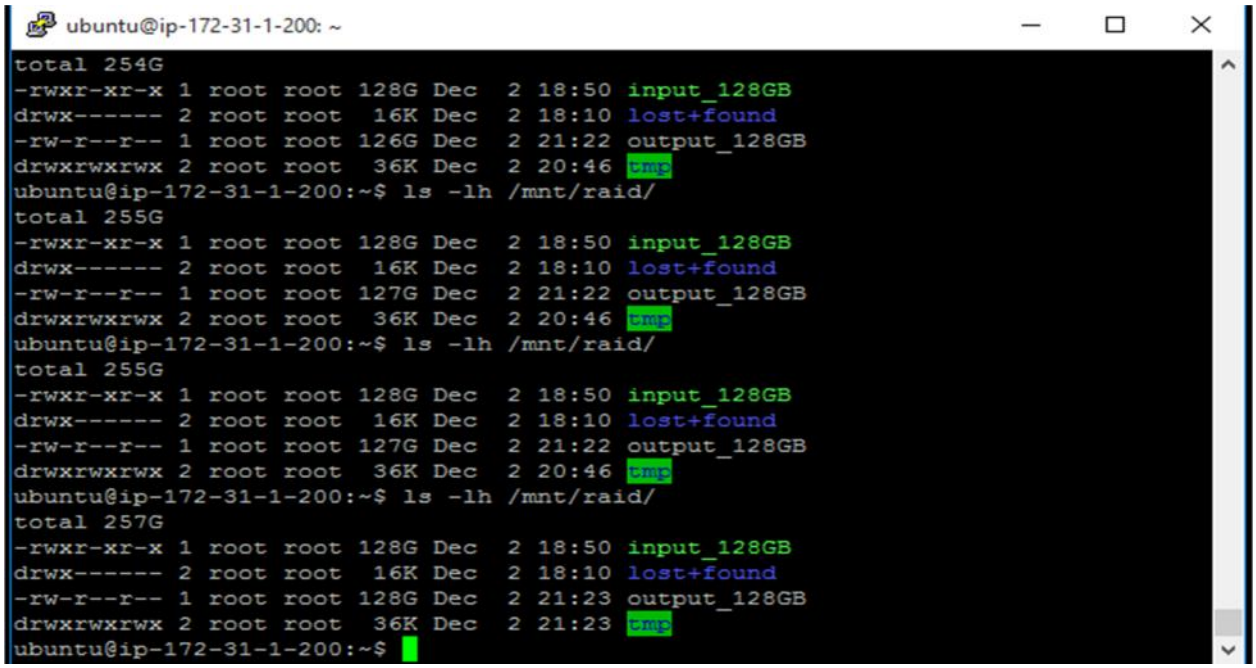
128GB Dataset



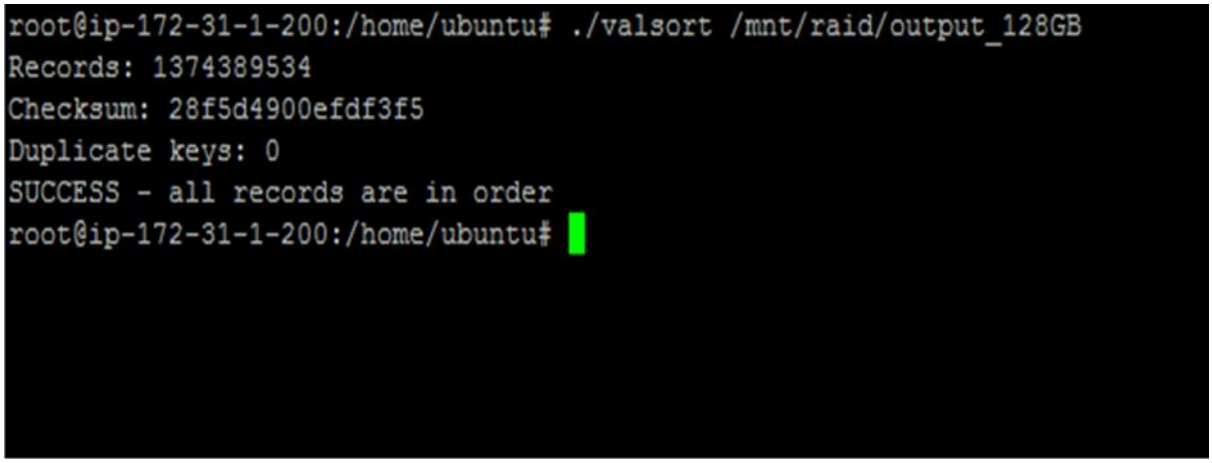
-Above screenshot is cluster information created for this experiment



-Above screenshot is command used to run the SharedMemoryTeraSort.jar file



-Above screenshot is directory information at mnt/raid path

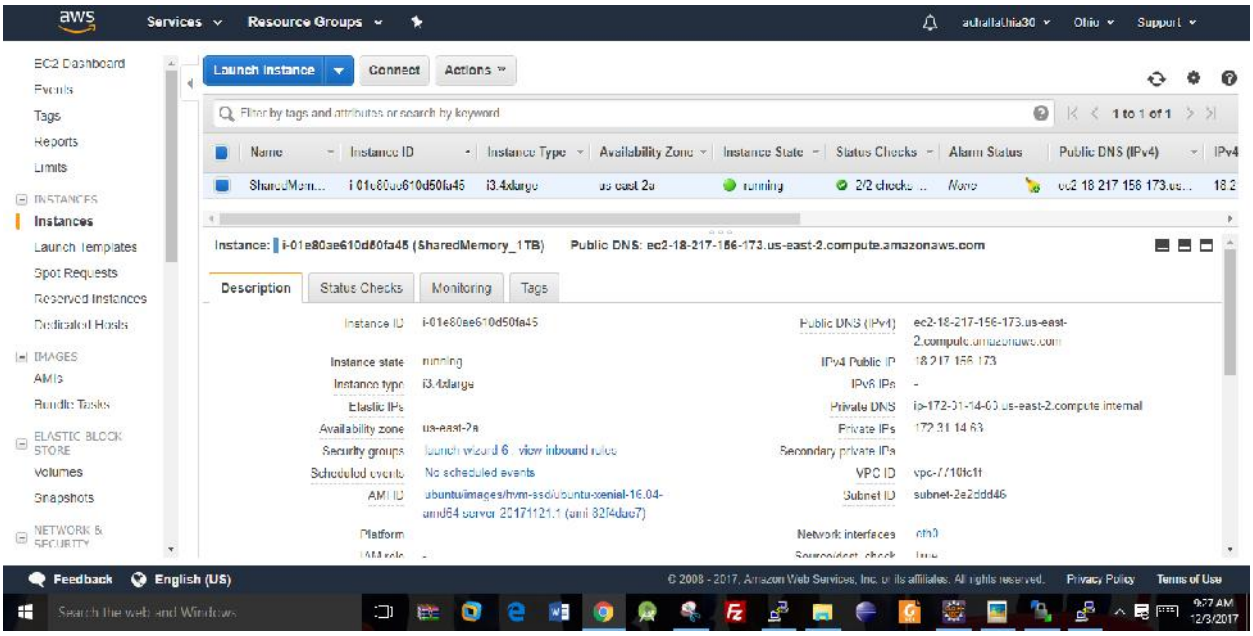


-Above screenshot is command for running valsort

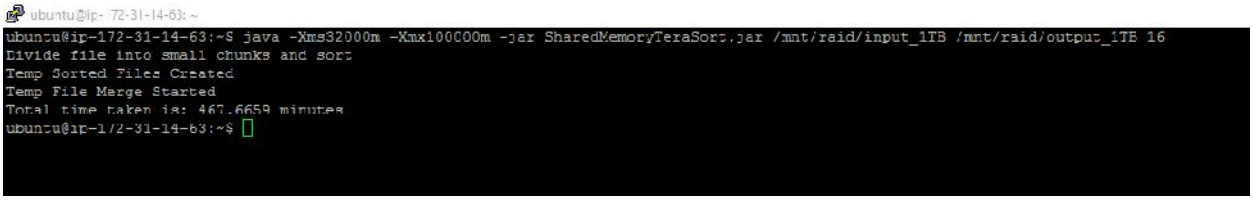
Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

1TB Dataset



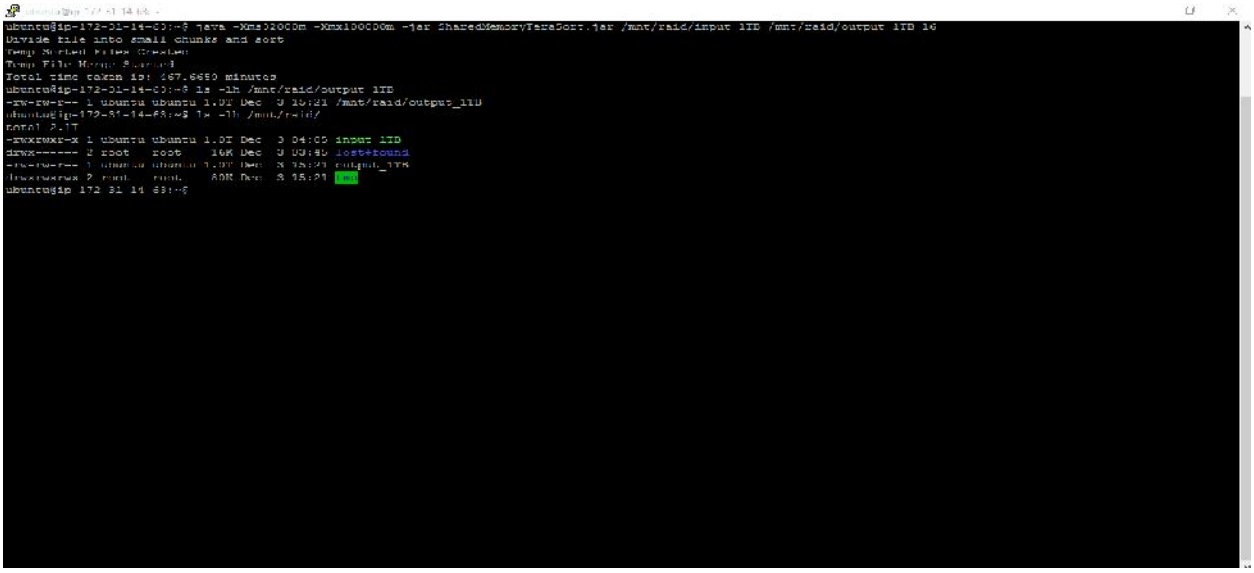
-Above screenshot is cluster information created for this experiment



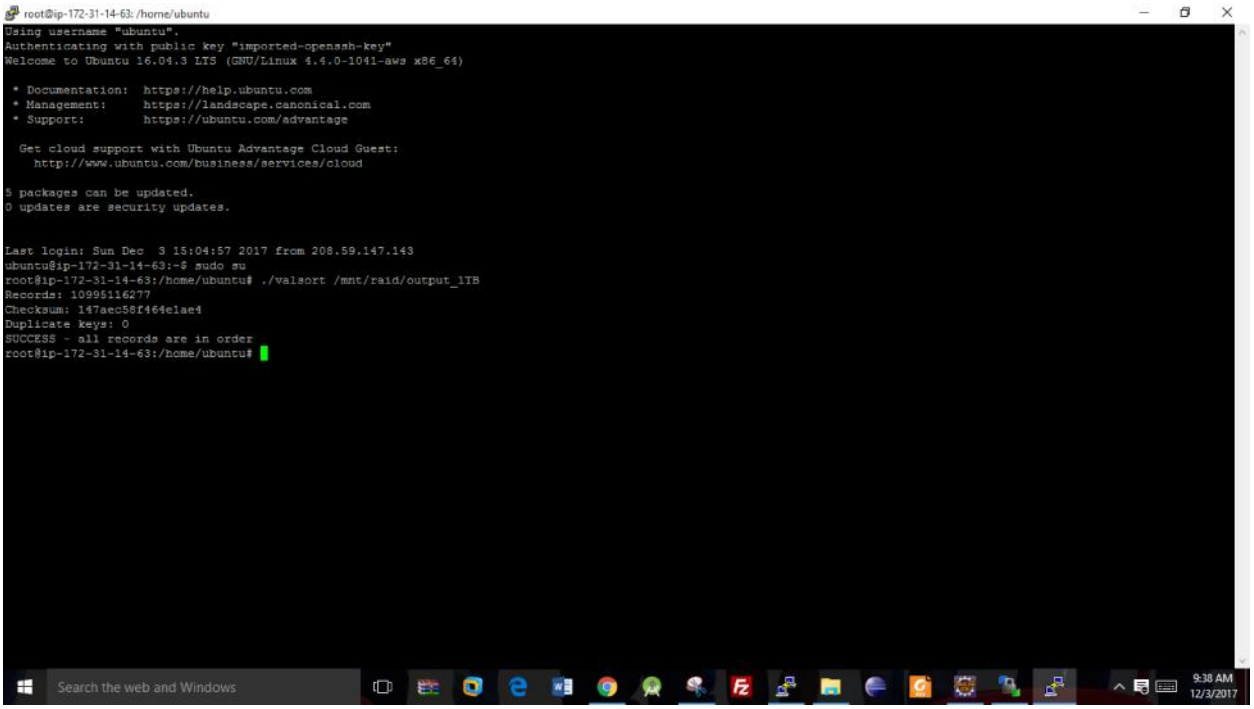
-Above screenshot is command used to run java SharedMemoryTeraSort.jar file

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



-Above screenshot is directory information at mnt/raid path



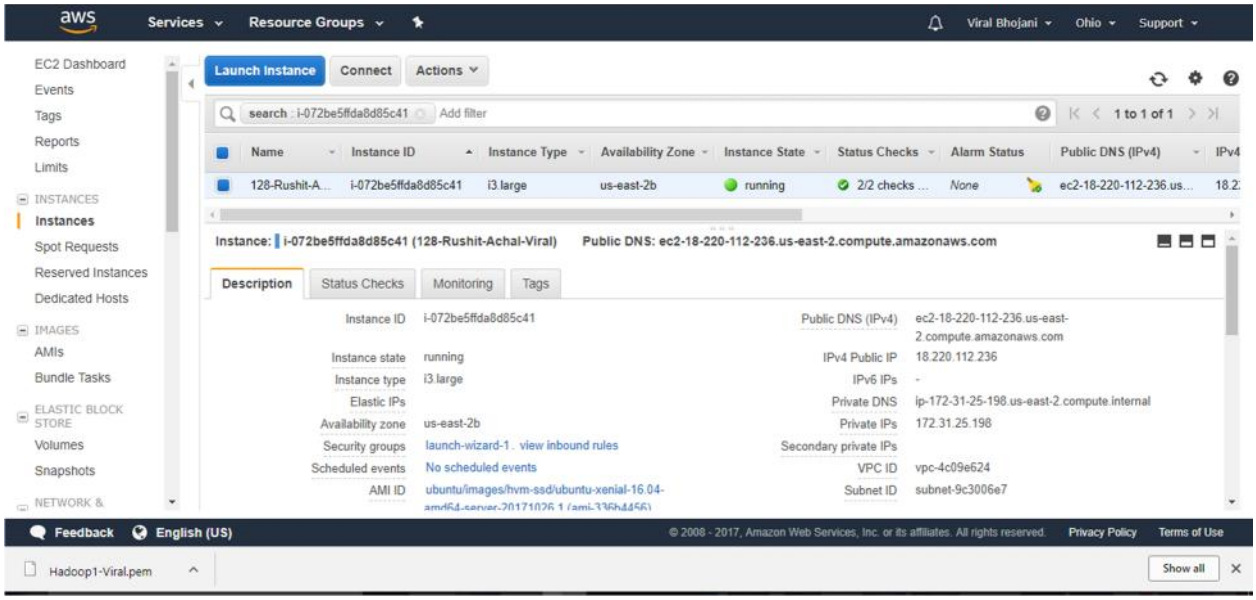
-Above screenshot is command for running valsort

