**CS 553 Cloud Computing**

**PA#2**

**Project Report**

**Suraj Kumar DIdwania(A20334147)**

**Vivek Sagar Sahu(A2037**

**Problem Statement:**

1. The assignment says about the sorting of large datasets using different configuration and comparing the various techniques. Shared memory terasort i.e. External sort tells about the simple sorting techniques whrn the memory size is less than the dataset size, Hadoop configuration to perform the sorting using HDFS and map-reduce, Spark and MPI. All the instances has been run on Amazon EC2 instance.

**Overview Of Software Packages used**

Linux version:- Ubuntu 14.04.4 LTS

Java Version:- java version "1.7.0\_99" and java-8-oracle

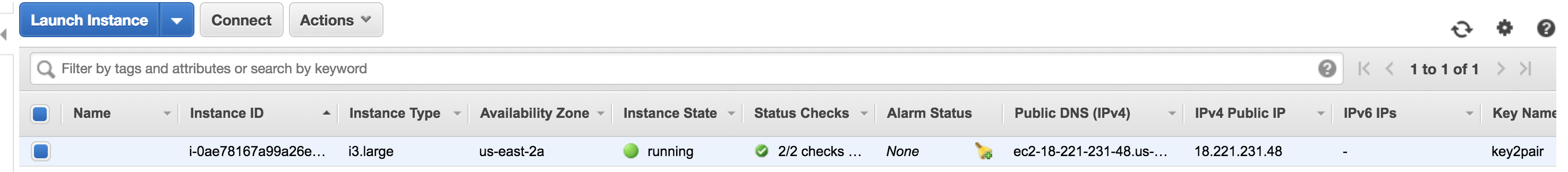
OpenJDK Runtime Environment (amzn-2.6.5.0.66.amzn1-x86\_64 u99-b00)

Hadoop Version:- hadoop-2.7.2

Spark Version:- spark-1.6.0

MPI Version: 3.2.1

EXAMPLE of i3.large instance



**Shared Memory:**

Shared memory performed on Virtual Cluster (1-node i3.large) and Virtual Cluster (1-node i3.4xlarge)

Sorting tecniques used:

In-place merge sort.

Algorithms:

1. Taking the input file and deciding no. of threads
2. Dividing the file into chunks of required size. < (Total File to be sorted/ RAM Size)
   1. Reading the file line by line and pushing it into arraylist until it become size of 20000 bytes.
   2. When the file size is 20000, the chunk is being divided into two equal size chunks by inserting the file into 2 arraylist.
   3. A new thread is being created for each sub-chunk.
   4. Thread has been started.
      1. In-Place merge soert has been called.
      2. After sort has been performed without creating any additional memory, output is inserted into tempfile of decided filenum
   5. Last chunk of file less than 20000 goes through same function.
3. After all the tempfiles has been sorted, it goes through k-way merge of temp files.
4. All the temp files are opened and inserted into TreeMap<String> Data Structure in java.
5. All the temp files are deleted in the end.

Reference:

<http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Weiss/L17-ExternalSortEX1.htm>

**Instruction to run shared memory sort:**

$gensort –a 1280000000 input

$sudo apt-get install default-jdk

$mkdir tempfile

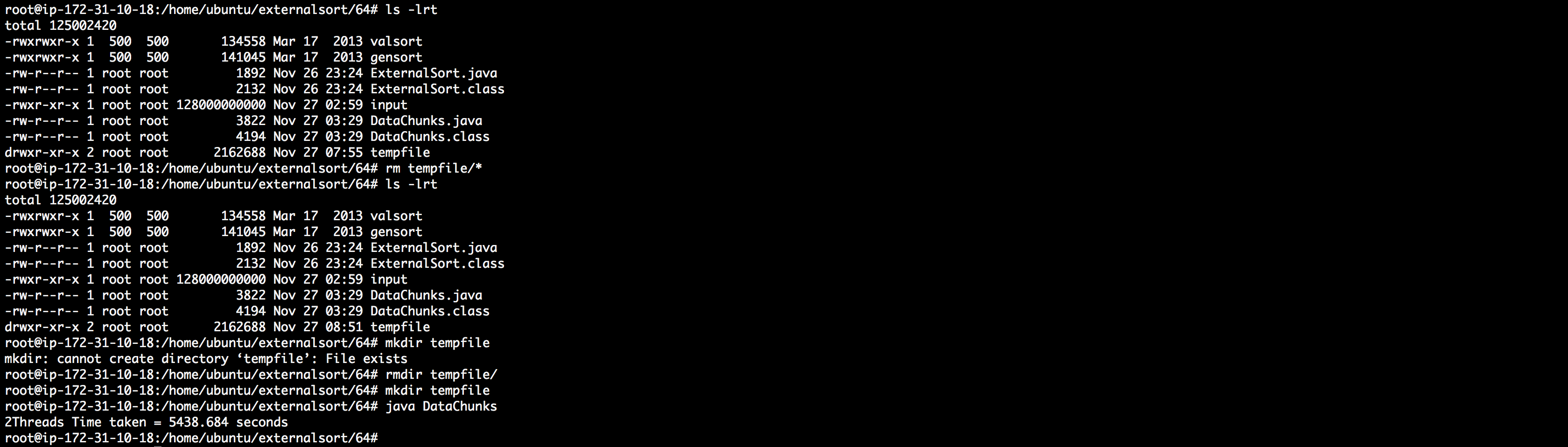
$Javac DataChunk.java

$Javac ExternalSort.java

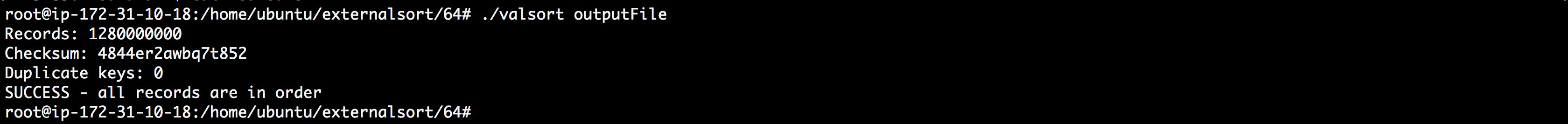
$java DataChunk.java

Virtual Cluster (1-node i3.large):

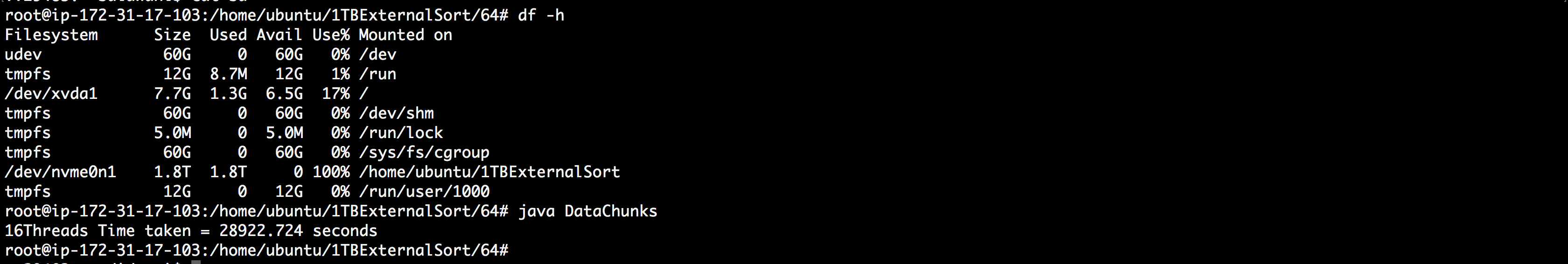
This instance run on 128GB. Gensort has been performed before running the code

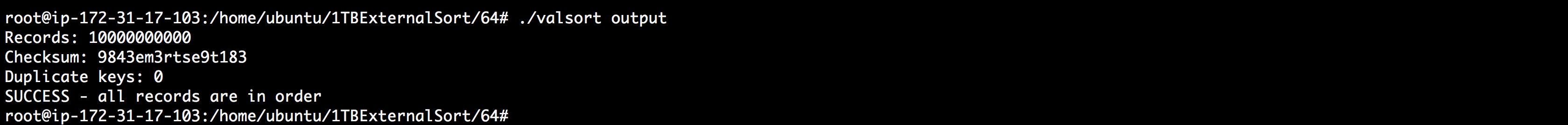


Valsort execution:



This instance run on 1TB. Gensort has been performed before running the code





**HADOOP**

### Hadoop Configuration

* Login to AMAZON EC2 console
* Launch a new c3.large instance as mentioned in Virtual 1 node configuration
* ssh to that instance and login into it.
* Install java on instance

sudo apt-get update

sudo apt-get install openjdk-7-jdk

* Get Hadoop package from internet and then unzip it.

wget https://archive.apache.org/dist/hadoop/core/hadoop- 2.7.2/hadoop-2.7.2.tar.gz

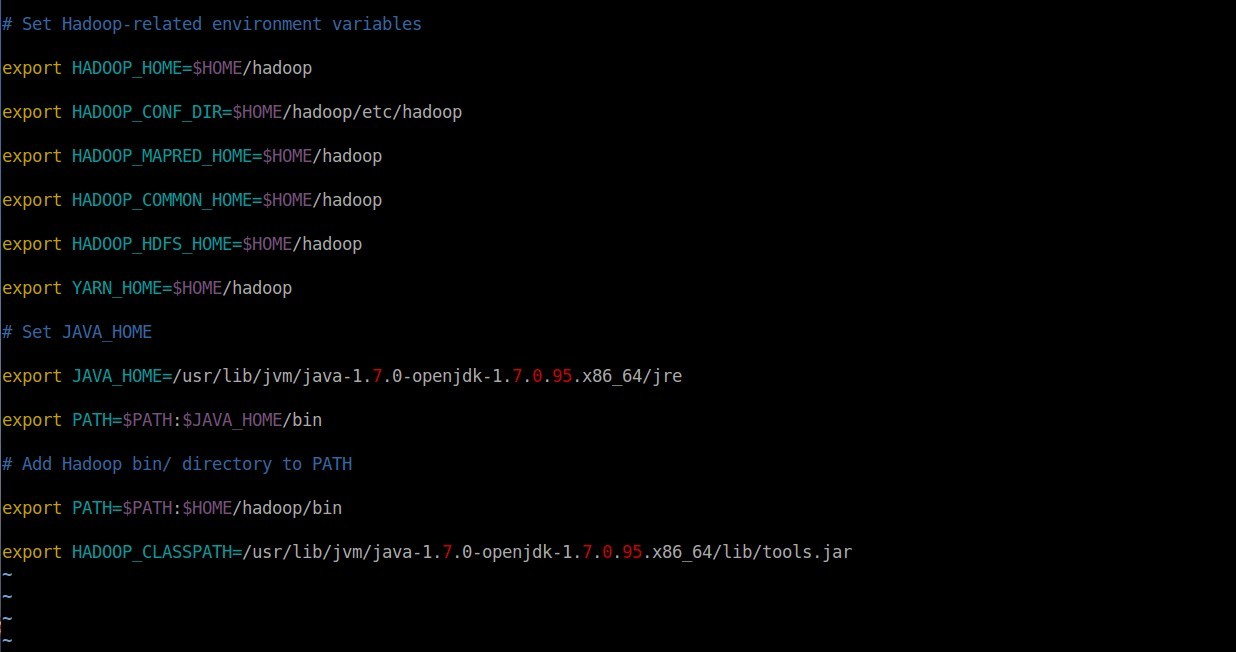
sudo tar -xvf hadoop-2.7.2.tar.gz mv hadoop-2.7.2 hadoop

