CS 553: CLOUD COMPUTING

Tera Sort Benchmarking

By

Saptarshi Banerjee (A20376116)

Vaibhav Aggarwal (A20374988)

Abhishek Vijhani (A20377670)

Shared Memory Sort

We have written the code for the Shared Memory Tera Sort in Java. Tera Sort is a popular benchmark that measures the amount of time to sort one terabyte of randomly distributed data on a given computer system. We have performed sorting on 128GB and 1 TB datasets. Using the key's ASCII Value, we have performed sorting. We have divided the original dataset into the block size (size of file/ Number of Temporary files). After that we have created unsorted chunks for each file and applied Quick Sort on these chunks, which provides me with sorted chunks as output. This part of the program has threads applied to it to parallelize the process of sorting. Then with the help of k-way external merge sort, I have merged all the Sorted Chunks of earlier phase into a final sorted output file.

Hadoop Tera Sort

We have written the code for Hadoop Sorting in Java. In this we have used Map-Reduce based implementation. It contains of two phases Sort and Shuffle Phase, which internally sorts the data without using any code. We have provided the mapper with input as key and values, and the Sort and Shuffle phase will categorize the data and sort them according to its Key. The reducer is used to give the output of the Sort and Shuffle Phase. We have tested the Hadoop code on 128 GB, 1 TB

Spark Tera Sort

We have written the code for Spark using the PySpark Shell, which uses Python. It also uses the Map-Reduce based implementation. When we install Spark on it automatically installs various features like R-Studio, Scala, Python. We have implemented sorting using Python sortByKey(), functionality, which sorts the data based on Key value (10 Bytes). Then we obtained the chunks of the sorted data.

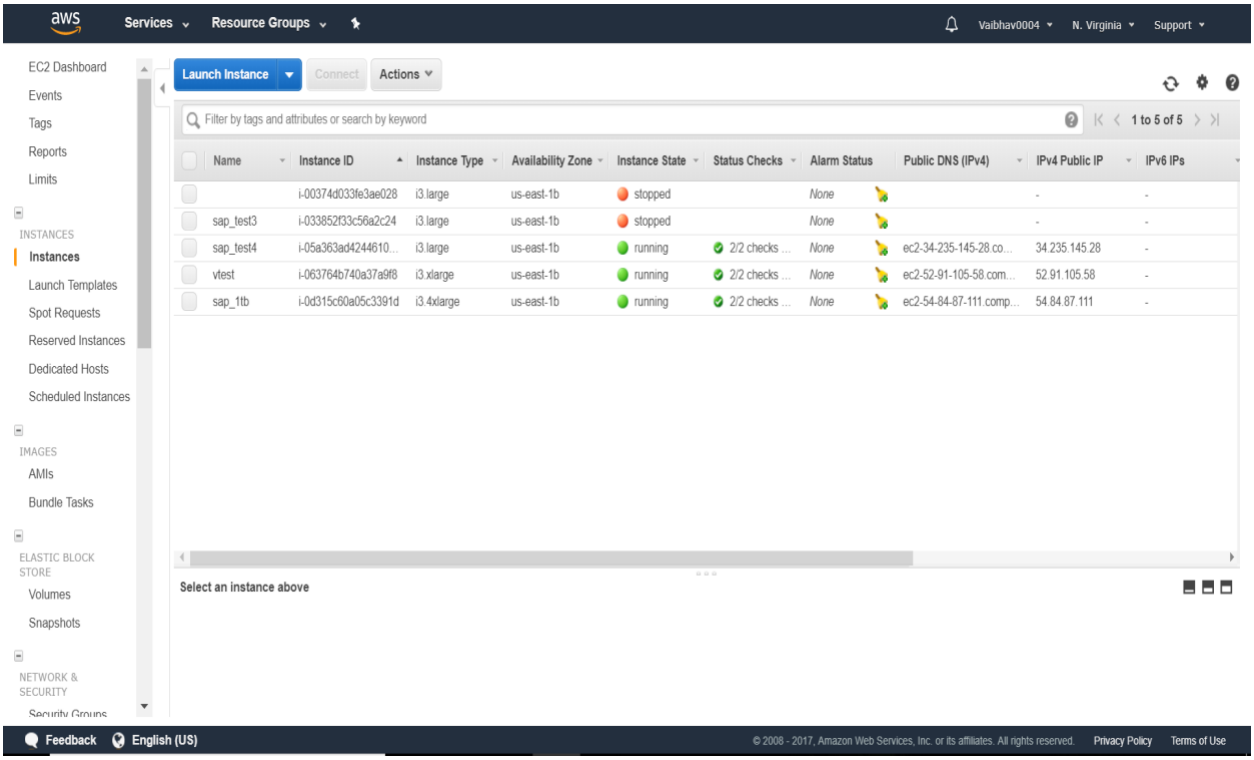
Performance is done on AWS instance (i3.large), using Ubuntu. The Specification for i3.large is

Instance Type	i3.large (Ubuntu)
Ram	15.25GB
Number of Cores	2(vCPU)
Storage	1 x 0.475 NVMe SSD
Networking Performance	Upto 10 Gigabit
Region	Us-east-1 (N. Virginia)
EBS Volume Used	500GB SSD

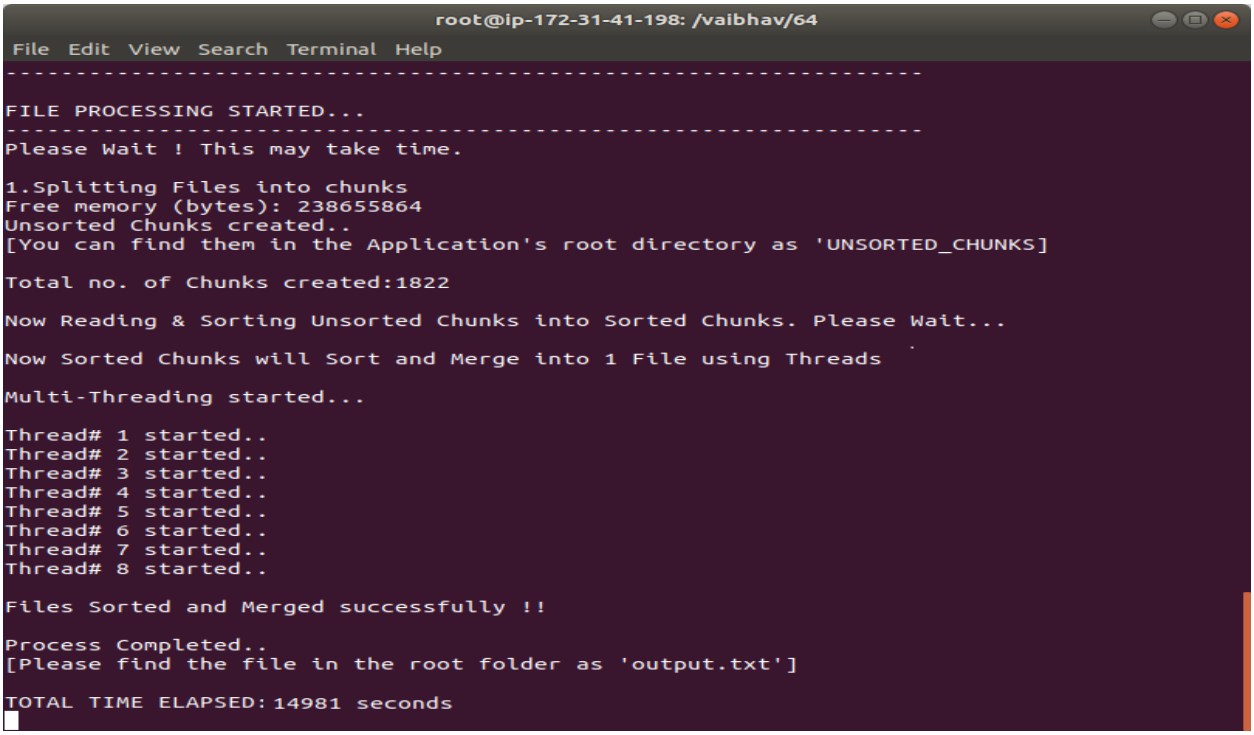
Performance is done on AWS instance (i3.4xlarge), using Ubuntu. The Specification for i3.4xlarge is

Instance Type	i3.4xlarge (Ubuntu)
Ram	122GB
Number of Cores	16(vCPU)
Storage	2 x 1.9 NVMe SSD
Networking Performance	Upto 10 Gigabit
Region	Us-east-1 (N. Virginia)
EBS Volume Used	2045GB SSD

