CS553 Cloud Computing

TeraSort on Hadoop/Spark

Shruti Gupta (A20381966)

Rohan Borde (A20375497)

Sagar Mane (A20379756)

Introduction:

This programming assignment covers the TeraSort application implemented in different ways: Java, Hadoop, Spark, and MPI. We creating sorting application which could read a large file and sort it in place We created 2 datasets, a small and a large dataset, which we will use to benchmark the different approaches to sorting: 128GB dataset and 1TB dataset.

For all benchmark, we experienced on Amazon EC2, 1-node i3.large, 1-node i3.4xlarge, 8-nodes i3.large.

System configuration for i3.large instance

Processor	High Frequency Intel Xeon E5-2686 v4 (Broadwell) Processors with base frequency of 2.3 GHz
vCPU	2
Mem (GiB)	15.25
Networking performance	Up to 10 Gigabit
Storage (tb)	1 x 0.475 NVMe SSD

System configuration for i3.4xlarge instance

Processor	High Frequency Intel Xeon E5-2686 v4 (Broadwell) Processors with base frequency of 2.3 GHz
vCPU	16
Mem (GiB)	122
Networking performance	Up to 10 Gigabit
Storage (TB)	2 x 1.9 NVMe SSD

Methodology

Software version

- Java – OpenJdk8
- Hadoop – Hadoop-2.7.4
- Spark – spark-2.2.0
- MPI – MPICH2 Plugin (http://star.mit.edu/cluster/docs/0.93.3/plugins/mpich2.html)
- Linux version:- Ubuntu 16.04 LTS
- Scala version: scala-2.11.6

Share-Memory

In Share-Memory Tera-sort application sorted in memory.

Implementation:

Sorting:

We are processing large dataset like 128GB and 1TB so this dataset is not fir inside the memory. For sorting this large dataset, we divide the file into small chunks of size, and process this small chunks in memory. We sort all small chunk block and store in temporary file for further processing.

Merging:

In this, we are using sorted temporary files, and merge all small files for creating one large sort file. We are using merge sort algorithm for implementing this.

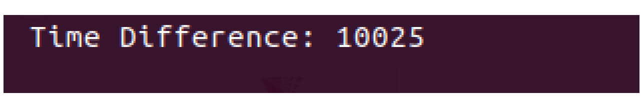
Observation:

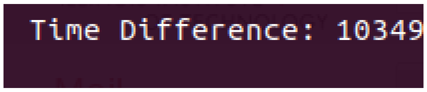
Thread	**Time(sec)**
1	**9915**
2	**10025**
4	**10349**
8	**10596**

We run the shar-memory application for 128GB data size and 1TB data size using different threads like 1 2 4 and 8. We observed that as we increase the numbers of threads, the time require for sorting all big data file is almost constant for. For i3.large, number of cores available are 2, so when the share-memory application run for 1 thread, it will utilizes all two core of processor for implementation. So, time taken by 1 thread is less. For 2 thread all work is distributed among all 2 cores but for thread 4 and 8, because of limitation of number of cores the time required to sort large file is almost same.

Screenshots:

1. 

2. 

3. 

4. 

Apache Hadoop:

Apache Hadoop is an open-source software framework used for distributed storage and processing of dataset of big data using the MapReduce programming model. Two components in Hadoop are HDFS and YARN.

HDFS create an abstraction of resources which allows store data of various format across a cluster. YARN, is processing unit of Hadoop. In HDFS, Namenode is a master node and Datanodes are slaves. Namenode contains the metanode about data stored in Data nodes.

Namenode:

- Master daemon
- Maintains and Manages DataNodes
- Records metadata
- Receive heartbeat and block report from all the Datanodes.

DataNode:

- Slave daemons
- Store actual data
- Serves read and write request

Implementation:

Map Phase:

File stored in HDFS are used by this phase as input and process that data for generating tuples represented as key value pairs. Main responsibility of Map phase is to generating <key,value> pair which can be used by reducer phase for further implementation.

Reduce Phase:

<kay,value> pair generated by Map phase is used by this phase as input. This phase combine similar keys and generating smaller dataset which can be stored in HDFS.

Observation:

Hadoop Sorting with i3.large and i3.4xlarge for 128GB and 1TB data size on 1 Node and 8 Node. We observed that as node increase from 1 to 8, the more amount parallelism is achieved in Hadoop, which result in the scale up the performance of sorting on large dataset.

Apache Spark

Apache spark is a fast and general engine for large-scale data processing. It is open-source cluster computing framework for real-time processing. Spark has ability to run on top of an existing Hadoop cluster using YARN. Spark can use Hadoop for storage and processing. Spark can perform cluster in-memory and it has its own cluster management computation, it uses Hadoop for storing purpose only.

