

# Design Document

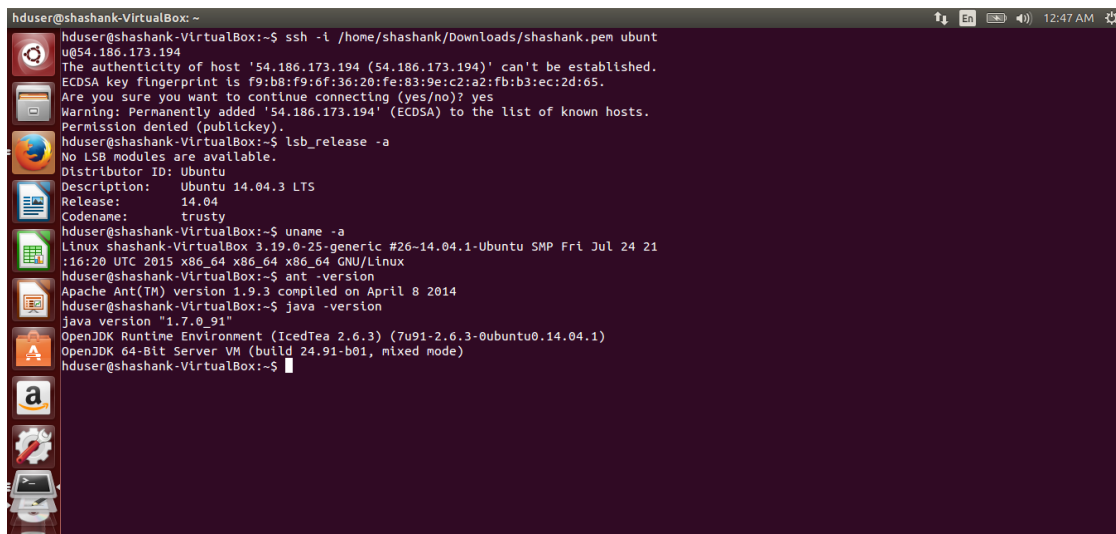
**Problem:** To sort a large dataset of 10GB using Shared Memory in JAVA, HADOOP and SPARK and 100GB using HADOOP and SPARK.

**Methodology:** The basic solution for sorting the large dataset efficiently by dividing the large datasets into smaller chunks and then combining them back in the sorted order.

**Versions:**

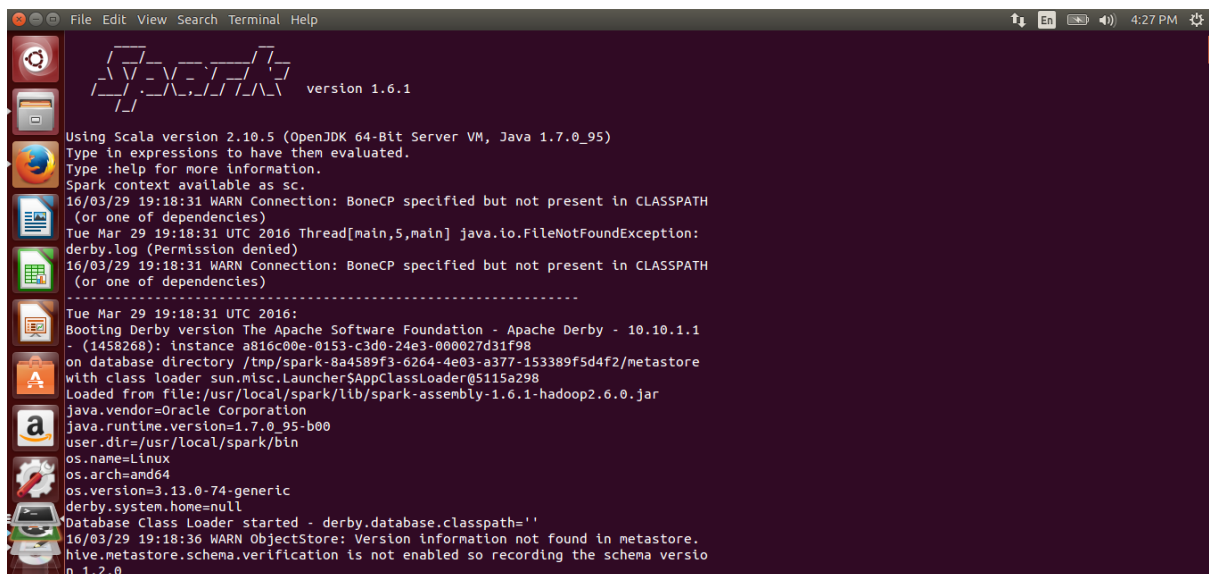
The following screenshots provides the Ubuntu version along with the ant version and the java version, Hadoop and spark version to run that had been used for running the sorting programs