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

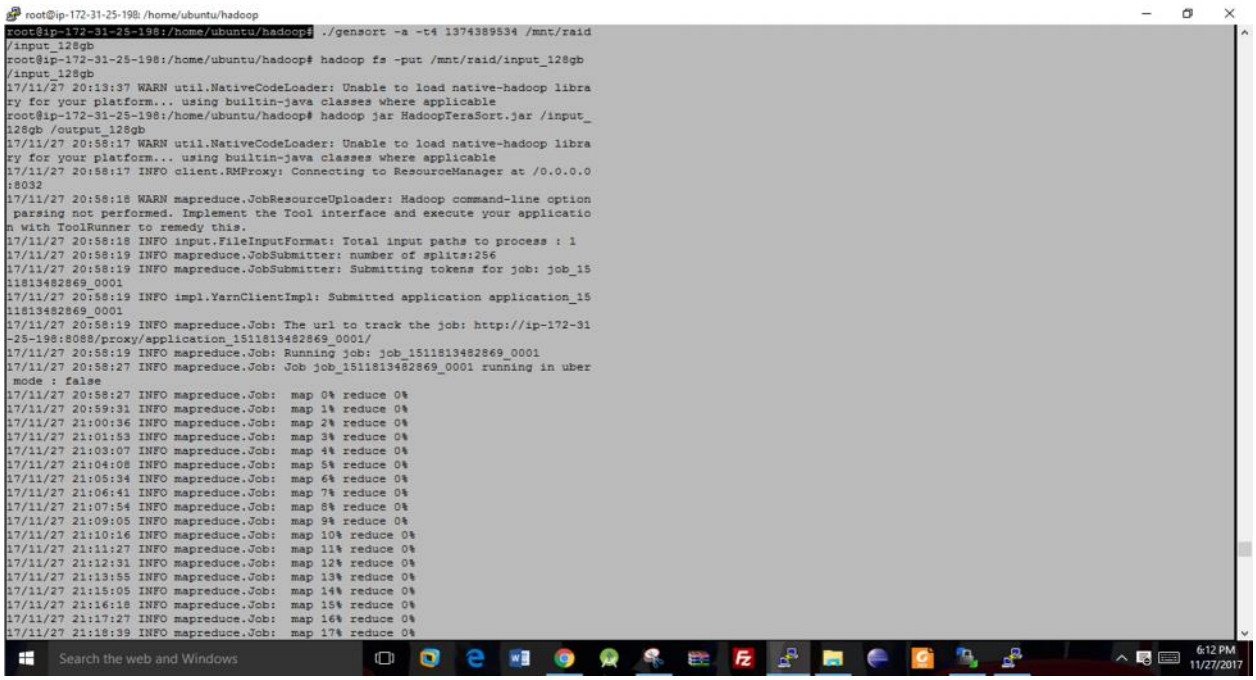
PA-2-REPORT

Hadoop Single Node

128GB Dataset



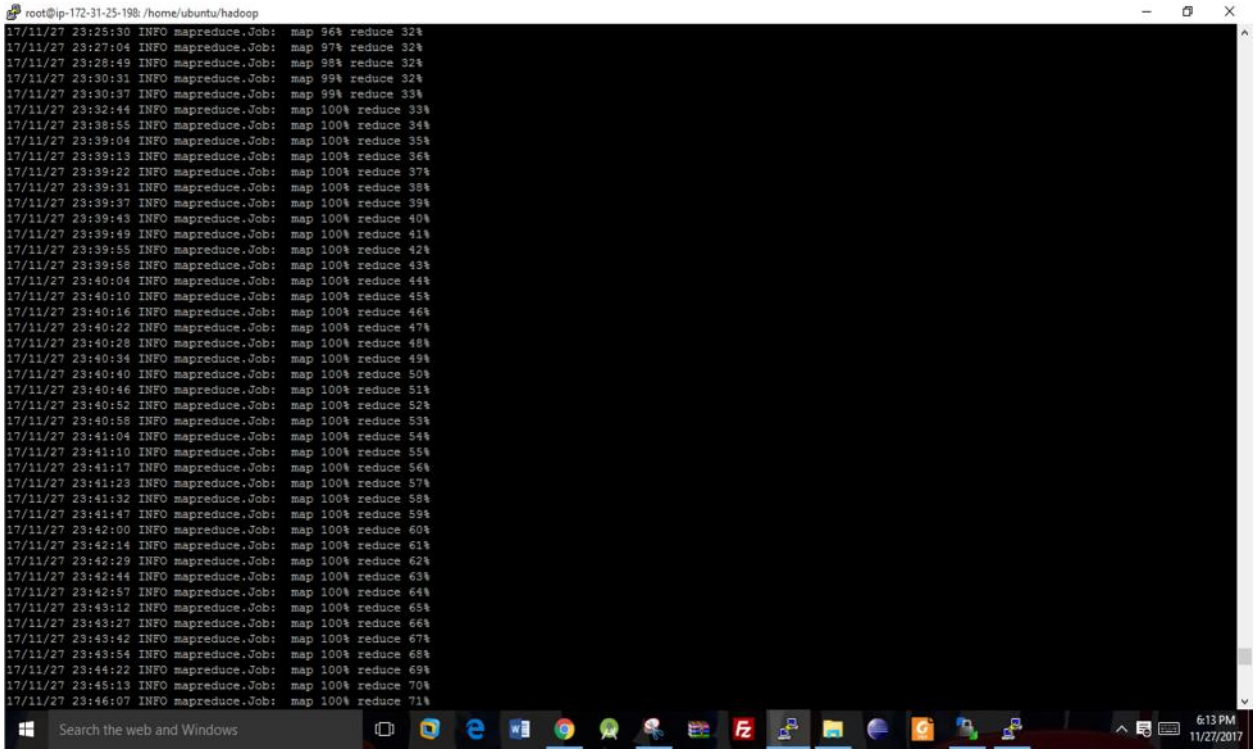
-Above screenshot is cluster information used in this experiment



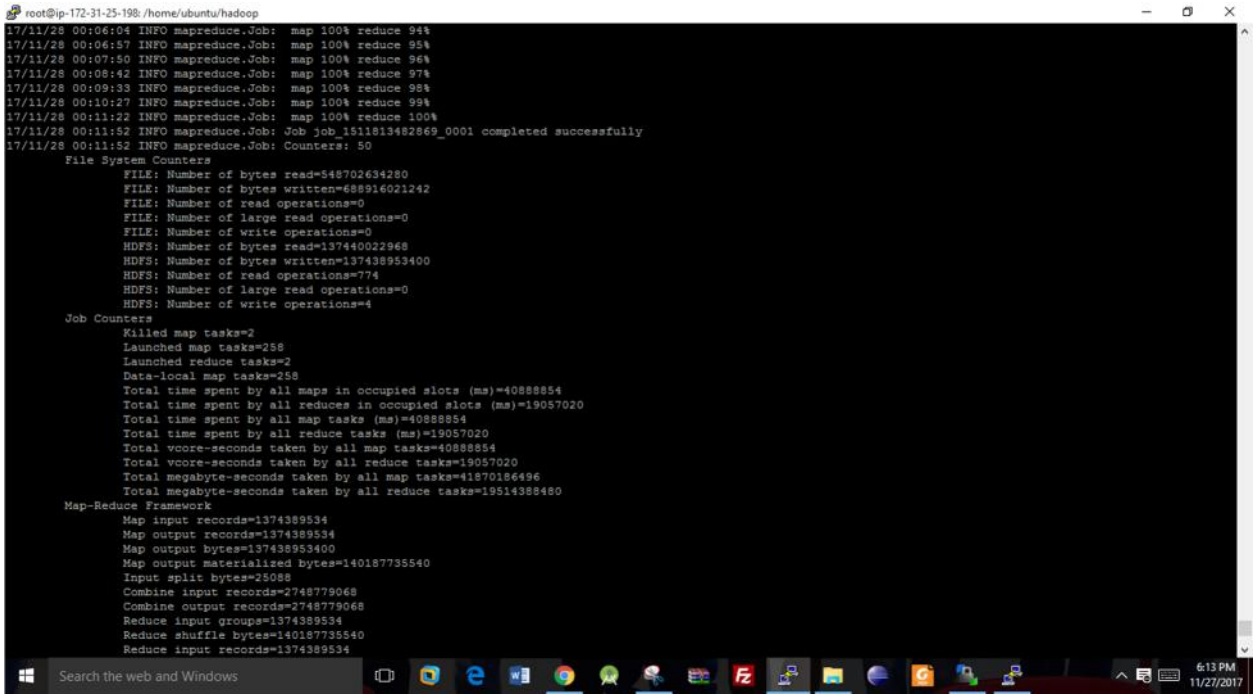
-Above screenshot is command used to run java HadopTeraSort.jar file

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



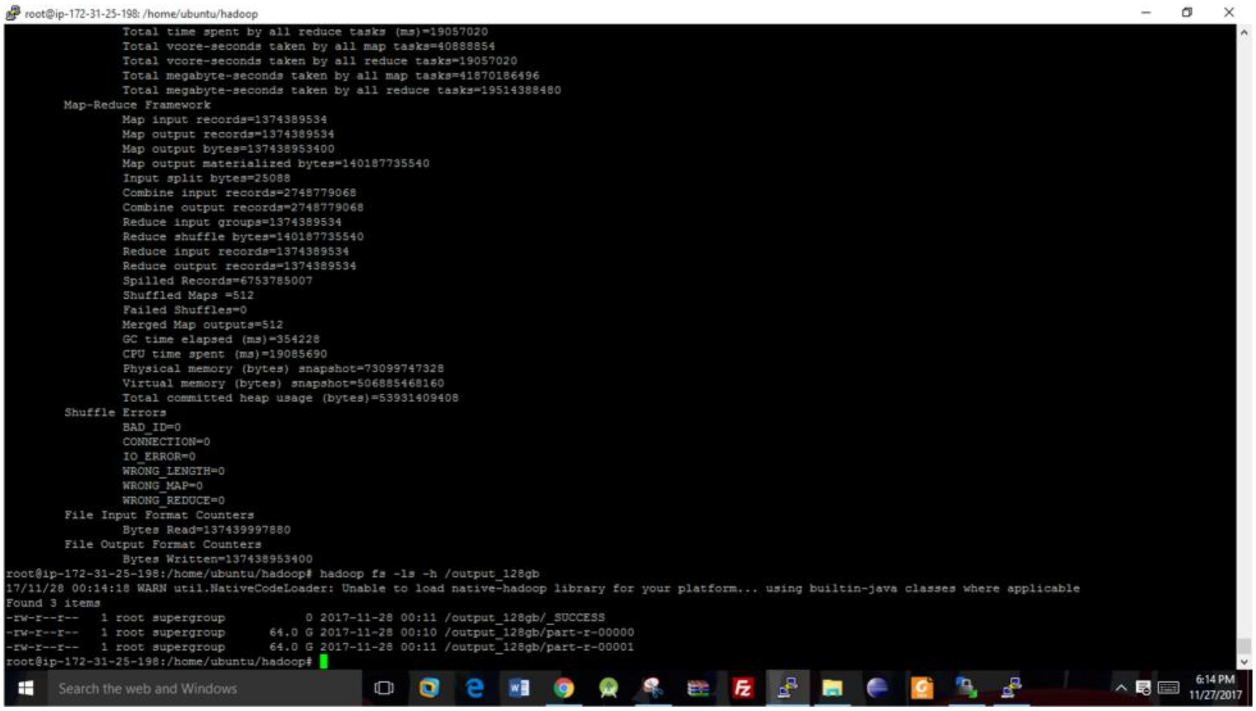
-Above screenshot is information of completion percentage of map and reduce class



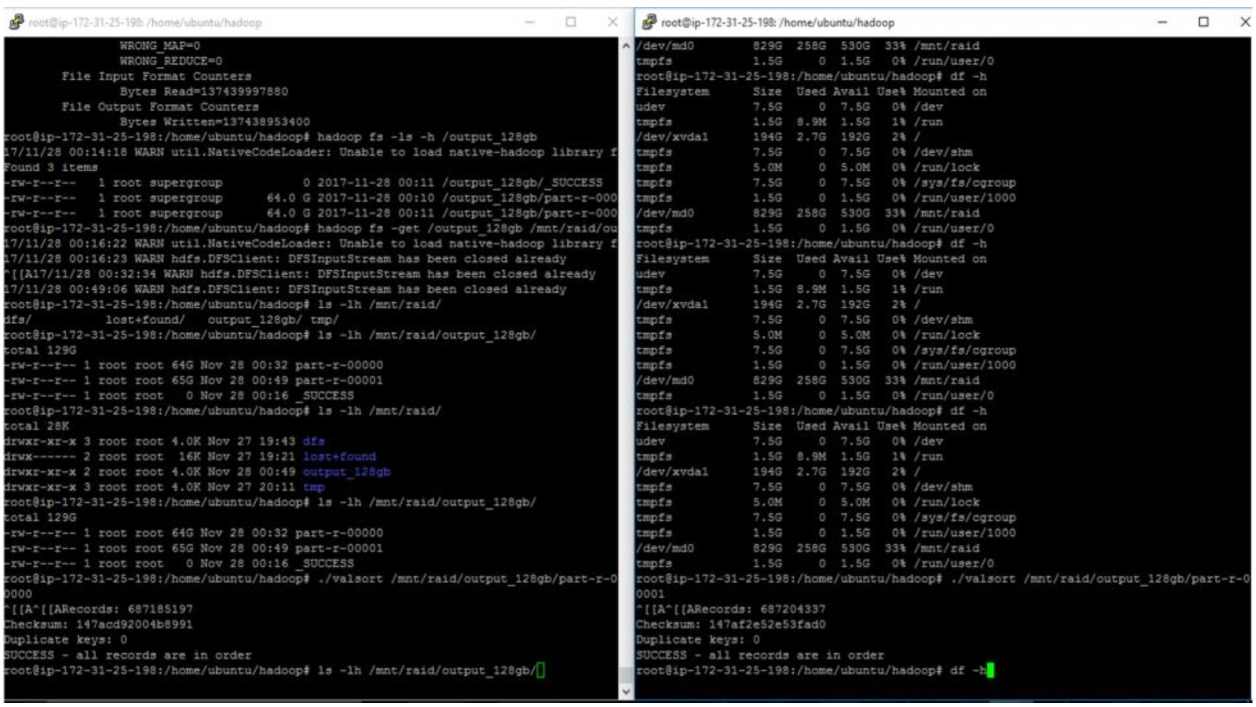
-Above screenshot is information after map reduce execution is completed