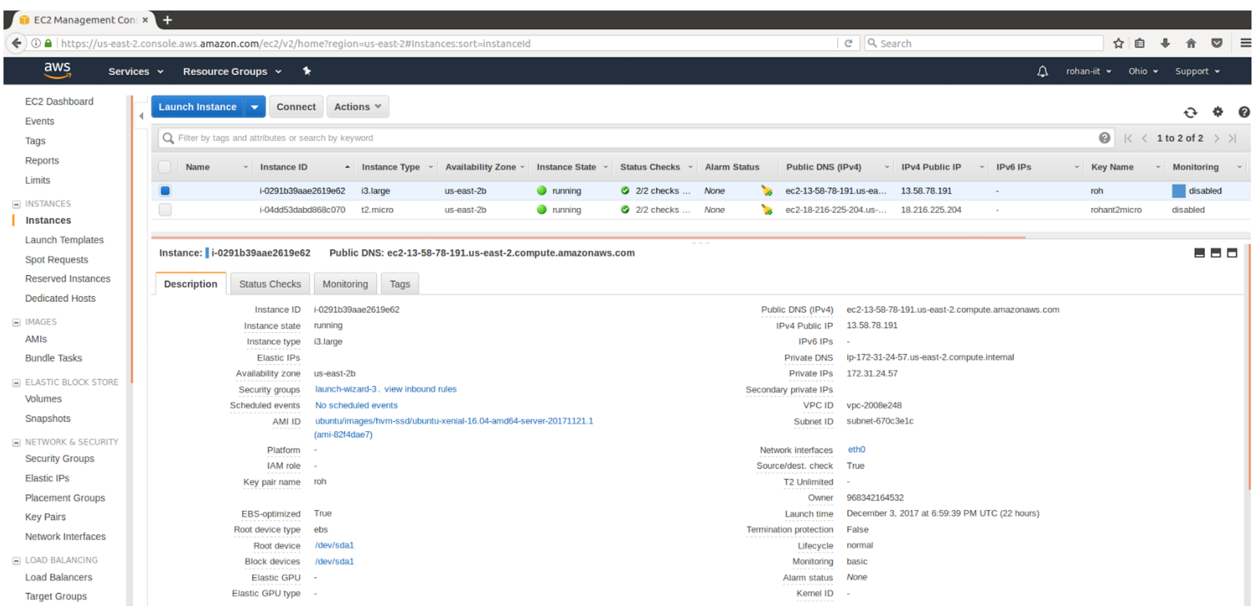
Resilient Distributed Dataset (RDD):

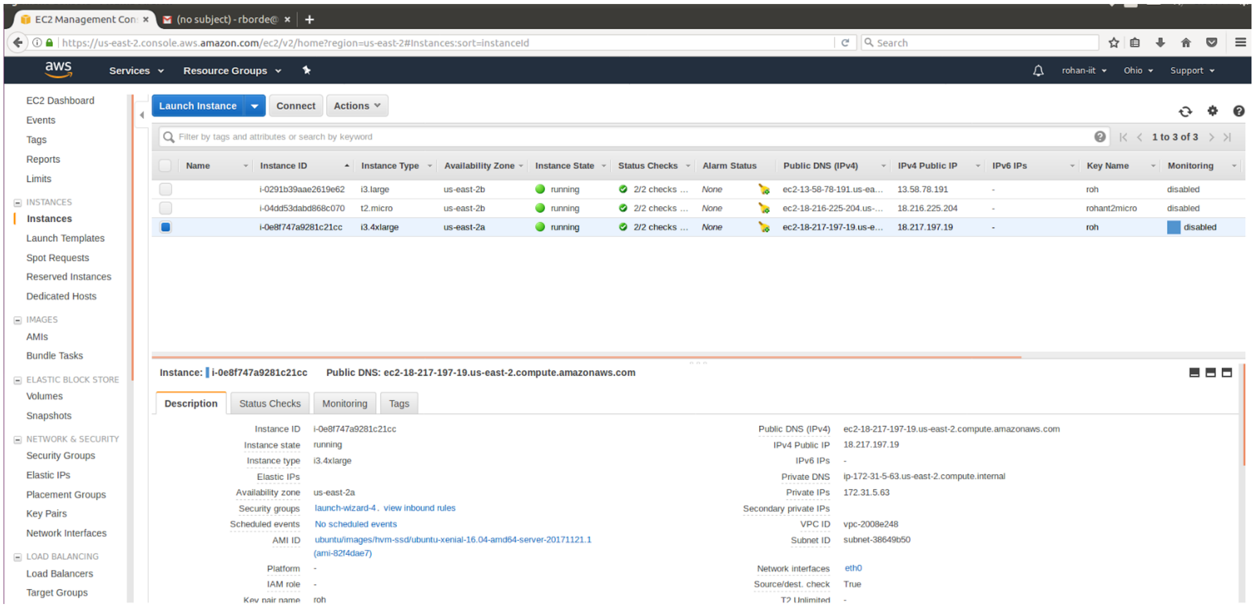
RDD is data structure of spark. It is a collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of cluster. It contains any type of Python, Java, or Scala objects.