Shared Memory Tera Sort



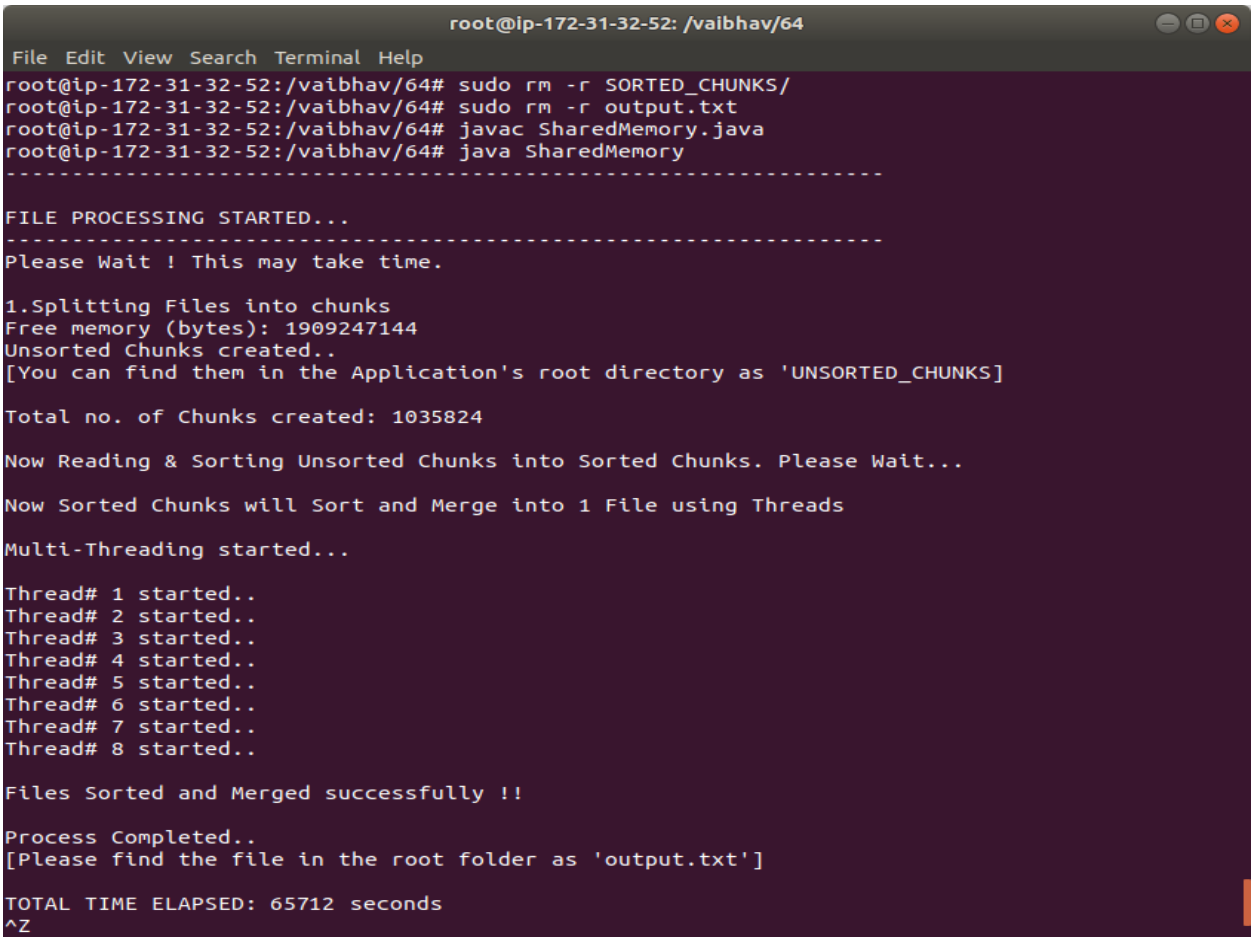
i3.large: 128 GB Using gensort



Total time taken 14981 Seconds

i3.4xlarge: 1 TB Using gensort

EBS volume used: 2045GB SSD



Total time taken 65712 Seconds

Hadoop Tera Sort

Configuring Hadoop on AWS

Here we have created i3.large instance on AWS

1. Login to new EC2 instance using ssh
   (ssh -i aws-key.pem ubuntu@54.183.30.236)

2. Login as root user to install base packages (java 8)
   (**sudo su**
   sudo apt-get install python-software-properties
   sudo add-apt-repository ppa:webupd8team/java
   sudo apt-get update
   sudo apt-get install oracle-java8-installer)

3. Check the java version
   (java –version)

4. Download latest stable Hadoop using wget from one of the Apache mirrors.
   wget http://www.trieuvan.com/apache/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
   (**tar xzf hadoop-2.6.0.tar.gz**)

5. Create a directory where the hadoop will store its data. We will set this directory path in hdfs-site.
   (mkdir hadoopdata)

6. Add the Hadoop related environment variables in your bash file.
   (vi ~/.bashrc
   export HADOOP_HOME=/home/ubuntu/hadoop-2.6.0
   export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
   export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
   export JAVA_HOME=/usr/lib/jvm/java-8-oracle
   PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin)

7. Save and exit and use this command to refresh the bash settings.
   (source ~/.bashrc)

8. Setting hadoop environment for password less ssh access. Password less SSH Configuration is a mandatory installation requirement. However it is more useful in distributed environment.
   (ssh-keygen -t rsa -P ''
   cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys)

9. check password less ssh access to localhost
   (ssh localhost)

10. exit from inner localhost shell
    (exit)

11. Set the hadoop config files. We need to set the below files in order for hadoop to function properly.
    core-site.xml
    hadoop-env.sh
    yarn-site.xml
    hdfs-site.xml
    mapred-site.xml

12. go to directory where all the config files are present (cd /home/ubuntu/hadoop-2.6.0/etc/Hadoop), Copy and paste the below configurations in core-site.xml
    (Add the following text between the configuration tabs.)
    (<property>
    <name>hadoop.tmp.dir</name>
    <value>/home/ubuntu/hadooptmp/hadoop-${user.name}</value>
    <description>A base for other temporary directories.</description>
    </property>
    <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
    </property>)

13. Copy and paste the below configurations in hadoop-env.sh

    get the java home directory using:

    (readlink -f `which java`)

    Example output: /usr/lib/jvm/java-8-oracle/jre/bin/java (NOTE THE JAVA_HOME PATH. JUST GIVE THE BASE DIRECTORY PATH)

Need to set JAVA_HOME in hadoop-env.sh
(export JAVA_HOME=/usr/lib/jvm/java-8-oracle)

14. Copy and paste the below configurations in mapred-site.xml

    copy mapred-site.xml from mapred-site.xml.template
    (cp mapred-site.xml.template mapred-site.xml
    vi mapred-site.xml)

    Add the following text between the configuration tabs.
    <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
    </property>

15. Copy and paste the below configurations in yarn-site.xml

    Add the following text between the configuration tabs.
    <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
    </property>

- Copy and paste the below configurations in hdfs-site.xml

  Add the following text between the configuration tabs.
  <property>
  <name>dfs.replication</name>
  <value>1</value>
  </property>
  <property><name>dfs.name.dir</name>
  <value>file:///home/ubuntu/hadoopdata/hdfs/namenode</value>
  </property>
  <property>
  <name>dfs.data.dir</name>
  <value>file:///home/ubuntu/hadoopdata/hdfs/datanode</value>
  </property>

16. Formatting the HDFS file system via NameNode (after installing hadoop, for the first time we have to format the HDFS file system to make it work)

    (hdfs namenode -format)

17. Issue the following commands to start Hadoop

(cd sbin/
./start-dfs.sh
./start-yarn.sh)

(start-all.sh)

OR you can separately start required services as below:

Name node:
(hadoop-daemon.sh start namenode)

Data node:
(hadoop-daemon.sh start datanode)

Resource Manager:
(yarn-daemon.sh start resourcemanager)

Node Manager:
(yarn-daemon.sh start nodemanager)

Job History Server:
(mr-jobhistory-daemon.sh start historyserver)

18. Once hadoop has started point your browser to http://localhost:50070/

19. Check for hadoop processes /daemons running on hadoop with Java Virtual Machine Process Status Tool.
    (jps)

20. OR you can check TCP and port details by using

    (sudo netstat -plten | grep java)

Configuring Hadoop on a Single Node

Here we are configuring the bashrc file

```
# 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
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_INSTALL=/sapto/hadoop-2.7.4
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
```

Now we will configure xml file

```
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/ubuntu/hadooptmp/hadoop-${user.name}</value>
<description>A base for other temporary directories.</description>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Configuring the HDFS file

Here we have kept the dfs replication 1

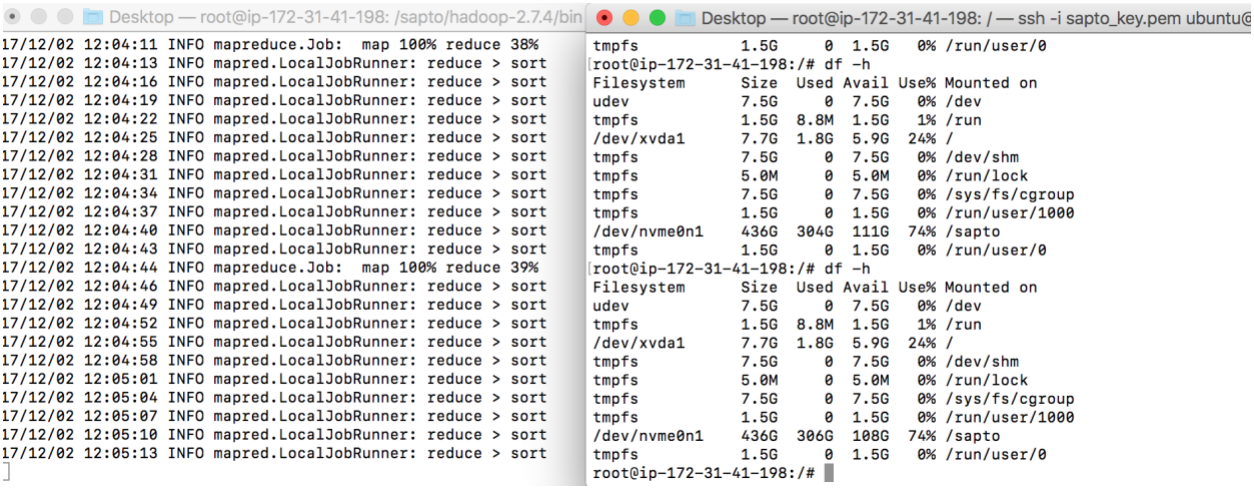
```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property><name>dfs.name.dir</name>
<value>file:///sapto/hadoopdata/hdfs/namenode</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:///sapto/hadoopdata/hdfs/datanode</value>
</property>
</configuration>
```

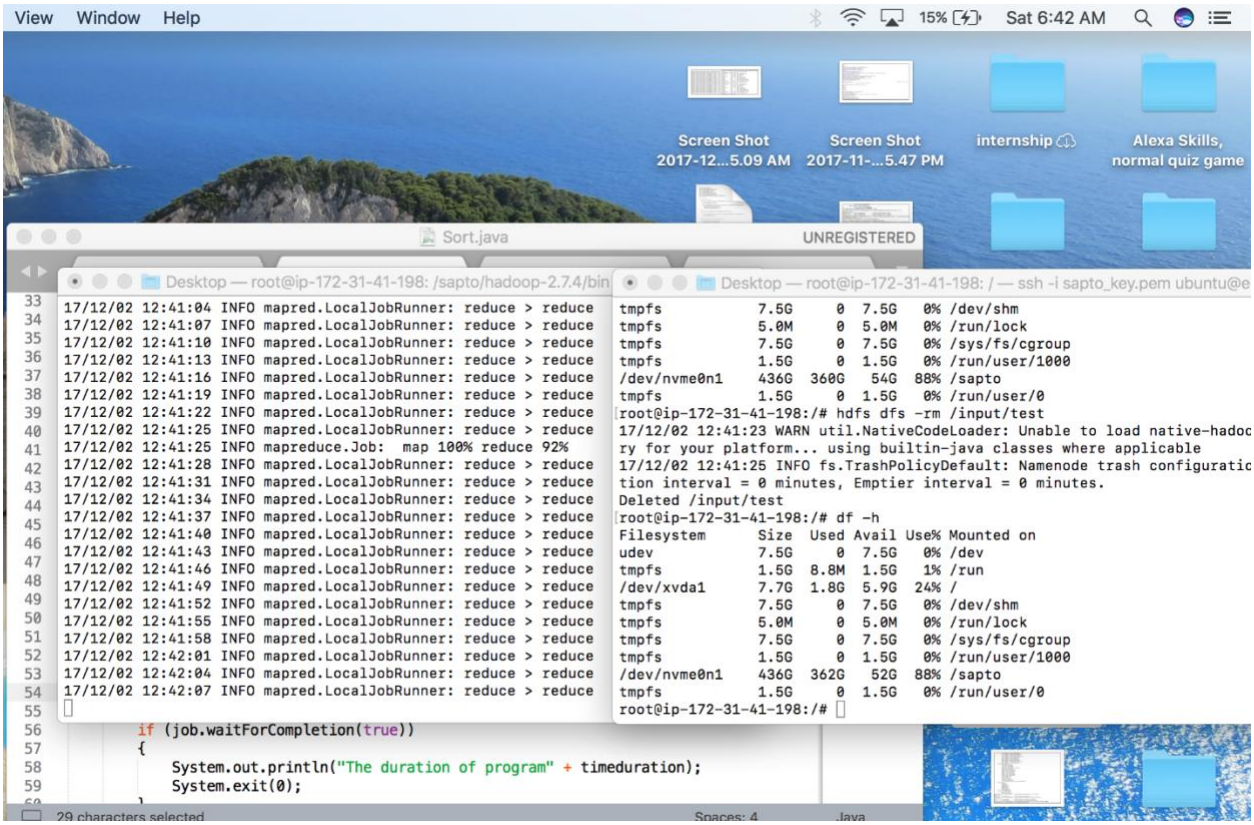
Hadoop Single Node 128GB

In this image, you can see the starting of Hadoop Job