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



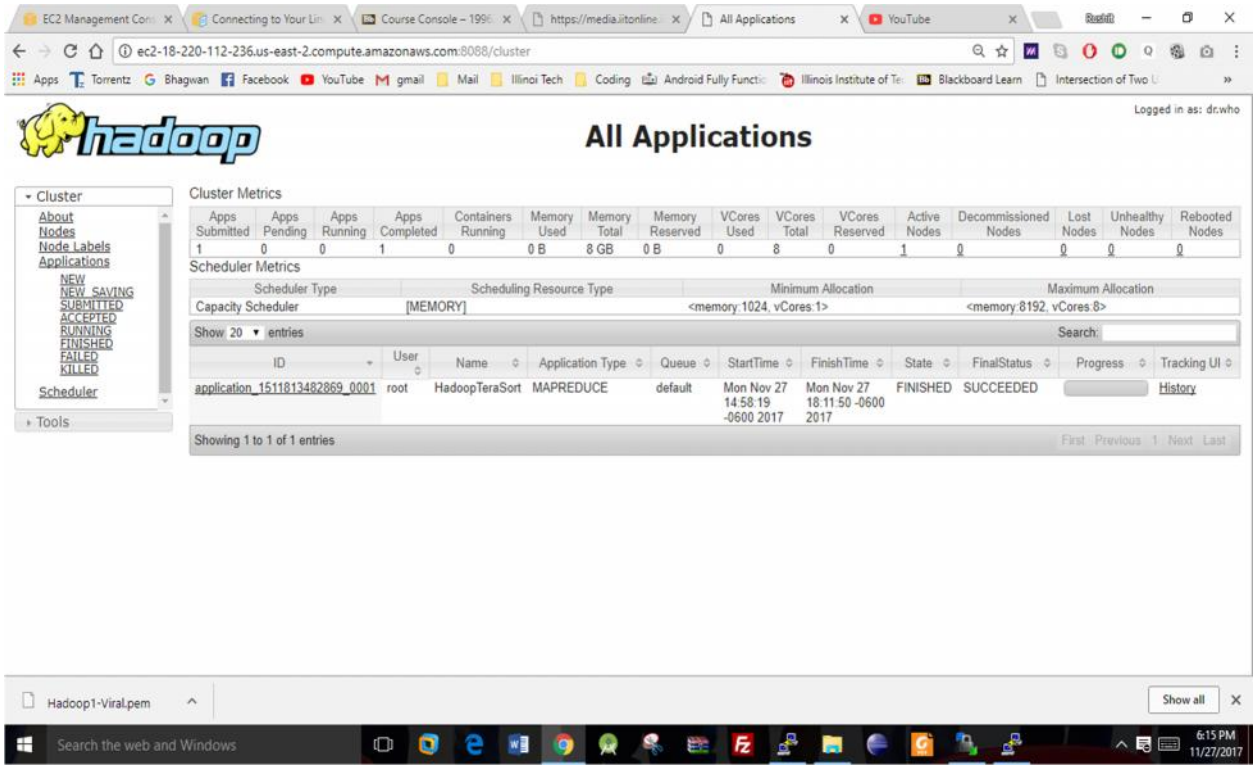
-Above screenshot is directory structure presented in Hadoop hdfs



-Above screenshot is command used to run valsort in both the part

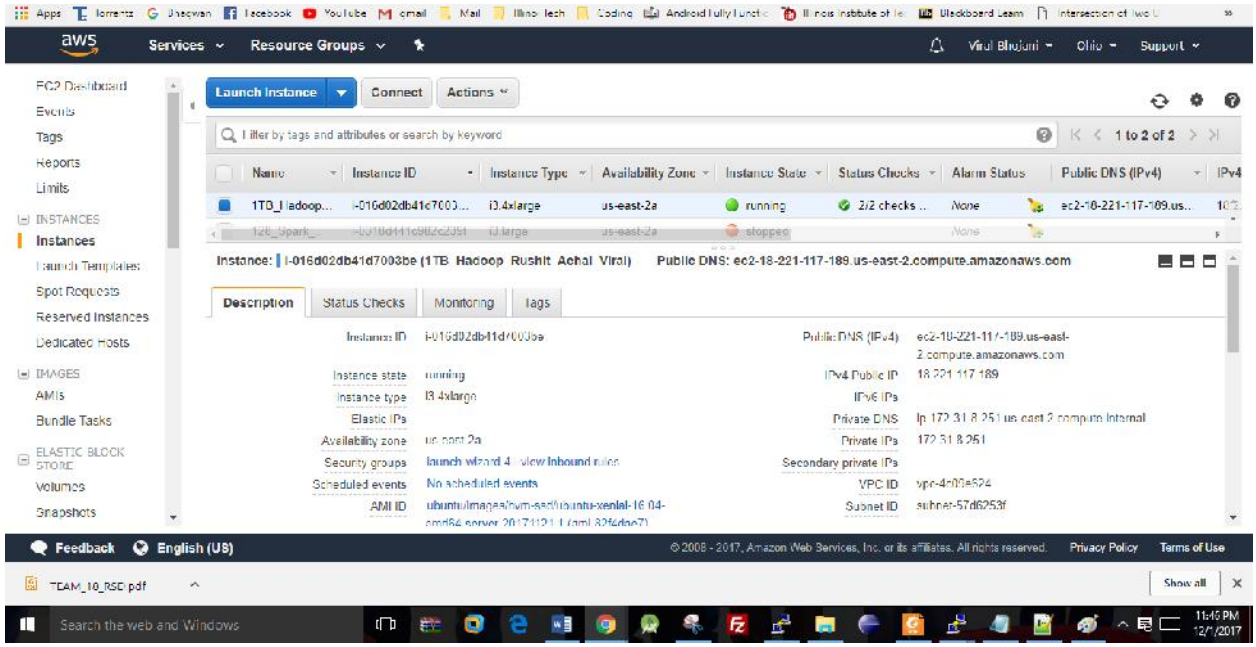
Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



-Above screenshot is Hadoop completion report with time required to finish the experiment

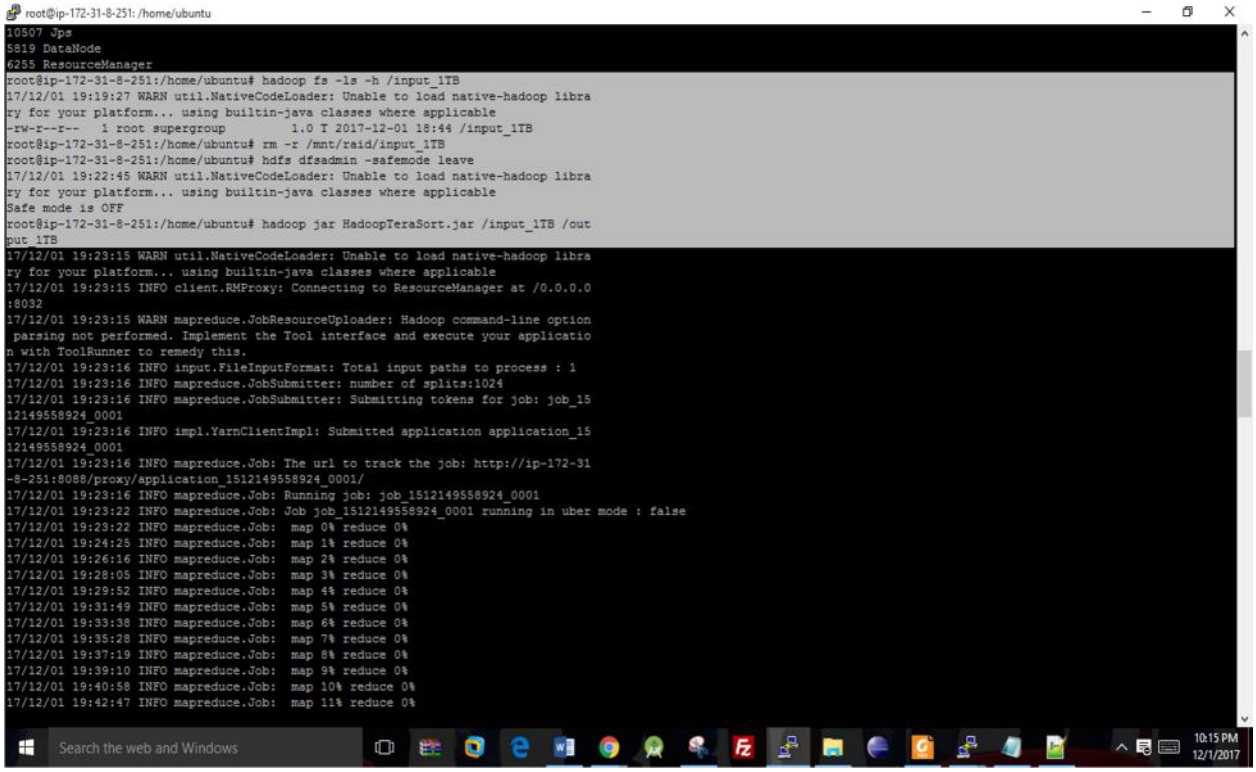
1TB Dataset



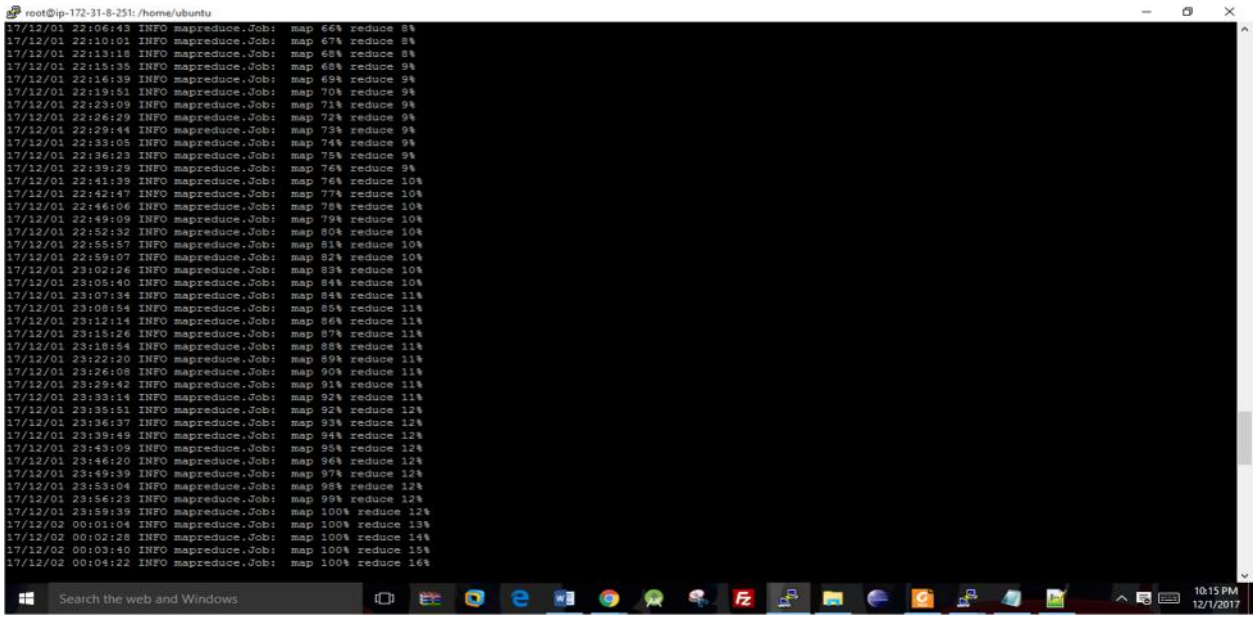
-Above screenshot is cluster information used in this experiment

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



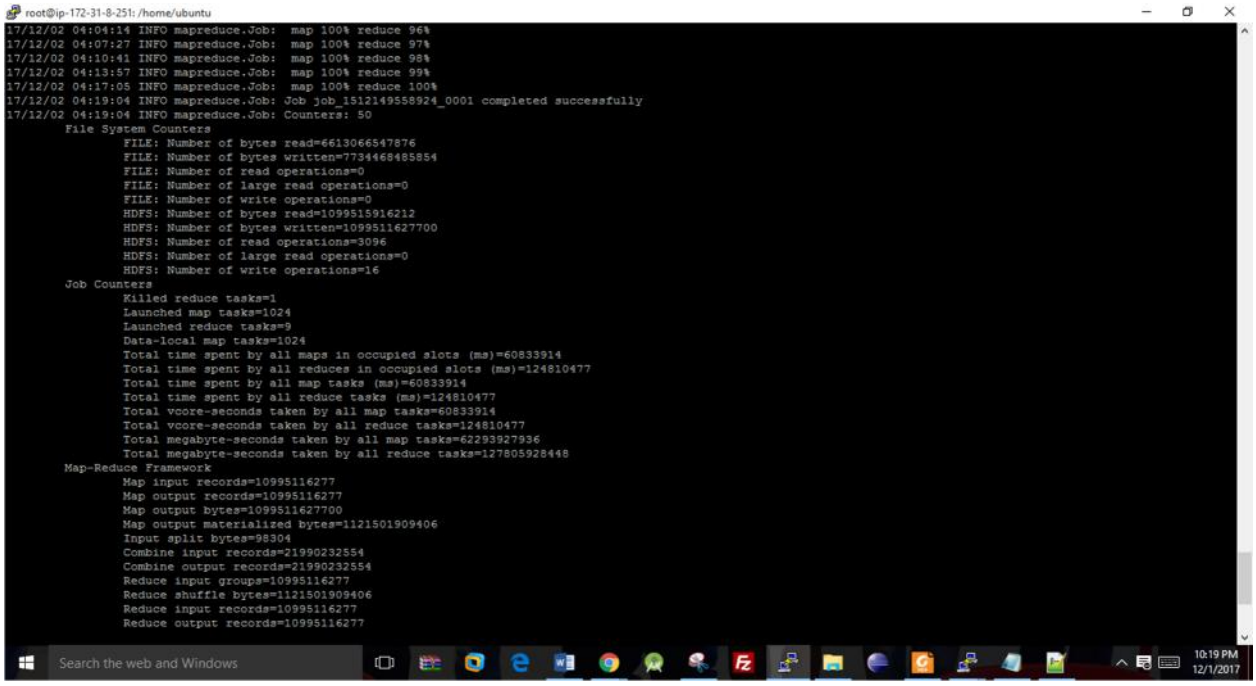
-Above screenshot is command used to run java HadopTeraSort.jar file



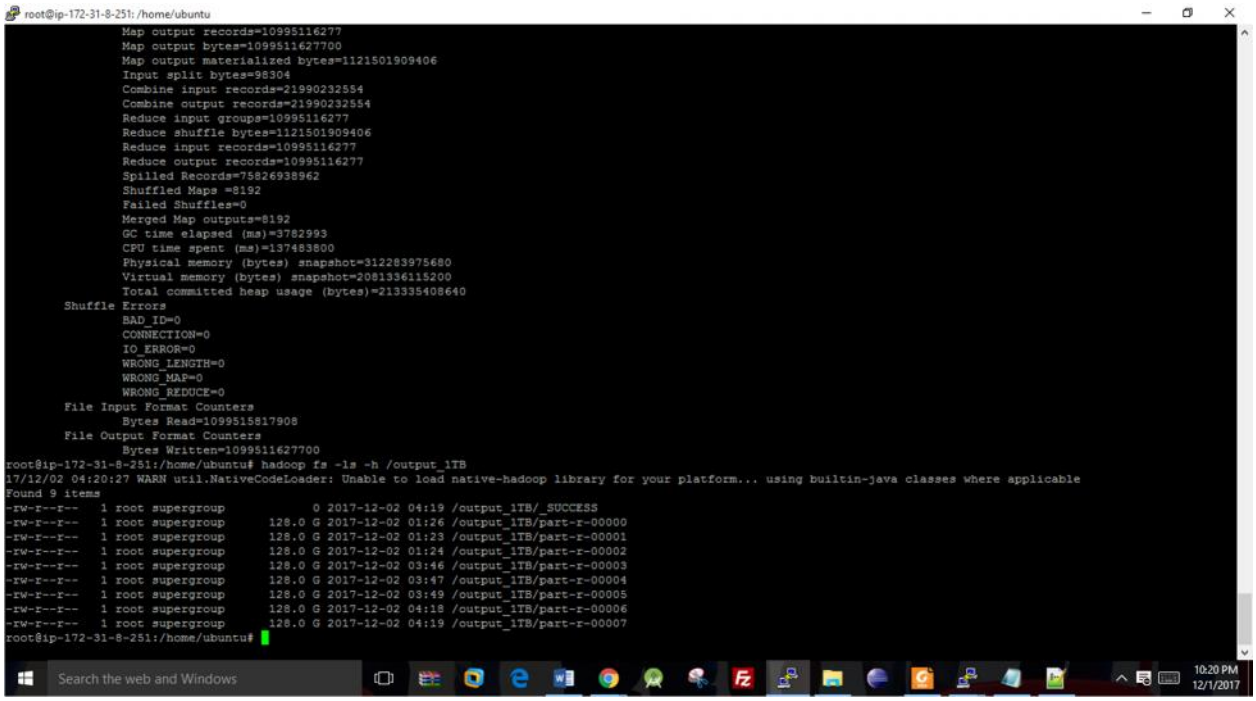
-Above screenshot is information of completion percentage of map and reduce class