**Hadoop:** Hadoop-2.7.2

A screenshot of a terminal window titled 'hduser@shashank-VirtualBox: ~'. The terminal shows the following commands and outputs:

```
hduser@shashank-VirtualBox:~$ ssh -l /home/shashank/Downloads/shashank.pem ubuntu@54.186.173.194
The authenticity of host '54.186.173.194 (54.186.173.194)' can't be established.
ECDSA key fingerprint is f9:b8:f9:6f:36:20:fe:83:9e:c2:a2:fb:b3:ec:2d:65.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.186.173.194' (ECDSA) to the list of known hosts.
Permission denied (publickey).
hduser@shashank-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.3 LTS
Release:        14.04
Codename:       trusty
hduser@shashank-VirtualBox:~$ uname -a
Linux shashank-VirtualBox 3.19.0-25-generic #26~14.04.1-Ubuntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
hduser@shashank-VirtualBox:~$ ant -version
Apache Ant(TM) version 1.9.3 compiled on April 8 2014
hduser@shashank-VirtualBox:~$ java -version
java version "1.7.0_91"
OpenJDK Runtime Environment (IcedTea 2.6.3) (7u91-2.6.3-0ubuntu0.14.04.1)
OpenJDK 64-Bit Server VM (build 24.91-b01, mixed mode)
hduser@shashank-VirtualBox:~$
```

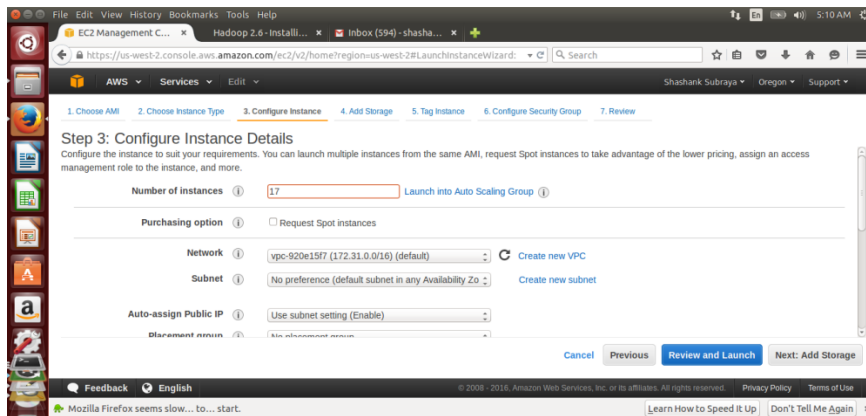
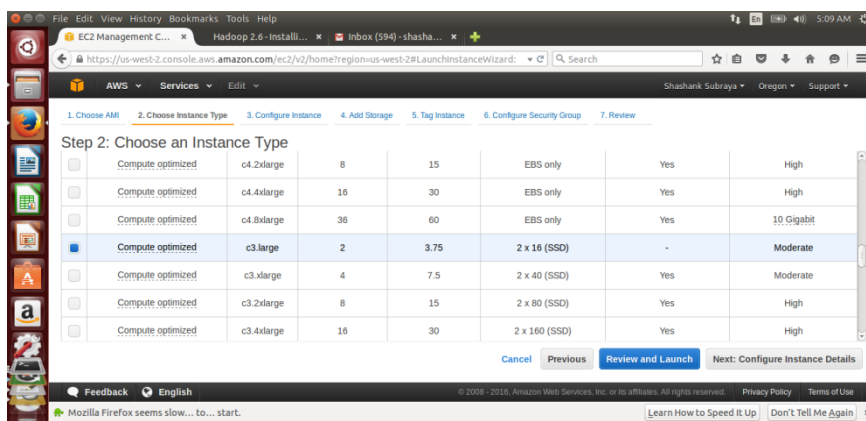
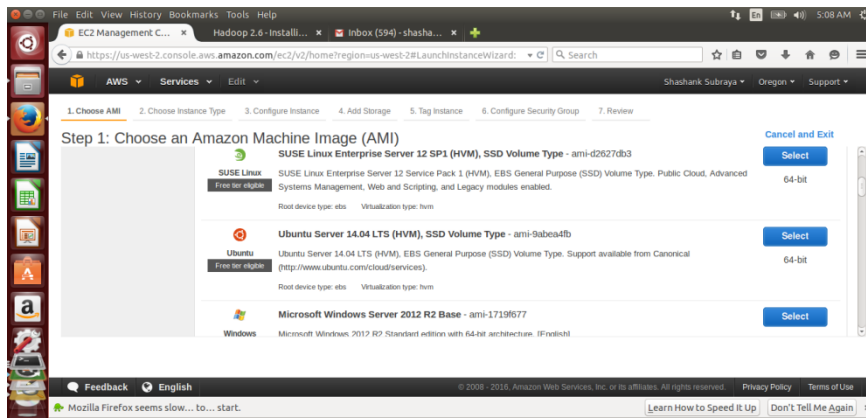
The terminal window has a dark purple background and a light blue title bar. On the left side, there is a vertical dock with several application icons including a gear, a folder, a terminal, a file manager, a web browser, and a printer.