```
        at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
root@ip-172-31-41-198:/sapto/hadoop-2.7.4/bin# hadoop jar sort1.jar SortHadoop
input/test3 /output/terasortresult
17/12/02 10:23:11 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ry for your platform... using builtin-java classes where applicable
17/12/02 10:23:11 INFO Configuration.deprecation: session.id is deprecated. Ins
ead, use dfs.metrics.session-id
17/12/02 10:23:11 INFO jvm.JvmMetrics: Initializing JVM Metrics with processNam
=JobTracker, sessionId=
17/12/02 10:23:12 WARN mapreduce.JobResourceUploader: Hadoop command-line optio
 parsing not performed. Implement the Tool interface and execute your applicati
n with ToolRunner to remedy this.
17/12/02 10:23:12 INFO input.FileInputFormat: Total input paths to process : 1
17/12/02 10:23:12 INFO mapreduce.JobSubmitter: number of splits:745
17/12/02 10:23:12 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_l
cal1940813515_0001
17/12/02 10:23:12 INFO mapreduce.Job: The url to track the job: http://localhos
:8080/
17/12/02 10:23:12 INFO mapreduce.Job: Running job: job_local1940813515_0001
17/12/02 10:23:13 INFO mapred.LocalJobRunner: OutputCommitter set in config nul
17/12/02 10:23:13 INFO output.FileOutputCommitter: File Output Committer Algori
hm version is 1
```

Hadoop Job running on way






```
l7/12/02 12:48:01 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:04 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:07 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:10 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:13 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:16 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:19 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:22 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:23 INFO mapreduce.Job:  map 100% reduce 100%
l7/12/02 12:48:25 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:28 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:31 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:34 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:37 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:40 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:43 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:46 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:48 INFO mapred.Task: Task:attempt_local1940813515_0001_r_00000
 is done. And is in the process of committing
l7/12/02 12:48:48 INFO mapred.LocalJobRunner: reduce > reduce
l7/12/02 12:48:48 INFO mapred.Task: Task attempt_local1940813515_0001_r_00000
 is allowed to commit now
l7/12/02 12:48:48 INFO output.FileOutputCommitter: Saved output of task 'atte
_local1940813515_0001_r_000000_0' to hdfs://localhost:9000/output/terasortres
```

Hadoop Job mapped and reduced 100%

```
        Combine input records=1000000000
        Combine output records=1000000000
        Reduce input groups=999999999
        Reduce shuffle bytes=101000004470
        Reduce input records=1000000000
        Reduce output records=999999999
        Spilled Records=4990604755
        Shuffled Maps =745
        Failed Shuffles=0
        Merged Map outputs=745
        GC time elapsed (ms)=106996
        Total committed heap usage (bytes)=399716122624
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
```

```
.5G     0  7.5G   0% /dev/shm
.0M     0  5.0M   0% /run/lock
.5G     0  7.5G   0% /sys/fs/cgroup
.5G     0  1.5G   0% /run/user/1000
36G  379G   36G  92% /sapto
.5G     0  1.5G   0% /run/user/0
-198:/sapto# df -h
ize  Used Avail Use% Mounted on
.5G     0  7.5G   0% /dev
.5G  8.8M  1.5G   1% /run
.7G  1.8G  5.9G  24% /
.5G     0  7.5G   0% /dev/shm
.0M     0  5.0M   0% /run/lock
.5G     0  7.5G   0% /sys/fs/cgroup
.5G     0  1.5G   0% /run/user/1000
36G  332G   82G  81% /sapto
.5G     0  1.5G   0% /run/user/0
-198:/sapto#
```

```
13515_0001_r_000000_0
17/12/02 12:48:48 INFO mapred.LocalJobRunner: reduce task executor complete.
17/12/02 12:48:50 INFO mapreduce.Job: Job job_local1940813515_0001 completed suc
cessfully
17/12/02 12:48:50 INFO mapreduce.Job: Counters: 35
        File System Counters
                FILE: Number of bytes read=38181308369819
                FILE: Number of bytes written=75856220738849
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=37404037074944
                HDFS: Number of bytes written=98999999901
                HDFS: Number of read operations=559501
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=748
        Map-Reduce Framework
                Map input records=1000000000
                Map output records=1000000000
                Map output bytes=99000000000
                Map output materialized bytes=101000004470
                Input split bytes=73010
                Combine input records=1000000000
                Combine output records=1000000000
```

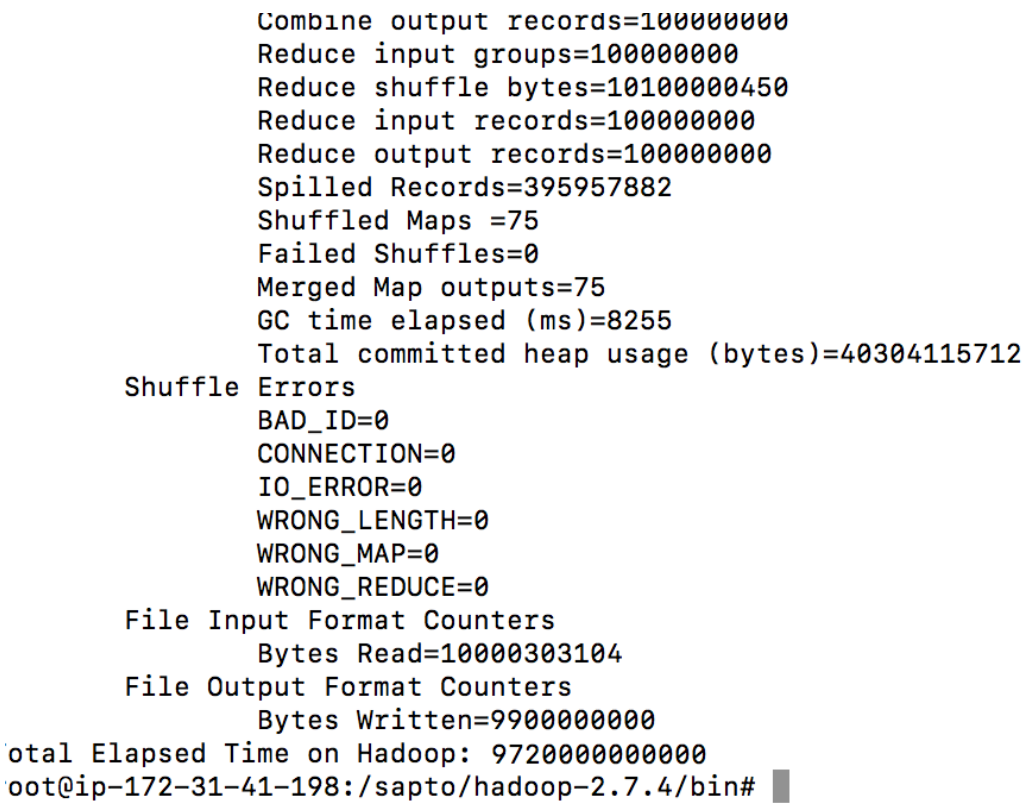
The Ending of Hadoop Job.

Screenshot of 128GB Output File

```
 !0<v9`:`          00000000000000000000000031290618  9999DDDDEEEE33331111AAAA222
00005555AAAAEEEE00009999
 !0<yzmJg          00000000000000000000000275F95BF  0000AAAA444400006666777766
0000BBBB2222AAAA44440000
 !0=#:U#q          00000000000000000000000001E724D94  FFFFAAAAEEEE5555EEEE3333333
EEEEAAAA77770000BBBBDDDD
 !0=)Z,x*          00000000000000000000000027D70205  8888CCCC7777AAAA11118888EEE
FFFF55556666777700002222
 !0=06=XB          00000000000000000000000002915078B  BBBBFFFF000044440000777700
DDDD44443333BBBB99994444
 !0=4nesb          00000000000000000000000002985F704  8888AAAA1111EEEE66660000111
BBBB11114444EEEE8888DDDD
 !0=<MM/]          00000000000000000000000001ED2AD1E  AAAA8888FFFF8888CCCC3333BBB
AAAAFFFF33330000DDDD3333
 !0=E1n(x          00000000000000000000000267C9410  0000BBBB111111111111AAAA888
BBBB999977779999BBBB1111
 !0=IbC>|          00000000000000000000000003150E986  AAAADDDD5555222299998888EEE
CCCCBBBB3333EEEE0000BBBB
 !0=Thpqa          00000000000000000000000000D26141D  EEEE888822224444000066666EEE
8888FFFF5555CCCCAAAAAAAA
 !0=ei@b>          00000000000000000000000001A94D1F3  AAAA777733330000999999FFF
AAAAAAAAEEEE99992222CCCC
 !0=hK*t(          00000000000000000000000022CC777A  333377779999DDDD777799998888
5555EEEEDDDDFFFFCCCC7777
```

		i-00374d033fe3ae028	i3.large	us-east-1b	stopped		None	
	sap_test3	i-033852f33c56a2c24	i3.large	us-east-1b	stopped		None	
	sap_test4	i-05a363ad4244610...	i3.large	us-east-1b	running	2/2 checks ...	None	ec2-34-235-145-28
	vtest	i-063764b740a37a9f8	i3.xlarge	us-east-1b	running	2/2 checks ...	None	ec2-52-91-105-58.c
	sap_1tb	i-0d315c60a05c3391d	i3.4xlarge	us-east-1b	running	2/2 checks ...	None	ec2-54-84-87-111.c

In is image you can see the total time elapsed on Hadoop Job



So for 128 Gb it took us around 2.7 hrs.

Hadoop Single Node 1TB

Here, you can see the starting time of Hadoop Job

```
:oot@ip-172-31-32-52:/sapto/hadoop-2.7.4/bin# hadoop jar sort.jar SortHadoop /ir
)ut/sortftb /output/sortftbout
.7/12/03 13:08:08 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
:y for your platform... using builtin-java classes where applicable
.7/12/03 13:08:09 INFO Configuration.deprecation: session.id is deprecated. Inst
ead, use dfs.metrics.session-id
.7/12/03 13:08:09 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName
=JobTracker, sessionId=
.7/12/03 13:08:09 WARN mapreduce.JobResourceUploader: Hadoop command-line optior
 parsing not performed. Implement the Tool interface and execute your applicatic
 with ToolRunner to remedy this.
.7/12/03 13:08:09 INFO input.FileInputFormat: Total input paths to process : 1
.7/12/03 13:08:09 INFO mapreduce.JobSubmitter: number of splits:75
.7/12/03 13:08:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_lc
:al478218440_0001
.7/12/03 13:08:09 INFO mapreduce.Job: The url to track the job: http://localhost
:8080/
.7/12/03 13:08:09 INFO mapreduce.Job: Running job: job_local478218440_0001
.7/12/03 13:08:09 INFO mapred.LocalJobRunner: OutputCommitter set in config null
.7/12/03 13:08:09 INFO output.FileOutputCommitter: File Output Committer Algorit
```

Hadoop Job running for 1 TB File

Desktop — root@ip-172-31-32-52: /sapto/hadoop-2.7.4/bin — ssh -i sapto_ke...