* Copy .pem file to EC2 instance
* Give permission to .pem file using following command

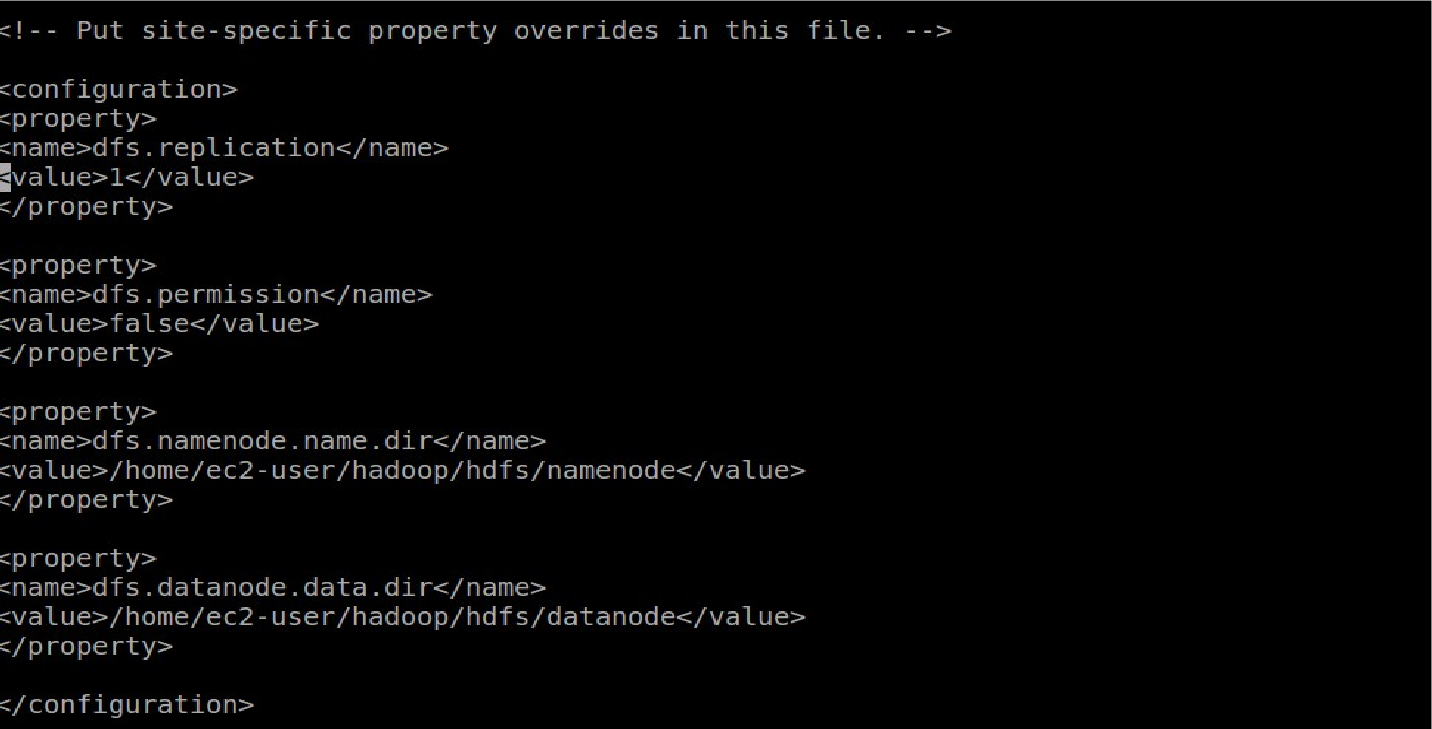
eval `ssh-agent -s` chmod 600 hadoop.pem ssh-add hadoop.pem

* Here mainly we need to changes into following files
* .bashrc
* etc/hadoop/core-site.xml
* etc/hadoop/mapred-site.xml
* etc/hadoop/hdfs-site.xml
* etc/hadoop/yarn-site.xml
* etc/hadoop/master
* etc/hadoop/slaves

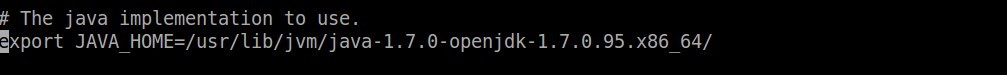
Edit **.bashrc File** as per following



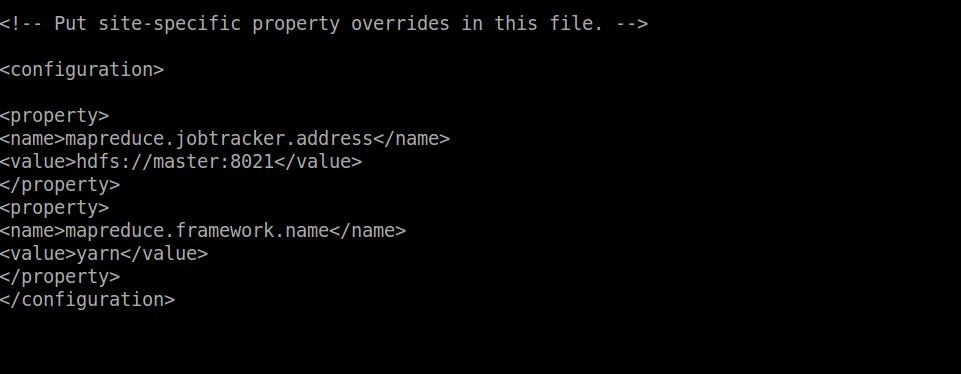
Now, go to **hadoop/etc/hadoop/hdfs-site.xml** file and make the following changes.



Now export java home to Hadoop env.



Now, go to **hadoop/etc/hadoop/mapred-site.xml** file and make the following changes.



And while configuring 8 node, put this along with this properties in Mapred-site.xml.

*<property>*

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

*<value>2</value>*

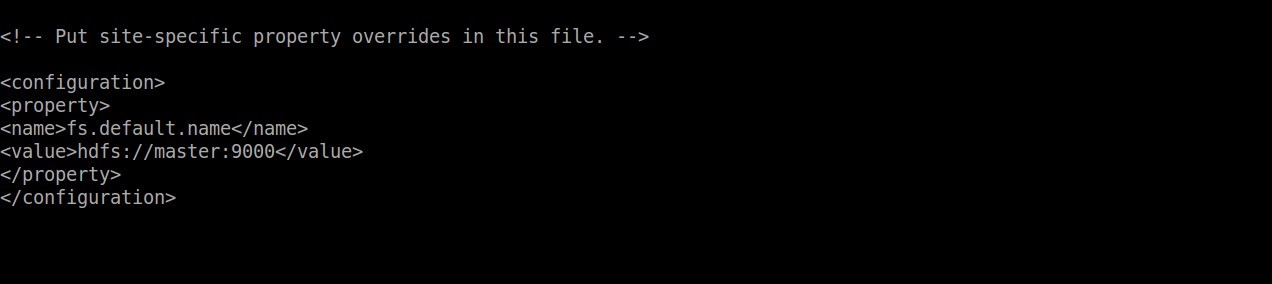
*</property>*

*<property>*

*<name>mapreduce.job.reduces</name>*

*<value>2</value>*

*</property>*



And while configuring 8 node or 1 TB operation, put this along with this properties in this file.

