# Performance Report

The following are the aspects in this assignment:

- Creating a Virtual Cluster using AMAZON WEB SERVICES(AWS)
- Implementing Shared-Memory Sort in C
- Implementing Sort in HADOOP
- Implementing Sort in SPARK
- Implementing Sort using MPI
- Implementing Sorting across 4 approaches (C, HADOOP, SPARK, MPI)
- Measuring the performance of all the above

### *Creating a Virtual Cluster using AMAZON WEB SERVICES (AWS):*

First, we login into the Amazon AWS service website (http://aws.amazon.com/) and we select EC2 from the network and services section. We select launch instance here. Then we configure our service to meet our requirements. We choose the AMI we need. Then we choose the instance type which can be micro, small, medium, large Xlarge, 2Xlarge, 4Xlarge or 8Xlarge. Then we configure the instance i.e. we can choose the instance to be either spot instance or on Demand instance. We then add storage to our selected instance and configure the security group. We add TCP, UDP and ICMP protocols from the security group. We then create the instance. We access the instance by connecting to the instance by using the public IP address of it and later we can configure the settings of the AMI that we are using.

### *Implementing Shared Memory Sort:*

We implement a C program to count the occurrence of each word in each input file of size 128GB and 1TB. We print the output into a file in the increasing order of the frequency of the word.

### *Implementing Sort in HADOOP:*

After downloading and installing HADOOP and extracting its components and adding the library paths to the bash we connect to the AWS instance using ssh –i key (IP address). The sort application is implemented using HADOOP for 128GB and 1TB on a single node and 1TB dataset on 8 nodes.

### *Implementing Sort in SPARK:*

The same Sort application is implemented using SPARK programming language for 128GB and 1TB on a single node and 1TB dataset on 8 nodes.

## Performance Calculation:

From the given data sets, all the four versions of TeraSort (Shared-Memory, Hadoop, Spark, and MPI), their performances are obtained on 1 node scale and 8 node scales with both the 128GB and 1TB dataset. These are compared, and performance graphs are drawn for them. The performance result of shared-memory sort is compared with the performance of Spark and Hadoop sorts which are implemented on 1 node as well as on 8 nodes.

## Sorting:

Implementation and evaluation the performance of sorting for all the 4 versions of sorts i.e. for Shared-Memory, HADOOP, SPARK, MPI on a 128GB and 1TB datasets are done. This sorting performance includes reading, sorting and writing data to disks.

## Creating an instance in Amazon(AWS):

- Log in into the Amazon AWS service website (http://aws.amazon.com/).
- Select EC2 from the network and services section.
- Select launch instance. Then configure the service and chose the best AMI to meet the requirements.
- Then choose the instance type which is i3.large and i3.4xlarge.
- Then we configure the instance i.e. we can choose the instance to be either spot instance or on Demand instance.
- Add storage to our selected instance.
- Configure the security group. We add TCP, UDP and ICMP protocols from the security group. We then create the instance. We access the instance by connecting to the instance by using the public IP address of it and later we can configure the settings of the AMI that we are using.

After getting connected to the AMI we now must install java on the virtual node.

- Update all the installed applications by using sudo apt-get update
- Now update all the installed applications by using sudo apt-get upgrade
- Install java by using sudo apt-get install default-jdk.
- Now check the java version by using java –version.

128GB size operations were carried out at pic 1 ip.



1Tb size operation were carried out at pic 2 ip.

**System configuration used on Amazon:**

**AMI: Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-3d50120d**

**Instance Type: i3.large, i3.4xlarge**

**Security Group: Created and used security group with SSH enabled for both inbound and outbound**

<u>LOGGING INTO YOUR INSTANCE</u>:

```
INVINCIBLES_SK@INVINCIBLE MINGW64 ~/Downloads
$ ssh -i amazon_key.pem ubuntu@ec2-18-221-204-249.us-east-2.compute.amazonaws.com
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-1041-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

1 package can be updated.
0 updates are security updates.


Last login: Tue Dec  5 00:22:39 2017 from 106.207.6.228
ubuntu@ip-172-31-17-91:~$
```

## *Shared-Memory Sort:*

Shared Memory sorting is known as external sorting, and it is required when the dataset which we need to sort do not fit into the main memory (RAM) and therefore they must reside in some slower external memory (say a hard drive). The Shared memory tera sort program was run on Amazon i3.large instance and i3.4xlarge instance by taking any random thread for both 128GB and 1TB datasets generated using gensort. After sorting, these datasets were validated using valsort.

### *Methodology:*

We take input of total size 128 GB and 1TB and then sort in the file.

Based on number of lines, we further split the data into equal number of chunks.

- Shared Memory sort is an application that will sort file-resident dataset and particularly those datasets that are larger than the allocated cluster memory.
- The application is implemented in C with multithreading functionality. The 128GB and 1TB dataset is divided into chunks.
- For measuring performance, the dataset is stored in i3.large and i3.4xlarge instance.

The below screenshot shows the i3.large instance created on which the shared memory sort application can be executed.
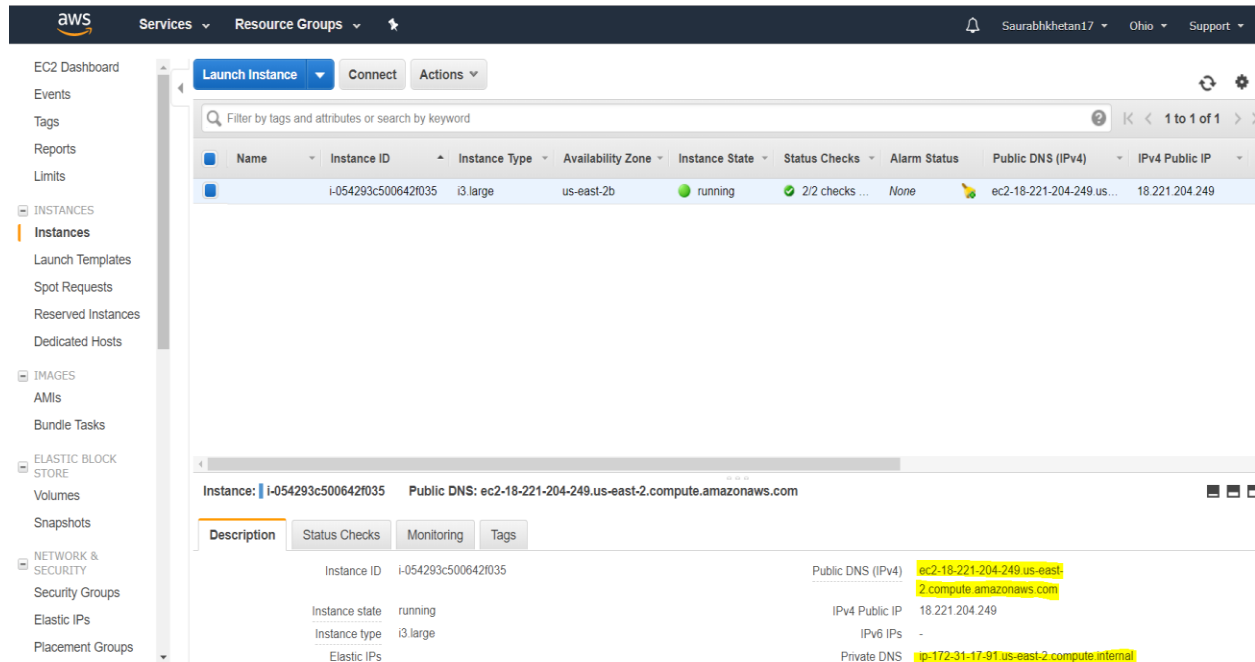
Once we launch the instance, one active node will be actively running on the Amazon AWS Dashboard.