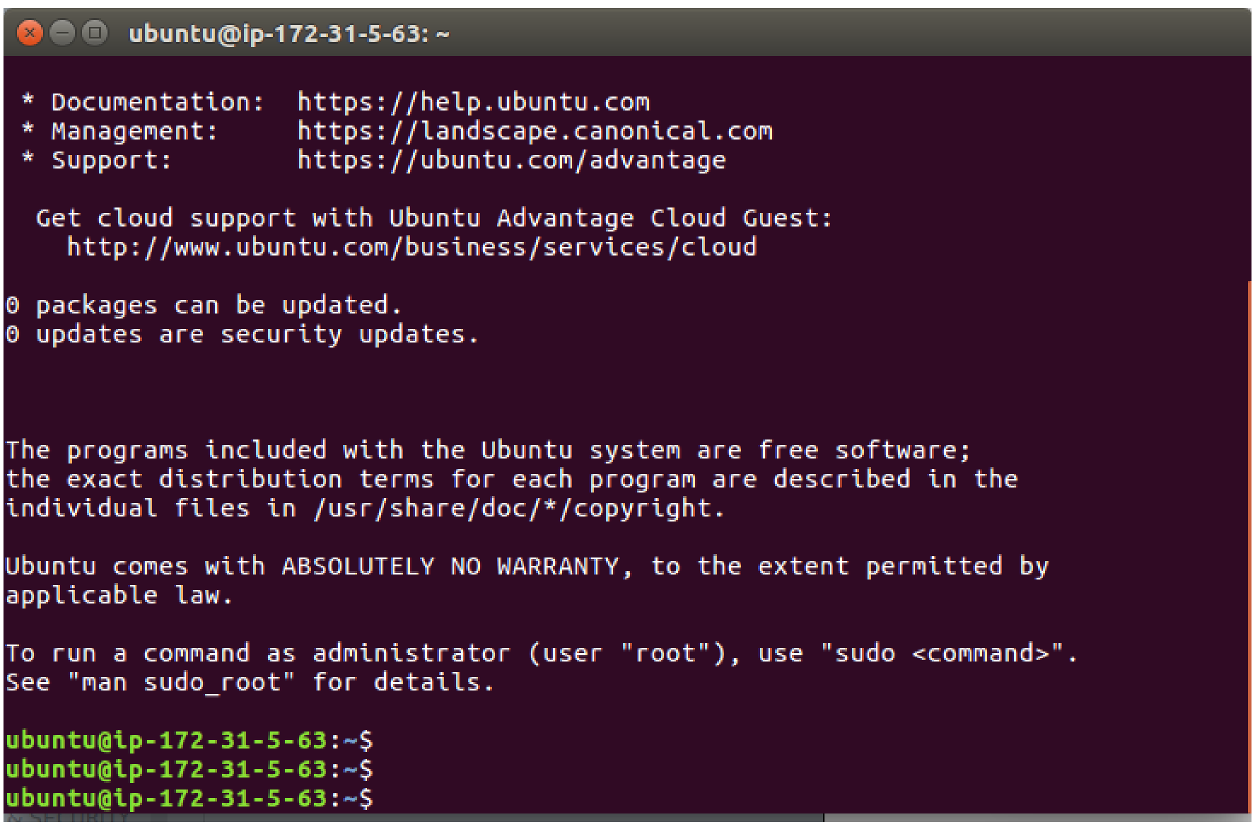
Observation:

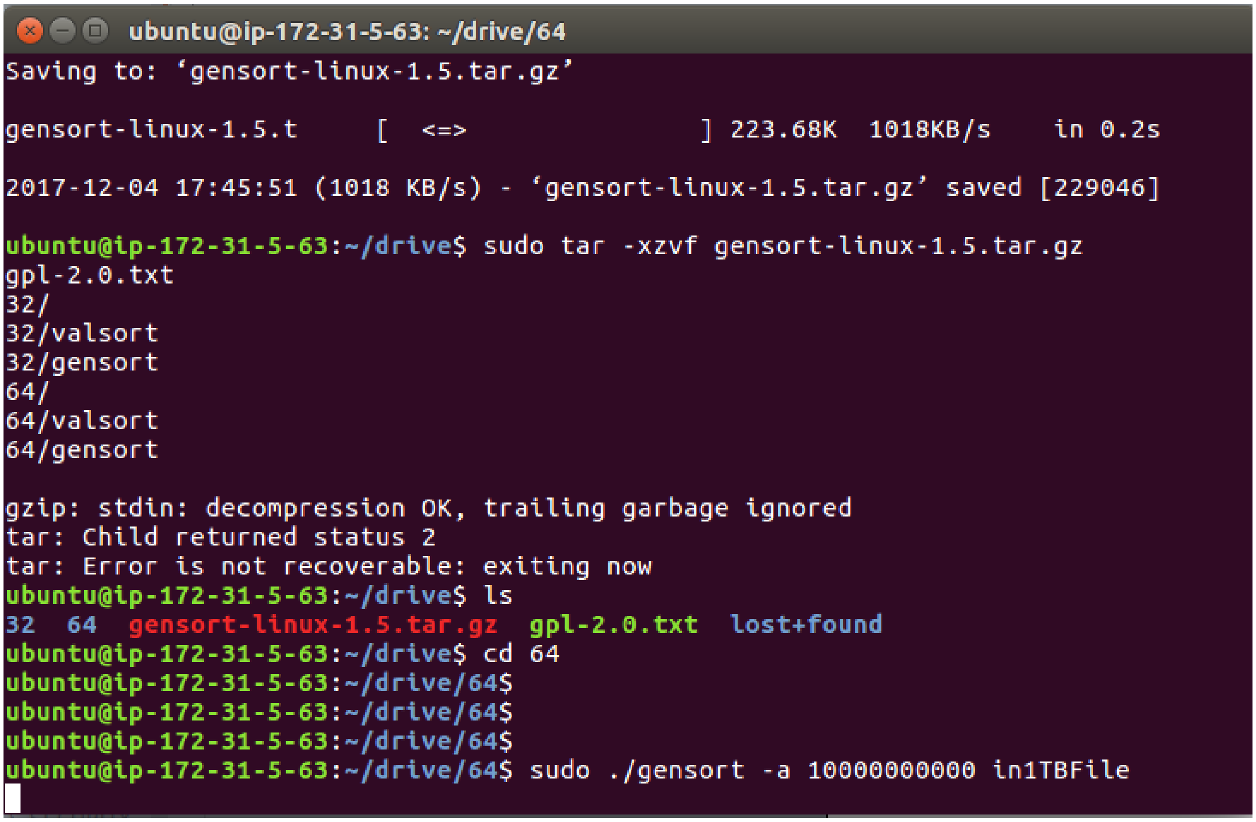
Spark sorting on 1 node take more time as compare to sorting on 8 node as shown in performance table.

Apache Hadoop and spark screenshot:




EC2 Management Con
(no subject) - rborde@
https://us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances:sort=instanceId
Services
Resource Groups
rohan-iit
Ohio
Support
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Launch Templates
Spot Requests
Reserved Instances
Dedicated Hosts
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
LOAD BALANCING
Load Balancers
Target Groups
Launch Instance
Connect
Actions
1 to 3 of 3
Name
Instance ID
Instance Type
Availability Zone
Instance State
Status Checks
Alarm Status
Public DNS (IPv4)
IPv4 Public IP
IPv6 IPs
Key Name
Monitoring
i-0291b39aae2619e62 i3.large us-east-2b running 2/2 checks ... None ec2-13-58-78-191.us-ea... 13.58.78.191 - roh disabled
i-04dd53dabd868c070 t2.micro us-east-2b running 2/2 checks ... None ec2-18-216-225-204.us-... 18.216.225.204 - rohant2micro disabled
i-0e8f747a9281c21cc i3.4xlarge us-east-2a running 2/2 checks ... None ec2-18-217-197-19.us-e... 18.217.197.19 - roh disabled
Instance: i-0e8f747a9281c21cc Public DNS: ec2-18-217-197-19.us-east-2.compute.amazonaws.com
Description
Status Checks
Monitoring
Tags
Instance ID i-0e8f747a9281c21cc
Instance state running
Instance type i3.4xlarge
Elastic IPs
Availability zone us-east-2a
Security groups launch-wizard-4. view inbound rules
Scheduled events No scheduled events
AMI ID ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20171121.1 (ami-82f4dae7)
Platform -
IAM role -
Key pair name roh
Public DNS (IPv4) ec2-18-217-197-19.us-east-2.compute.amazonaws.com
IPv4 Public IP 18.217.197.19
IPv6 IPs -
Private DNS ip-172-31-5-63.us-east-2.compute.internal
Private IPs 172.31.5.63
Secondary private IPs
VPC ID vpc-2008e248
Subnet ID subnet-38649b50
Network interfaces eth0
Source/dest. check True
T2 Unlimited -