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



-Above screenshot is information once map reduce execution is completed



-Above screenshot is directory information in Hadoop hdfs

```
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00000
Records: 1374354514
Checksum: 28f5b88d38567053
Duplicate keys: 0
SUCCESS - all records are in order
```

```
ubuntu@ip-172-31-8-251:~$ sudo su
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00001
Records: 1374307037
Checksum: 28f510cf82b2b115
Duplicate keys: 0
SUCCESS - all records are in order
```

```
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00002
Records: 1374360153
Checksum: 28f59f9cf96c9a9d
Duplicate keys: 0
SUCCESS - all records are in order
```

```
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00003
Records: 1374414727
Checksum: 28f60651892253bb
Duplicate keys: 0
SUCCESS - all records are in order
```

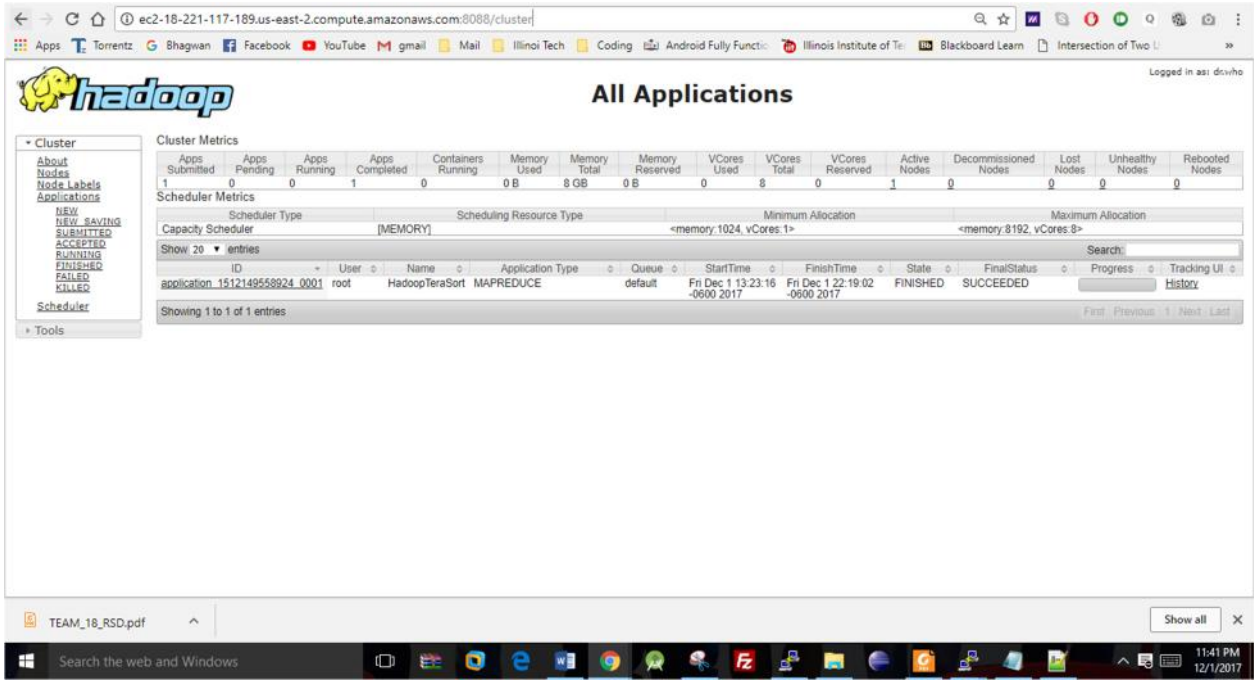
```
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00004
Records: 1374487578
Checksum: 28f6bd3585974edc
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00005
^[[ARecords: 1374369130
Checksum: 28f5ce3e37d6e67b
Duplicate keys: 0
SUCCESS - all records are in order
```

```
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00006
Records: 1374414956
Checksum: 28f60c56bd8e62de
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-8-251:/home/ubuntu# ./valsort /mnt/raid/output_1TB/part-r-00007
Records: 1374408182
Checksum: 28f5beffb6b4cd4e
Duplicate keys: 0
SUCCESS - all records are in order
```

-Above screenshots are commands used to run valsort in all 8 parts

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

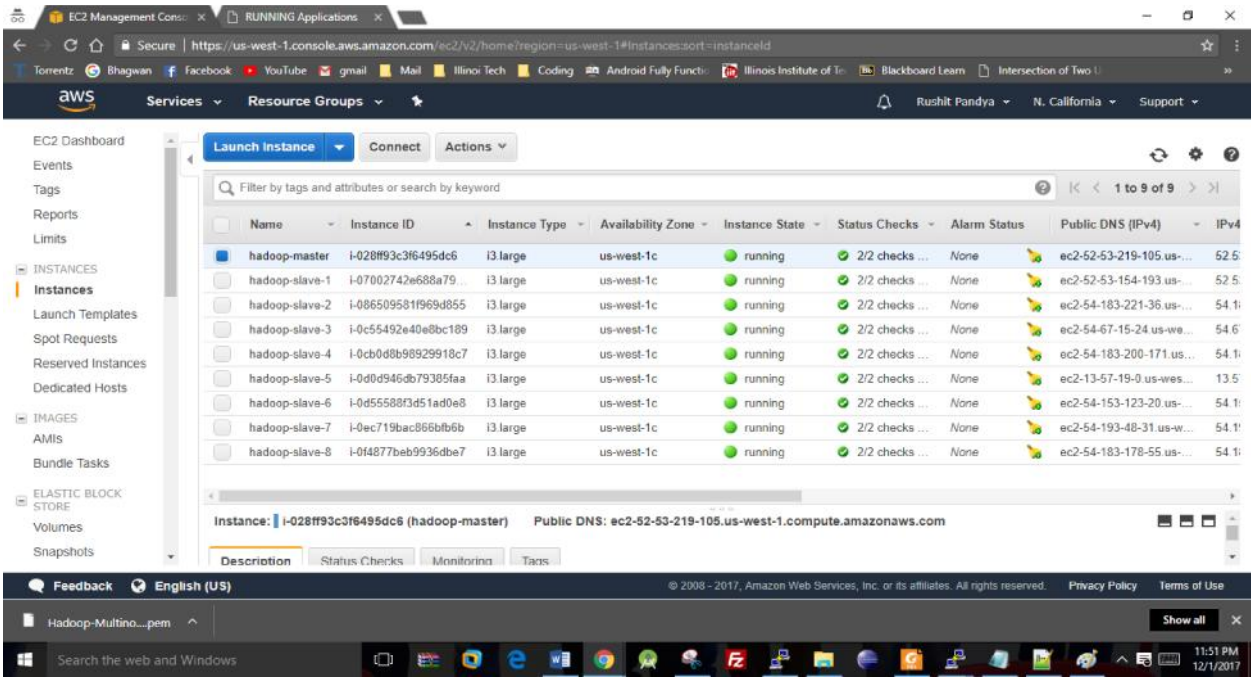
PA-2-REPORT



-Above screenshot is Hadoop completion report with time required to finish the experiment

Hadoop 8-Node Cluster

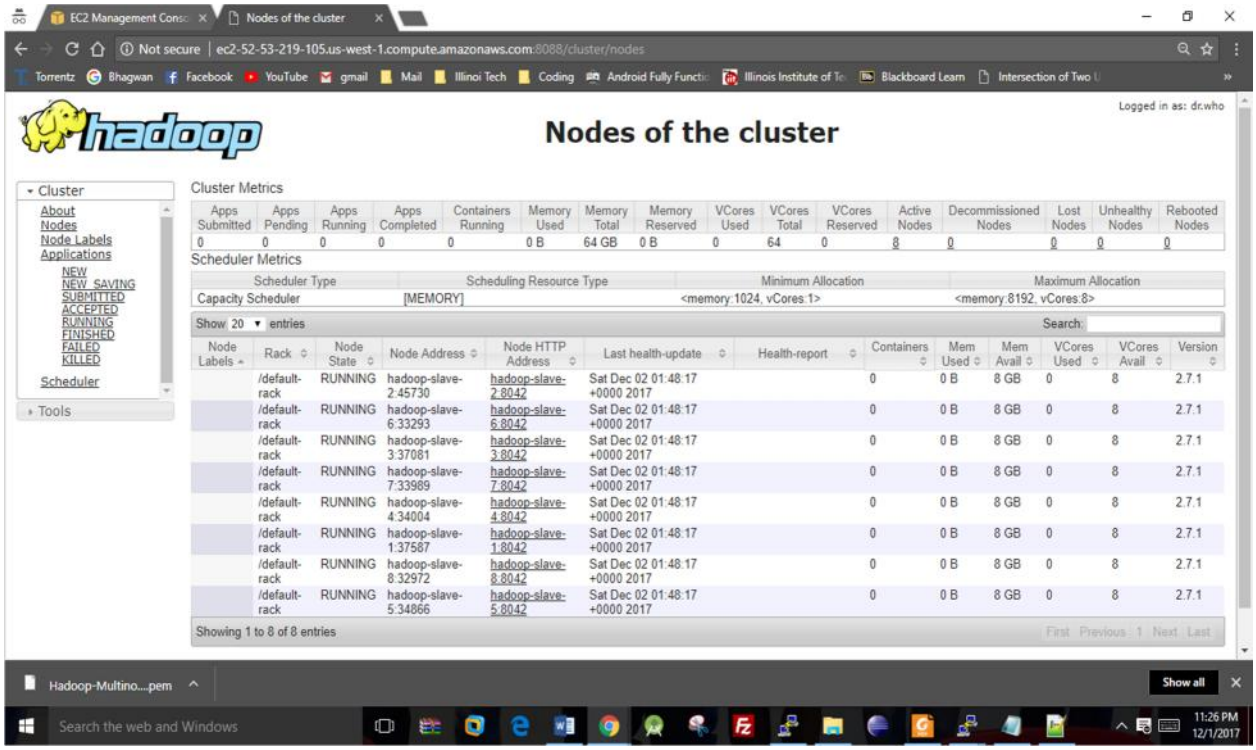
1TB Dataset



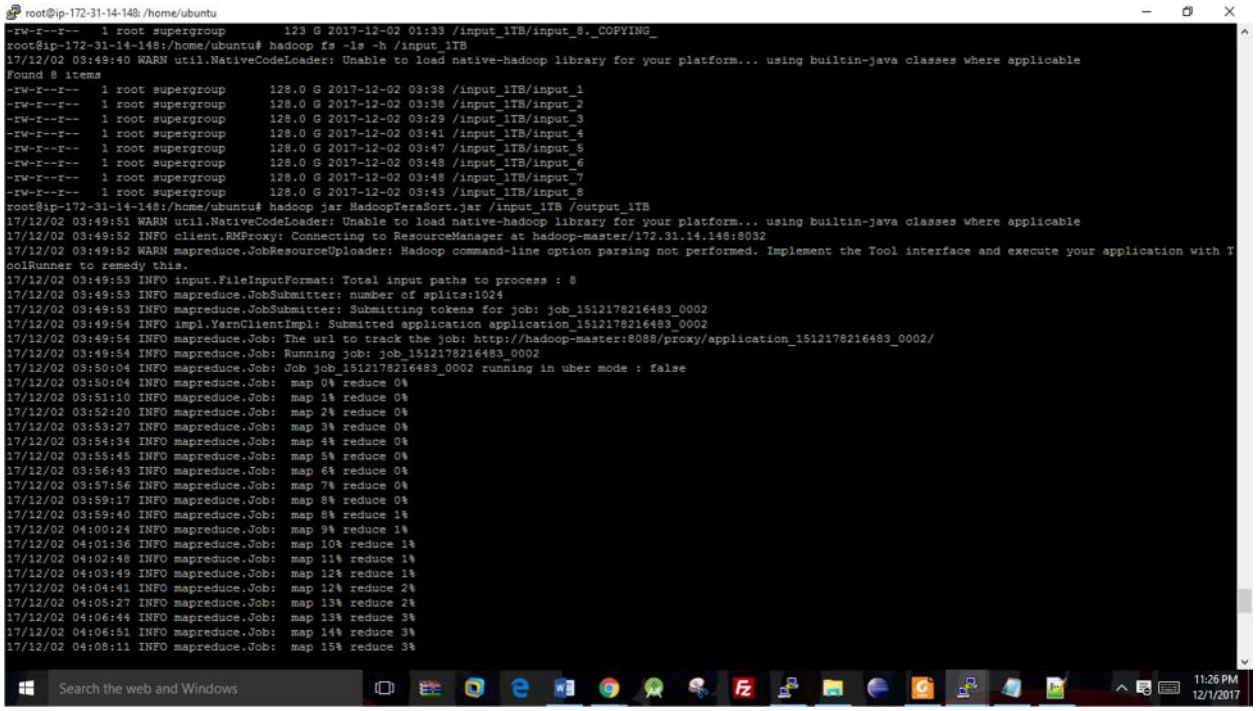
-Above screenshot is cluster information used in this experiment

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



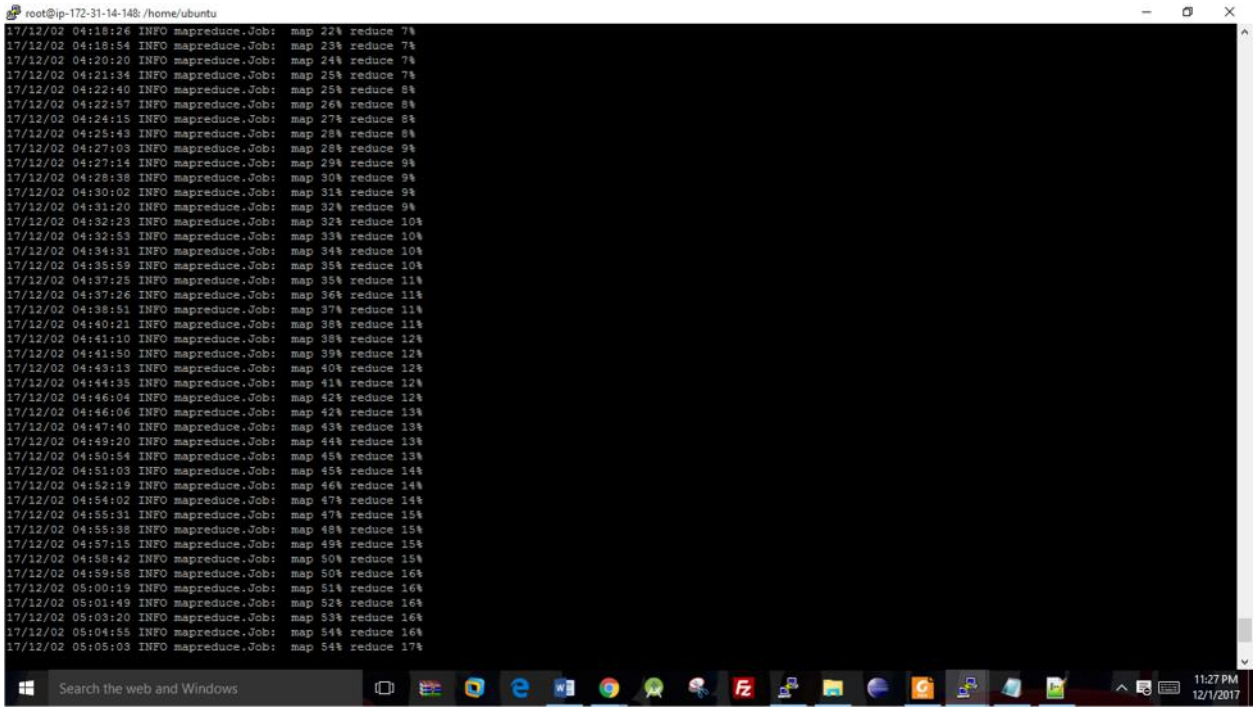
-Above screenshot is information of nodes in this cluster