To connect this above instance followed the following steps.

We use chmod 400 My_Key.pem to make sure your private key file isn't publicly viewable.

- Use the ssh command to connect to the instance.
- After connection established we need to install Java on our virtual cluster: sudo apt-get update
- Install Default Java: sudo apt-get install openjdk-7-jdk

To mount the EBS volume attached to the system, we follow the mentioned below steps:

**Sudo yum install mdadm**

We use the mdadm command to create a logical RAID device from the newly attached Amazon EBS volumes.

We can list the devices on our instance using lsblk command.

To create a RAID 0 array, execute the following command: **sudo mdadm --create --verbose /dev/md0 --level=0 --name=RAID_NAME --raid-devices=No_Of_Devices Device1 Device2**

Create a file system on your RAID array, and give that file system a label to use when you mount it later: **sudo mkfs.ext4 -L RAID_NAME /dev/md0**

Create a mount point for your RAID array**: sudo mkdir -p /Saurabh**

Finally, mount the RAID device on the mount point that you created: **sudo mount LABEL=MAPREDUCE_RAID /SAURABH**

```
MINGW64:/c/Users/saura/Downloads
root@ip-172-31-17-91:/Saurabh# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7.5G     0  7.5G   0% /dev
tmpfs           1.5G  9.0M  1.5G   1% /run
/dev/xvda1      7.7G  1.8G  6.0G  23% /
tmpfs           7.5G     0  7.5G   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           7.5G     0  7.5G   0% /sys/fs/cgroup
tmpfs           1.5G     0  1.5G   0% /run/user/1000
/dev/md0        829G  245G  543G  32% /Saurabh
tmpfs           1.5G     0  1.5G   0% /run/user/0
root@ip-172-31-17-91:/Saurabh#
```

### *Outputs:*

### *Shared-Memory 128GB sorting using 1-node i3.large instance:*

We need to create an input data of 128GB for which we use Gensort by using the following command:
      **./gensort –a 1280000000 input.txt**

The output of the above command will be a dataset of 128GB which will be stored in the directory where gensort file is located.

The application can be executed using the following command,

**gcc shared_mem.c -o sharedmemory**

**./sharedmemory**



```
MINGW64:/c/Users/saura/Downloads
root@ip-172-31-17-91:/Saurabh# gcc shared_mem.c -o sharedmemory
root@ip-172-31-17-91:/Saurabh# ./sharedmemory
root@ip-172-31-17-91:/Saurabh# Start Multithreading for external sort

 128GB input file external sorting

Reading Input file

Split and sorting done

Starting merge phase

Sorting Records done after dividing into chunks

 Total calculated time for External Sorting: 12401 sec
root@ip-172-31-17-91:/Saurabh#
```

The output of the application will be stored in the /Saurabh/  path as given in the execution command.

The output ''**outputfile**' can be evaluated using valsort using the below command,

**. /valsort outpufile**



```
MINGW64:/c/Users/saura/Downloads
root@ip-172-31-17-91:/Saurabh/64# cat abc.txt
root@ip-172-31-17-91:/Saurabh/64# ./valsort outputFile
Records: 1280000000
Checksum: 1484eo3qwde7i863
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-17-91:/Saurabh/64#
root@ip-172-31-17-91:/Saurabh/64#
```

*Shared-Memory 1TB sorting using 1-node i3.4xlarge instance:*



```
MINGW64:/c/Users/saura/Downloads
root@ip-172-31-41-228:/Shiksha# gcc shared_mem.c -o sharedmemory
root@ip-172-31-41-228:/Shiksha# ./sharedmemory
root@ip-172-31-41-228:/Shiksha# Start Multithreading for external sort

 1 TB input file external sorting

Reading Input file

Split and sorting done

Starting merge phase

Sorting Records done after dividing into chunks

 Total calculated time for External Sorting: 47964 sec
root@ip-172-31-41-228:/Shiksha#
```

valsort for 1 TB file:

```
root@ip-172-31-41-228:/Shiksha/64# ./valsort outputFile
Records: 10000000000
Checksum : 5621em3qwde7k963
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-41-228:/Shiksha/64#
root@ip-172-31-17-91:/Saurabh/64#
```

## *HADOOP*

### *Hadoop Installation:*

After logging into i3.large instance, firstly I have run following commands to make SSH Passwordless:

ssh-keygen -f ~/.ssh/id_rsa -t rsa -P ""

cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

Then I have implemented RAID0 to combine two raid devices using following commands:

sudo apt-get install mdadm

sudo umount -l /dev/xvdb

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

sudo mkfs -t ext4  /dev/md0

sudo mkdir -p /Saurabh

sudo mount LABEL=Saurabh /Saurabh

After RAID0, I have updated the necessary packages on i3.large instance using following command:

$ sudo apt-get update

Next, Latest version of java has been installed using following command:

$ sudo apt-get install openjdk-7-jdk

After that, Hadoop (2.7.4) has been downloaded and extracted on the instance and the permissions are given to Hadoop for read, write and execute using following commands:

wget http://apache.claz.org/hadoop/common/hadoop-2.7.4/hadoop-2.7.4.tar.gz

sudo tar -zxvf hadoop-2.7.4.tar.gz

sudo mv hadoop-2.7.4 hadoop

### *Hadoop Version Used:*

We have used Hadoop-2.7.4 for sorting experiments. And  have downloaded it from following website:

**http://apache.mirrors.tds.net/hadoop/common/hadoop-2.7.4/hadoop-2.7.4.tar.gz**

### *Setting up Environment Variables:*

In the environment variables, We have added some Hadoop and Java environment variables to .bashrc and source them to the current shell session. These are shown below:

sudo vi .bashrc


Have added following export statements in .profile:

export JAVA_HOME=/usr

export PATH=$PATH:$JAVA_HOME/bin

export HADOOP_HOME=/usr/local/hadoop

export PATH=$PATH:$HADOOP_HOME/bin

export HADOOP_CONF_DIR=/usr/local/hadoop/etc/Hadoop

Then I have loaded these environment variables by sourcing the profile using following command:

**source .bashrc**

### *Hadoop Configuration:*

Following are the configuration changes which are common to all nodes i.e., namenode and datanodes:

$HADOOP_CONF_DIR/hadoop-env.sh

$HADOOP_CONF_DIR/core-site.xml

$HADOOP_CONF_DIR/yarn-site.xml

$HADOOP_CONF_DIR/mapred-site.xml

1. hadoop-env.sh



```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.
#export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol.  Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}

# Extra Java CLASSPATH elements.  Automatically insert capacity-scheduler.
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
  if [ "$HADOOP_CLASSPATH" ]; then
    export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
  else
    export HADOOP_CLASSPATH=$f
  fi
done

# The maximum amount of heap to use, in MB. Default is 1000.
#export HADOOP_HEAPSIZE=
#export HADOOP_NAMENODE_INIT_HEAPSIZE=""

# Extra Java runtime options.  Empty by default.
export HADOOP_OPTS="$HADOOP_OPTS -Djava.net.preferIPv4Stack=true"

"hadoop-env.sh" 99L, 4276C
```

```bash
# Extra Java runtime options.  Empty by default.
export HADOOP_OPTS="$HADOOP_OPTS -Djava.net.preferIPv4Stack=true"


# Command specific options appended to HADOOP_OPTS when specified
export HADOOP_NAMENODE_OPTS="-Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS}  -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender} $HADOOP_NAMENODE_OPTS"
export HADOOP_DATANODE_OPTS="-Dhadoop.security.logger=ERROR,RFAS $HADOOP_DATANODE_OPTS"

export HADOOP_SECONDARYNAMENODE_OPTS="-Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS}  -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender} $HADOOP_SECONDARYNAMENODE_OPTS"

export HADOOP_NFS3_OPTS="$HADOOP_NFS3_OPTS"
export HADOOP_PORTMAP_OPTS="-Xmx512m $HADOOP_PORTMAP_OPTS"


# The following applies to multiple commands (fs, dfs, fsck, distcp etc)
export HADOOP_CLIENT_OPTS="-Xmx512m $HADOOP_CLIENT_OPTS"
#HADOOP_JAVA_PLATFORM_OPTS="-XX:-UsePerfData $HADOOP_JAVA_PLATFORM_OPTS"


# On secure datanodes, user to run the datanode as after dropping privileges.
# This **MUST** be uncommented to enable secure HDFS if using privileged ports
# to provide authentication of data transfer protocol.  This **MUST NOT** be
# defined if SASL is configured for authentication of data transfer protocol
# using non-privileged ports.
export HADOOP_SECURE_DN_USER=${HADOOP_SECURE_DN_USER}


# Where log files are stored.  $HADOOP_HOME/logs by default.
#export HADOOP_LOG_DIR=${HADOOP_LOG_DIR}/$USER


# Where log files are stored in the secure data environment.
export HADOOP_SECURE_DN_LOG_DIR=${HADOOP_LOG_DIR}/${HADOOP_HDFS_USER}


###
# HDFS Mover specific parameters
###
# Specify the JVM options to be used when starting the HDFS Mover.
# These options will be appended to the options specified as HADOOP_OPTS
# and therefore may override any similar flags set in HADOOP_OPTS
#
# export HADOOP_MOVER_OPTS=""


###
# Advanced Users Only!
###


# The directory where pid files are stored. /tmp by default.
# NOTE: this should be set to a directory that can only be written to by
#       the user that will run the hadoop daemons.  Otherwise there is the
#       potential for a symlink attack.
export HADOOP_PID_DIR=${HADOOP_PID_DIR}
export HADOOP_SECURE_DN_PID_DIR=${HADOOP_PID_DIR}


# A string representing this instance of hadoop. $USER by default.
export HADOOP_IDENT_STRING=$USER
```

## 2. Core-site.xml:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/Saurabh/hadoop/hd-tmp/</value>
    </property>
</configuration>
~
~
```

## 3. mapred-site.xml

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>localhost:9001</value>
    </property>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapred.tasktracker.map.tasks.maximum</name>
        <value>4</value>
        <final>true</final>
    </property>
    <property>
        <name>mapred.tasktracker.reduce.tasks.maximum</name>
        <value>4</value>
        <final>true</final>
    </property>

</configuration>
~
```

## 4. .bashrc

```bash
# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_INSTALL=/Saurabh/hadoop
export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
export PATH=$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
#HADOOP VARIABLES END

# Insert these lines into your ~/.bash_profile:
export SPARK_HOME=/Saurabh/spark
PATH=$PATH:$SPARK_HOME/bin
export PATH
# Then exit the text editor and return to the command line
```

## 5. yarn-site.xml

```xml
<?xml version="1.0"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<configuration>

<!-- Site specific YARN configuration properties -->

    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value> ec2-18-221-204-249.us-east-2.compute.amazonaws.com </value>
    </property>
</configuration>
```

6. hdfs-site.xml

```
MINGW64:/c/Users/saura/Downloads
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
            <name>dfs.namenode.name.dir</name>
            <value>file://${hadoop.tmp.dir}/dfs/name</value>
    </property>
    <property>
            <name>dfs.datanode.data.dir</name>
            <value>file://${hadoop.tmp.dir}/dfs/data</value>
    </property>
</configuration>
```

*Two properties which were added, i.e., dfs.permissions, if it is set to true then only HDFS user can modify the data. Therefore, I have kept it "false". Secondly dfs.replicaton is used to specify how many copies of data can be made.*

*The current path where data on the Name Node will reside does not exist, so I have created this before starting HDFS. It is done using following command:*

**sudo mkdir -p /Saurabh/hadoop/hd-tmp/namenode**

**sudo chown -R ubuntu $HADOOP_HOME**

After this, Namenode needs to be formatted. I have used following command for that:

*$ hdfs namenode –format*

**_Checking the Hadoop installation by starting dfs and yarn:_**
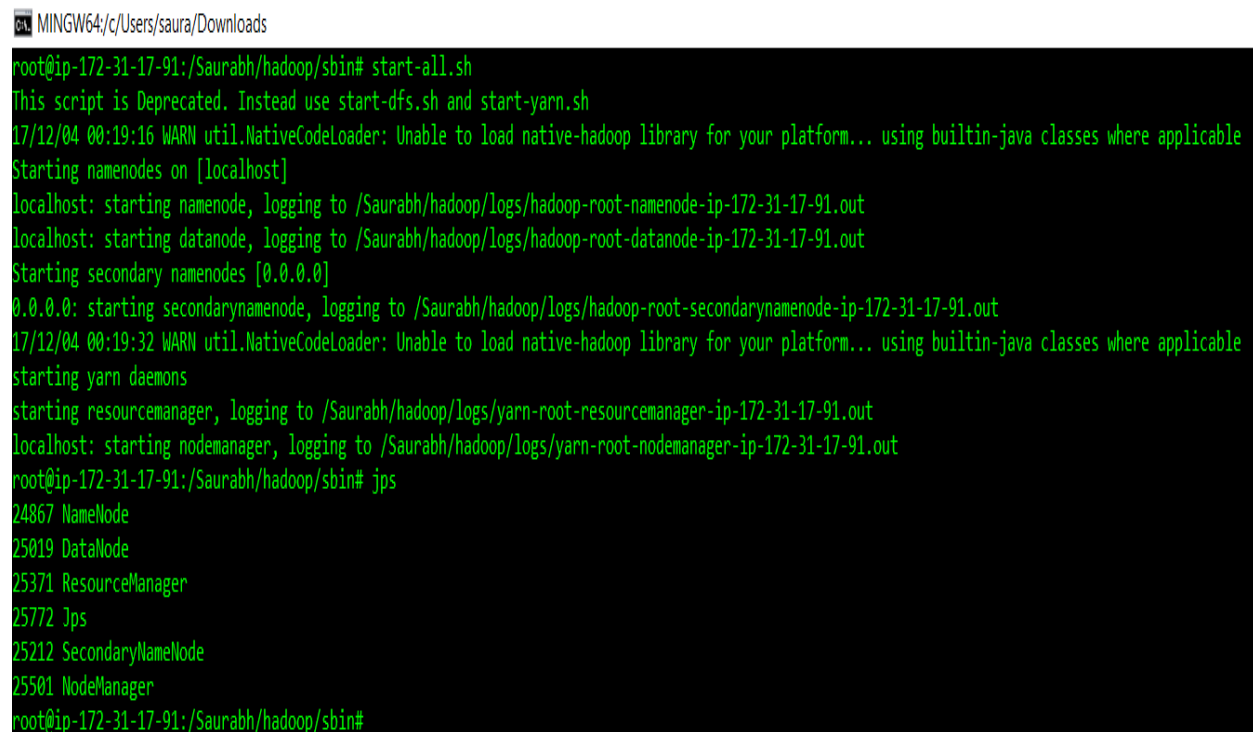
$HADOOP_HOME/sbin/start-dfs.sh

$HADOOP_HOME/sbin/start-yarn.sh

After starting both, we can check by running jps command:

$ jps

The following screenshot shows the successful start of Hadoop-2.7.4.