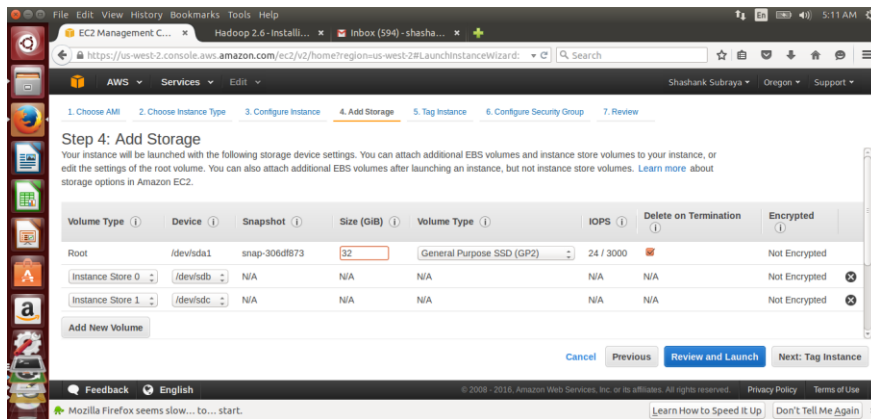


## Setting up Virtual Clusters:

A virtual cluster of 17 nodes is set up in amazon web services.  
[aws.amazon.com](http://aws.amazon.com)

Under ec2 services, we created instance using Launch Instances tab. Next Ubuntu Server LTS AMI [Amazon Machine Image] was selected.





Hadoop:

In the Hadoop framework. The configurations made are as follows:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install default-jdk
$ sudo apt-get install ssh
$ sudo addgroup hadoop
$ sudo adduser --ingroup hadoop hduser
$ su hduser
$ sudo adduser hduser sudo
$ sudo su hduser
$ wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.7.2/hadoop-2.7.2.tar.gz
$ tar -xvzf hadoop-2.7.2.tar.gz
$ ssh-keygen -t rsa -P ""
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
$ ssh localhost
$ sudo mkdir /usr/local/hadoop
$ sudo mv * /usr/local/hadoop/
$ sudo chown -R hduser:hadoop /usr/local/hadoop
$ update-alternatives --config java
$ vim ~/.bashrc

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_INSTALL=/usr/local/hadoop
```

```
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"
#HADOOP VARIABLES END
```

```
$ source ~/.bashrc
```

```
$ vim /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

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

```
$ sudo mkdir -p /app/hadoop/tmp && sudo chown hduser:hadoop /app/hadoop/tmp
```

```
$ vim /usr/local/hadoop/etc/hadoop/core-site.xml
```

```
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>

  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:54310</value>
    <description>The name of the default file system. A URI whose
      scheme and authority determine the FileSystem implementation. The
      uri's scheme determines the config property (fs.SCHEME.impl) naming
      the FileSystem implementation class. The uri's authority is used
      to
      determine the host, port, etc. for a filesystem.</description>
  </property>
</configuration>
```

```
$ cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
```

```
$ vim /usr/local/hadoop/etc/hadoop/mapred-site.xml
```

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:54311</value>
    <description>The host and port that the MapReduce job tracker runs
      at. If "local", then jobs are run in-process as a single map
      and reduce task.
    </description>
  </property>
</configuration>
```

```
$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode && sudo mkdir -p
/usr/local/hadoop_store/hdfs/datanode && sudo chown -R hduser:hadoop
```