```
17/12/03 21:44:16 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:19 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:22 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:25 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:28 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:31 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:34 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:37 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:40 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:43 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:46 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:49 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:52 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:55 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:44:58 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:45:01 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:45:04 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:45:07 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:45:10 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:45:10 INFO mapreduce.Job:  map 100% reduce 63%
17/12/03 21:45:13 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:45:16 INFO mapred.LocalJobRunner: reduce > sort
17/12/03 21:45:19 INFO mapred.LocalJobRunner: reduce > sort
```

Here, we are generating sort file of 1TB

```
u should have received a copy of the GNU General Public License
ong with this program; if not, write to the Free Software Foundation,
c., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.

rsion 1.5, cvs $Revision: 1.14 $
MP Pump version 1.2.3, svn: 130
ot@ip-172-31-32-52:/sapto1/64# ./gensort -a -t8 1000000000000 sortftb
```

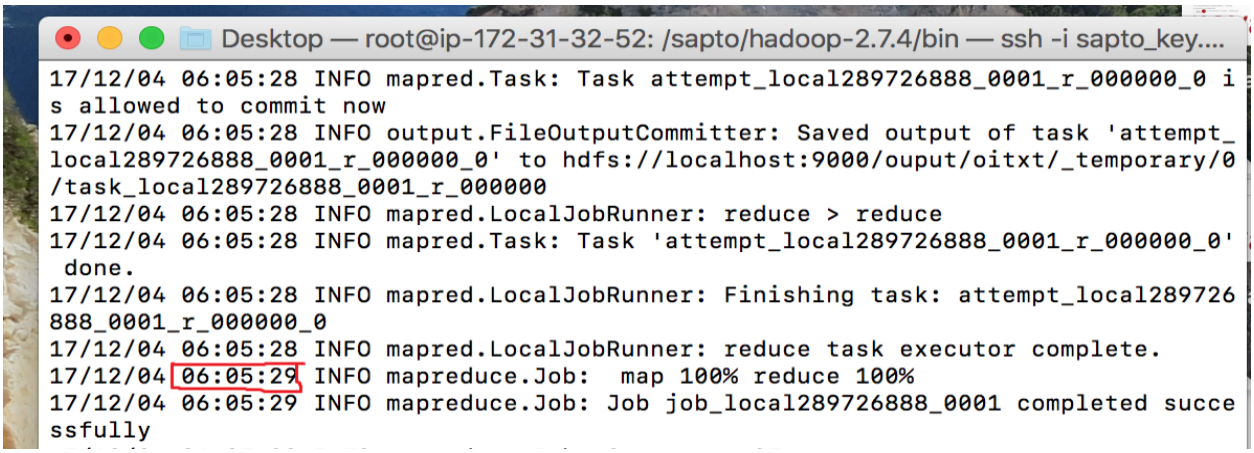
Here, Reduce Jobs are running for 1 Single node

```
17/12/03 22:44:53 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:44:56 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:44:59 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:02 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:05 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:08 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:11 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:14 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:17 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:20 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:23 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:26 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:29 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:32 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:35 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:38 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:41 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:44 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:47 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:50 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:53 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:56 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:45:59 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:02 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:05 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:08 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:11 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:14 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:17 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:20 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:23 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:26 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:29 INFO mapred.LocalJobRunner: reduce > reduce
17/12/03 22:46:32 INFO mapred.LocalJobRunner: reduce > reduce
```

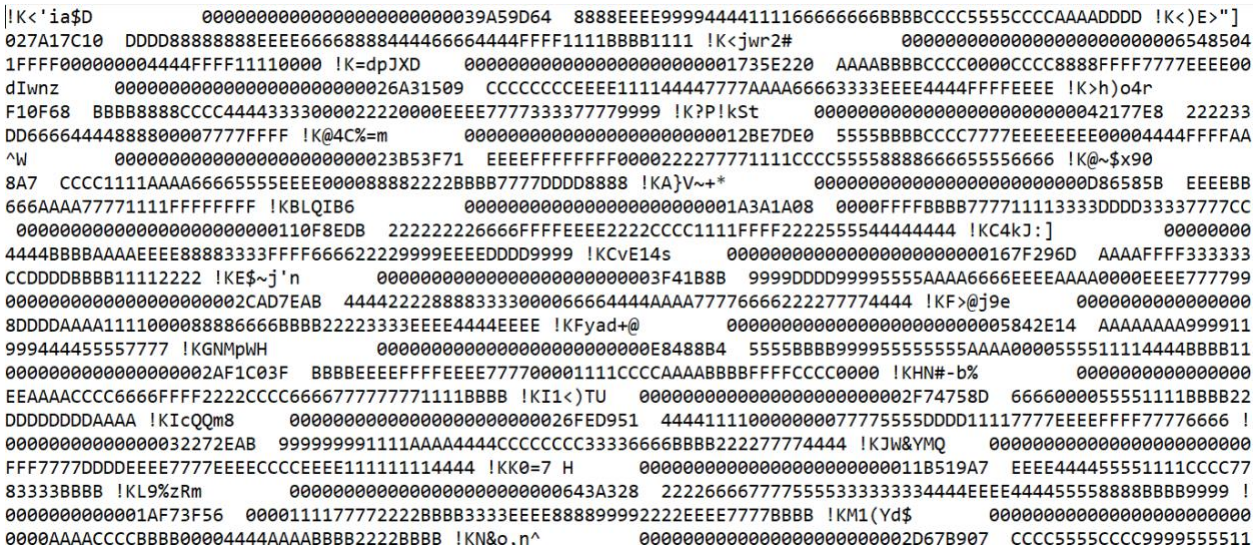
```
                HDFS: Number of bytes written=296999999010
                HDFS: Number of read operations=5013118
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2239
        Map-Reduce Framework
                Map input records=3000000000
                Map output records=3000000000
                Map output bytes=297000000000
                Map output materialized bytes=303000013416
                Input split bytes=225836
                Combine input records=3000000000
                Combine output records=3000000000
                Reduce input groups=2999999990
                Reduce shuffle bytes=303000013416
                Reduce input records=3000000000
                Reduce output records=2999999990
                Spilled Records=16851737032
                Shuffled Maps =2236
                Failed Shuffles=0
                Merged Map outputs=2236
                GC time elapsed (ms)=383451
                Total committed heap usage (bytes)=1199164096512
        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=300009154560
        Total Time Elapsed time on Hadoop : 64800000000000
```

So, the total time required by Hadoop to sort 1 Tb data is around 18.5 hours

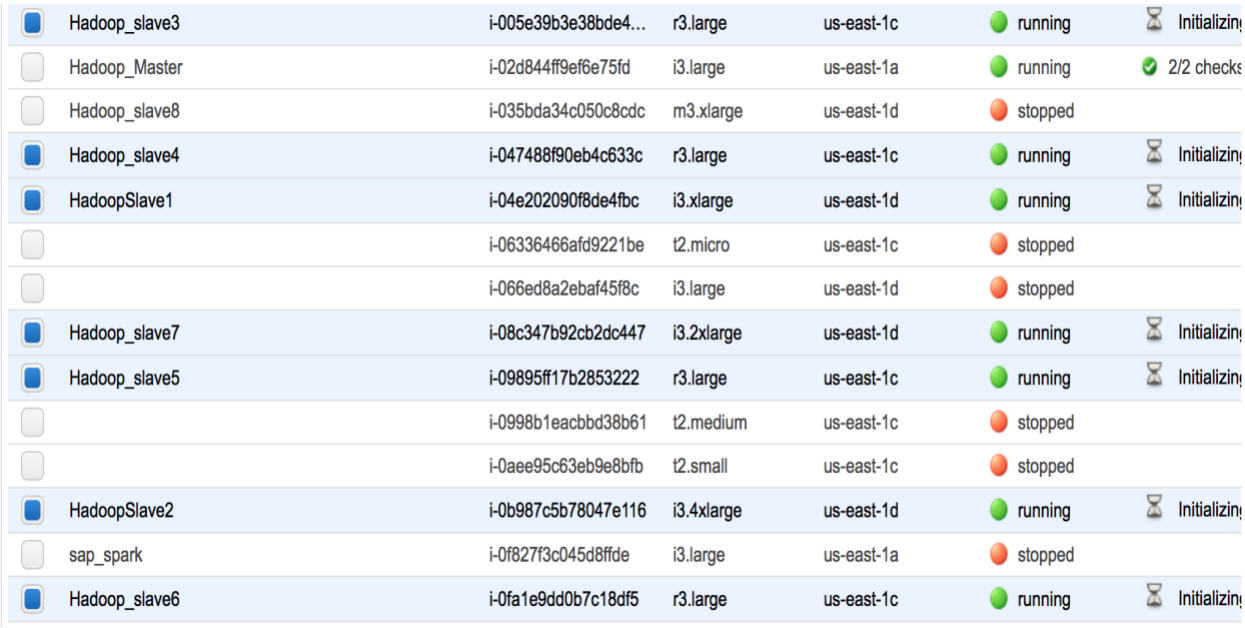
Here, you can see the ending time of Hadoop Job



Screenshot of 1TB Output File



Configuring Hadoop on a Multi Node



Hadoop_slave3	i-005e39b3e38bde4...	r3.large	us-east-1c	running	Initializing
Hadoop_Master	i-02d844ff9ef6e75fd	i3.large	us-east-1a	running	2/2 checks
Hadoop_slave8	i-035bda34c050c8cdc	m3.xlarge	us-east-1d	stopped	
Hadoop_slave4	i-047488f90eb4c633c	r3.large	us-east-1c	running	Initializing
HadoopSlave1	i-04e202090f8de4fbc	i3.xlarge	us-east-1d	running	Initializing
	i-06336466afd9221be	t2.micro	us-east-1c	stopped	
	i-066ed8a2ebaf45f8c	i3.large	us-east-1d	stopped	
Hadoop_slave7	i-08c347b92cb2dc447	i3.2xlarge	us-east-1d	running	Initializing
Hadoop_slave5	i-09895ff17b2853222	r3.large	us-east-1c	running	Initializing
	i-0998b1eacbbd38b61	t2.medium	us-east-1c	stopped	
	i-0aee95c63eb9e8bfb	t2.small	us-east-1c	stopped	
HadoopSlave2	i-0b987c5b78047e116	i3.4xlarge	us-east-1d	running	Initializing
sap_spark	i-0f827f3c045d8ffde	i3.large	us-east-1a	stopped	
Hadoop_slave6	i-0fa1e9dd0b7c18df5	r3.large	us-east-1c	running	Initializing

We can see Hadoop Master and Hadoop Slaves running. We configured the IPs of each and every slave.

Here we are configuring the .bashrc file