For gensort, commands ran were:

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

tar -zvxf gensort-linux-1.5.tar.gz

Generate 128 GB file on remote machine and put it into hdfs using the following commands:

./gensort -a 1280000000 inputdata

hdfs dfs -put inputdata /input/

The program was ran using:

hadoop jar TeraSorting.jar TeraSorting /input/inputdata /Output/output

## Outputs:

### Hadoop 128 GB sorting using 1-node i3.large instance:

Starting the sorting of 128GB file on Hadoop:



```
MINGW64:/c/Users/saura/Downloads
root@ip-172-31-17-91:/Saurabh/hadoop/share/hadoop# hadoop jar TeraSorting.jar TeraSorting /input/inputdata /output/output
17/12/04 00:59:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/12/04 00:59:46 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/12/04 00:59:46 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/12/04 00:59:46 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
17/12/04 00:59:46 INFO input.FileInputFormat: Total input files to process : 1
17/12/04 00:59:47 INFO mapreduce.JobSubmitter: number of splits:954
17/12/04 00:59:47 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1117338954_0001
17/12/04 00:59:47 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
17/12/04 00:59:47 INFO mapreduce.Job: Running job: job_local1117338954_0001
17/12/04 00:59:47 INFO mapred.LocalJobRunner: OutputCommitter set in config null
17/12/04 00:59:47 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/04 00:59:47 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
17/12/04 00:59:47 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
17/12/04 00:59:48 INFO mapreduce.Job: Job job_local1117338954_0001 running in uber mode : false
17/12/04 00:59:48 INFO mapreduce.Job:  map 0% reduce 0%
17/12/04 00:59:48 INFO mapred.LocalJobRunner: Waiting for map tasks
17/12/04 00:59:48 INFO mapred.LocalJobRunner: Starting task: attempt_local1117338954_0001_m_000000_0
17/12/04 00:59:48 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/04 00:59:48 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
17/12/04 00:59:48 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
17/12/04 00:59:48 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/input/in:0+134217728
17/12/04 00:59:48 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
17/12/04 00:59:48 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
17/12/04 00:59:48 INFO mapred.MapTask: soft limit at 83886080
17/12/04 00:59:48 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
17/12/04 00:59:48 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
17/12/04 00:59:48 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
17/12/04 00:59:51 INFO mapred.MapTask: Spilling map output
17/12/04 00:59:51 INFO mapred.MapTask: bufstart = 0; bufend = 72112614; bufvoid = 104857600
17/12/04 00:59:51 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 23271028(93084112); length = 2943369/6553600
17/12/04 00:59:51 INFO mapred.MapTask: (EQUATOR) 75055974 kvi 18763988(75055952)
17/12/04 00:59:54 INFO mapred.MapTask: Finished spill 0
17/12/04 00:59:54 INFO mapred.MapTask: (RESET) equator 75055974 kv 18763988(75055952) kvi 18028160(72112640)
17/12/04 00:59:54 INFO mapred.LocalJobRunner:
17/12/04 00:59:54 INFO mapred.MapTask: Starting flush of map output
17/12/04 00:59:54 INFO mapred.MapTask: Spilling map output
17/12/04 00:59:54 INFO mapred.MapTask: bufstart = 75055974; bufend = 29619204; bufvoid = 104857600
17/12/04 00:59:54 INFO mapred.MapTask: kvstart = 18763988(75055952); kvend = 16338652(65354608); length = 2425337/6553600
17/12/04 00:59:56 INFO mapred.MapTask: Finished spill 1
17/12/04 00:59:56 INFO mapred.Merger: Merging 2 sorted segments
17/12/04 00:59:56 INFO mapred.Merger: Down to the last merge-pass, with 2 segments left of total size: 135559964 bytes
17/12/04 00:59:57 INFO mapred.Task: Task:attempt_local1117338954_0001_m_000000_0 is done. And is in the process of committing
17/12/04 00:59:57 INFO mapred.LocalJobRunner: map > sort
17/12/04 00:59:57 INFO mapred.Task: Task 'attempt_local1117338954_0001_m_000000_0' done.
17/12/04 00:59:57 INFO mapred.LocalJobRunner: Finishing task: attempt_local1117338954_0001_m_000000_0
17/12/04 00:59:57 INFO mapred.LocalJobRunner: Starting task: attempt_local1117338954_0001_m_000001_0
17/12/04 00:59:57 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/04 00:59:57 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
17/12/04 00:59:57 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
17/12/04 00:59:57 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/input/in:134217728+134217728
17/12/04 00:59:57 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
```

Result/Output for 128 GB on 1 node on HADOOP:

```
17/12/04 08:00:13 INFO mapreduce.Job:  map 100% reduce 100%
17/12/04 08:00:13 INFO mapreduce.Job: Job job_local611976903_0001 completed successfully
17/12/04 08:00:13 INFO mapreduce.Job: Counters: 35
        File System Counters
                FILE: Number of bytes read=63027972233954
                FILE: Number of bytes written=125389835904492
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=61270698782720
                HDFS: Number of bytes written=128000000000
                HDFS: Number of read operations=915846
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=957
        Map-Reduce Framework
                Map input records=1280000000
                Map output records=1280000000
                Map output bytes=128000000000
                Map output materialized bytes=13056005724
                Input split bytes=97308
                Combine input records=1280000000
                Combine output records=1280000000
                Reduce input groups=1280000000
                Reduce shuffle bytes=130560005724
                Reduce input records=1280000000
                Reduce output records=1280000000
                Spilled Records=6490800117
                Shuffled Maps =954
                Failed Shuffles=0
                Merged Map outputs=954
                GC time elapsed (ms)=143063
                Total committed heap usage (bytes)=512040108032
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1280000286782
        File Output Format Counters
                Bytes Written=128000000000
root@ip-172-31-17-91:/Saurabh/hadoop/share/hadoop# ls -lrt
```

Time elapsed from test.txt:

```
root@ip-172-31-17-91:/Saurabh/hadoop/share/hadoop# vi test.txt
root@ip-172-31-17-91:/Saurabh/hadoop/share/hadoop# cat test.txt
Time elapsed in sec:15260.23
root@ip-172-31-17-91:/Saurabh/hadoop/share/hadoop#
```

Valsort for 128GB on 1 node:



```
MINGW64:/c/Users/saura/Downloads
root@ip-172-31-17-91:/Saurabh/64# ./valsort part-r-00000
Records: 1280000000
Checksum:5631qw8wcipc294
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-17-91:/Saurabh/64#
```

| Dataset(GB) | Number of Nodes | Execution time(seconds) |
|-------------|-----------------|-------------------------|
| 128         | 1               | 15260.23                |