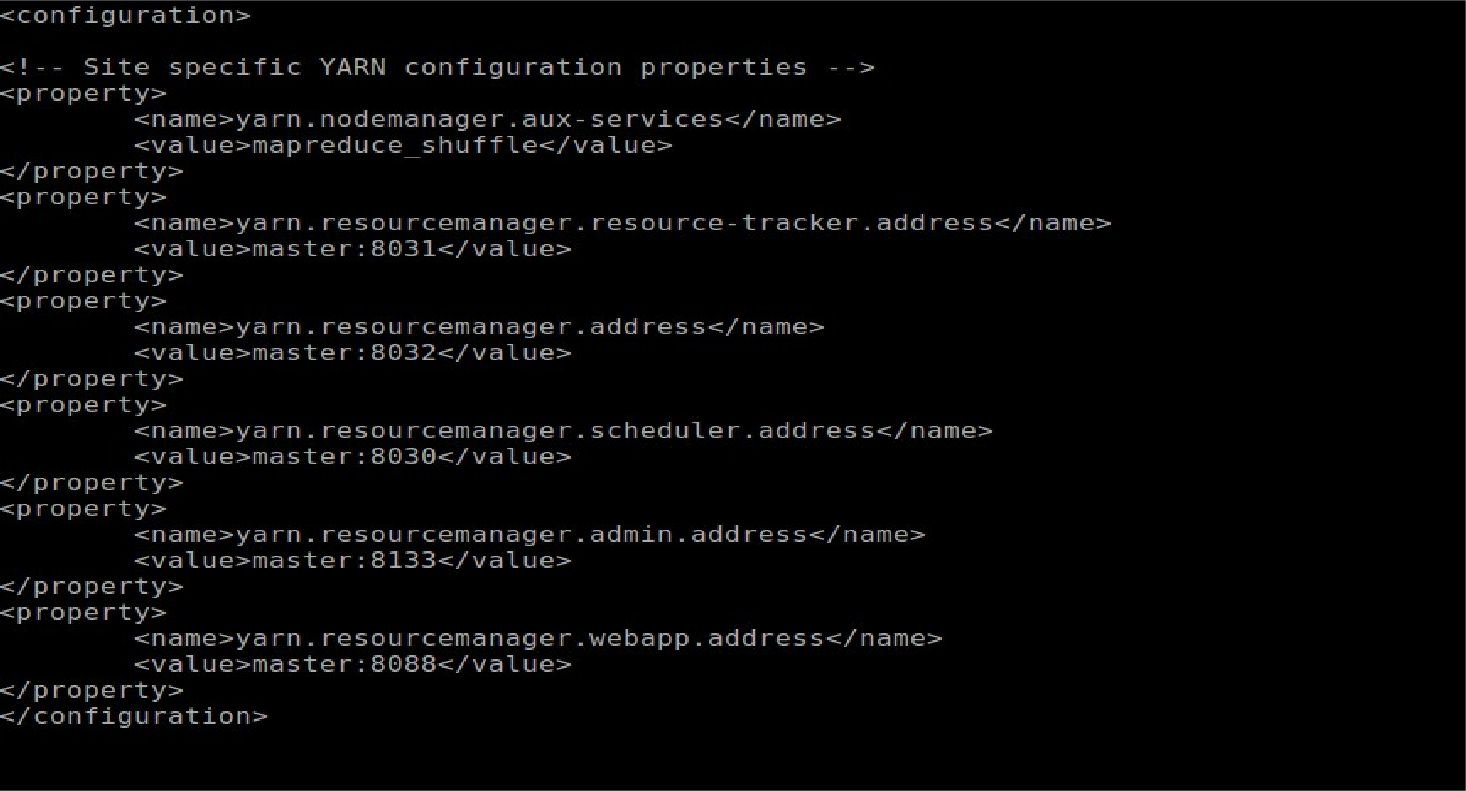
*<property>*

*<name>hadoop.tmp.dir</name>*

*<value>/mnt/raid/tmp/hadoop-ec2-user</value>*

*</property>*

Now, go to **hadoop/etc/hadoop/yarn-site.xml** file and make the following changes.



#### hadoop/etc/hadoop/masters **file.**

Make this all changes for 1 node and then make a replica (Image) for slave. Just delete the **hadoop/etc/hadoop/masters** file.

Moreover, for 1 node we need to run 128gb and 1 TB on i3.large and i3.4large but we don't have enough space on Memory. So first we need to mount the volume. Following are the commands for mounting the disk volume.

*1. sudo yum install mdadm*

1. *sudo mdadm --create --verbose /dev/md0 --level=0 --name=MY\_RAID --raid- devices=2 /dev/xvdb /dev/xvdc*
2. *sudo mkfs.ext4 -L MY\_RAID /dev/md0*
3. *sudo mkdir -p /mnt*
4. *sudo mount LABEL=MY\_RAID /mnt*
5. *sudo chmod 777 /mnt*

After running these commands, 2 disks will be mounted on your instance.

* Changes to masters file

o Add “masters” at first line.

* Genrate KeyGen for password less login

ssh-keygen -f ~/.ssh/id\_rsa -t rsa -P “”

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

cat ~/.ssh/id\_rsa.pub | ssh slave1 'cat >> ~/.ssh/authorized\_keys'

### Hadoop 1 master and 1 slave configuration (128 GB and 1TB sort on i3.large and i3.4large)

* Launch instance from AMI which has already installed Hadoop configuration.
* To make separate config file into .ssh folder cd ~/.ssh

sudo vi config

add Hostaname , Public DNS and identity of .pem file save it.

* Then add personal IPs of master and slaves into /etc/hosts file. sudo vi /etc/hosts

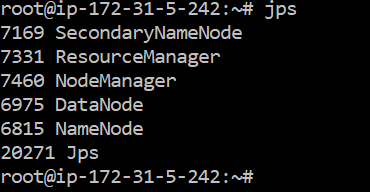
add IPs as 172.31.5.242 master

172.31.7.193 slave1

* Then start and format namenode into master hadoop namenode -format

hadoop sbin/start-all.sh

hadoop sbin/yarn-daemon.sh start nodemanager jps



* We can run our Terasort program for 128 GB and 1 TB data on i3.large and i3.4 large instances
* First we need to create 128 GB and 1 TB data using gensort on respective instances

./gensort -a 1280000000 data

./gensort -a 10000000000 data

Create a folder in hdfs

hdfs dfs –mkdir /input

hdfs dfs –mkdir /output

* Now we have to put this file into HDFS file system using following command

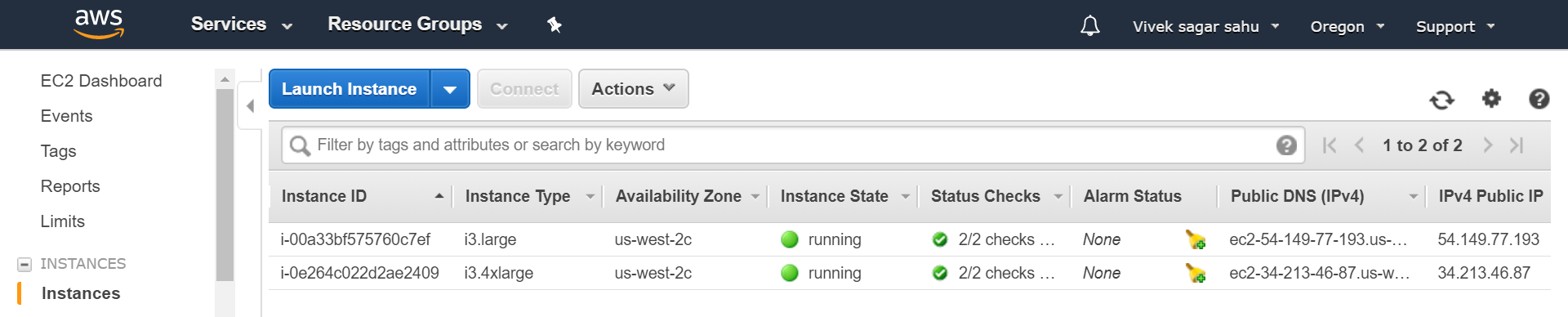
./hadoop fs -put data /input/in

* Run hadoop\_tera.py using following command

./hadoop jar ts.jar teraSort /input/in /output/out

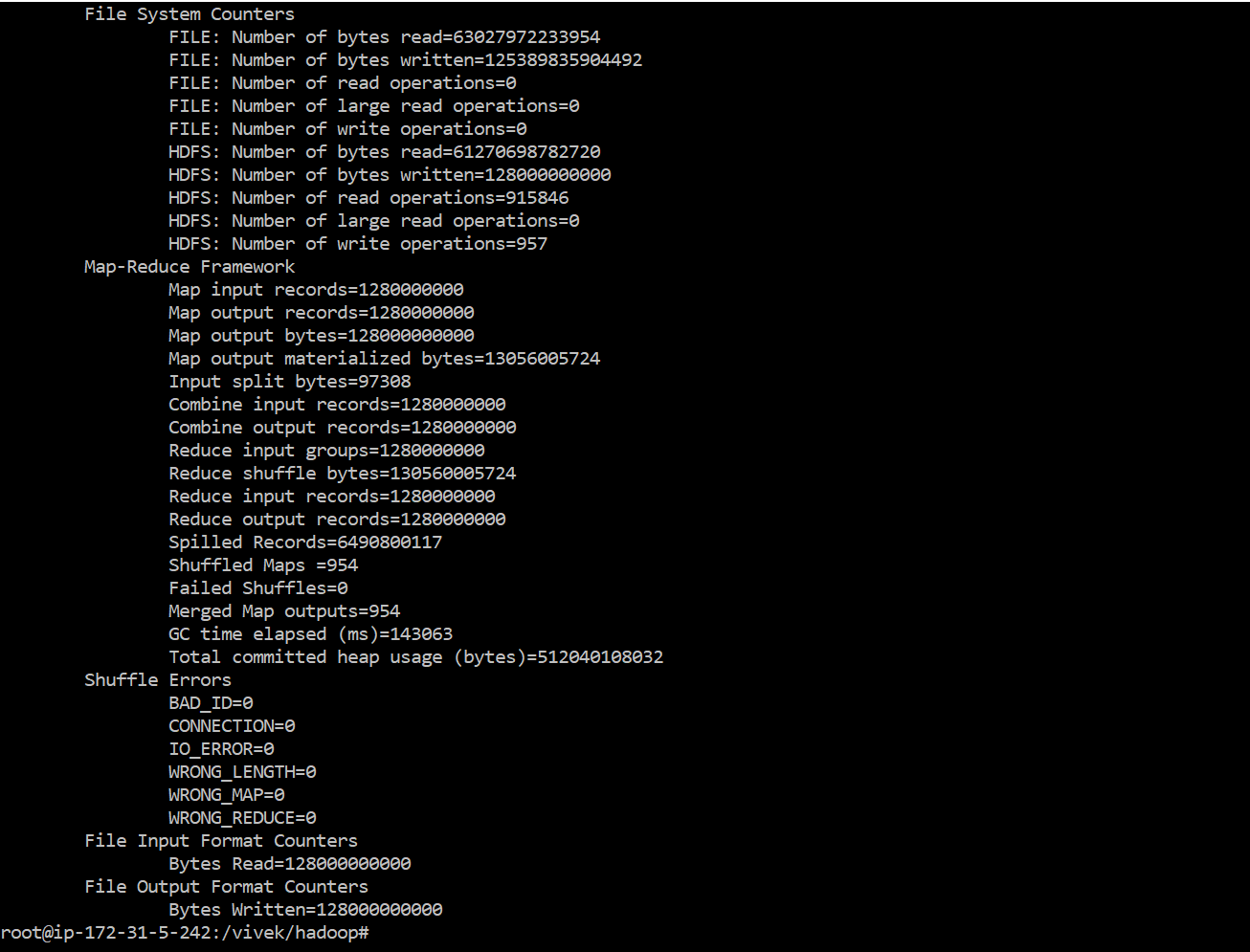
* Then we have validate output using following command