```
# 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
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_INSTALL=/sapto/hadoop-2.7.4
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
```

Now we will configure xml file

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://master:9001</value>
</property>
</configuration>
```

Configuring the HDFS file

Here we have kept the dfs replication 1

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.permissions</name>
<value>false</value>
</property>
</configuration>
```

Configuring the YARN file

Here, we have changed the port number of slaves

```
<?xml version="1.0"?>
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value>master:8038</value>
</property>
<property>
<name>yarn.resourcemanager.address</name>
<value>master:8039</value>
</property>
<property>
<name>yarn.resourcemanager.webapp.address</name>
<value>master:8086</value>
</property>
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>master:8037</value>
</property>
<property>
<name>yarn.resourcemanger.admin.address</name>
<value>master:8033</value>
</property>
</configuration>
```

Hadoop Multi Node 1TB

Here, we are generating the 1TB Data

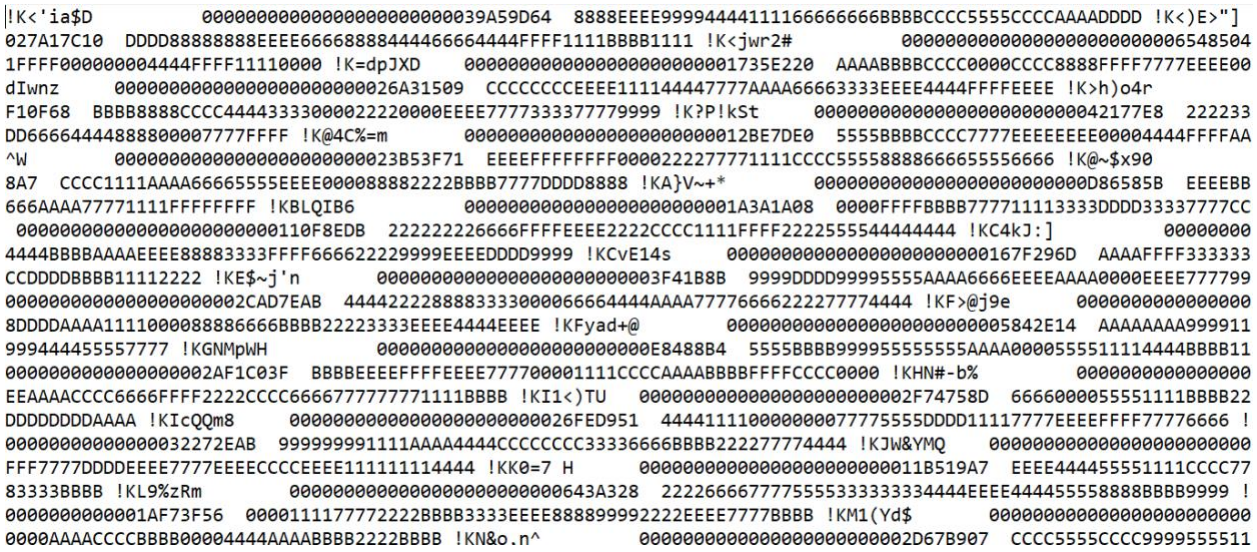
```
u should have received a copy of the GNU General Public License
ong with this program; if not, write to the Free Software Foundation,
c., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.

rsion 1.5, cvs $Revision: 1.14 $
MP Pump version 1.2.3, svn: 130
ot@ip-172-31-32-52:/sapto1/64# ./gensort -a -t8 1000000000000 sortftb
```

Here, you can see the ending time of the Hadoop Job

```
                HDFS: Number of bytes written=296999999010
                HDFS: Number of read operations=5013118
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2239
        Map-Reduce Framework
                Map input records=3000000000
                Map output records=3000000000
                Map output bytes=297000000000
                Map output materialized bytes=303000013416
                Input split bytes=225836
                Combine input records=3000000000
                Combine output records=3000000000
                Reduce input groups=2999999990
                Reduce shuffle bytes=303000013416
                Reduce input records=3000000000
                Reduce output records=2999999990
                Spilled Records=16851737032
                Shuffled Maps =2236
                Failed Shuffles=0
                Merged Map outputs=2236
                GC time elapsed (ms)=383451
                Total committed heap usage (bytes)=1199164096512
        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=300009154560
        Total Time Elapsed time on Hadoop : 54000000000000
```

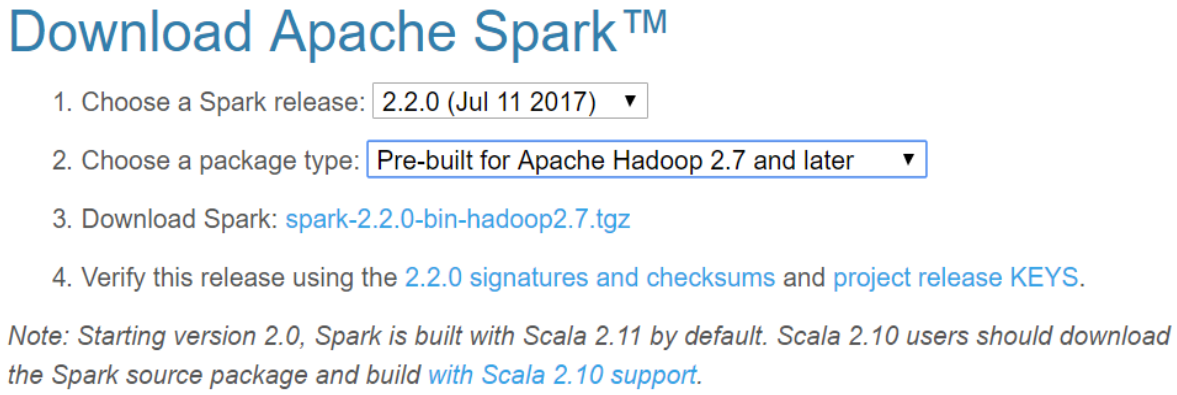
Here the time taken by Hadoop to complete sorting of 1 TB using multi nodes is around 16 hours.

Screenshot of 1TB Output File



Configuring Spark on AWS

1. In a web browser from your local development environment, visit the Apache Spark Download Page. We need to generate a download link which we can access from our EC2 instance.



The Download Link in the 3rd bullet dynamically updates based on your choices.

2. Right Click on the download link and copy it into your clipboard so it can be pasted onto your EC2 instance. From your EC2 instance , type these commands

```
#Download Spark to the ec2-user's home directory
cd ~
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
# Create a symbolic link to make it easier to access
sudo ln -fs spark-2.2.0-bin-hadoop2.7 /opt/spark
```

3. To complete the installation, set the SPARK_HOME environment variable so it takes effect when you login into EC2 instance.

```
# 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.
```

4. You need to reload the environment variables

```
# Reload environment variables
source ~/.bash_profile
```

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

5. Spark is not shy about presenting reams of informational output to the console as it runs. To make the output of your applications easier to spot, you can optionally set up Log4j configuration file and suppress some of the spark logging output. You should maintain logging at the INFO level during the Sparkour tutorials, as we will review some information such as Master URLs in the log files.

   # Create a Log4J configuration file from the provided template.
   cp $SPARK_HOME/conf/log4j.properties.template
   $SPARK_HOME/conf/log4j.properties
   vi $SPARK_HOME/conf/log4j.properties
   # (on line 19 of the file, change the log level from INFO to ERROR)
   # (Note that this will suppress some of the output needed in the tutorials)
   # log4j.rootCategory=ERROR, console
   # Save the file and exit the text editor.

Configuring Spark on a Single Node

Here we are configuring the .bashrc file

```
# 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
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_INSTALL=/sapto/hadoop-2.7.4
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
export SPARK_HOME=/sapto/spark-2.2.0-bin-hadoop2.7
export PATH=$PATH:$SPARK_HOME/bin
```

Now we will configure xml file

```
<configuration>
<property>
 <name>hadoop.tmp.dir</name>
 <value>/home/ubuntu/hadooptmp/hadoop-${user.name}</value>
 <description>A base for other temporary directories.</description>
</property>
<property>
 <name>fs.default.name</name>
 <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

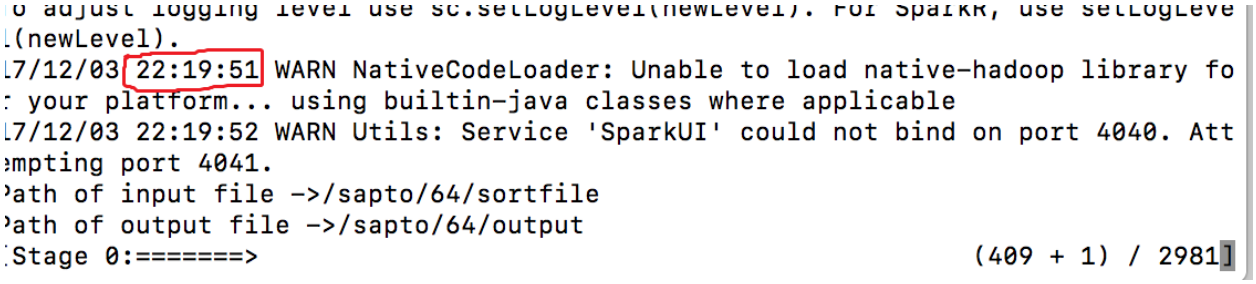
Configuring the HDFS file

Here we have kept the dfs replication 1

```
<configuration>
<property>
 <name>dfs.replication</name>
 <value>1</value>
</property>
<property><name>dfs.name.dir</name>
 <value>file:///sapto/hadoopdata/hdfs/namenode</value>
</property>
<property>
 <name>dfs.data.dir</name>
 <value>file:///sapto/hadoopdata/hdfs/datanode</value>
</property>
</configuration>
```