## Hadoop 1 TB sorting using 1-node i3.4xlarge instance:

This was ran on a different instance having a different ip address.



```
MINGW64:/c/Users/saura/Downloads
root@ip-172-31-41-228:/Shiksha/hadoop/share/hadoop# hadoop jar TeraSorting.jar TeraSorting /input/1TB_inputdata /output/output
17/12/04 00:59:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/12/04 00:59:46 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.session-id
17/12/04 00:59:46 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
17/12/04 00:59:46 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
17/12/04 00:59:46 INFO input.FileInputFormat: Total input files to process : 1
17/12/04 00:59:47 INFO mapreduce.JobSubmitter: number of splits:954
17/12/04 00:59:47 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1117338954_0001
17/12/04 00:59:47 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
17/12/04 00:59:47 INFO mapreduce.Job: Running job: job_local1117338954_0001
17/12/04 00:59:47 INFO mapred.LocalJobRunner: OutputCommitter set in config null
17/12/04 00:59:47 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/04 00:59:47 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
17/12/04 00:59:47 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
17/12/04 00:59:48 INFO mapreduce.Job: Job job_local1117338954_0001 running in uber mode : false
17/12/04 00:59:48 INFO mapreduce.Job:  map 0% reduce 0%
17/12/04 00:59:48 INFO mapred.LocalJobRunner: Waiting for map tasks
17/12/04 00:59:48 INFO mapred.LocalJobRunner: Starting task: attempt_local1117338954_0001_m_000000_0
17/12/04 00:59:48 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/04 00:59:48 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
17/12/04 00:59:48 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
17/12/04 00:59:48 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/input/in:0+134217728
17/12/04 00:59:48 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
17/12/04 00:59:48 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
17/12/04 00:59:48 INFO mapred.MapTask: soft limit at 83886080
17/12/04 00:59:48 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
17/12/04 00:59:48 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
17/12/04 00:59:48 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
17/12/04 00:59:51 INFO mapred.MapTask: Spilling map output
17/12/04 00:59:51 INFO mapred.MapTask: bufstart = 0; bufend = 72112614; bufvoid = 104857600
17/12/04 00:59:51 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 23271028(93084112); length = 2943369/6553600
17/12/04 00:59:51 INFO mapred.MapTask: (EQUATOR) 75055974 kvi 18763988(75055952)
17/12/04 00:59:54 INFO mapred.MapTask: Finished spill 0
17/12/04 00:59:54 INFO mapred.MapTask: (RESET) equator 75055974 kv 18763988(75055952) kvi 18028160(72112640)
17/12/04 00:59:54 INFO mapred.LocalJobRunner:
17/12/04 00:59:54 INFO mapred.MapTask: Starting flush of map output
17/12/04 00:59:54 INFO mapred.MapTask: Spilling map output
17/12/04 00:59:54 INFO mapred.MapTask: bufstart = 75055974; bufend = 29619204; bufvoid = 104857600
17/12/04 00:59:54 INFO mapred.MapTask: kvstart = 18763988(75055952); kvend = 16338652(65354608); length = 2425337/6553600
17/12/04 00:59:56 INFO mapred.MapTask: Finished spill 1
17/12/04 00:59:56 INFO mapred.Merger: Merging 2 sorted segments
17/12/04 00:59:56 INFO mapred.Merger: Down to the last merge-pass, with 2 segments left of total size: 135559964 bytes
17/12/04 00:59:57 INFO mapred.Task: Task:attempt_local1117338954_0001_m_000000_0 is done. And is in the process of committing
17/12/04 00:59:57 INFO mapred.LocalJobRunner: map > sort
17/12/04 00:59:57 INFO mapred.Task: Task 'attempt_local1117338954_0001_m_000000_0' done.
17/12/04 00:59:57 INFO mapred.LocalJobRunner: Finishing task: attempt_local1117338954_0001_m_000000_0
17/12/04 00:59:57 INFO mapred.LocalJobRunner: Starting task: attempt_local1117338954_0001_m_000001_0
17/12/04 00:59:57 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
17/12/04 00:59:57 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
17/12/04 00:59:57 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
17/12/04 00:59:57 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/input/in:134217728+134217728
17/12/04 00:59:57 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
```

```
MINGW64:/c/Users/saura/Downloads

17/12/04 04:54:39 INFO output.FileOutputCommitter: Saved output of task 'attempt_local181930459_0001_r_000000_0' to hdfs://localhost:9000/output5/_temporary/0/task_local181930459_0001_r_00
17/12/04 04:54:39 INFO mapred.LocalJobRunner: reduce > reduce
17/12/04 04:54:39 INFO mapred.Task: Task 'attempt_local181930459_0001_r_000000_0' done.
17/12/04 04:54:39 INFO mapred.LocalJobRunner: Finishing task: attempt_local181930459_0001_r_000000_0
17/12/04 04:54:39 INFO mapred.LocalJobRunner: reduce task executor complete.
17/12/04 04:54:40 INFO mapreduce.Job:  map 100% reduce 100%
17/12/04 04:54:40 INFO mapreduce.Job: Job job_local181930459_0001 completed successfully
17/12/04 04:54:40 INFO mapreduce.Job: Counters: 35
        File System Counters
                FILE: Number of bytes read=7945385517000
                FILE: Number of bytes written=12896795722000
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=5848096700
                HDFS: Number of bytes written=1000000000
                HDFS: Number of read operations=280
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=20
        Map-Reduce Framework
                Map input records=10000000000
                Map output records=10000000000
                Map output bytes=1020000000000
                Map output materialized bytes=1040000048
                Input split bytes=800
                Combine input records=0
                Combine output records=0
                Reduce input groups=10000000000
                Reduce shuffle bytes=1040000048
                Reduce input records=10000000000
                Reduce output records=10000000000
                Spilled Records=29395200041
                Shuffled Maps =8
                Failed Shuffles=0
                Merged Map outputs=8
                GC time elapsed (ms)=256
                Total committed heap usage (bytes)=4414504900060
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1000000000000
        File Output Format Counters
                Bytes Written=1000000000000
17/12/04 04:54:40 INFO terasort.TeraSort: done
root@ip-172-31-41-228:/Shiksha/hadoop/share/hadoop#
```

```
root@ip-172-31-41-228:/Shiksha/hadoop/share/hadoop# cat test.txt
Time elapsed in sec:39564.18
root@ip-172-31-41-228:/Shiksha/hadoop/share/hadoop#
```

```
root@ip-172-31-41-228:/Shiksha/64# ./valsort part-r-00000
Records: 10000000000
Checksum:5631qw8wcipc294
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-41-228:/Shiksha/64#
```

| Dataset(TB) | Number of Nodes | Execution time(seconds) |
|---|---|---|
| 1 | 1 | 39564.18 |

## Setting up 8 nodes virtual cluster:

The Hadoop 8 node cluster is same as the single node cluster when compared with respect to installation of Hadoop but the configuration files for 8 nodes are different for master node.

While creating the instances we provided EBS volume to each slave node that will provide enough memory to store the temp files if the reducers fail and have to revert back to start.

Once all the instances are granted , our Amazon AWS Dashboard will show up 9 active instances with 1 master and 8 slave nodes.

As done on single node virtual cluster, we connect the master node on terminal using SSH and install and configure the Hadoop files to set up the 8 active slave nodes that will be linked with it.

### Configuration for Master Node

We connect to the master node using SSH on Public DNS of the Master Node Instance.

The host file for the master will have its private IP followed by its public DNS. Similarly it will have the same information of all the slaves that it needs to connect with.

The Host file of Slaves will contain its own Private IP and Public DNS along with master's Private IP and Public DNS.