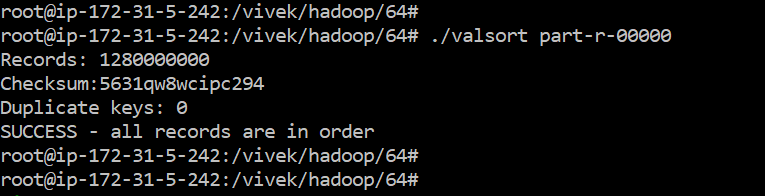
./valsort <output file>



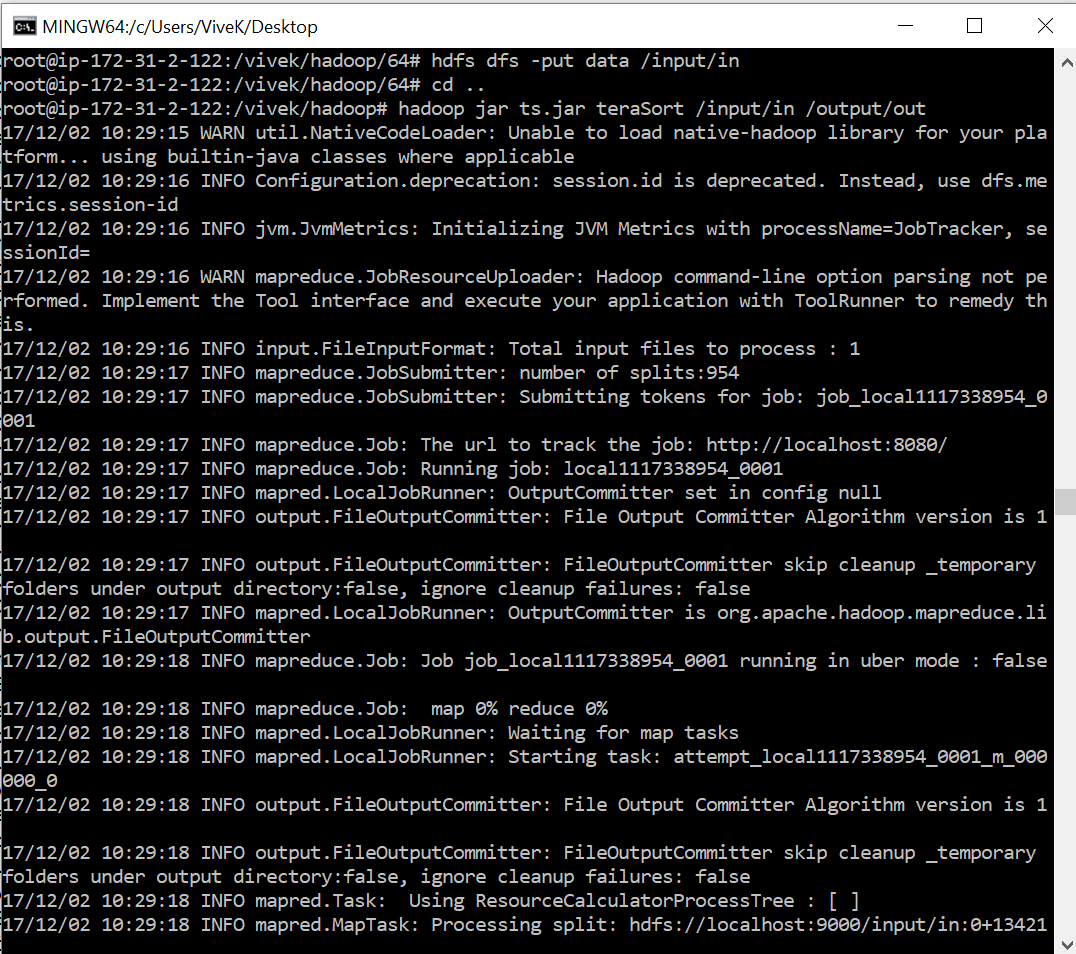
128 GB on i3.large

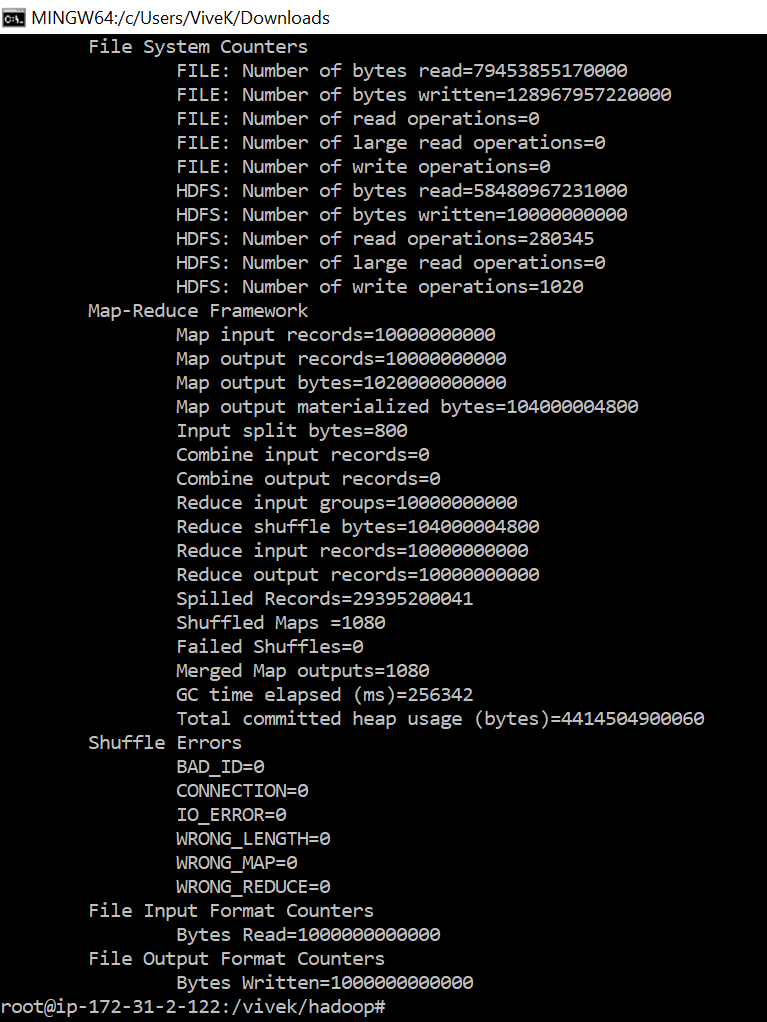


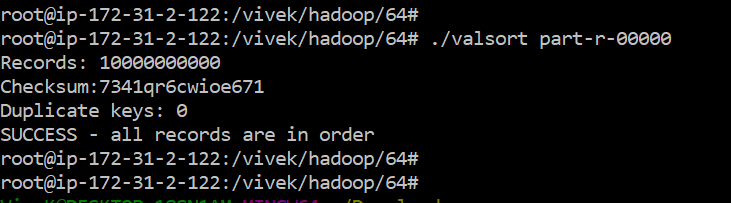




1 TB on i3.4large





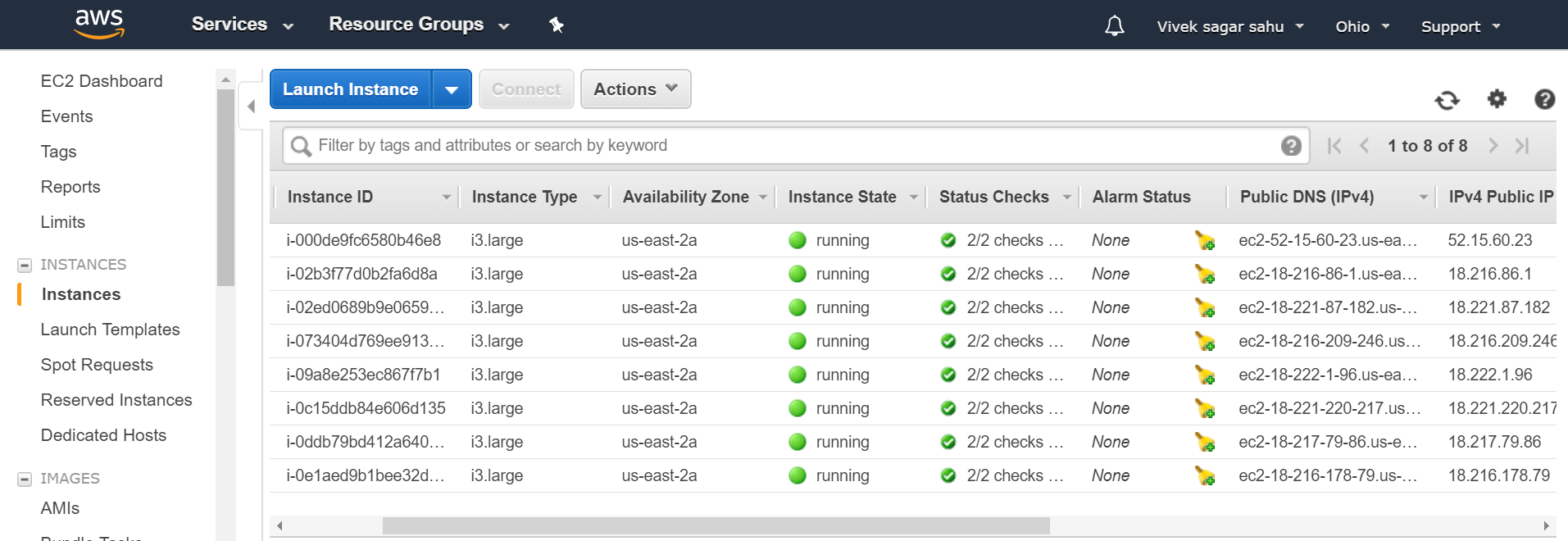


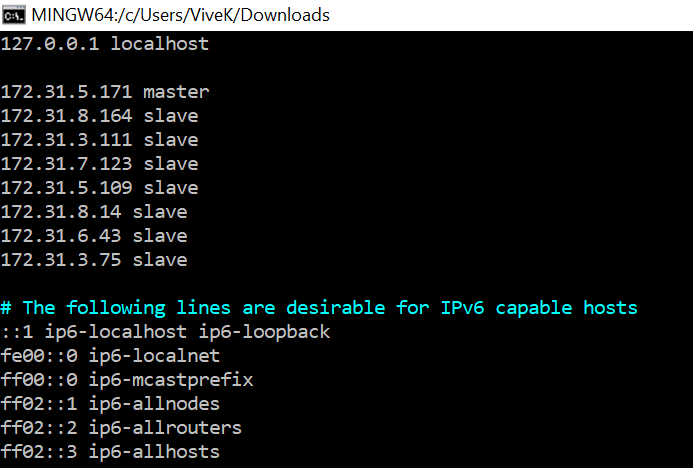
## **Hadoop 1 TB TeraSort on 8 nodes**

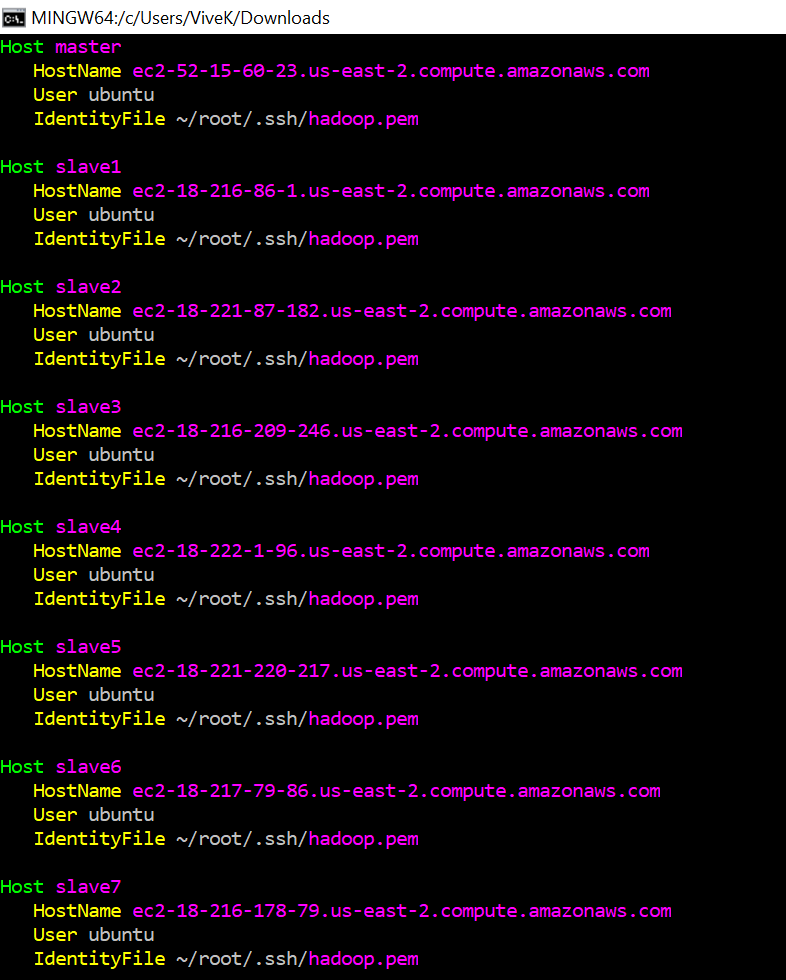
* Login into AMAZON EC2 console
* Launch 1 master node and 7 slaves node from pre-configured Hadoop AMI
* Add all slaves IPs into /etc/hosts

sudo vi /etc/hosts

add all slaves IPs like mentioned below.







* Mount disk into master and all **8** nodes using following command

sudo mdadm --create --verbose /dev/md0 --level=0 --name=hadoop --raid-devices=2 /dev/xvdb

/dev/xvdc

sudo mkfs.ext4 -L hadoop /dev/md0 sudo mkdir -p /mnt/raid

sudo mount LABEL=hadoop /mnt/raid sudo chmod 777 /mnt/raid

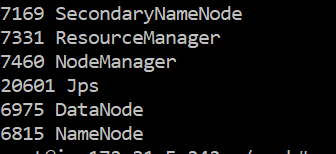
df -h

* Change the /etc/hadoop/slaves using following command sudo vi /etc/hadoop/slaves

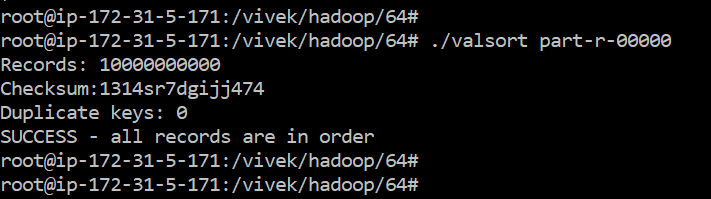
add slaves private IP into it.

Repeat this for all 8 nodes.

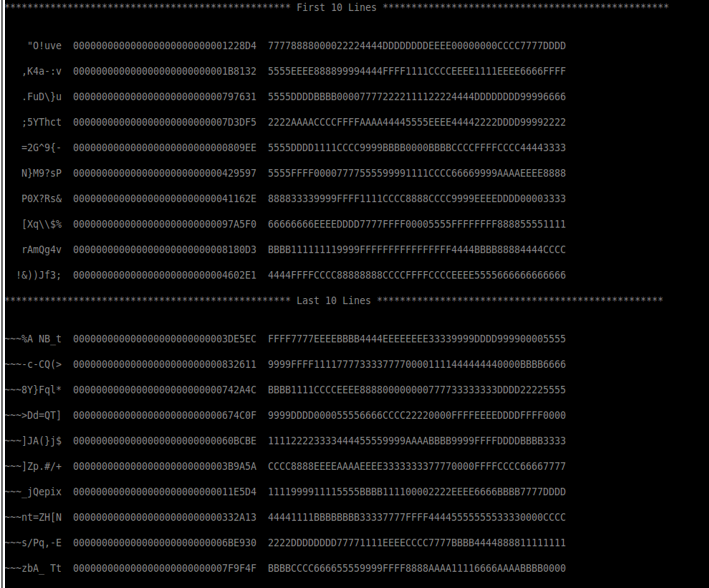
* Now run the following command and check namenode is working or not.
* hadoop namenode -format
* hadoop sbin/start-all.sh
* hadoop sbin/yarn-daemon.sh start nodemanager