ubuntu@ip-172-31-5-63: ~
* 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
0 packages can be updated.
0 updates are security updates.
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
ubuntu@ip-172-31-5-63:~$
ubuntu@ip-172-31-5-63:~$
ubuntu@ip-172-31-5-63:~$


ubuntu@ip-172-31-5-63: ~/drive/64
Saving to: ‘gensort-linux-1.5.tar.gz’
gensort-linux-1.5.t [<=>] 223.68K 1018KB/s in 0.2s
2017-12-04 17:45:51 (1018 KB/s) - ‘gensort-linux-1.5.tar.gz’ saved [229046]
ubuntu@ip-172-31-5-63:~/drive$ sudo tar -xzvf gensort-linux-1.5.tar.gz
gpl-2.0.txt
32/
32/valsort
32/gensort
64/
64/valsort
64/gensort
gzip: stdin: decompression OK, trailing garbage ignored
tar: Child returned status 2
tar: Error is not recoverable: exiting now
ubuntu@ip-172-31-5-63:~/drive$ ls
32 64 gensort-linux-1.5.tar.gz gpl-2.0.txt lost+found
ubuntu@ip-172-31-5-63:~/drive$ cd 64
ubuntu@ip-172-31-5-63:~/drive/64$
ubuntu@ip-172-31-5-63:~/drive/64$
ubuntu@ip-172-31-5-63:~/drive/64$
ubuntu@ip-172-31-5-63:~/drive/64$ sudo ./gensort -a 10000000000 in1TBFile

```
ubuntu@ip-172-31-5-63: ~
nvme1n1 259:0    0  1.7T  0 disk
ubuntu@ip-172-31-5-63:~$ cd drive
ubuntu@ip-172-31-5-63:~/drive$
ubuntu@ip-172-31-5-63:~/drive$ ls
32  64  gensort-linux-1.5.tar.gz  gpl-2.0.txt  lost+found
ubuntu@ip-172-31-5-63:~/drive$ cd 64
ubuntu@ip-172-31-5-63:~/drive/64$
ubuntu@ip-172-31-5-63:~/drive/64$ ls
gensort  in1TBFile  valsort
ubuntu@ip-172-31-5-63:~/drive/64$ ls -l
total 976562864
-rwxrwxr-x 1  500  500         141045 Mar 17  2013 gensort
-rwxr-xr-x 1 root root 1000000000000 Dec  4 18:09 in1TBFile
-rwxrwxr-x 1  500  500         134558 Mar 17  2013 valsort
ubuntu@ip-172-31-5-63:~/drive/64$ ls -l --block-size=M
total 953675M
-rwxrwxr-x 1  500  500      1M Mar 17  2013 gensort
-rwxr-xr-x 1 root root 953675M Dec  4 18:09 in1TBFile
-rwxrwxr-x 1  500  500      1M Mar 17  2013 valsort
ubuntu@ip-172-31-5-63:~/drive/64$
ubuntu@ip-172-31-5-63:~/drive/64$
ubuntu@ip-172-31-5-63:~/drive/64$
ubuntu@ip-172-31-5-63:~/drive/64$ cd
ubuntu@ip-172-31-5-63:~$
ubuntu@ip-172-31-5-63:~$
ubuntu@ip-172-31-5-63:~$
```

```
ubuntu@ip-172-31-5-63: ~
scala-2.11.6/doc/licenses/mit_jquery-layout.txt
scala-2.11.6/doc/licenses/mit_jquery.txt
scala-2.11.6/doc/licenses/bsd_jline.txt
scala-2.11.6/doc/License.rtf
scala-2.11.6/lib/
scala-2.11.6/lib/scala-parser-combinators_2.11-1.0.3.jar
scala-2.11.6/lib/scala-reflect.jar
scala-2.11.6/lib/akka-actor_2.11-2.3.4.jar
scala-2.11.6/lib/scala-continuations-library_2.11-1.0.2.jar
scala-2.11.6/lib/config-1.2.1.jar
scala-2.11.6/lib/scalap-2.11.6.jar
scala-2.11.6/lib/scala-xml_2.11-1.0.3.jar
scala-2.11.6/lib/scala-continuations-plugin_2.11.5-1.0.2.jar
scala-2.11.6/lib/scala-actors-migration_2.11-1.1.0.jar
scala-2.11.6/lib/jline-2.12.1.jar
scala-2.11.6/lib/scala-library.jar
scala-2.11.6/lib/scala-compiler.jar
scala-2.11.6/lib/scala-actors-2.11.0.jar
scala-2.11.6/lib/scala-swing_2.11-1.0.1.jar
ubuntu@ip-172-31-5-63:~$ mv scala-2.11.6 /usr/local/scala
mv: cannot move 'scala-2.11.6' to '/usr/local/scala': Permission denied
ubuntu@ip-172-31-5-63:~$ sudo mv scala-2.11.6 /usr/local/scala
ubuntu@ip-172-31-5-63:~$ vi .bashrc
ubuntu@ip-172-31-5-63:~$ scala version
The program 'scala' is currently not installed. You can install it by typing:
sudo apt install scala
ubuntu@ip-172-31-5-63:~$ scala -version
The program 'scala' is currently not installed. You can install it by typing:
sudo apt install scala
ubuntu@ip-172-31-5-63:~$ source ~/.bashrc
ubuntu@ip-172-31-5-63:~$ scala -version
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL
ubuntu@ip-172-31-5-63:~$
```