The core site for Master will have the same set of entries as done for single node virtual cluster.

The core-site.xml for slave will contain only the default entries i.e. its own file system name having value as the Public DNS.

mapred-site.xml for Master node will contain the jobtracker address and all the cluster temp address. Another important property that I defined in the mapred-site.xml is the number of reducers that the Hadoop map-reduce sort program will use while generating the output file.

We have set the number of reducers to be 30 in order to tune mapreduce parallelism and even if the node fails the reduces can still be executed in the single wave. Note that this property is ignored when the jobtracker address is local.

For Slaves, the mapred-site.xml will have all the default properties of the Hadoop setup done for single node virtual cluster.

The reducers defined in the master node are also defined in the slaves.

The hdfs-site.xml for master and slaves have default property of replication factor 1

The Slaves file of Master node will have all the Public DNS of the Slaves.

The Slaves file of Slave node will have its own and master node's Public DNS address.

Once the configuration is done on master and slave node, we need to generate a ssh private key that will authorize all the DNS of Master and Slaves.

To access permission write eval: `**ssh-agent –s**`

**ssh-add keypair.pem**

If it's restricted, then chmod 400 keypair.pem and then above type instructions.

To authorize all the DNS we use the below command,

**ssh-keygen -t rsa**

**ssh-copy-id -i ~/.ssh/id_rsa.pub PUBLIC_DNS**

**chmod 0600 ~/.ssh/authorized_keys**

**Just the configuration we had done!!**

## _Hadoop 1 TB sorting using 8-node i3.large instance:_

Below mentioned are the steps to mount an EBS volume to our system.

**Sudo yum install mdadm**

We use the mdadm command to create a logical RAID device from the newly attached Amazon EBS volumes.

We can list the devices on our instance using lsblk command.

To create a RAID 0 array, execute the following command: **sudo mdadm --create --verbose /dev/md0 --level=0 --name=RAID_NAME --raid-devices=No_Of_Devices Device1 Device2**

Create a file system on your RAID array, and give that file system a label to use when you mount it later. **sudo mkfs.ext4 -L RAID_NAME /dev/md0**

Create a mount point for your RAID array: **sudo mkdir -p /mnt/raid**

Finally, mount the RAID device on the mount point that you created:
     **sudo mount LABEL=MAPREDUCE_RAID /mnt/raid**

The mentioned steps for mounting an EBS volume are shown in the below screenshot

We will be running an hadoop mapreduce algorithm of sorting 1TB of data . To generate  1TB of data , we used gensort and stored all the data in the mounted raid disk.

Once this data is generated, we will format the namenode as done previously for the single node cluster.

Once the node is formatted, The Hadoop clusters are ready and can be started.

Start the Hadoop cluster using following two commands:

**./start-dfs.sh**   This command will start all the Hadoop namenodes, datanodes, jobtracker and tasktrackers

**./start-yarn.sh**  This command will start the resource managers and the node managers.

You can type in jps command to check whether the above nodes have been started.

The Data Nodes, Name Nodes will start up on all the Virtual Clusters defined in the localhost of the master nodes.

Create an Input folder in the HDFS from where the Hadoop Mapreduce job will gather the input.

To create an input directory in the HDFS, we use the below commands,

**hdfs dfs -mkdir /input**

Once the input directory is created, we transfer the 1TB of input stored in Mounted Disk to the HDFS using below command,

**hdfs dfs -put inputdata  /input/input**

We can check if all the nodes are active and Hadoop setup is correctly configured on all the nodes.

Once confirmed we can start our map-reduce jar file that we used for sorting on single node virtual cluster.

Map_Reduce_TeraSort is the name of the jar file that contains the map-reduce algorithm implemented.

Map_Reduce is the name of the class

/input/input – Input directory of the text file

/output/output – Output Directory for the sorted file.

The Hadoop 8 Node Virtual Cluster can be stopped using the below commands,

**./stop-dfs.sh     ./stop-yarn.sh**

# SPARK:

Apache Spark is an open source cluster computing framework. It was originally developed at the University of California, Berkeley's AMPLab. However later, the Spark codebase was donated to Apache Software Foundation that has maintained it since. Spark provides an interface for programming the entire clusters with implicit data parallelism and fault-tolerance. Spark runs in-memory on the cluster, and it is not tied to Hadoop's MapReduce two-stage paradigm. This property of spark makes repeated access to the same data much faster.

## Spark Installation Steps:

**Over the Hadoop configured instance:**

```
# Download Spark to the ec2-user's home directory
cd /Saurabh
```

```
wget                https://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-
hadoop2.7.tgz


# Unpack Spark in the /opt directory
sudo tar zxvf spark-2.2.0-bin-hadoop2.7.tgz -C /opt


# Insert these lines into your ~/.bash_profile:
export SPARK_HOME=/opt/spark
PATH=$PATH:$SPARK_HOME/bin
export PATH
# Then exit the text editor and return to the command line.
```

Make the bashrc file as the source.

```
# Confirm that spark-submit is now in the PATH.
spark-submit --version
# (Should display a version number)
```

Run it by:

```
# Start the Scala shell
cd $SPARK_HOME
$SPARK_HOME/bin/spark-shell
```

Now use the command:

```
scala> :load spark_sort.scala
```

For 128 GB:

**Start**



**Result**

```
MINGW64:/c/Users/saura/Downloads                                                                    _

root@ip-172-31-17-91:/Saurabh/spark/bin# spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/12/04 16:11:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/12/04 16:11:12 WARN SparkConf: In Spark 1.0 and later spark.local.dir will be overridden by the value set by the cluster manager (via SPARK_LOCAL_DIRS in mesos/standalone and LOCAL_DIRS in YARN).
17/12/04 16:11:19 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://172.31.17.91:4040
Spark context available as 'sc' (master = local[*], app id = local-1512403873071).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.0
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> :load spark_sort.scala
Loading spark_sort.scala...
start_time: Long = 1512403914486
sortFile: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/input/input MapPartitionsRDD[1] at textFile at <console>:24
sortedobj: org.apache.spark.rdd.RDD[(String, String)] = ShuffledRDD[6] at sortByKey at <console>:26
end_time: Long = 1512410538772
Total time taken :6624286 ms
Total time taken in seconds: 6624.286 seconds

scala> sys.exit
root@ip-172-31-17-91:/Saurabh/spark/bin#
```