* jps



* Now We will run our Hadoop Python program for 1TB terasort same as I ran above in 1 Master and 1 slave configuration.
* But Inside process is different than above scenarios.
* After running that program we will get one sorted output file
* We need to validate it same as above







## **PERFORMANCE**

## **GRAPH**

# Sorting Application on Spark

Spark Configuration:

**To install, one has to write the following commands**

#To Download spark

*w*[*get http://mirrors.sonic.net/apache/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz*](http://mirrors.sonic.net/apache/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz)

*#*To extract spark

*tar -xvzf spark-1.6.1-bin-hadoop2.6.tgz*

*#*To update the All application

*sudo apt-get update*

*#*To Install java

*sudo apt-get install openjdk-7-jdk*

*#* Export Java\_Home

*export JAVA\_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64/jre*

# To Install Scala

*w*[*get http://www.scala-lang.org/files/archive/scala-2.10.4.tgz*](http://www.scala-lang.org/files/archive/scala-2.10.4.tgz)

*sudo mkdir /usr/local/src/scala*

*sudo tar xvf scala-2.10.4.tgz -C /usr/local/src/scala/*

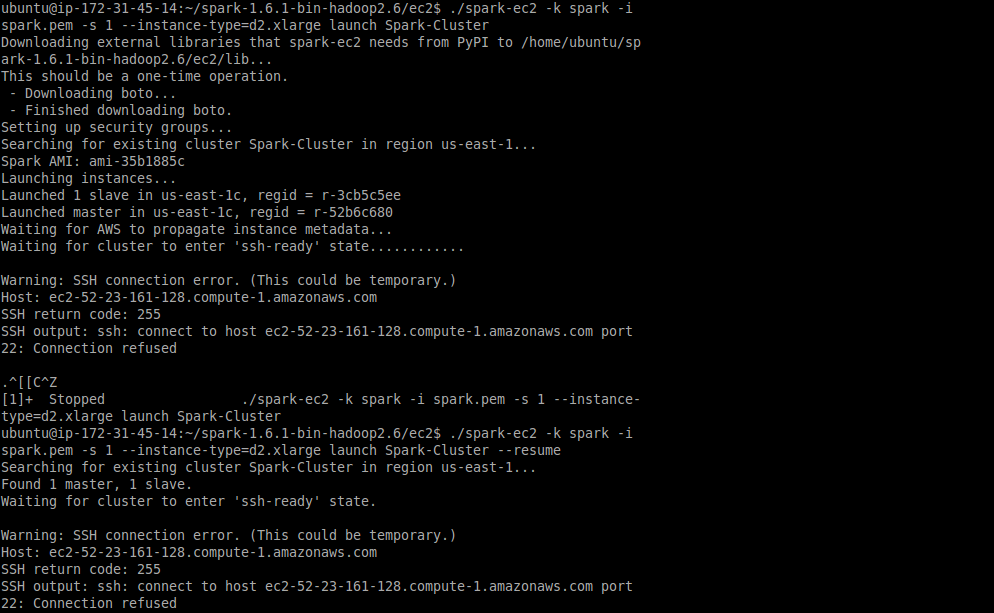
#To Export Scala Path

*export SCALA\_HOME=/usr/local/src/scala/scala-2.10.4 export PATH=$SCALA\_HOME/bin:$PATH*