Spark Processing the jobs:

Below screenshot shows the sorting of 1TB file job in progress for spark.

```
 --- Processing started ---
[Stage 0:==========>                                        (6055 + 16) / 29803]
```

```
Spark session available as 'spark'.
Loading BigDataSortSpark.scala...
import org.apache.spark._
import org.apache.spark.SparkContext._
defined object BigDataSortSpark
 --- Processing started ---
[Stage 1:=========>                                         (2768 + 16) / 29803]
```

```
 --- Processing started ---
[Stage 1:===>                                               (1984 + 16) / 29803]
```

-- Spark Virtual Cluster (1-node i3.large)

```
ubuntu@ip-172-31-24-57:~/drive/output128$ head -10 part-00000
   !4+ABv   0000000000000000000000000017F7E829  EEEE3333444411112222888833334444666633332222DDDDEEEE
   "O!uve   00000000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
   %!$sU(   000000000000000000000000002E6C821C  22223333777744445555111199999CCCC4444EEEEFFFF11115555
   &5rX|X   00000000000000000000000000399BC288  5555CCCCBBBB99999999DDDD111100001111EEEE7777DDDD9999
   'ic%So   0000000000000000000000000031F06B7D  EEEEBBBBAAAA8888DDDDDDDD77772222444411116666 4444AAAA
   *0G1Io   000000000000000000000000003B5E85A1  1111AAAA9999CCCCBBBB11119999111133339999 1111AAAA6666
   ,(GhT_   000000000000000000000000002D0172DC  1111CCCC1111DDDDCCCCEEEE9999CCCC8888CCCCFFFF55555555
   0*vYm3   0000000000000000000000000026D61578  DDDD7777AAAAEEEEEEEE6666AAAA2222CCCC5555555522229999
   2C>)8d   0000000000000000000000000026C79E66  444400001111CCCC6666BBBB555577776666CCCC2222AAAABBBB
   8tU30;   000000000000000000000000003FC4ABD1  77772222DDDD77772222AAAA33336666EEEE88880000FFFF6666
ubuntu@ip-172-31-24-57:~/drive/output128$ tail -10 part-00000
 "V51g`Dn+   000000000000000000000000003F35AA33  4444FFFF111166662222BBBBCCCC9999000022224444EEEECCCC
 "V54{UM}A   000000000000000000000000004AD7B851  8888DDDD33332222BBBBEEEE444444445555EEEE888877776666
 "V58URG[e   00000000000000000000000000340776A3  9999DDDDAAAAAAAABBBBBBBB7777DDDD7777333355557777CCCC
 "V5B4Np<>   0000000000000000000000000042693915  BBBB7777DDDD0000AAAA9999BBBBCCCCAAAABBBB999977772222
 "V5BD21dE   0000000000000000000000000048F1B95E  22227777777711116666AAAA33334444DDDD9999111199993333
 "V5B~Q+9e   00000000000000000000000000000E8447A4  DDDDAAAA8888AAAA6666EEEE7777BBBBAAAA2222DDDD6666DDDD
 "V5WO)?Yc   000000000000000000000000003E7C5EE5  11115555BBBB7777333311114444FFFF0000CCCCBBBB22222222
 "V5[$DY1n   000000000000000000000000003879F566  999933337777FFFF1111444411110000999933337777AAAABBBB
 "V5\]mZ6*   000000000000000000000000004715C93C  EEEE22229999EEEE00004444999966663333CCCC111177775555
 "V5^.ZIgL   000000000000000000000000001D21A47D  DDDDBBBBFFFF00005555FFFF222200001111FFFF55554444AAAA
```

Spark Spark Virtual Cluster (1-node i3.large) output:

```
Spark session available as 'spark'.
Loading BigDataSortSpark.scala...
import org.apache.spark._
import org.apache.spark.SparkContext._
defined object BigDataSortSpark
 --- Processing started ---
Big data sorting time = 7126798 milisec
```