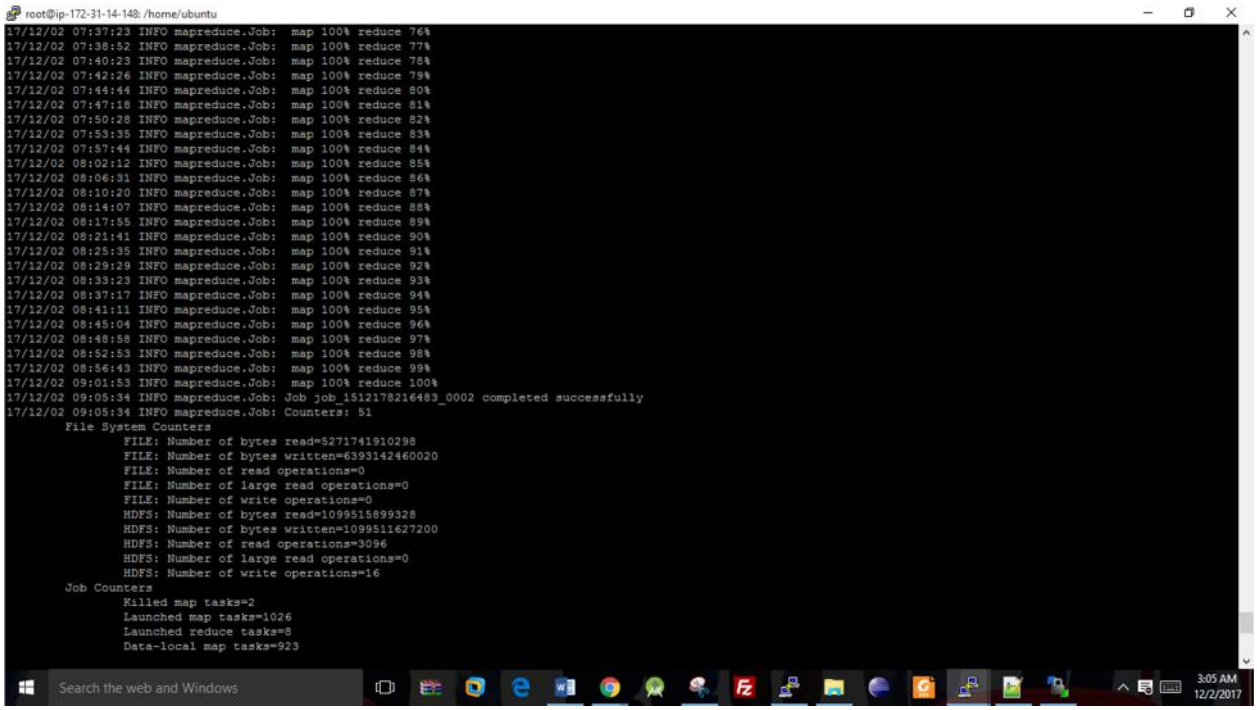
-Above screenshot is command used to run java HadopTeraSort.jar file

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



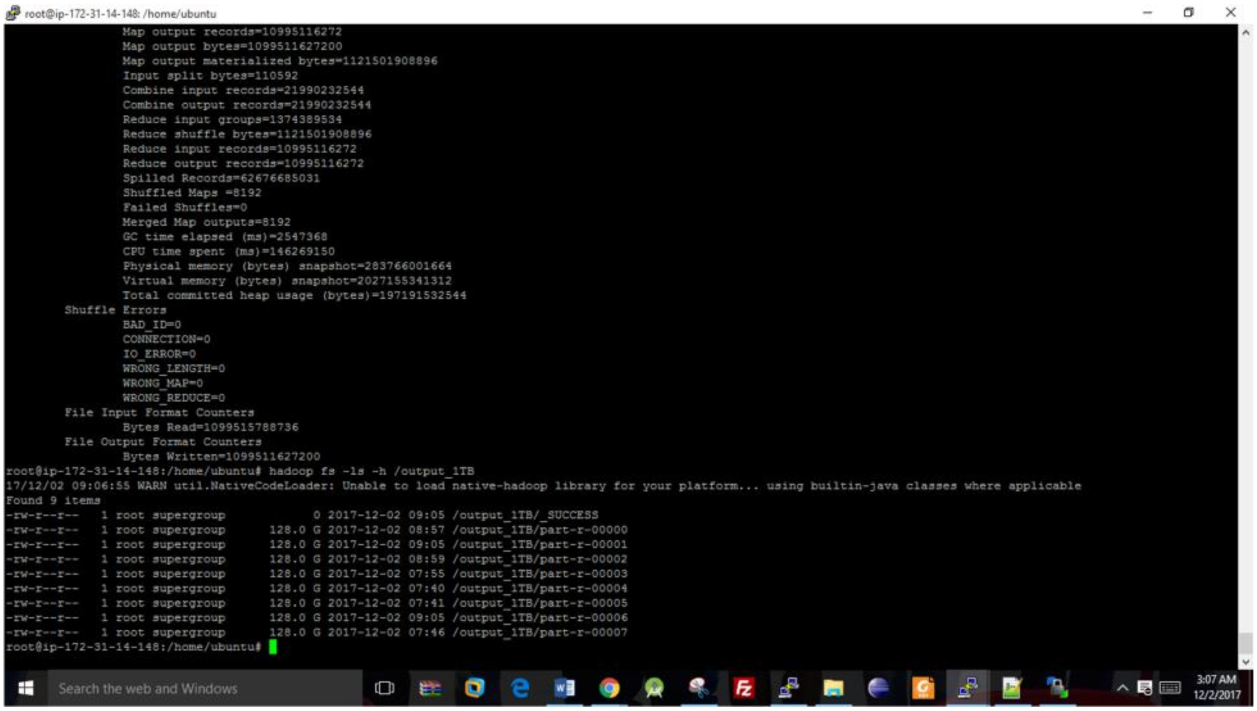
-Above screenshot is information of completion percentage of map and reduce class



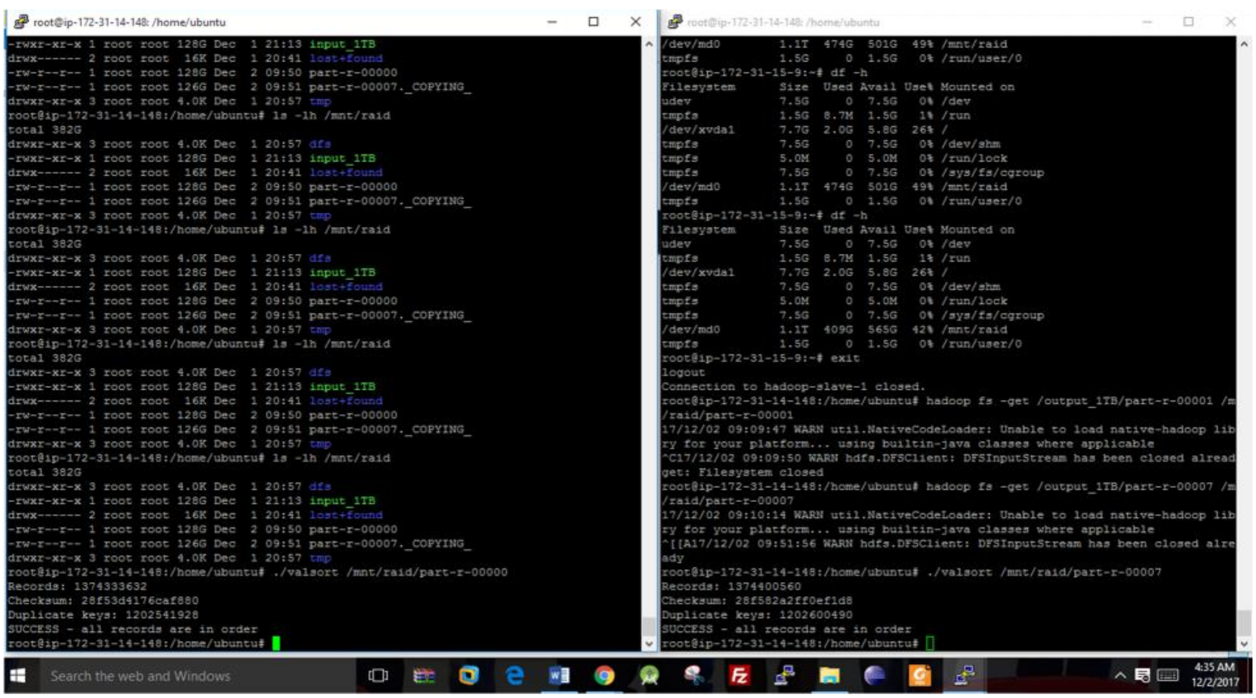
-Above screenshot is information once map reduce execution is completed

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



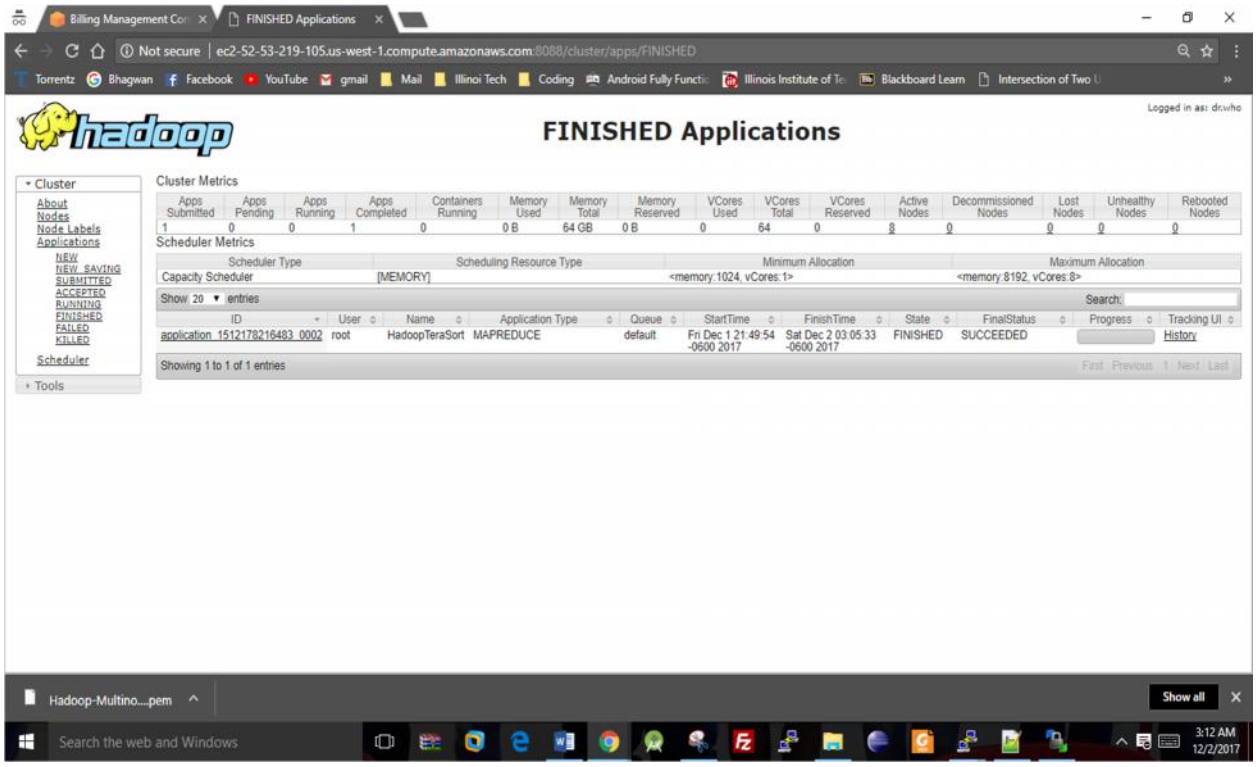
-Above screenshot is directory structure in Hadoop hdfs



-Above screenshots is list of output part files of total size 1TB. Hence, we have shown valsort on only first and last part of output file which shows that all records are in order

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

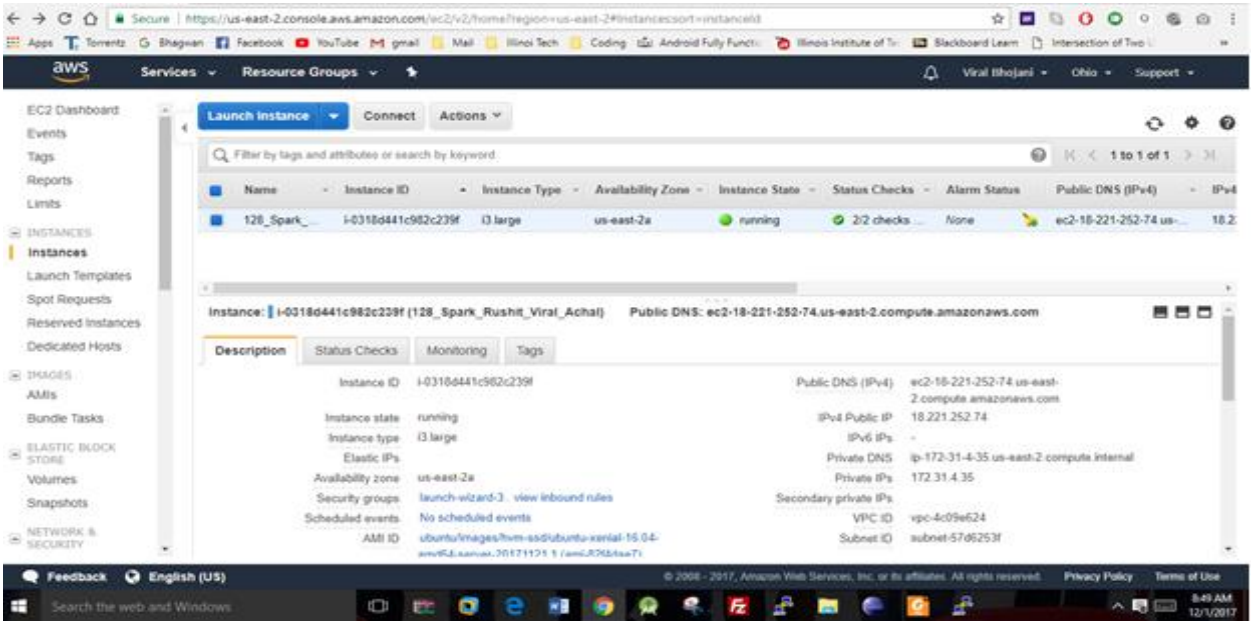
PA-2-REPORT



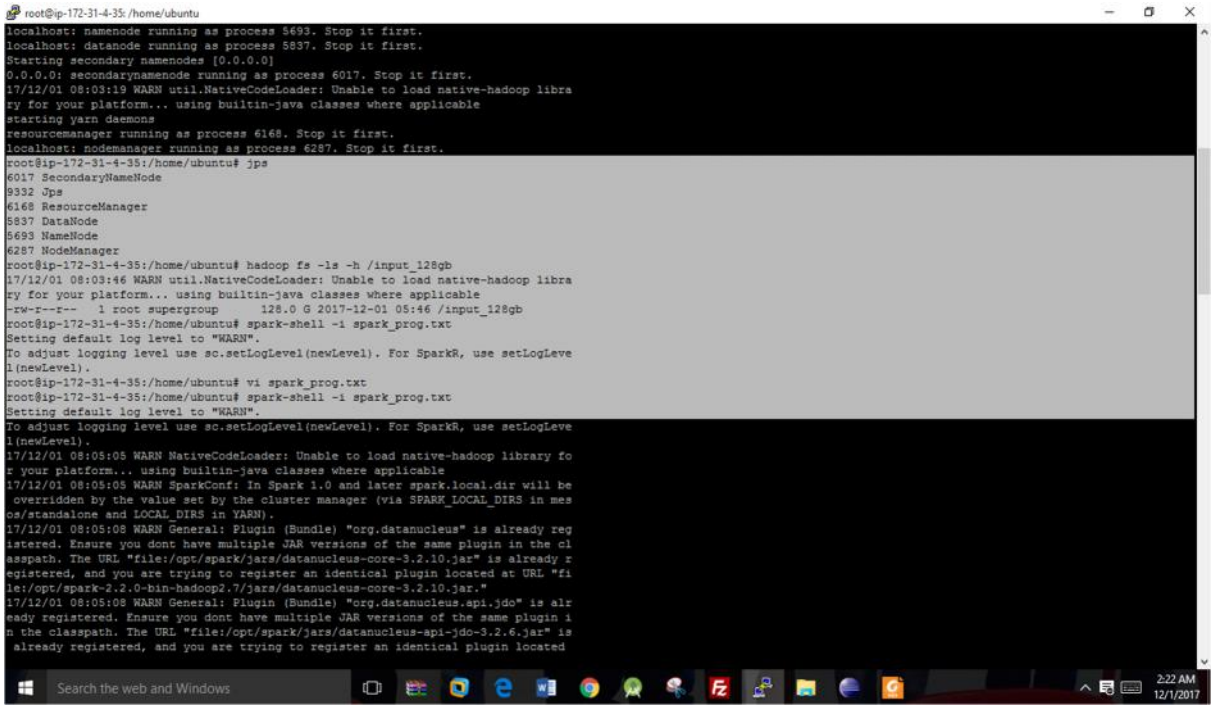
-Above screenshot is Hadoop completion report with time required to finish the experiment