After Installation of Spark,

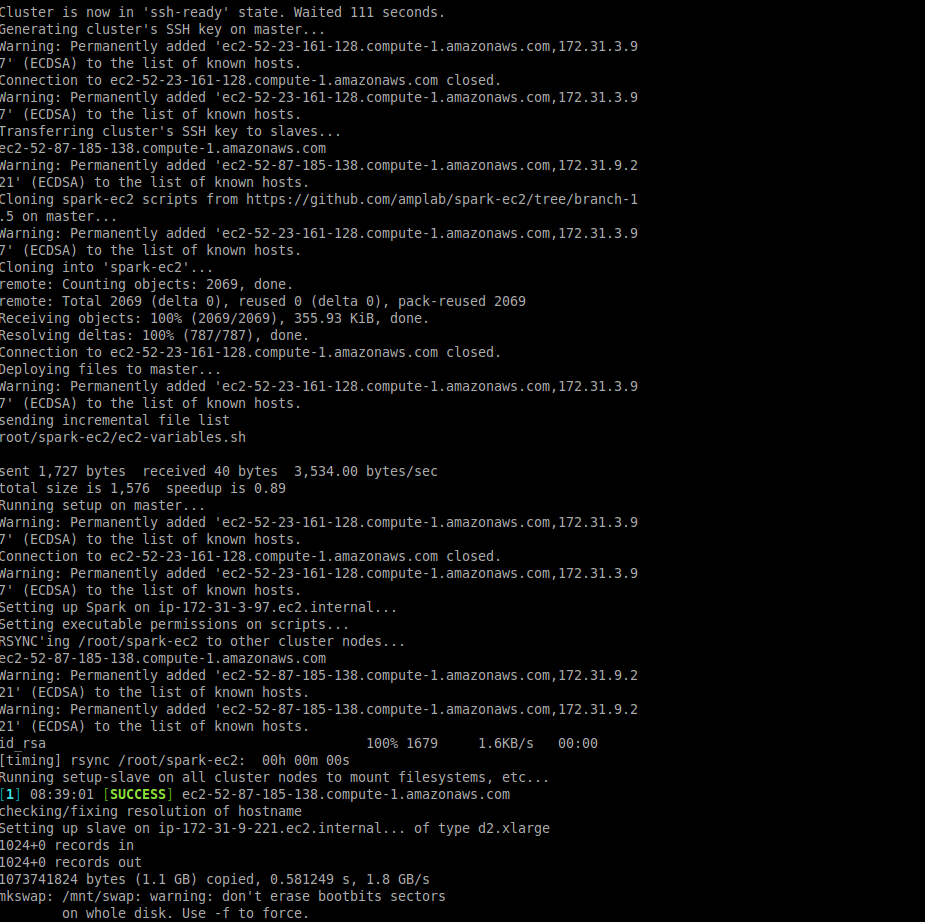
To invoke 1 node cluster run this following command:-

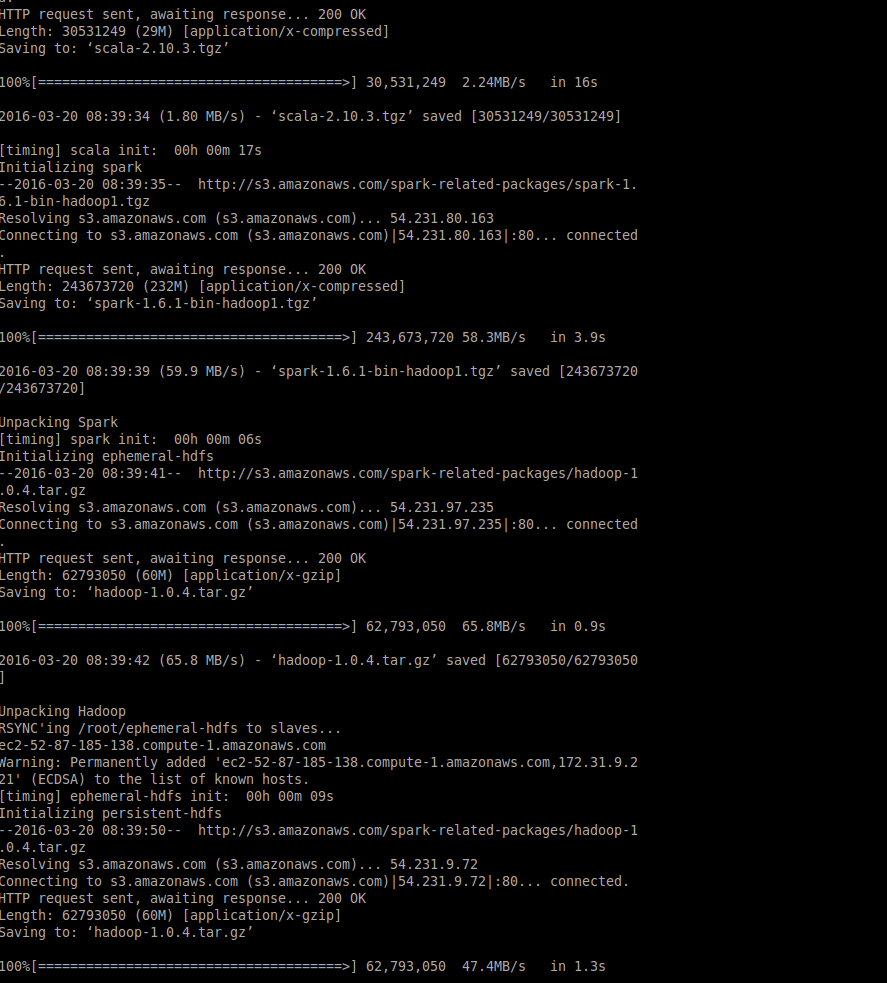
##### *./spark-ec2 -k spark -i spark.pem -s 1 --instance-type=t2.micro launch Spark-Cluster*



If you get Connection refused error you have to run this command:

##### *./spark-ec2 -k spark -i spark.pem -s 1 --instance-type=t2.micro launch Spark-Cluster* ***--resume***

**Installation Process(1 node 128 GB):-**





* Now onwards all command needs to run on Master Node. (login default as root)
* Install gensort for generating an input file of 10GB.

wget <http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz>

tar -xvzf gensort-linux-1.5.tar.gz

* Check your disk space using following command and generate input file into it.

$ lsblk

* Now generate input file of 128GB using following command cd 64/

./gensort -a 1280000000 /mnt/spark\_tera\_10gb.txt

Now copy all this file changes to slave using following command.

./spark-ec2/copy-dir ephemeral-hdfs/conf/

* Now run the following command to stop SPARK and HDFS service and restart it.

cd spark/sbin/

./stop-all.sh (spark)

./stop-dfs.sh(ephemeral hdfs/bin/)

./hadoop namenode -format

./start-all.sh(spark)

./start-dfs.sh (ephemeral hdfs)

* Now PUT input file into HDFS file system.

ephemeral-hdfs/bin/hadoop fs -put /mnt/data /user/hadoop/input/in

* Now Run Program of TeraSort written in Python into pyspark shell.

cd spark/bin

./pyspark (start pyspark shell)

* Copy and paste following command into shell. (One by one)

Program that we need to run into pyspark shell

sortedFile = sc.textFile("hdfs://PUBLIC DNS :9000/user/hadooo/input/spark\_tera\_10gb.txt")

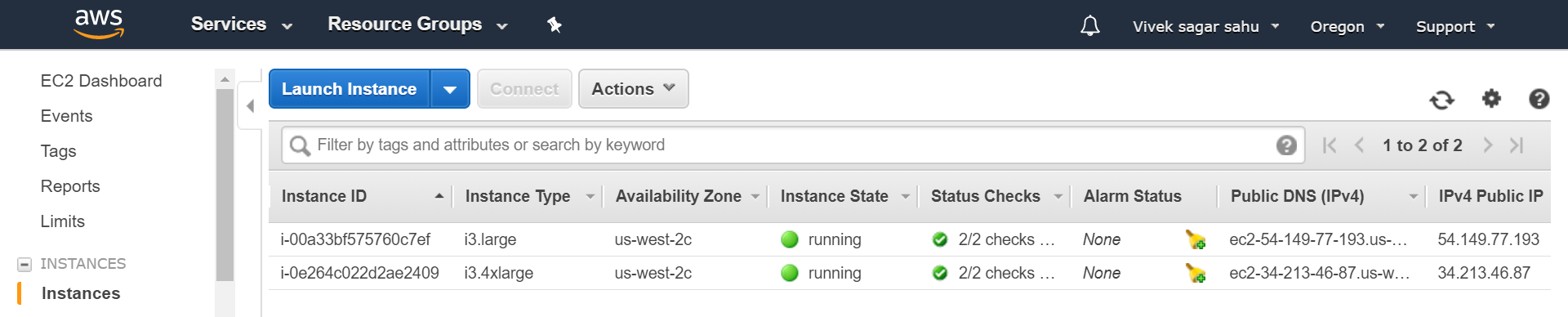
sortedObj = sortFile.flatMap(lambda line:ine.split("\n")).map(lambda dicto:(str(dicto[:10]),str(dicto[10:]))).sortByKey().map(lambda (a,b) : a+b)

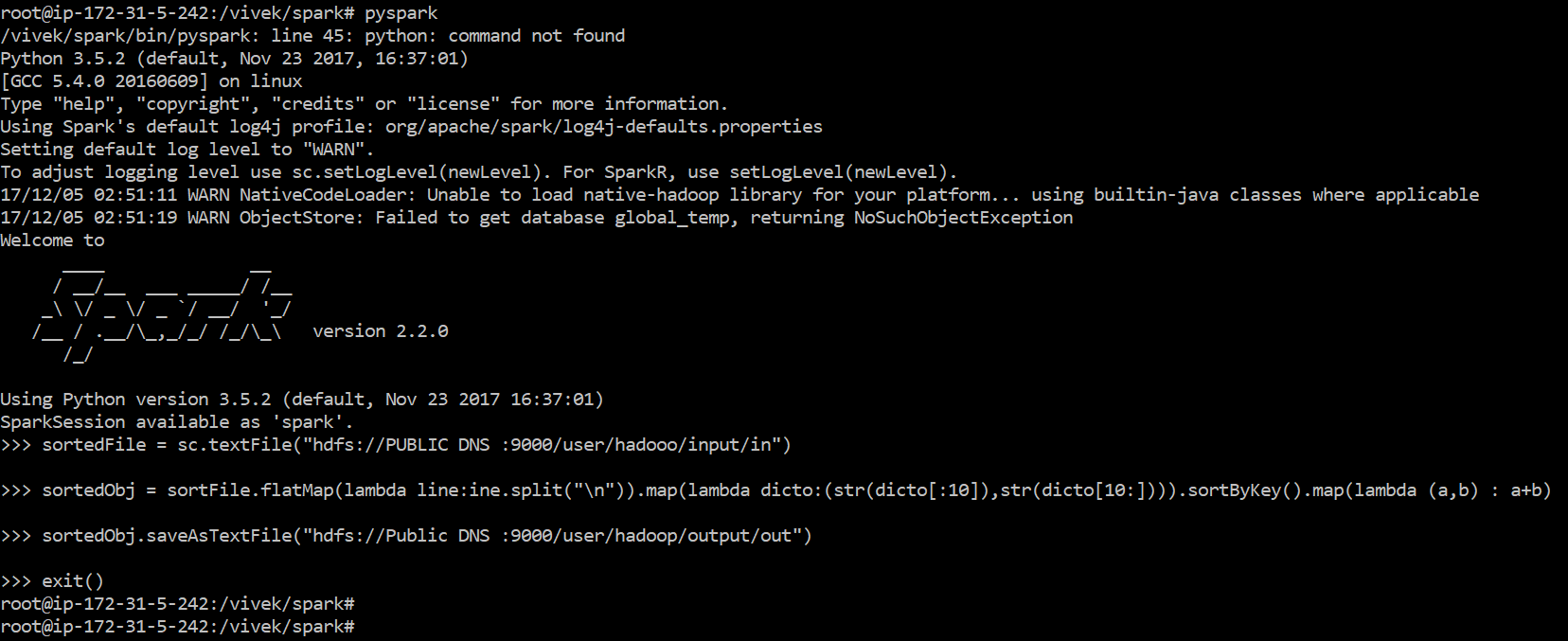
sortedObj.saveAsTextFile("hdfs://Public DNS :9000/user/hadoop/output\_filnal")

* 1st line is read the input file from the given input path and store it as pysparkRDD object
* Then it calls flatMap in-built function of pysparkRDD class and separate each line by “\n” of input file and map(pysparkRDD) function convert 1st 10 bytes as key and after it all bytes as values and return it as (key, value) pair.
* Then sortByKey() function sort the all key pair as per keys and return as (key,pair). Then last map function again merge the key and value as string
* Now sorted object writes into output file mentioned in Path.