Spark Spark Virtual Cluster (1-node i3.4xlarge) output:

```
Spark session available as 'spark'.
Loading BigDataSortSpark.scala...
import org.apache.spark._
import org.apache.spark.SparkContext._
defined object BigDataSortSpark
 --- Processing started ---
Big data sorting time = 33951113 milisec
```

Spark Spark Virtual Cluster (8-nodes i3.large) output:

```
Spark session available as 'spark'.
Loading BigDataSortSpark.scala...
import org.apache.spark._
import org.apache.spark.SparkContext._
defined object BigDataSortSpark
 --- Processing started ---
Big data sorting time = 15946216 milisec
```

-- hadoop Virtual Cluster (1-node i3.large)

```
ubuntu@ip-172-31-24-57:~/drive/output$ head -10 part-00000
  !&S3/]]   000000000000000000000000002145D78  8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
  !*NXt>o   00000000000000000000000002A00EE08  2222EEEE777766668888BBBB4444AAAA33331111EEEE55559999
  !+cqLJP   000000000000000000000000032148131  77778888FFFF4444AAAA4444FFFF7777AAAA0000AAAA99996666
  !,=U#,9   000000000000000000000000019072E3  33332222FFFFBBBB0000FFFFAAAA666655553333DDDD3333CCCC
  !.*G<W~   00000000000000000000000002FC516F8  6666CCCCBBBB111144446666BBBB33337777BBBBDDDDAAAA9999
  !0f[ITd   0000000000000000000000000003CAAB4B  9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
  !64O>Nx   00000000000000000000000000C297475  BBBB44443333DDDD77771111AAAA8888AAAAAAAA333311112222
  !7h}W4^   00000000000000000000000001B209AE2  EEEEAAAADDDD11115555DDDD3333DDDD5555333355557777FFFF
  !CDYuIp   00000000000000000000000001D0D4CE0  DDDD444400006666BBBB5555FFFFCCCC4444DDDDFFFFAAAA1111
  !F*ykr'   00000000000000000000000002B7E4320  FFFF3333FFFF4444666644448888FFFF1111EEEE000066661111
ubuntu@ip-172-31-24-57:~/drive/output$ tail -10 part-00000
 %+K#`@,T\   0000000000000000000000000034C133EB  4444DDDD44446666FFFFDDDD3333AAAADDDDEEEE999933334444
 %+K%Kl<gT   0000000000000000000000000317E9628  AAAA7777BBBB11116666DDDD77774444AAAABBBB9999BBBB9999
 %+K867@-r   0000000000000000000000000037EBB375  22225555888833331111BBBB44449999FFFFAAAACCCC11112222
 %+K9B^ar%   00000000000000000000000001E251E44  000099992222BBBB8888FFFFEEEE000088889999EEEE4444DDDD
 %+KB}]Q!|   00000000000000000000000000307AC7BD  8888CCCC7777CCCCDDDDEEEEDDDDEEEEDDDDFFFF00000000AAAA
 %+KD DO$Z   000000000000000000000000000426994D9  FFFFAAAADDDDBBBB333322224444000BBBB66665555AAAAEEEE
 %+KHhsjGX   0000000000000000000000000265C3343  6666BBBBFFFF00007777888800002222BBBBDDDDAAAADDDDCCCC
 %+KMn5;'Q   00000000000000000000000003B077DC0  666644448888AAAAFFFF222244444444EEEE7777BBBB44441111
 %+KQ^%zy:   00000000000000000000000002A91A87B  BBBB44443333FFFF7777DDDDBBBBEEEEFFFF66669999AAAA4444
 %+KSqYigf   000000000000000000000000000145249D  AAAA8888BBBB999955551111CCCC77770000777788882222AAAA
```

MPI:

Message passing interface (MPI) is message passing libraries. It is message passing parallel programming model.

MPI Flow

MPI_Init: Initialize MPI execution environment
MPI_Comm_rank: Determine the rank of calling process in the communicator

MPI_Comm_size: Determine the size of group associated with communicator
MPI_Scatter: send the data from one process to other process in a communicator
MPI_Gather: Gather together all values from a group of process
MPI_Barrier: Blocks until all processes in the communicator have reached this routine.
MPI_Finalize: Terminates MPI execution environment