Spark Single Node Cluster

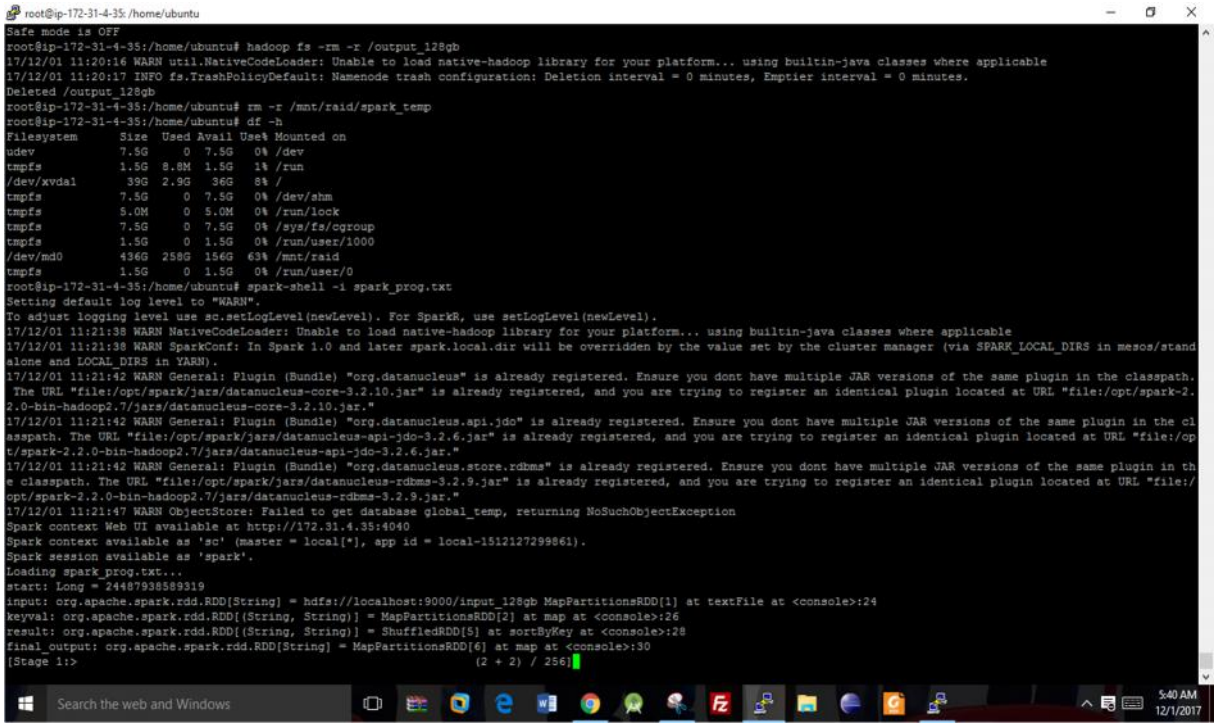
128GB Dataset



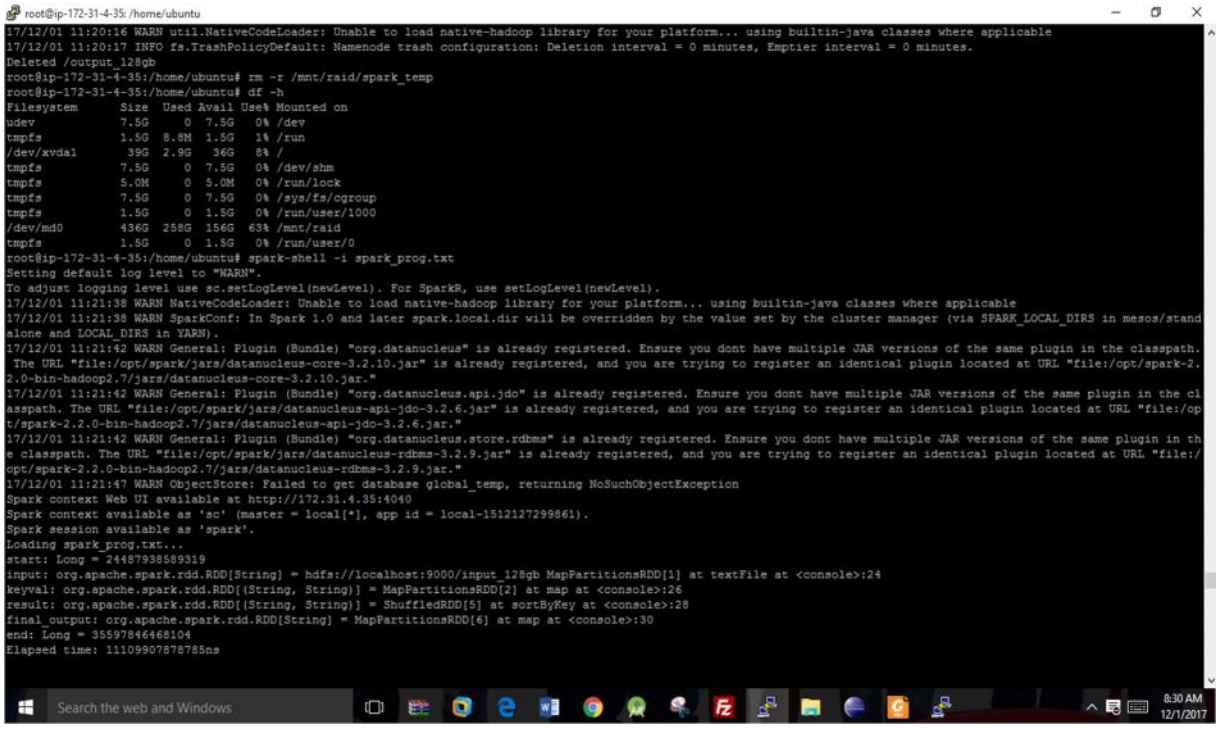
-Above screenshot is cluster information used in this experiment



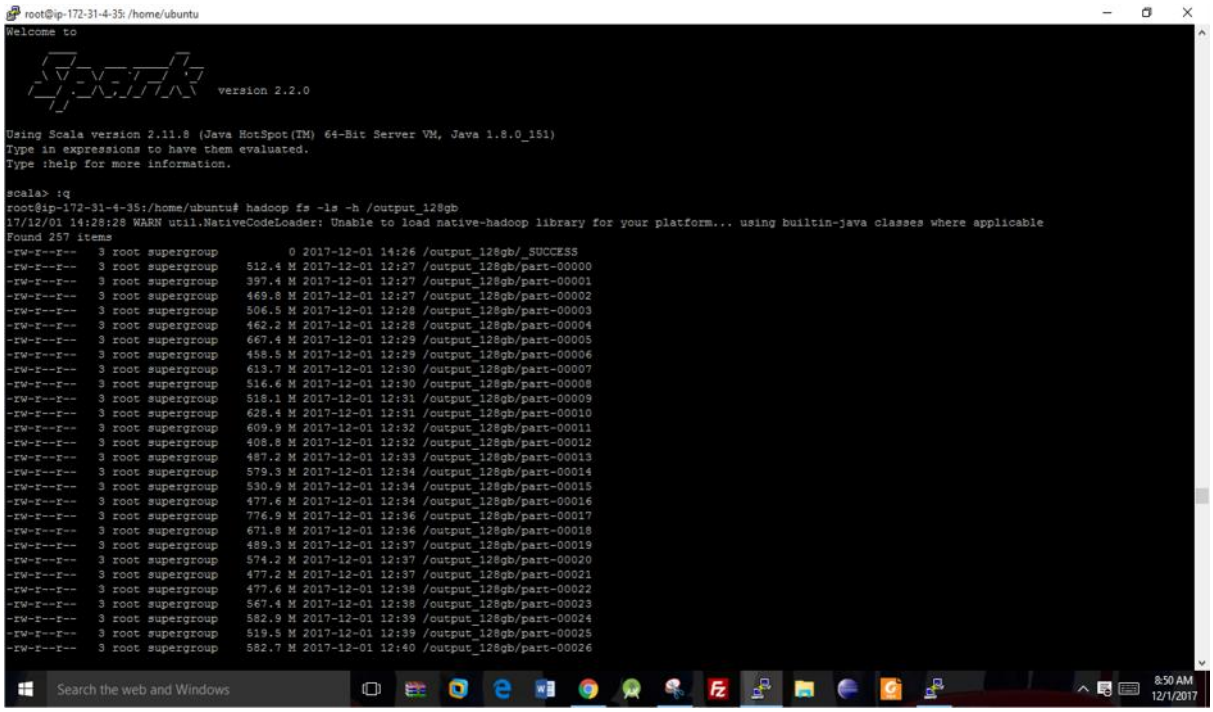
-Above screenshot is command used to run spark spark_prog.txt



-Above screenshot is information that spark execution is started



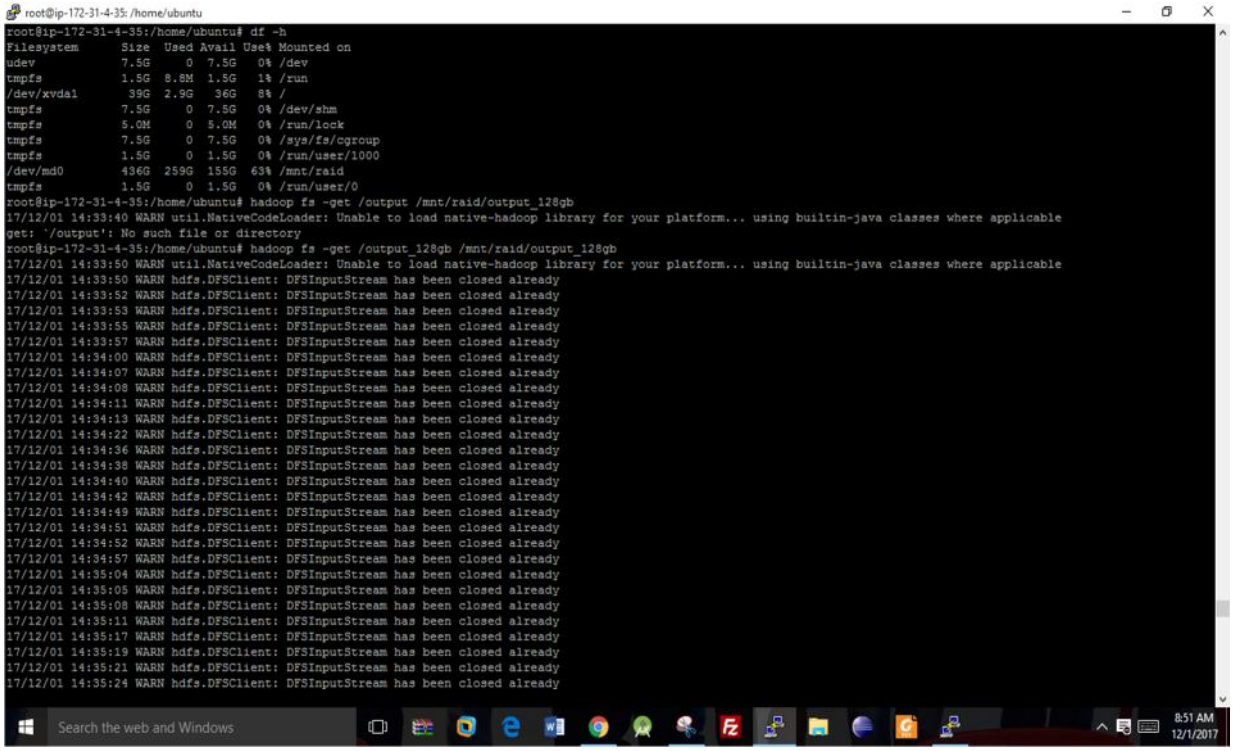
-Above screenshot is information that spark execution is completed



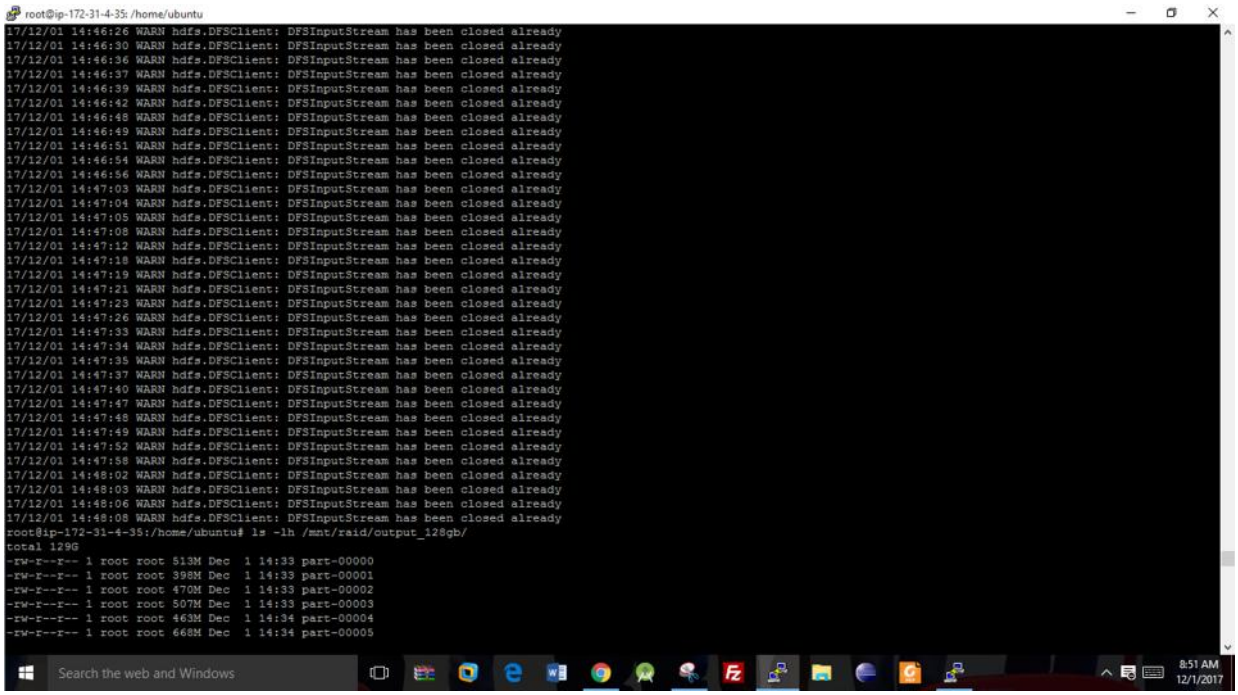
-Above screenshot is information that 257 files being created each file in the range of 400 to 600 MB making it 128GB data