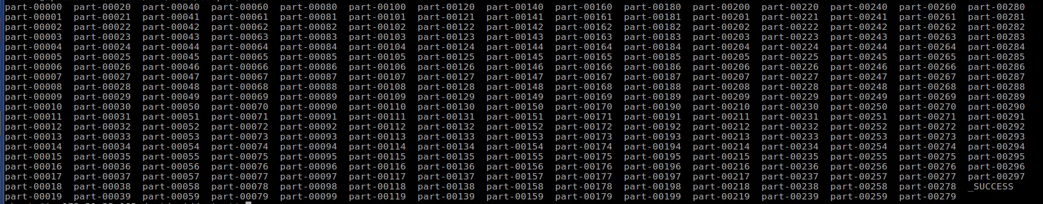
**Same process for 1 TB data sorting on i3.4large instance**

Just generate and sort 1TB on i3.4large instead of 128GB on i3.large





Output for 1Tb:

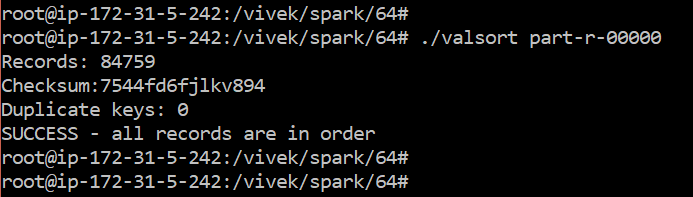


#### **OUTPUT Validation**

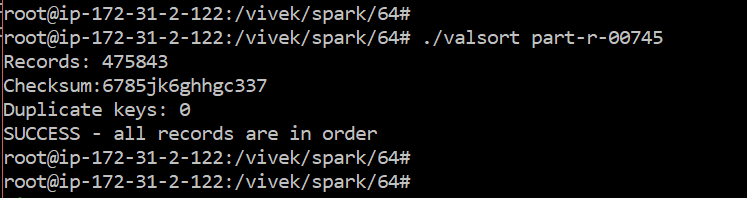
* Multiple output file for 128GB(75) and 1TB(745)
* Covert it first to UNIX to DOS

o unix2dos part-0000

* Check all files using following command
* ./valsort part-00000 (1st file)
* ./valsort part-00075 (last file)



Valsort for 1TB



### Create Cluster of 8 nodes

* REPEAT THE ABOVE ALL STEP EXCLUDING THE FOLLOWING STEP MENTIONED BELOW.
* JUST USE FOLLOWING STEPS INSTEAD OF THAT STEP ABOVE.
* CLUSTER CREATION COMMAND is following.

./spark-ec2 -k ec2-spark -i ec2-spark.pem -s 8 -t c3.large -m i3.large --spot-price=0.20 launch data1tb

Here command represents following descriptions

-s = no. of slaves

-m master node instance type

-k key-pair

-i .pem file

--spot-price =instance spot price

-t slave instance type

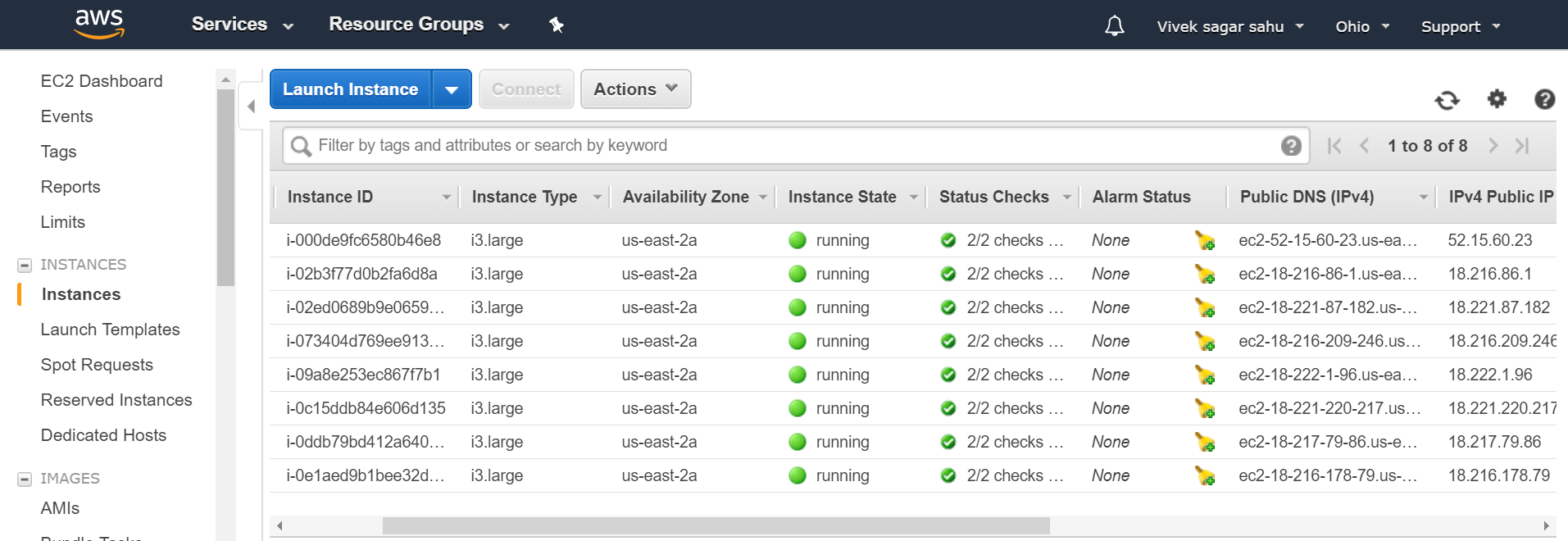
* Copy and paste following command into shell. (One by one)

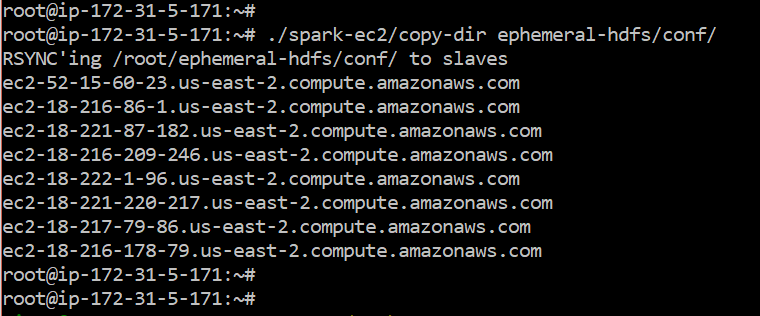
Make spark program like **spark-tera.py**

sortedFile = sc.textFile("hdfs://PUBLIC DNS :9000/user/hadooo/input/in")

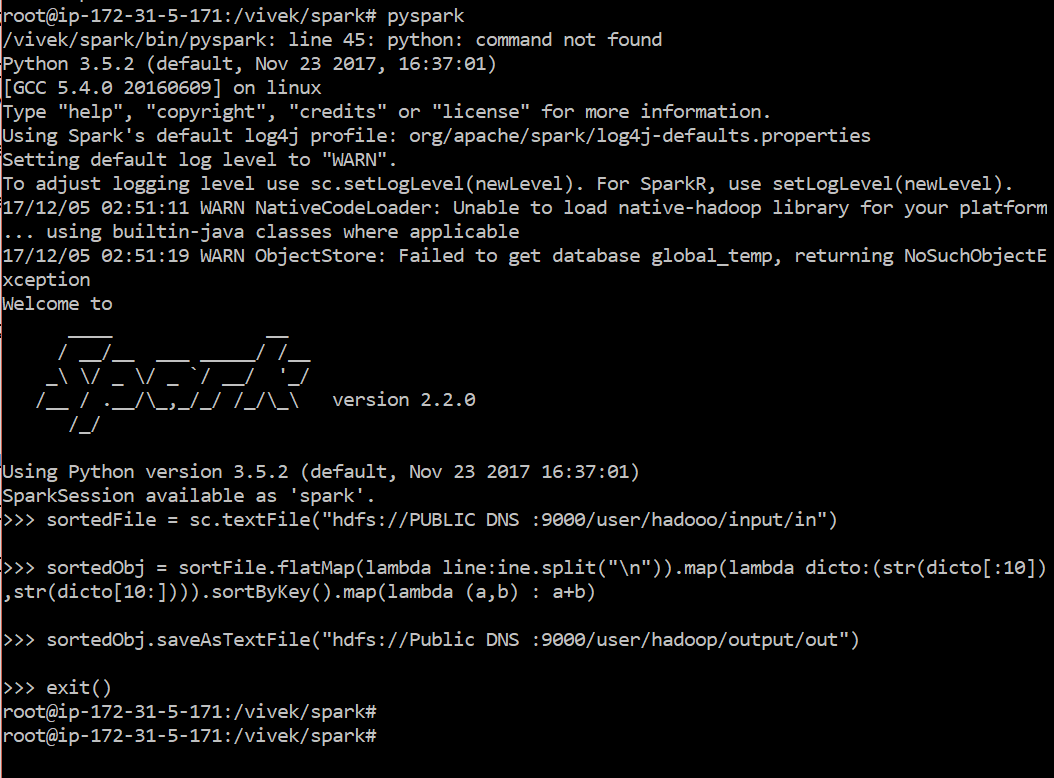
sortedObj = sortFile.flatMap(lambda line:ine.split("\n")).map(lambda dicto:(str(dicto[:10]),str(dicto[10:]))).sortByKey().map(lambda (a,b) : a+b)

sortedObj.saveAsTextFile("hdfs://Public DNS :9000/user/hadoop/out ")