Valsort on spark 128 gb data:



```
MINGW64:/c/Users/saura/Downloads

root@ip-172-31-17-91:/Saurabh/spark/bin# hdfs dfs -getmerge /output/output/part* part-00000
17/12/04 16:33:07 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
root@ip-172-31-17-91:/Saurabh/spark/bin# ls -lrt
total 996208
-rw-r--r-- 1  500  500        1012 Jun 30 23:09 spark-submit.cmd
-rw-r--r-- 1  500  500        1128 Jun 30 23:09 spark-submit2.cmd
-rwxr-xr-x 1  500  500        1040 Jun 30 23:09 spark-submit
-rwxr-xr-x 1  500  500        1065 Jun 30 23:09 spark-sql
-rw-r--r-- 1  500  500        1010 Jun 30 23:09 spark-shell.cmd
-rw-r--r-- 1  500  500        1530 Jun 30 23:09 spark-shell2.cmd
-rwxr-xr-x 1  500  500        3017 Jun 30 23:09 spark-shell
-rw-r--r-- 1  500  500        1000 Jun 30 23:09 sparkR.cmd
-rw-r--r-- 1  500  500        1014 Jun 30 23:09 sparkR2.cmd
-rwxr-xr-x 1  500  500        1039 Jun 30 23:09 sparkR
-rw-r--r-- 1  500  500        1012 Jun 30 23:09 spark-class.cmd
-rw-r--r-- 1  500  500        2467 Jun 30 23:09 spark-class2.cmd
-rwxr-xr-x 1  500  500        3196 Jun 30 23:09 spark-class
-rw-r--r-- 1  500  500         988 Jun 30 23:09 run-example.cmd
-rwxr-xr-x 1  500  500        1030 Jun 30 23:09 run-example
-rw-r--r-- 1  500  500        1002 Jun 30 23:09 pyspark.cmd
-rw-r--r-- 1  500  500        1493 Jun 30 23:09 pyspark2.cmd
-rwxr-xr-x 1  500  500        2989 Jun 30 23:09 pyspark
-rw-r--r-- 1  500  500        2133 Jun 30 23:09 load-spark-env.sh
-rw-r--r-- 1  500  500        1909 Jun 30 23:09 load-spark-env.cmd
-rwxr-xr-x 1  500  500        1933 Jun 30 23:09 find-spark-home
-rw-r--r-- 1  500  500         899 Jun 30 23:09 beeline.cmd
-rwxr-xr-x 1  500  500        1089 Jun 30 23:09 beeline
drwxr-xr-x 5 root root        4096 Dec  4 16:11 metastore_db
-rw-r--r-- 1 root root         694 Dec  4 16:11 derby.log
-rw-r--r-- 1 root root         406 Dec  4 16:16 spark_sort.scala
-rw-r--r-- 1 root root        1539 Dec  4 16:17 result.txt
-rw-r--r-- 1 root root         994 Dec  4 16:29 part
-rw-r--r-- 1 root root  1020000000 Dec  4 16:33 part-00000
root@ip-172-31-17-91:/Saurabh/spark/bin# mv part-00000 /Saurabh/64
root@ip-172-31-17-91:/Saurabh/spark/bin# cd /Saurabh/64
root@ip-172-31-17-91:/Saurabh/64# ls
gensort  inputdata  part-00000  valsort
root@ip-172-31-17-91:/Saurabh/64# ./valsort part-00000
Records: 1280000000
Checksum: 4dd5c561ed2c702
Duplicate keys: 0
SUCCESS - all records in order
root@ip-172-31-17-91:/Saurabh/64#:
```

For 1 TB data(performed on second instance):

## Result:

```
root@ip-172-31-41-228:/Shiksha/spark/bin# spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/12/04 16:11:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/12/04 16:11:12 WARN SparkConf: In Spark 1.0 and later spark.local.dir will be overridden by the value set by the cluster manager (via SPARK_LOCAL_DIRS in mesos/standalone and LOCAL_DIRS in YARN).
17/12/04 16:11:19 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://172.31.41.228:4040
Spark context available as 'sc' (master = local[*], app id = local-1512403873071).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.0
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala> :load spark_sort.scala
Loading spark_sort.scala...
start_time: Long = 1234107954476
sortFile: org.apache.spark.rdd.RDD[String] = hdfs://localhost:9000/input/input MapPartitionsRDD[1] at textFile at <console>:24
sortedobj: org.apache.spark.rdd.RDD[(String, String)] = ShuffledRDD[6] at sortByKey at <console>:26
sortedobj: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at map at <console>:27
end_time: Long = 1234138016716
Total time taken : 3006224 ms
Total time taken in seconds : 30062.24 seconds

scala> sys.exit
root@ip-172-31-41-228:/Shiksha/spark/bin#
```

## Valsort on sorted 1 TB data output:

```
root@ip-172-31-41-228:/Shiksha/spark/bin# mv part-00000 /Shiksha/64
root@ip-172-31-41-228:/Shiksha/spark/bin# cd /Shiksha/64
root@ip-172-31-41-228:/Shiksha/64# ls
gensort  inputdata  part-00000  valsort
root@ip-172-31-41-228:/Shiksha/64# ./valsort part-00000
Records: 10000000000
Checksum: 2de6n161ed2c4p1
Duplicate keys: 0
SUCCESS - all records in order
root@ip-172-31-41-228:/Shiksha/64#:
```

## Performance evaluation of TeraSort:

| Experiment (instance/dataset) | Shared Memory TeraSort | Hadoop TeraSort | Spark TeraSort | MPI TeraSort |
|---|---|---|---|---|
| Compute Time (sec) [1xi3.large 128GB] | 12401 sec | 15260.23 seconds | 6624.286 seconds | |
| Data Read (GB) [1xi3.large 128GB] | 3840000000 | 1280000000 | 66.2 | |
| Data Write (GB) [1xi3.large 128GB] | 3840000000 | 1280000000 | 66.2 | |
| I/O Throughput (MB/sec) [1xi3.large 128GB] | 61.93 MB/sec | 16.77 MB/sec | 19.98 MB/sec | |
| Compute Time (sec) [1xi3.4xlarge 1TB] | 47964 sec | 39564.18 | 30062.24 sec | |
| Data Read (GB) [1xi3.4xlarge 1TB] | 30000000000 | 10000000000 | 523.5 GB | |
| Data Write (GB) [1xi3.4xlarge 1TB] | 30000000000 | 10000000000 | 523.5 GB | |
| I/O Throughput (MB/sec) [1xi3.4xlarge 1TB] | 125.09 MB/sec | 50.55 MB/sec | 66.4 MB/sec | |
| Compute Time (sec) [8xi3.large 1TB] | N/A | | | |
| Data Read (GB) [8xi3.large 1TB] | N/A | | | |
| Data Write (GB) [8xi3.large 1TB] | N/A | | | |
| I/O Throughput (MB/sec) [8xi3.large 1TB] | N/A | | | |
| Speedup (weak scale) | 1.495 | 1.33 | 1.30 | |
| Efficiency (weak scale) | 50.49% | 66.8% | 70.2% | |

## Graph Showing Comparison in Computation Time For Spark, Hadoop and shared Memory for Config 1 and Config 2:
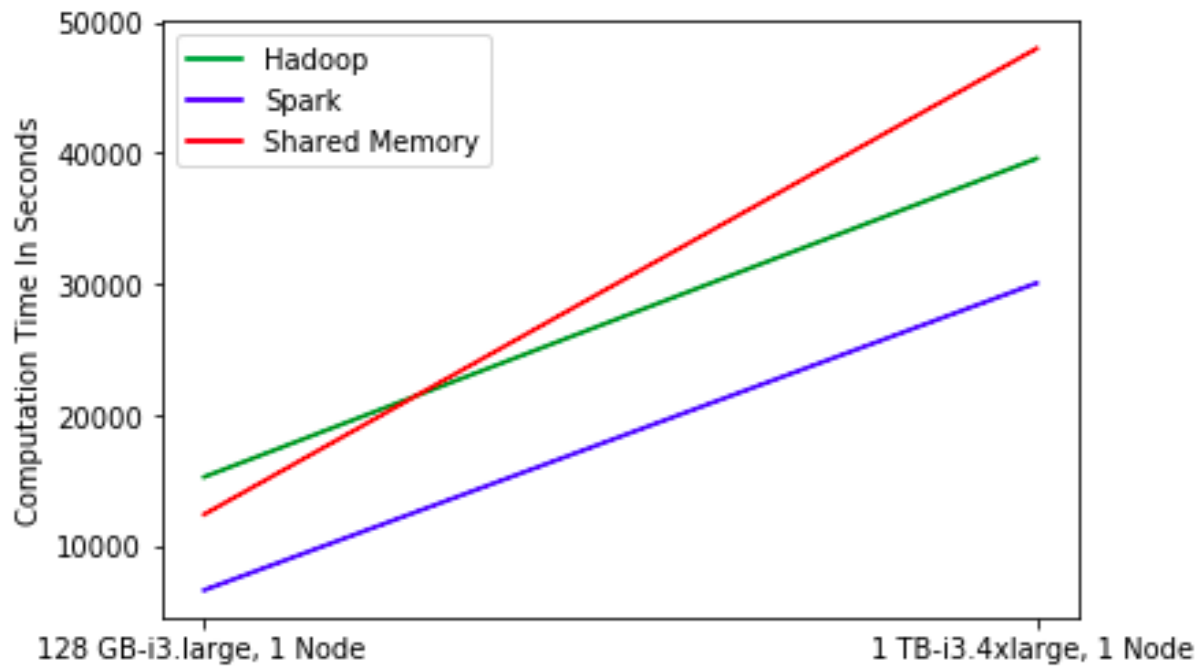