Spark Single Node 128GB

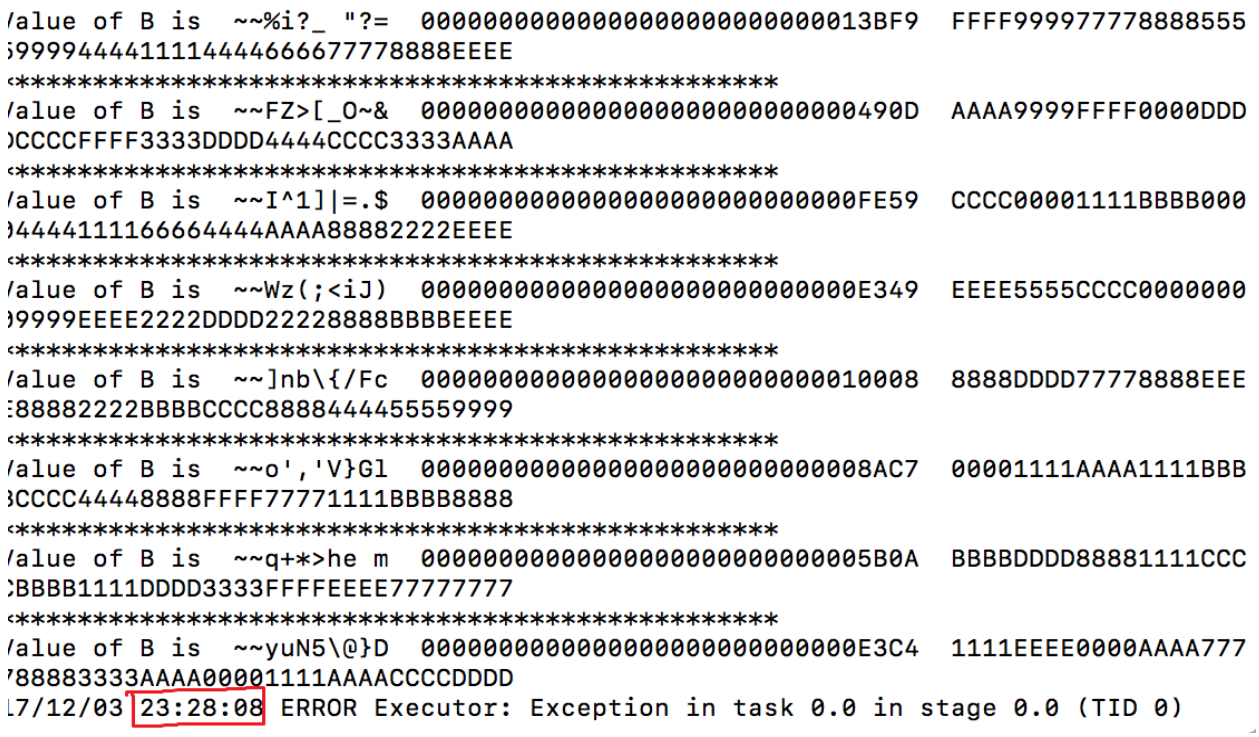
Here, we are starting the Spark Job, you can see the start time of Spark in the image



Spark Output file for 128GB

```
**************************************************
Value of B is   "v*NX#W+/  00000000000000000000000001D7B1  44446666EEEE8888999
9BBBB666699998888777888811116666
**************************************************
Value of B is   "v4<73S,u  00000000000000000000000434898  777788880000EEEE222
2AAAA8888DDDDEEEEBBBB111188889999
**************************************************
Value of B is   "v8%kmcD   00000000000000000000000475597  9999FFFF44447777999
900004444222299990DDDDAAAAEEEE8888
**************************************************
Value of B is   "v:BDC9aP  00000000000000000000000003BA1DE  666677779999AAAA888
86666EEEECCCC66668888333311113333
**************************************************
Value of B is   "v:e<jPY~  00000000000000000000000007C5502  222200001111BBBB666
633330000EEEE3333CCCC3333DDDDFFFF
**************************************************
Value of B is   "v;hAL&t-  00000000000000000000000036C993  BBBBFFFF2222CCCCFFF
F777733339999777000OBBBB8888CCCC
**************************************************
Value of B is   "v>_c+d}U  00000000000000000000000008817B0  5555000088889999444
4777766669999DDDD5555FFFF99991111
**************************************************
Value of B is   "vC.Dg<OJ  00000000000000000000000033DD7F  8888222277778888CCC
CDDDDCCCCCCCC8888EEEE222288880000
```

In this image, you can see the ending time of Spark

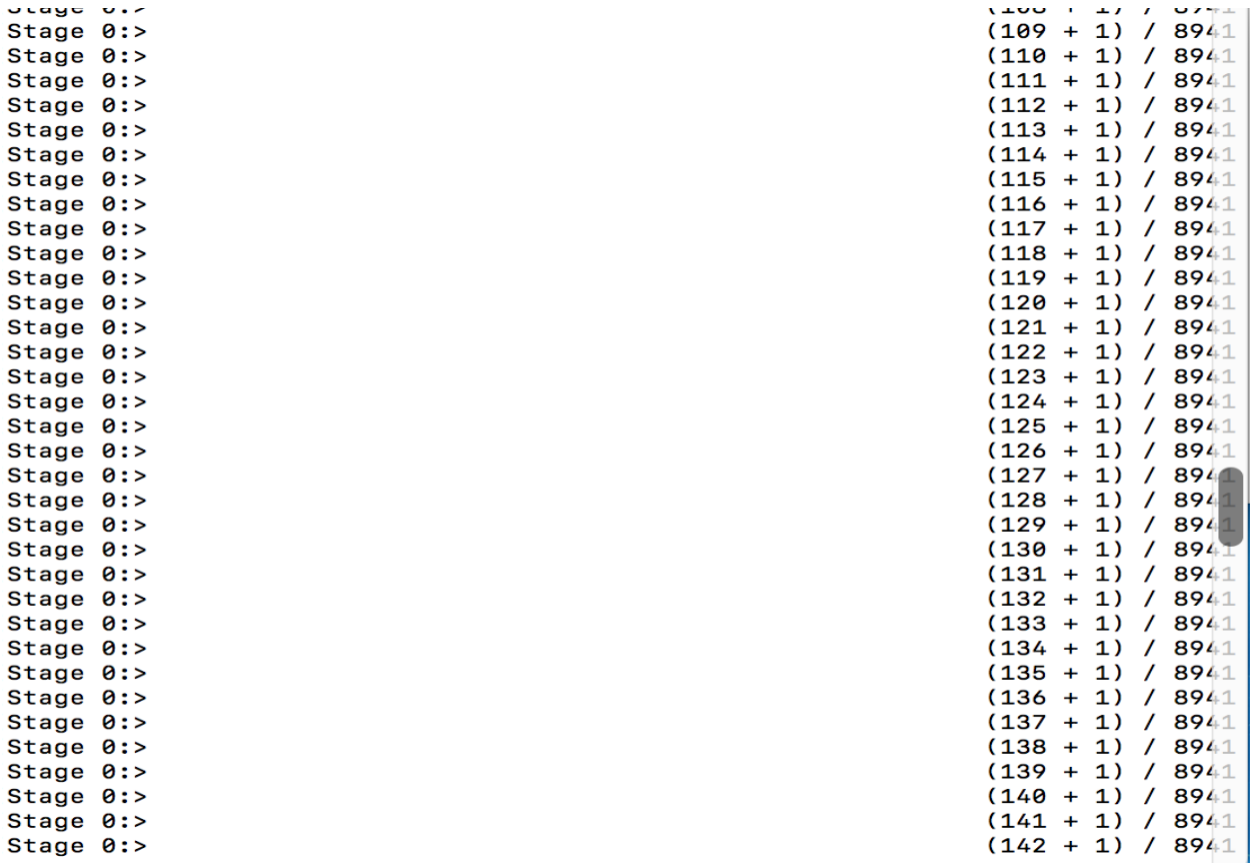


Total time taken for spark to sort 128 GB data is 1.2 Hours

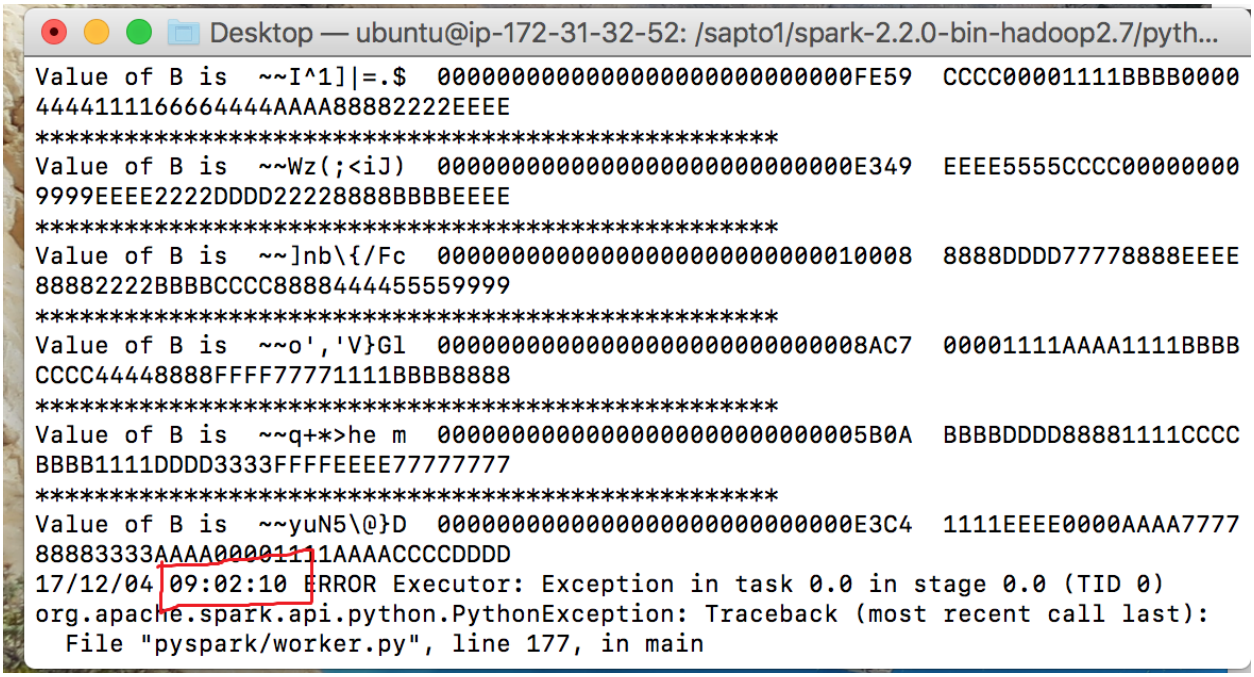
Spark Single Node 1TB

Here, we are starting the Spark Job, you can see the start time of Spark in the image.

```
Jsing Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLev
el(newLevel).
17/12/04 01:15:07 WARN NativeCodeLoader: Unable to load native-hadoop library f
or your platform... using builtin-java classes where applicable
17/12/04 01:15:09 WARN Utils: Service 'SparkUI' could not bind on port 4040. At
tempting port 4041.
Path of input file ->/sapto1/64/sortftb
Path of output file ->/sapto1/output.txt
[Stage 0:>                                                          (0 + 0) / 8941
[Stage 0:>                                                          (0 + 1) / 8941
[Stage 0:>                                                          (1 + 1) / 8941
[Stage 0:>                                                          (2 + 1) / 8941
[Stage 0:>                                                          (3 + 1) / 8941
[Stage 0:>                                                          (4 + 1) / 8941
[Stage 0:>                                                          (5 + 1) / 8941
]
```