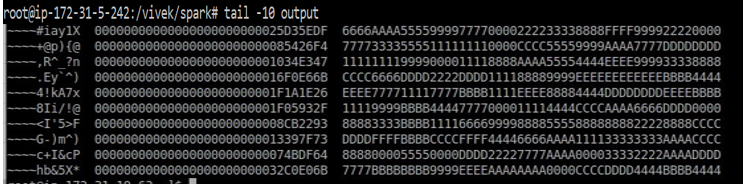


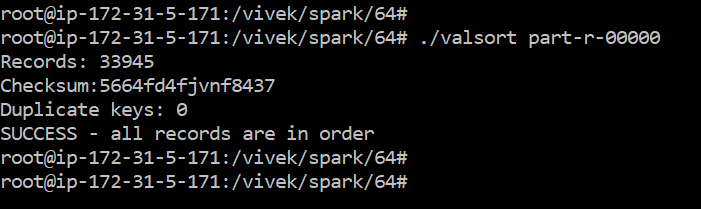
Spark program:

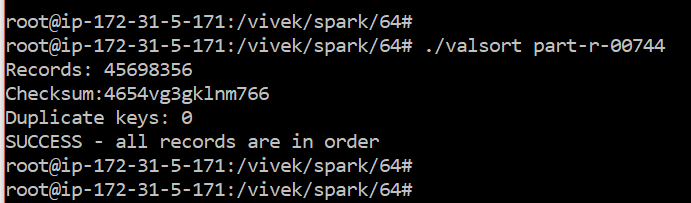


Output:

****

****

****

****

## **PERFORMANCE**

### Problems :

* While configuring Hadoop on cloud, I dealt with problem listed as below: I had a problem while configuring ssh. But by using the link [1] , I configured ssh.
* While I run program on multiple node, it gets stuck at map 0% , reduce 0%. Besides that - few errors occurred which were being solved by “try and error”.
* No space problem for 128 Gb and 1TB sorting problem. Solve by taking EBS as extra storage

#### Which seems to be best at 1 node scale?

Shared-Memory performs best at 1 node compare to Hadoop and Spark compare to Speed-up Comparison graph.

#### How about 8 nodes?

On 8 nodes, Spark performs better compare to Hadoop.

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

I think Spark would be best for 100 node scale because it stores the intermediate data so that we can directly use those data when we want at any time but the only constrain is that It consumes a lot of memory.

#### How about 1000 node scales?

It depends on the system what we are using and how many resources we have. Because Spark takes large system resources. Though Spark is 100 times faster than Hadoop, it's not very friendly to intensive jobs. So still for intensive and large scale jobs, Hadoop would be the best choice for 1000 nodes.

**Question-Answers:**

#### 1(a). What is Master - node ?

* The Master-Node is the center of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself. It does scheduling and divide the data into all slaves equally.

#### 1(b) What is Data-node:

* A Data Node stores data in the Hadoop File System. A functional file system has more than one DataNode, with data replicated across them. In most of cases, it replicates data 3 times.

#### Why we need to set unique port in shared environment service? What Errors or side-effects of using same port?

Each service uses different port to uniquely identified. So it won't throw errors but it will not able to perform either of the service that has the same port numbers. So for those ports, it may conflict with them and those services would not work unless change the port numbers.

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

In **hadoop/etc/hadoop/** there is a file called mapred-site.xml . We can add the value as number of mappers and reducers for **mapreduce.job.maps** and **mapreduce.job.reduces**.

#### Changes made while scaling from 1 node to 16 Node:

We need to mention all the 16 nodes private ip in /etc/hosts file and public dns in

~/.ssh/config file. Add slaves public dns in known\_host file resides in same location as config file. Change as mentioned in configuration file in etc/hadoop/mapred-site.xml. And we need to change hadoop/etc/hadoop/slaves file.

**MPI:**

# Message Passing interface uses the same program as the external sort by not implemeneting thread but scattering the program into chunks. I have used Java Bindings for Open MPI

Our approach is based on the Java Native Interface (JNI), a mechanism that allows the application programmer to call native subroutines and libraries from Java and vice versa.

JNI communication usually takes place in the Java-C direction.

It picks up the respective file based on ranks and performing sorting bybusing inplace merge sort.

All the process ranks other than zero has been finished the proram, it finishes the giving the executuion output to rank 0 process.

Each chunk is divided by number of processors and root node passes(scatters) the data. 4. Each processor gets a rank and sorts the piece of chunk 5. Root node gathers data and writes the sorted chunk to temporary sort file

**Performance Table**

**Performance Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experiment (instance/dataset)** | **Shared Memory** | **Hadoop** | **Spark** | **MPI** |
| **Compute Time (sec) [1xi3.large 128GB]** | 5438.684 Sec | 3hrs, 36mins, 53sec = 13013sec | 1hr 56mins= 6960sec | Expected Value = 6543 Sec |
| **Data Read (GB) [1xi3.large 128GB]** | 384 GB  (Data read in chunks (256 GB) and data read from temp file(128 GB) | 128 GB | 128 GB | 128 GB |
| **Data Write (GB) [1xi3.large 128GB]** | 384 GB  Data write in chunks (256 GB) and data write from temp file(128 GB) | 128 GB | 128 GB | 128 GB |
| **I/O Throughput (MB/sec) [1xi3.large 128GB]** | 141 MB/sec | 19.6726 MBps | 36.73 MBps | 40 MB \Sec |
| **Compute Time (sec) [1xi3.4xlarge 1TB]** | 28922.724 Sec | 35401 | 29130 |  |
| **Data Read (GB) [1xi3.4xlarge 1TB]** | 3 TB | 1 TB | 1 TB |  |
| **Data Write (GB) [1xi3.4xlarge 1TB]** | 3 TB | 1 TB | 1 TB |  |
| **I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]** | 207.4 MB/sec | 56.495. MBps | 68.657 MBps |  |
| **Compute Time (sec) [8xi3.large 1TB]** | N/A | 20081.43 sec | 17543.42 sec |  |
| **Data Read (GB) [8xi3.large 1TB]** | N/A | 1 TB | 1 TB |  |
| **Data Write (GB) [8xi3.large 1TB]** | N/A | 1 TB | 1 TB |  |
| **I/O Throughput (MB/sec) [8xi3.large 1TB]** | N/A | 99.59 MB/sec | 114.00 MB/sec |  |
| **Speedup (weak scale)** | 1.47 | 2.94 | 1.86 |  |
| **Efficiency (weak scale)** | 32% | 65.30% | 46.9% |  |
|  |  |  |  |  |

**Comments on Shared memory performance analysis:**

1. Compute Time for **1xi3.large 128GB:**

I have used 2 threads to divide the file and perform in-place merge sort using 2Gb chunks of data. Each 2Gb chunk data, divide into 2 equal sized data. Each equal sized data has been performed merge sort and sorted output is done into temp file.

K-Way merge files performed using Tree Map Data structure in Java.

FileReader in Java reads the file line by line and hence increase the reading of data.

Trail version: 1GB chunks has been created as it is less than RAM capacity, the output was not good as expected.

1. Data Read (GB) [1xi3.large 128GB]

Data has been read by reading the file and pushing it into 2GB chunks i.e. 128 GB. Again 128GB data again read onto chunks divided by thread size.

At the time of k-way merge, file read and pushed onto output file. So totally 128\*3= 384 GB.

1. Data Write (GB) [1xi3.large 128GB:

Data has been read by writing the file and pushing it into 2GB chunks i.e. 128 GB. Again 128GB data again write onto chunks divided by thread size.

At the time of k-way merge, file read and pushed onto output file. So totally 128\*3= 384 GB.

1. **I**/O Throughput (MB/sec) [1xi3.large 128GB]

Throughput = (384+384)/5438.684 = 141 MB/sec

**Comments on Shared memory on 1 TB performance analysis:**

1. Compute Time for **1xi3.4xlarge 1TB:**

I have used 16 threads to divide the file and perform in-place merge sort using 2Gb chunks of data and used same approach as befor. Each 2Gb chunk data, divide into 2 equal sized data. Each equal sized data has been performed merge sort and sorted output is done into temp file.

K-Way merge files performed using Tree Map Data structure in Java.

FileReader in Java reads the file line by line and hence increase the reading of data.

Trail version: 1GB chunks has been created as it is less than RAM capacity, the output was not good as expected.

1. Data Read (GB) [1xi3.4xlarge 1TB]

Data has been read by reading the file and pushing it into 2GB chunks i.e. 1TB Again 1000GB data again read onto chunks divided by thread size.

At the time of k-way merge, file read and pushed onto output file. So totally 1000\*3= 3000 GB.

1. Data Write (GB) [1xi3.4xlarge 1TB:

Data has been read by writing the file and pushing it into 2GB chunks i.e. 1TB GB. Again 1TB data again write onto chunks divided by thread size.

At the time of k-way merge, file read and pushed onto output file. So totally 1000\*3= 3 TB.

1. **I**/O Throughput (MB/sec) [1xi3.large 128GB]

Throughput = (1000+1000)/28922 = 207.4 MB/sec

Scale up:

Scaled up by 1.47 times.

Performance: Performance increased by 32%

**Answers:**

1. Which seems to be best at 1 node scale?

According to our data, external sort seems to perform best among all four.

1. How about 8 nodes?

Spark would perform best among all four types.

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

Spark will be the best option.

1. How about 1000 node scales ?

Spark as spark uses in memory processing.

1. what can you learn from the CloudSort benchmark ?

Looking at the cost, we can calculate the cheapest way we can sort large datasets and uses limited resources. Best platform has been calculated.