From the above graph, we can say that Spark shows best performance even for huge volume of data. Shared memory shows worst performance as expected for 1 TB of data.

## Graph Showing Throughput Time For Spark, Hadoop and shared Memory for Config 1 and Config 2:
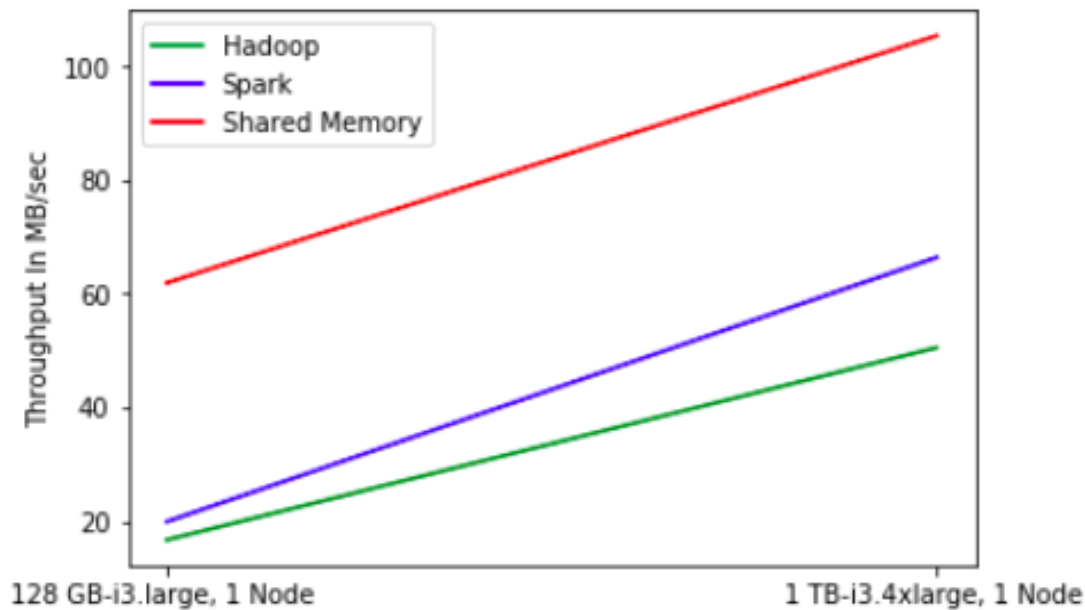


Throughput for Shared Memory is almost thrice the values for Hadoop and spark. Hence we can conclude that throughtput is much much higher than for other two.

## Speedup Comparison:

For 1-node i3.large, we can compare shared memory, Hadoop and Spark

Speedup:

1. Hadoop/Shared-Memory = 0.889

2. Spark/Shared-Memory = 0.869

## Conclusions:

We have performed shared memory sorting, sorting using Hadoop and sorting using Spark.

By looking at all above values we can say that spark performance is better than that of Hadoop and shared-memory. Ideally shared-memory's performance at one node should be better because it does not have any startup cost associated with it while in case of Hadoop and spark they have to setup master and slaves communication to manage the work through message exchange. When we want to process huge data in Gb's and Tb's. We should use spark as it uses RDD's (Resilient Distributed Datasets)

to process data and makes effective use of main memory. Spark could be 10 of 100 times faster than Hadoop in long run as it has better architecture of data processing and management.

## Prediction for 100 and 1000 nodes scale:

For 100 nodes and 1000 nodes if made available on those experiments then the performance will increase gradually every time any time extra node is added. This is because the virtual cores that is the CPU performance will increase each time any extra node is added.

## Sort Benchmark:

### Comparison with the winners:

Below are the results for Hadoop experiments for 2013 winners:

2013, 1.42 TB/min

**Hadoop**
102.5 TB in 4,328 seconds
2100 nodes x
(2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks)
Thomas Graves
Yahoo! Inc.

Below are the results for Spark experiments for 2014 winners:

2014, 4.27 TB/min

**Apache Spark**
100 TB in 1,406 seconds
207 Amazon EC2 i2.8xlarge nodes x
(32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD)
Reynold Xin, Parviz Deyhim, Xiangrui Meng,
Ali Ghodsi, Matei Zaharia
Databricks

We check the results of the sort benchmark and for the year 2013 and 2014, the winners of the benchmarking are Hadoop and Apache Spark respectively.

Comparing the result of Hadoop system that won the 2013 award for sort, the configuration of the system is much better than the instances that we use for our experiments. Hence the performance is incomparable as the system specification is better for the Daytona Hadoop. The Hadoop system 2013 winner produced a throughput of 1.42TB/min which is very high as compared to the throughput that we receive with those experiments.

Similarly, Apache Spark winner of 2014 produces a throughput of 4.27Tb/min which has system of 32 Virtual cores and a memory of 244 GB with 6400 GB SSD. This configuration is pretty less as compared to i3.large or i3.4xlarge instances on which this experiments have been tested.

Hence we get lesser throughput gradually with respect to the specifications on which both the experiments have been performed.

## *CloudSort Benchmark:*

This is benchmark based on external sorting basically to produce intended results on cloud environments with huge IO workload. All the huge cloud platforms gradually write off the initial cost, still having large profitability by running the cloud services. Thus this paper defines the efficiency of external sort from total cost ownership i.e. asking itself the minimum cost for sorting a fixed number of records on any public cloud.

Thus it explains the advantages of public cloud that satisfy some issues found in the other sort benchmarks. Those advantages are as follows:

*Accessibility:* Cloud sort levels the users. Even the groups with limited budget can develop their implementation and run their benchmarks.

*Affordability:* Comparison of AWS pricing for running sort benchmark is manageable for groups with modest budgets.

*Auditability:* Provides good detective work on the part of auditors.

The paper on Cloud Sort also states about the rules defined for the public cloud on the factors such as Pricing, Storage, Reliability and Reporting.

Hence, it can be concluded that the cloud platforms are poorly provisioned for IO intensive workloads. The cloud sort benchmark will help to initiate innovation in the public cloud for supporting IO Intensive tasks.

As the cloud grows gradually, the benchmark will provide you best platforms for building your data fields but also find efficient sort implementation from total cost of ownership perspective.