```
/usr/local/hadoop_store
```

```
$ vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default block replication.
      The actual number of replications can be specified when the file is
      created.
      The default is used if replication is not specified in create time.
    </description>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
  </property>
</configuration>
```

**hadoop namenode -format**

For 16 nodes enter the public dns of amazon of the master instead of localhost in core-site.xml, mapred-site.xml and yarn-site.xml for all 16 nodes and enter the public dns of the slave node in the slave folder of that slave node. The slave folder in the master node must have the public dns of the master and the all the 16 slaves

conf/core-site.xml: The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.

conf/mapred-site.xml: The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.

conf/hdfs-site.xml: Provides the block replication to be provided.

conf/slaves: Provides the slave public DNS . In master node has the public DNS of master and all its slaves.

conf/Master: Provides the Public DNS of the master.

- 1) Master: Namenode, JobTracker of the node which will run hadoop sort operation using the other slave nodes

Slave: {DataNode, TaskTraker}, ..... {DataNode, TaskTraker} are the other nodes are where Hadoop data is stored and where data processing takes place.

- 2) We need to set the unique available ports to those configurations on shared environment as each nodes would be performing its specified tasks. Giving unique ports would reduce traffic on single. By not doing so an error could occur like a Connection Bind Exception
- 3) We can set the number of Map and Reduce tasks to be run by using the JobConf's `conf.setNumMapTasks(int num)` and `conf.setNumReduceTasks(int num)` for map and reduce tasks respectively.

#### Code Flow:

Here the main class `HadoopTeraSort.java` calls the Mapping class which extends the map function. The map functions consists of the input data coming from the dataset .Each row of this dataset is divided to key value pairs into the map function. Implicit sort is performed here based on the key and the result is then sent to the reducer. The reducer is then takes the data from the mapper and places in the output file. Note, the i/p, o/p files are obtained from the HDFS which are of 10GB and 100GB for 1 node and 16 nodes respectively.

#### Spark:

For the spark framework Spark 1.6.1 version is installed and the configurations are made.

Download and extract the latest version and extract it. Then `~/ .bashrc` file is given a path from where the spark shell can be run.

```
vim ~/.bashrc
```

```
export PATH=$PATH:/usr/local/spark/bin
```

```
source ~/.bashrc
```

For 16 nodes we generate access key\_id and the secret\_access\_key from amazon aws. Then download the root.csv from it and place it from where are going to generate 16 nodes. Also copy the key file. Install aws configuration and then enter the access\_key\_id and security\_access\_key in aws configure and also export it as such in `~/ .bashrc`. Then generate the 16 nodes using the following command:

```
./spark-ec2 --key-pair=YOUR_KEY_NAME --identity-file=YOUR_KEY.pem --region=us-west-2 -  
-slaves=16 --instance-type=c3.large --ebs-vol-size=500 --spot-price=0.03 launch spark
```

One master and 16 slave node instances are created in amazon

#### Code Flow:

In Spark the input file is taken from HDFS which has a 10GB and a 100GB dataset for 1 node and 16 nodes respectively. The input file is fetched from the HDFS. Then using scala this

fetches input file is used in map which splits the datasets into key value pairs. Then we perform the sortbykey operation which sorts the data based on the key. The key here is nothing but the first 10 characters in each row of the dataset. The sorted output is then placed in other file HDFS. Note, the while writing in the o/p file. The data is split into number of files in the o/p directory in the sorted order

#### Shared Memory using JAVA

In this the input dataset is generated through gensort. The input data is then divided into chunks of data. These chunks of data are sorted and are then merged together to an output file. The output data can be validated using valsort.

Comparison between Hadoop, Spark, Shared Memory Java:

Shared Memory JAVA	Hadoop	Spark
The input data set is divided into smaller chunks and is processed using threads	File handling and Processing are done by two functions Mapper and Reducer	We use RDD to handle the data set
The data is sorted and then merged into single large o/p file	The mapper transfers the key-value input to the intermediate key-value	The map functions splits the data into key-value pairs
This process is slower compared to hadoop and spark for larger nodes	The Reducer shuffles the data	The data is sorted using sortByKey function
	For Hadoop running in 16 nodes the sorting time taken would be less than that by 1 node	For Spark running in 16 nodes the sorting time taken would be less than that by 1 node

Conclusion: For higher nodes Spark framework is the best option for sorting as it executes batch processing jobs at a much faster pace than Hadoop MapReduce framework just by merely cutting down on the number of reads and writes to the disc.

Therefore for 16, 100 and 1000 nodes can be spark is definitely a better option than Hadoop Mapreduce. However for 1 node Hadoop is a better option.

The sort rate of 2014 Hadoop MR and Spark is 1.42 TB/min and 4.27TB/min 102.5TB and 102.5 TB and 100 TB respectively with a sort rate/node of 0.67 GB/min and 20.7 GB/min respectively as compared to

CloudSort benchmark measures the total cost of ownership for external sorts performed in a cloud environment



#### Problems Faced:

1. While sorting the large dataset , got an error :  
Java.Memory.Exception: heap out of space;  
Sol: can increase the memory by `-Xmx2048g`
2. Hadoop Configuration for 16 nodes was too complex
3. Cannot calculate the sort time for Hadoop. Can only calculate the total execution time.