In this image, you can see the ending time of Spark, and Output file of Spark



Total time taken is by spark to sort 1TB data is 9 Hours

Configuring Spark on a Multi Node

```
Desktop — root@ip-172-31-46-168: /sapto/spark-2.2.0-bin-hadoop2.7 — ssh...
l(newLevel).
l7/12/03 12:58:27 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
l7/12/03 12:58:28 WARN Utils: Service 'SparkUI' could not bind on port 4040. Att
empting port 4041.
l7/12/03 12:58:35 WARN ObjectStore: Version information not found in metastore.
hive.metastore.schema.verification is not enabled so recording the schema versio
n 1.2.0
l7/12/03 12:58:35 WARN ObjectStore: Failed to get database default, returning No
SuchObjectException
l7/12/03 12:58:36 WARN ObjectStore: Failed to get database global_temp, returnin
g NoSuchObjectException
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.0
      /_/

Using Python version 2.7.12 (default, Nov 20 2017 18:23:56)
SparkSession available as 'spark'.
>>>
[2]+  Stopped                 pyspark
root@ip-172-31-46-168:/sapto/spark-2.2.0-bin-hadoop2.7#
```

Here we are configuring the bashrc file

```
# 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
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_INSTALL=/sapto/hadoop-2.7.4
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
export SPARK_HOME=/sapto/spark-2.2.0-bin-hadoop2.7
export PATH=$PATH:$SPARK_HOME/bin
```

Now we will configure xml file

```
<configuration>
<property>
 <name>fs.default.name</name>
 <value>hdfs://master:9001</value>
</property>
</configuration>
```

Configuring the HDFS file

Here we have kept the dfs replication 1

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.permissions</name>
<value>false</value>
</property>
</configuration>
```

Configuring the YARN file

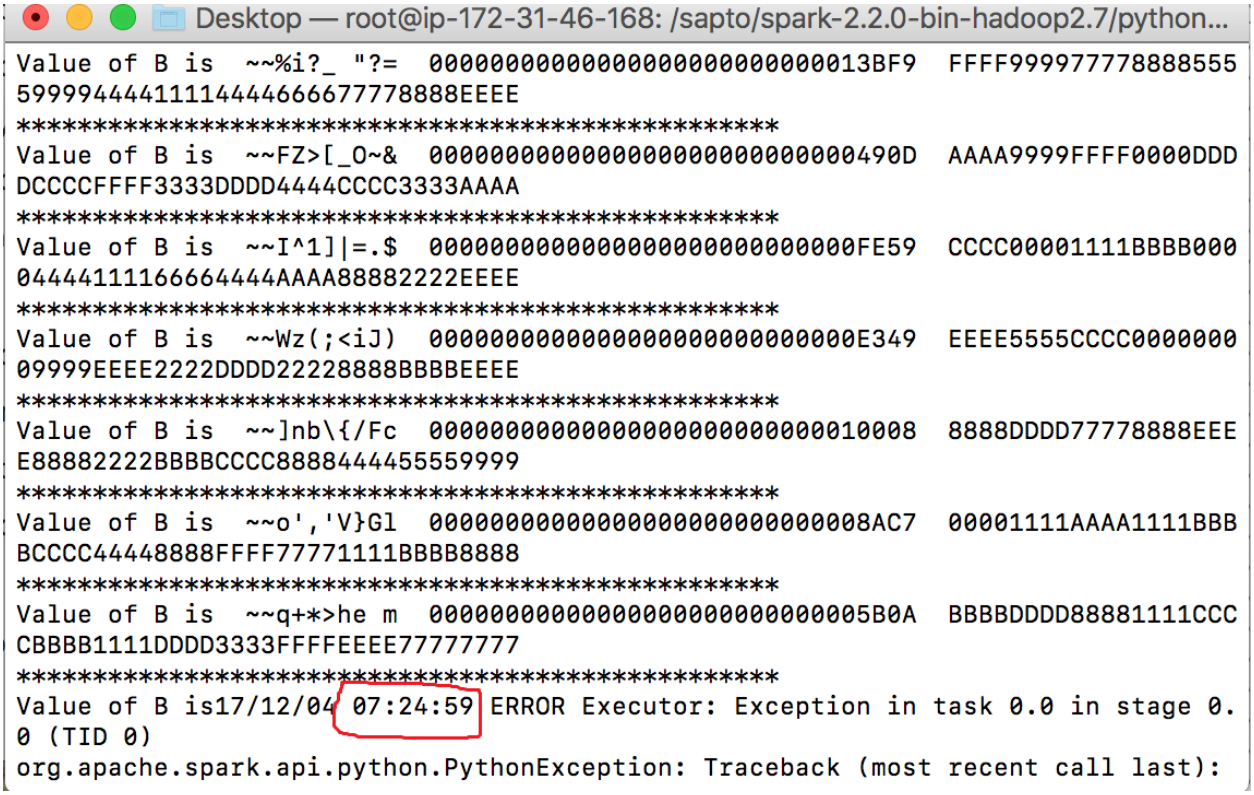
```
<?xml version="1.0"?>
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value>master:8038</value>
</property>
<property>
<name>yarn.resourcemanager.address</name>
<value>master:8039</value>
</property>
<property>
<name>yarn.resourcemanager.webapp.address</name>
<value>master:8086</value>
</property>
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>master:8037</value>
</property>
<property>
<name>yarn.resourcemanger.admin.address</name>
<value>master:8033</value>
</property>
</configuration>
```

Spark Multi Node 1TB

Here, we are starting the spark job, you can see the starting time of spark in the image