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

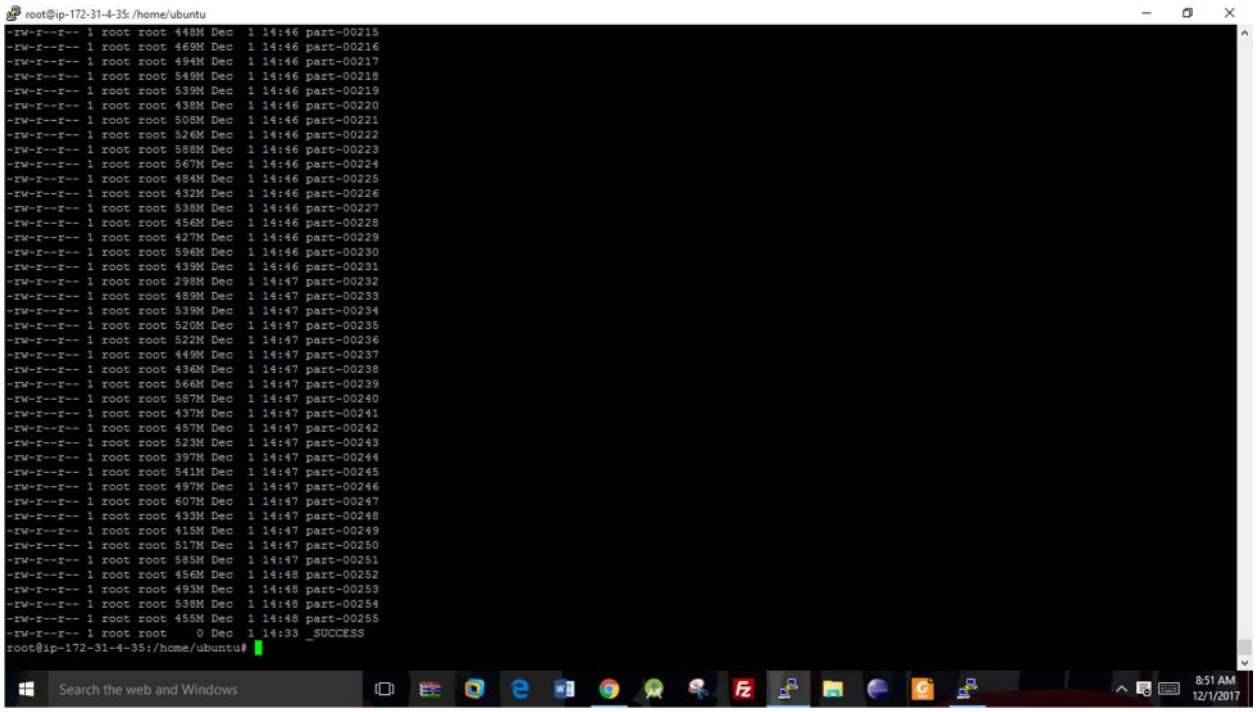


-Above screenshot is information for getting all 257 files from Hadoop hdfs into our file system

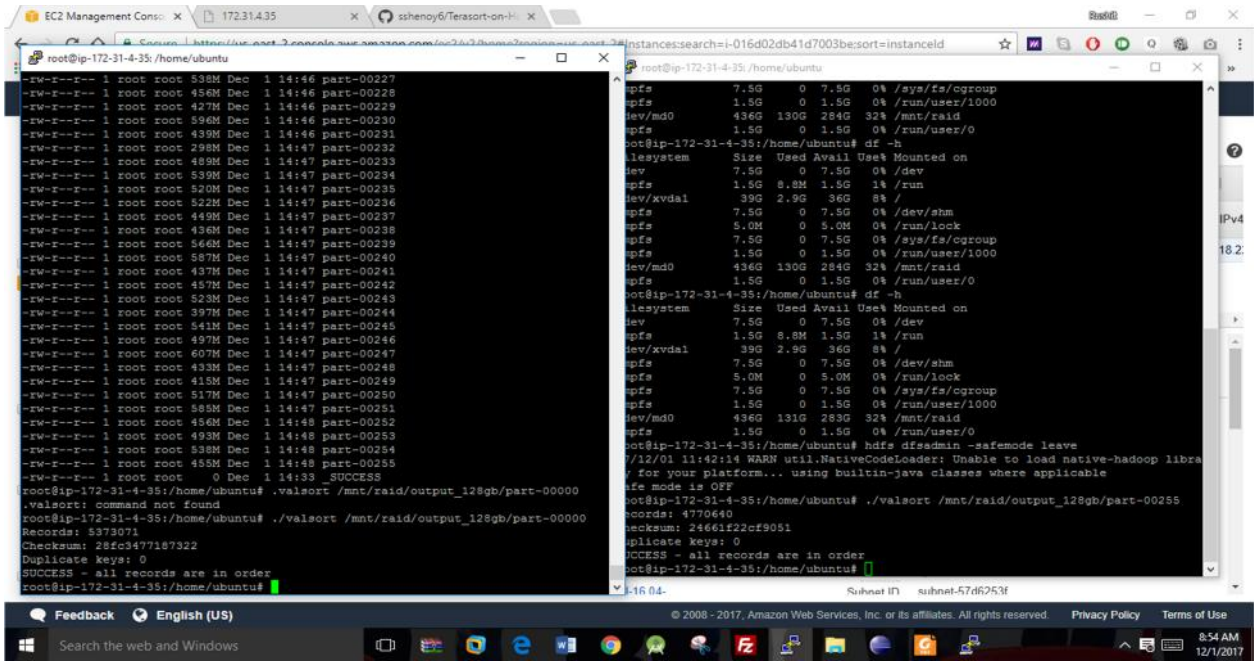


Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



-Above two screenshots are directory structure mnt/raid/output_128gb

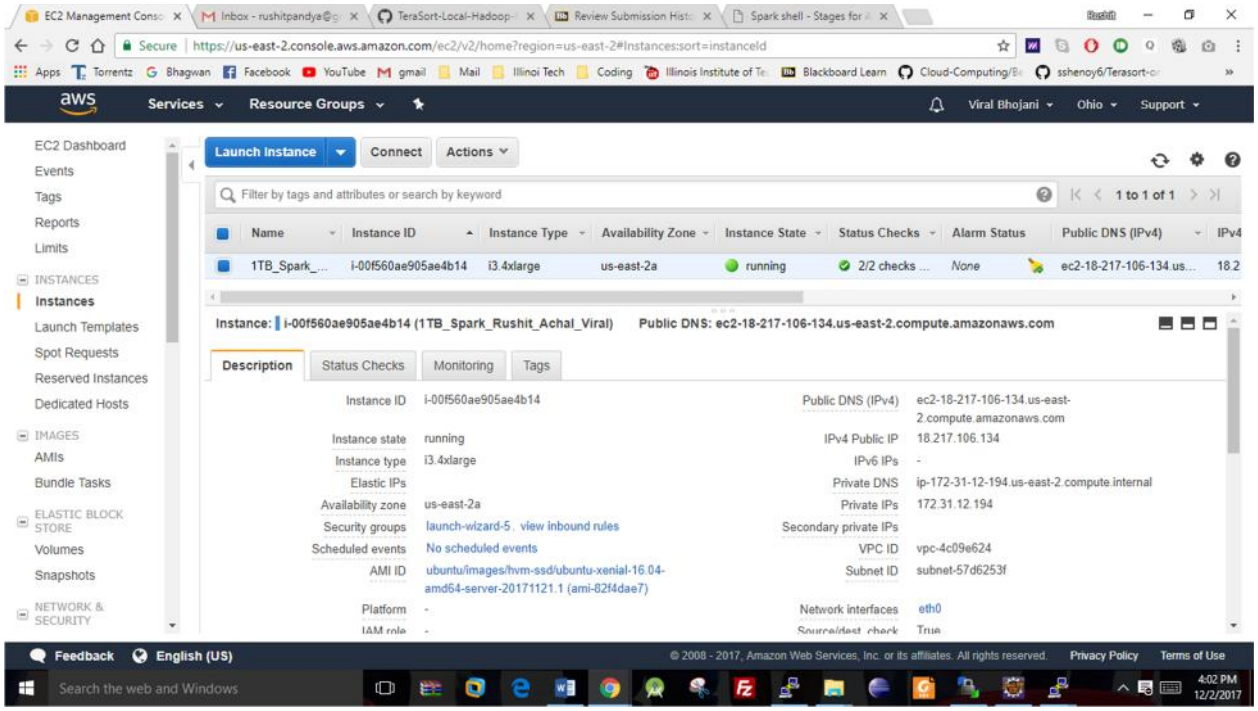


-Above screenshots infers list of output part files of total size 129G. Hence, we have shown valsort on only first and last part of output file which shows that all records are in order.

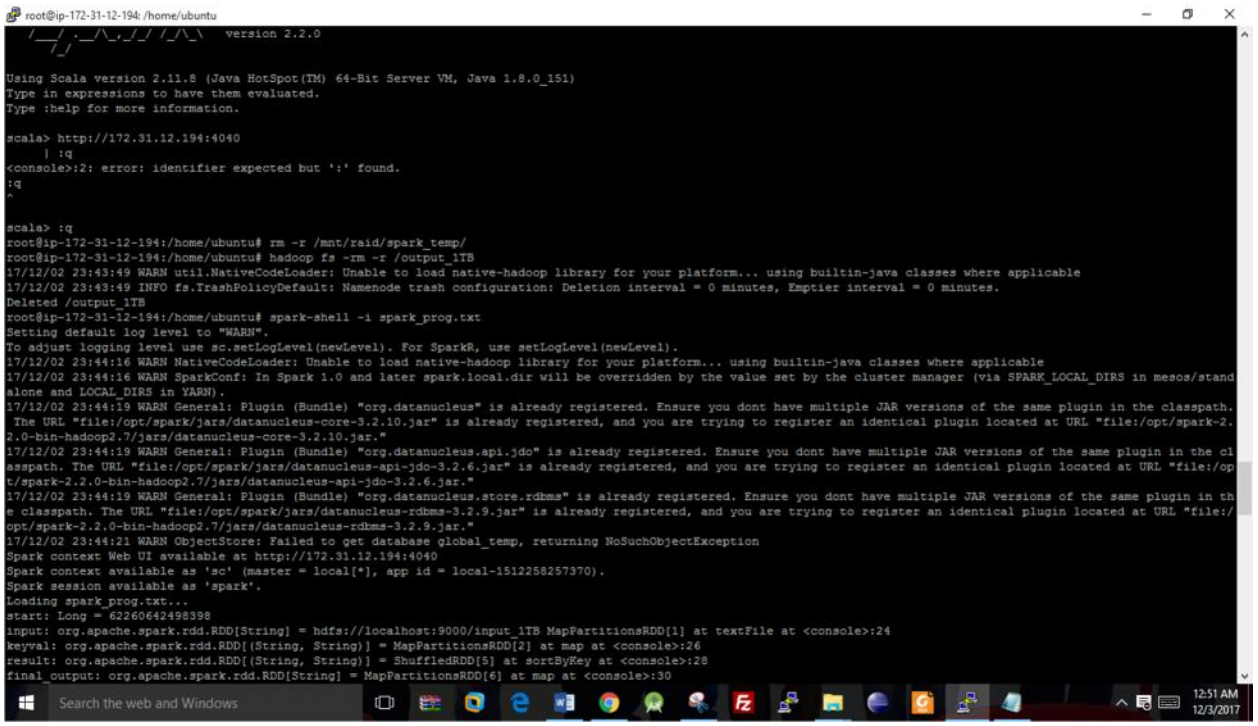
Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

1TB Dataset



-Above screenshot is cluster information used in this experiment

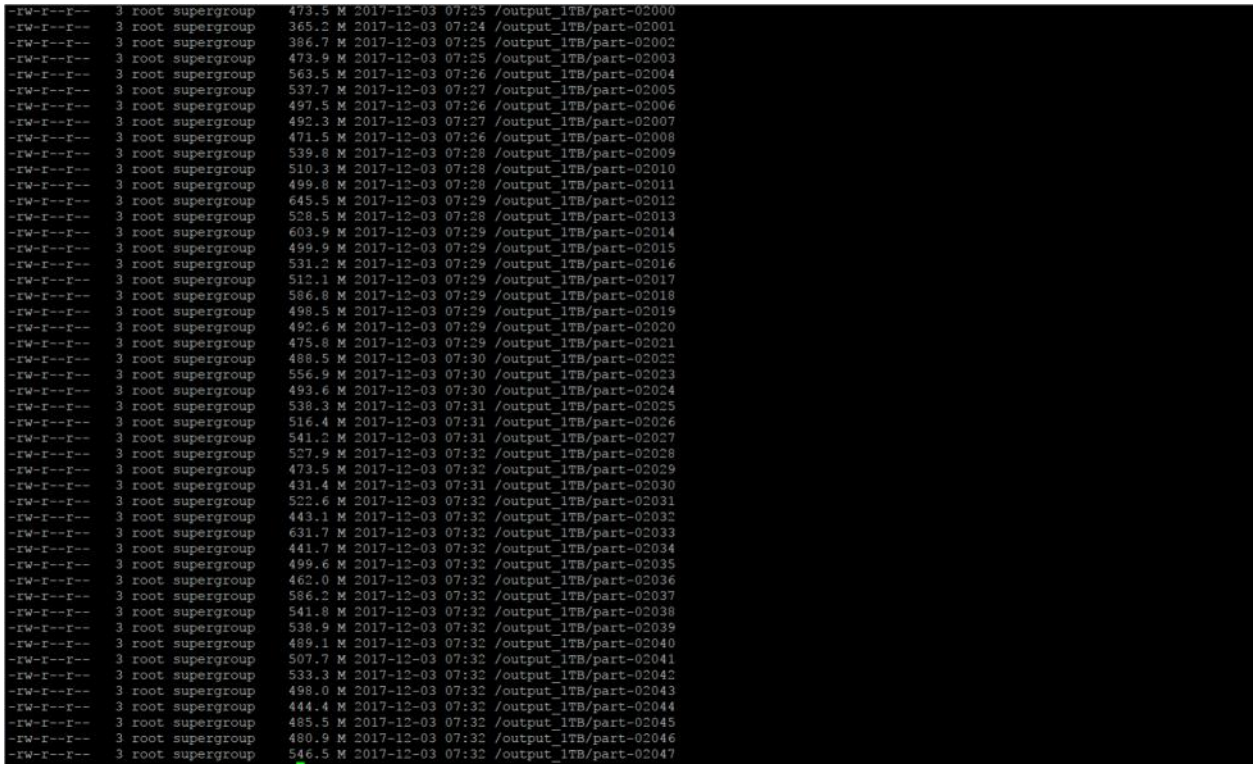


-Above screenshot is command used to run spark spark_prog.txt

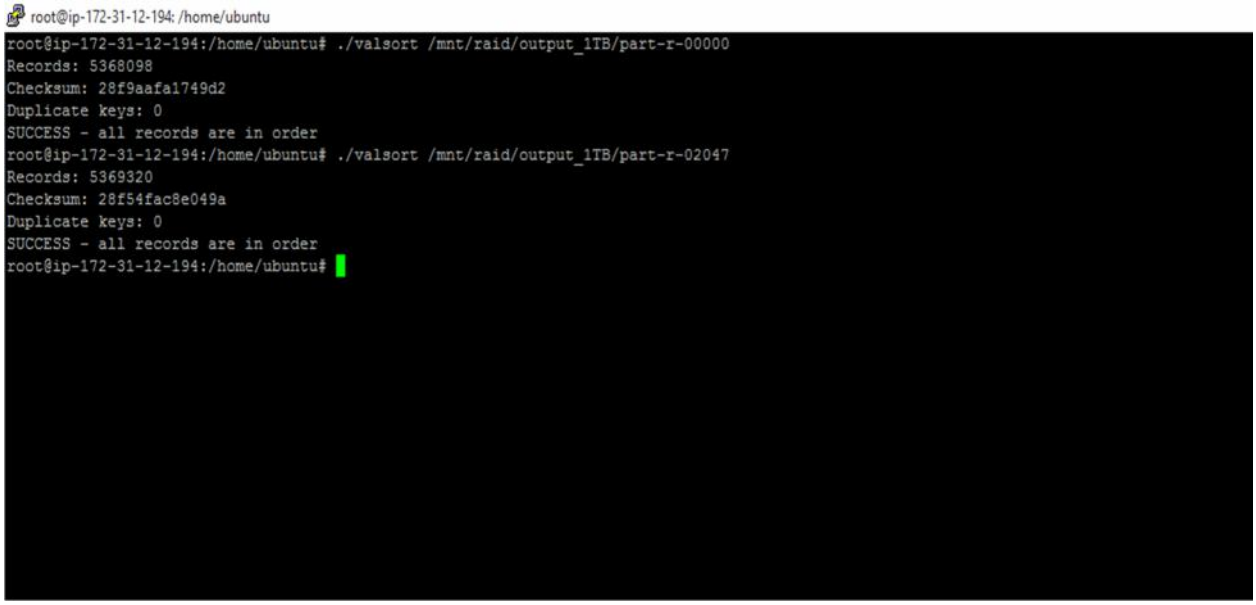
Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT

-Above screenshot is information that the spark is started



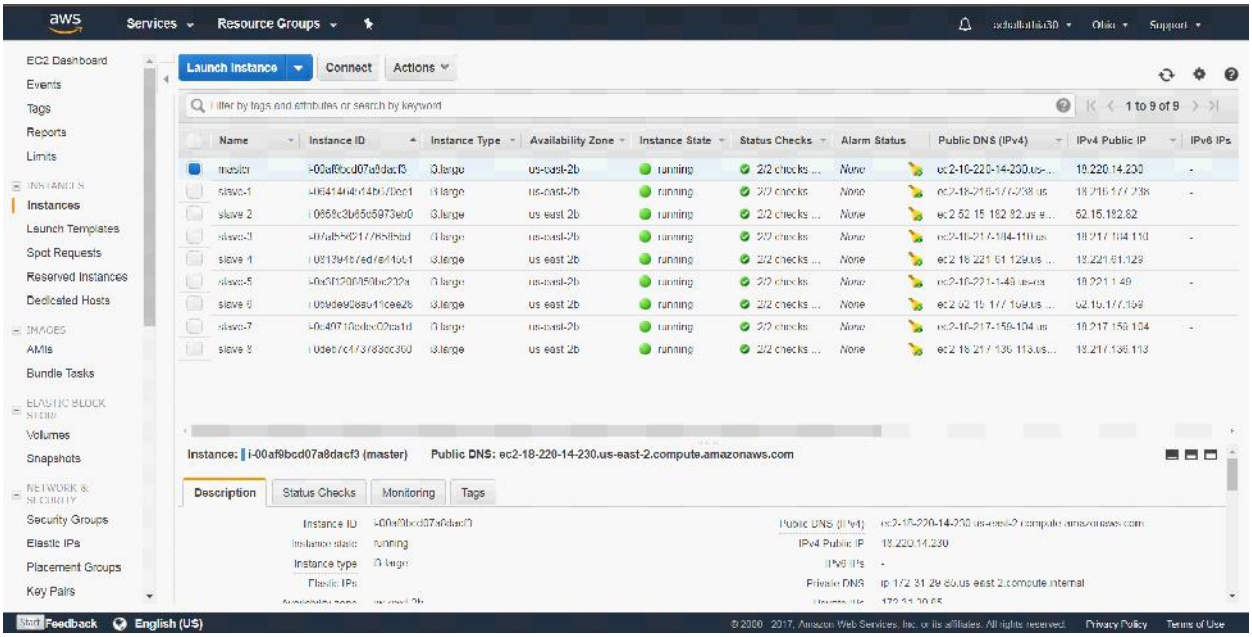
-Above screenshot is information of 2048 file each with the size in the range of 512 MB making it up to 1 TB



-Above screenshots infers that list of output part files of total size 1TB (2048 * 512MB). Hence, we have shown valsort on only first and last part of output file which shows that all records are in order

Spark 8 Node Cluster

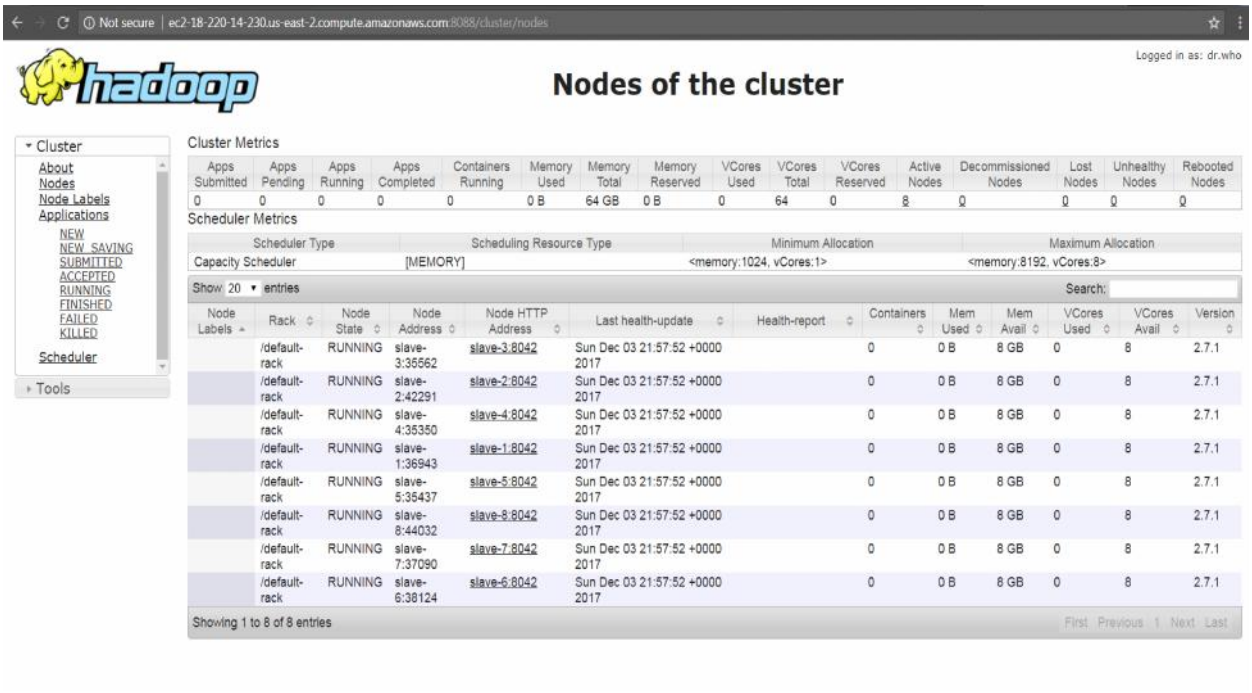
→We have run TeraSort for 10GB of dataset on 8 node cluster for Spark. The reason behind this is it takes almost 10 hours to put 1TB of data on hdfs. So, we have proportionally calculated results on basis of 10GB results and shown in performance evaluation.



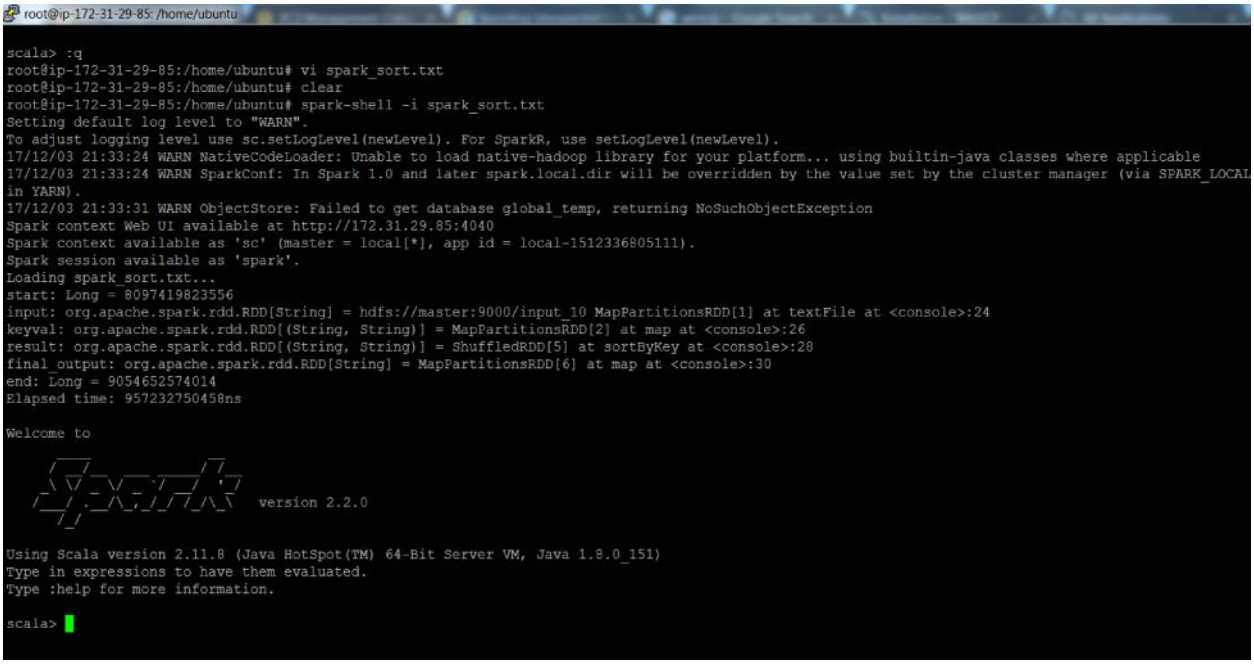
-Above screenshot is cluster information used in this experiment

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT



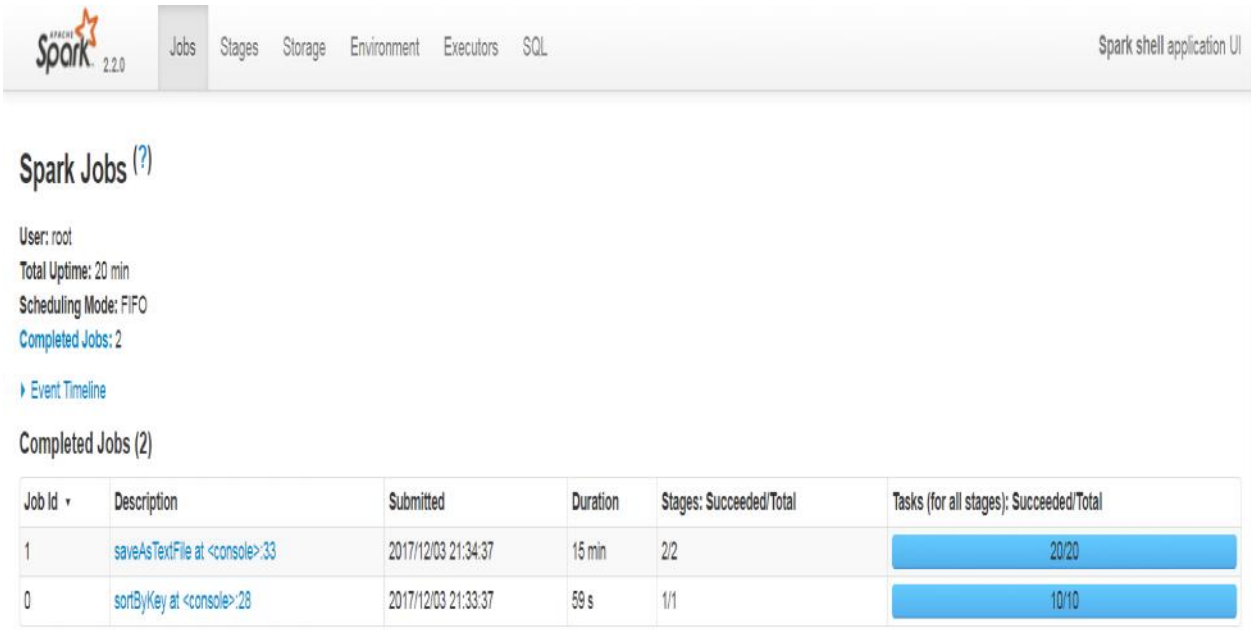
-Above screenshot is information of node available in the cluster

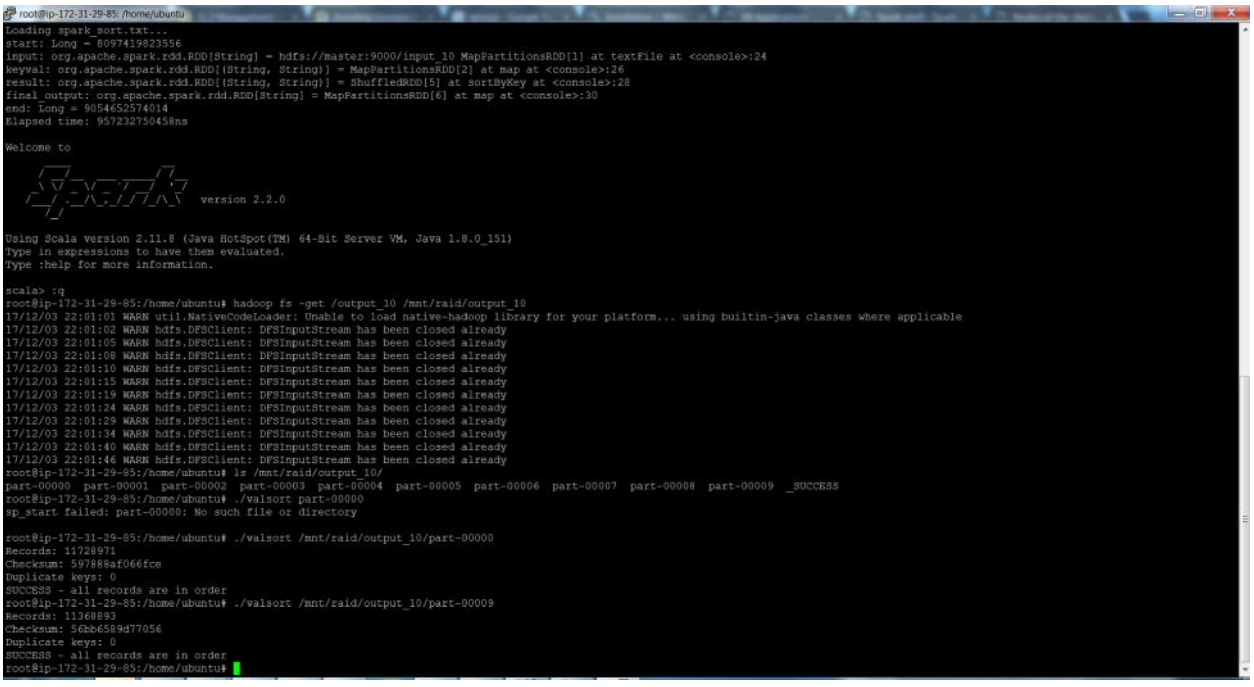


-Above screenshot is information that spark is started

Rushit Pandya A20381916
Achal Lathia A20375208
Viral Bhojani A20380100

PA-2-REPORT





-Above screenshots infers that list of output part files. Hence, we have shown valsort on only first and last part of output file which shows that all records are in order