```
root@ip-172-31-46-168:/sapto/spark-2.2.0-bin-hadoop2.7/python# python spark_sort
.py /sapto/64/sortfile /sapto/output.txt
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
17/12/04 00:52:31 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
17/12/04 00:52:32 WARN Utils: Service 'SparkUI' could not bind on port 4040. Att
empting port 4041.
17/12/04 00:52:32 WARN Utils: Service 'SparkUI' could not bind on port 4041. Att
empting port 4042.
Path of input file ->/sapto/64/sortfile
Path of output file ->/sapto/output.txt
[Stage 0:>                                                          (5 + 1) / 2981]
```

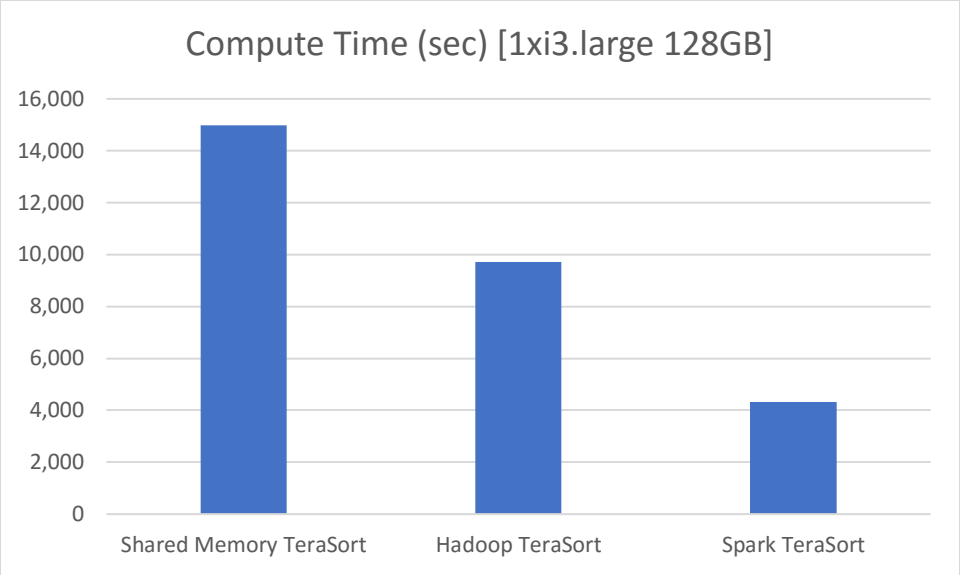
In this image you can see the ending time of spark.



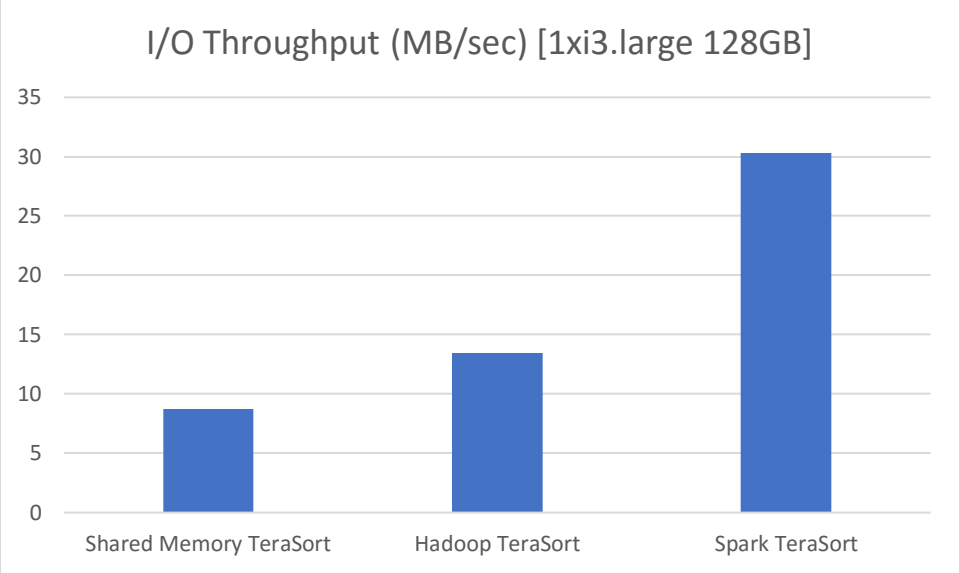
Total time taken by spark to sort 1 TB data in multimode is around 7 hours from the starting and ending time.

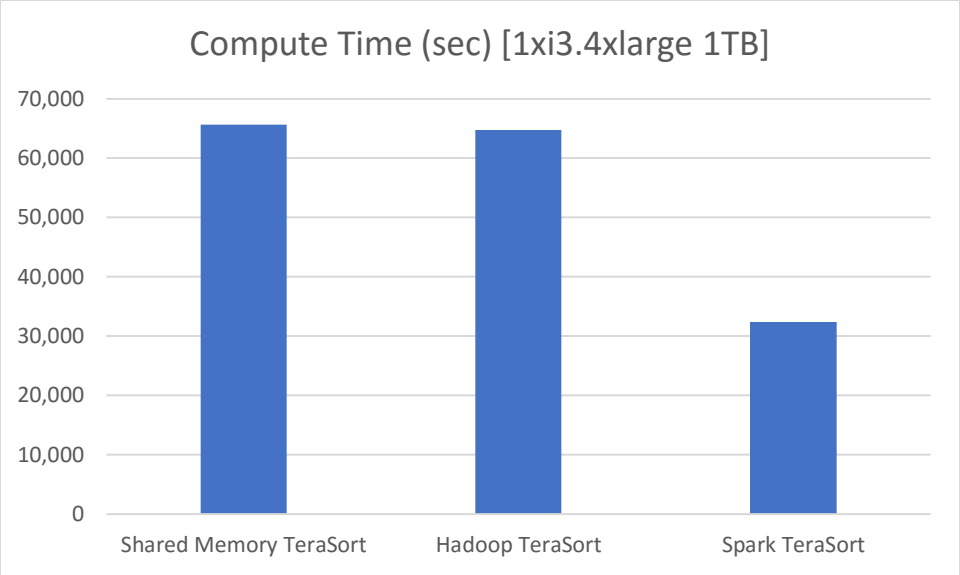
PERFORMANCE EVALUATION

Experiment (instance/dataset)	Shared Memory Tera Sort	Hadoop Tera Sort	Spark Tera Sort
Compute Time (sec) [1xi3.large 128GB]	14,981	9720	4320
Data Read (GB) [1xi3.large 128GB]	128	128	128
Data Write (GB) [1xi3.large 128GB]	128	124	122
I/O Throughput (MB/sec) [1xi3.large 128GB]	8.74	13.48	30.3
Compute Time (sec) [1xi3.4xlarge 1TB]	65,712	64800	32400
Data Read (GB) [1xi3.4xlarge 1TB]	1,024	1,024	1,024
Data Write (GB) [1xi3.4xlarge 1TB]	1,024	988	975
I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]	15.95	15.80	31.60
Compute Time (sec) [8xi3.large 1TB]	N/A	57,600	28,800
Data Read (GB) [8xi3.large 1TB]	N/A	1,024	1,024
Data Write (GB) [8xi3.large 1TB]	N/A	988	975
I/O Throughput (MB/sec) [8xi3.large 1TB]	N/A	17.77	35.5
Speedup (Weak Scale)	The speedup goes down as the number of thread increases. It is not affected for thread 2,3,4 but it significantly goes down for 7,8 threads.	2.2Mb/sec With increase from 1 node to 8 nodes, we observe that the speedup increases by 2.2 Mb/sec for Hadoop	5.2Mb/sec With increase from 1 node to 8 nodes, we observe that the speedup increases by 5.2Mb/sec
Efficiency (Weak Scale)	Efficiency is affected in same manner, down for more thread used.	Efficiency increases by 13.2%	Efficiency increases by 17.7%

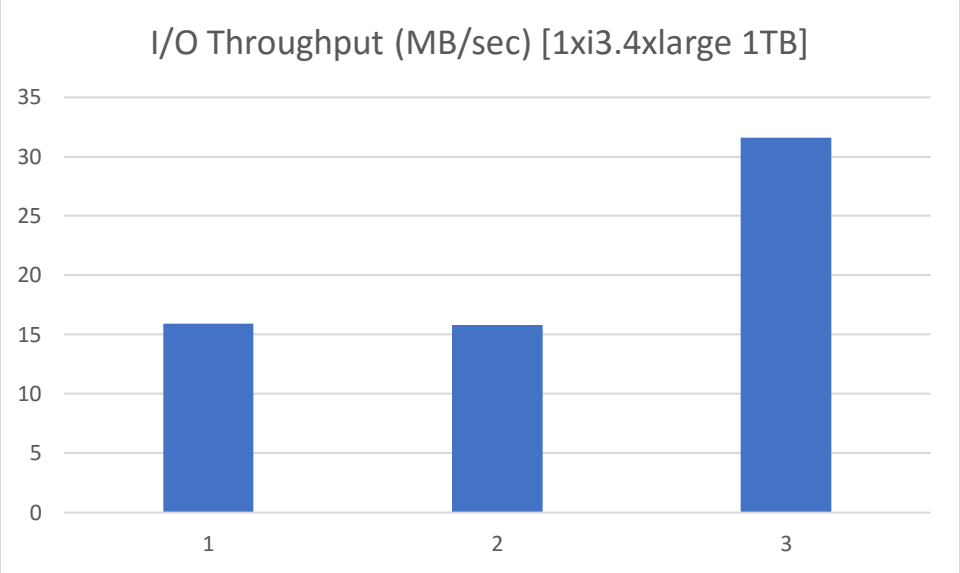


The Graph clearly shows Spark is most efficient in sorting big volumes of data

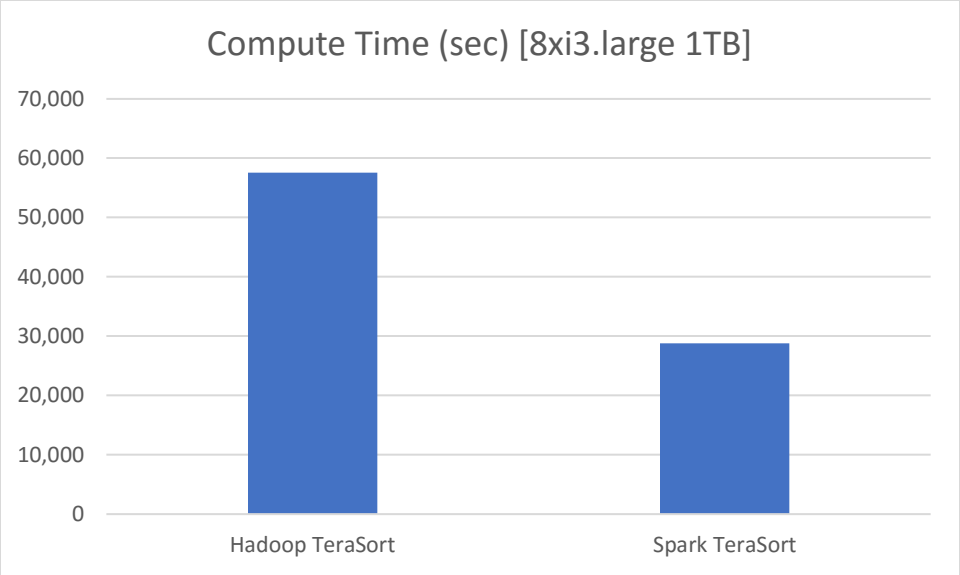




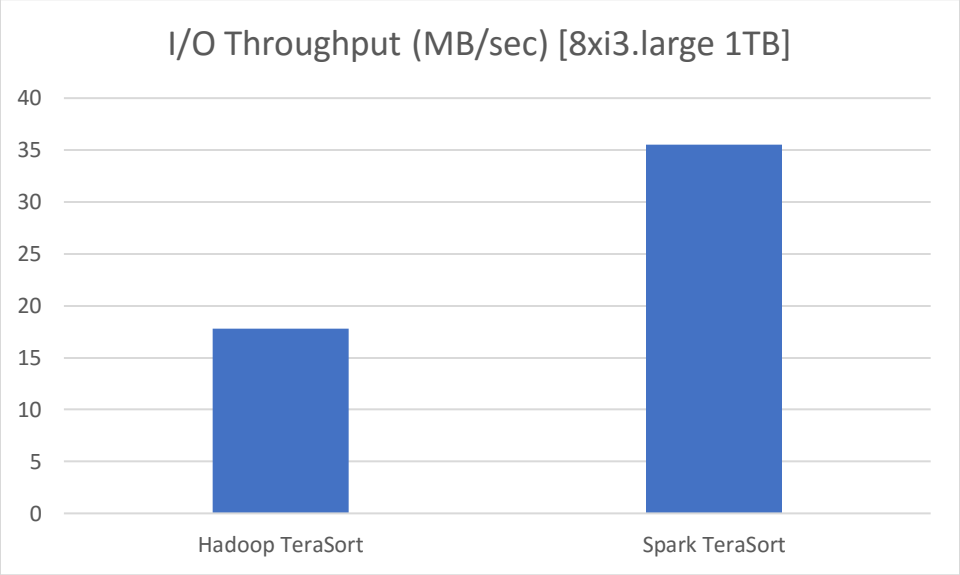
Even for 1TB spark comes out to be the best, Hadoop second best and shared last.



Graph for Multi-node(Hadoop and Spark)



Also in the Multi-node 1TB sorting, spark is better



Conclusion

So, we draw the conclusion that Apache spark is the best of all for sorting big volumes of data. It is much faster and efficient than Hadoop. Hadoop is much better than Shared memory for sorting large volumes of data.

Spark seems to be the best for 1 node scale with a throughput of 30.3 Mb/Sec compared 13.48 Mb/sec and 8.74 Mb/sec for Hadoop and shared respectively.

In 8 Nodes, we see that spark is better and achieves a slightly better efficiency than Hadoop. According to me for 100 node scales or 1000 node scale if we use both apache Hadoop and spark together we will get the best results.

Our benchmarking closely aligns with the result with the winners using apache Hadoop and Spark. It is clearly evident apache spark is the winner. It is 100 times faster than Hadoop and very efficient.

All the big companies are now backing to use apache spark to perform big data operations.

References

- http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/
- https://www.edureka.co/blog/install-hadoop-single-node-hadoop-cluster
- http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/
- http://grepcode.com/file/repo1.maven.org/maven2/org.apache.hadoop/hadoop-mapreduce-examples/2.6.0/org/apache/hadoop/examples/terasort/TeraGen.java
- https://github.com/hparik11/TeraSort_on_Hadoop-Spark/blob/master/Code/SortHadoop.java
- https://github.com/facebookarchive/hadoop-20/blob/master/src/examples/org/apache/hadoop/examples/terasort/TeraGen.java
- https://github.com/kunaldhande/Projects/blob/master/CS553%20-%20TeraSort/SourceCode/hadoop/HadoopSort.java
- https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html
- http://www.chinacloud.cn/upload/2014-01/14010410467139.pdf
- http://www.dreamincode.net/forums/topic/223614-how-to-create-a-shell-script-to-compile-and-execute-a-java-file/
- https://stackoverflow.com/questions/14211076/reading-shared-memory-data-using-java-that-is-written-by-c
- https://github.com/shalinc/Cloud-Sorting-Large-Dataset/blob/master/Code/SharedMemoryScript.sh
- http://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/setup-install.html
- http://www.ordinal.com/gensort.html