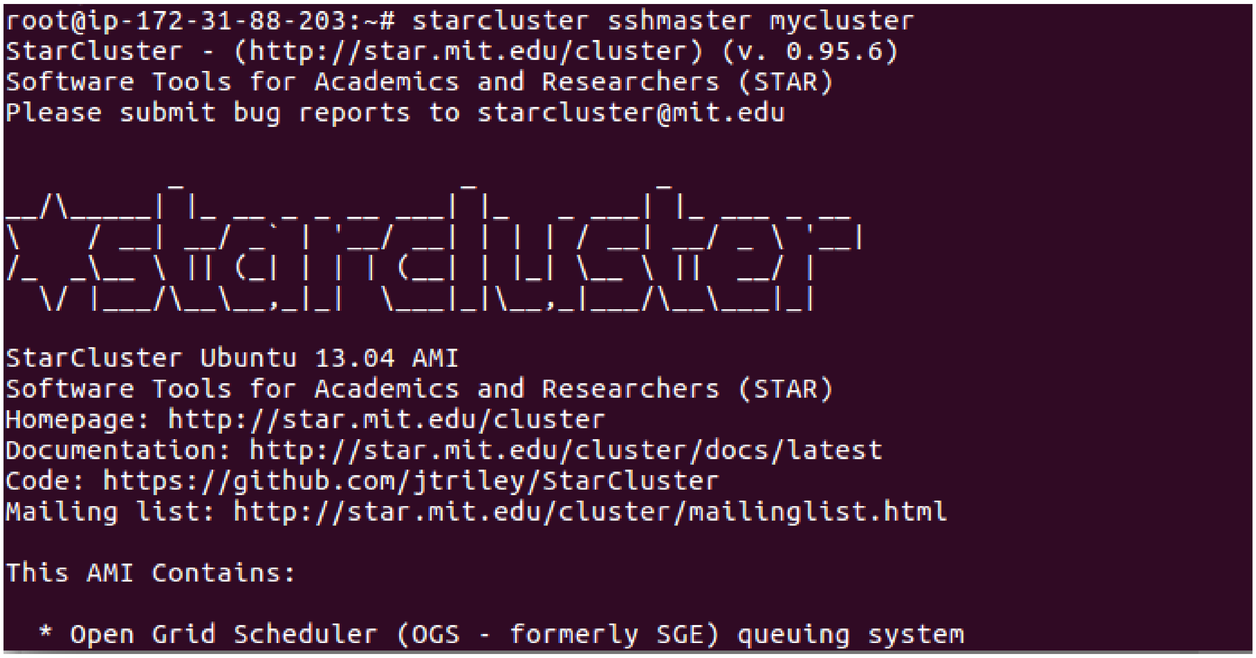
```
ubuntu@ip-172-31-88-203:~$ starcluster help
StarCluster - (http://star.mit.edu/cluster) (v. 0.95.6)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

!!! ERROR - config file /home/ubuntu/.starcluster/config does not exist

Options:
--------
[1] Show the StarCluster config template
[2] Write config template to /home/ubuntu/.starcluster/config
[q] Quit
```

```
root@ip-172-31-88-203:~# starcluster start -x mycluster
StarCluster - (http://star.mit.edu/cluster) (v. 0.95.6)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Validating existing instances...
>>> Validating cluster template settings...
>>> Cluster template settings are valid
>>> Starting cluster...
>>> Waiting for cluster to come up... (updating every 30s)
>>> Waiting for all nodes to be in a 'running' state...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for SSH to come up on all nodes...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for cluster to come up took 0.102 mins
>>> The master node is ec2-34-229-58-204.compute-1.amazonaws.com
>>> Configuring cluster...
>>> Running plugin starcluster.clustersetup.DefaultClusterSetup
>>> Configuring hostnames...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Creating cluster user: sgeadmin (uid: 1001, gid: 1001)
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
```



```
root@ip-172-31-88-203:~# starcluster sshmaster mycluster
StarCluster - (http://star.mit.edu/cluster) (v. 0.95.6)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

StarCluster Ubuntu 13.04 AMI
Software Tools for Academics and Researchers (STAR)
Homepage: http://star.mit.edu/cluster
Documentation: http://star.mit.edu/cluster/docs/latest
Code: https://github.com/jtriley/StarCluster
Mailing list: http://star.mit.edu/cluster/mailinglist.html

This AMI Contains:

  * Open Grid Scheduler (OGS - formerly SGE) queuing system
```

Performance Evaluation:

Experiment Instance/dataset	Shared Memory TeraSort	Hadoop TeraSort	Spark TeraSort	MPI TeraSort
Compute Time (sec) [1xi3.large 128GB]	9915	14450	7126	15616
Data Read (GB) [1xi3.large 128GB]	396	128	67.5	384
Data Write (GB) [1xi3.large 128GB]	396	128	67.5	384
I/O Throughput (MB/sec) [1xi3.large 128GB]	79.88	17.71	18.94	49.18
Compute Time (sec) [1xi3.4xlarge 1TB]	31200	38753	33951	41225
Data Read (GB) [1xi3.4xlarge 1TB]	3095	1000	525.6	3105
Data Write (GB) [1xi3.4xlarge 1TB]	3095	1000	525.6	3105
I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]	198.39	51.60	30.96	150.64

Compute Time (sec) [8xi3.large 1TB]	N/A	20435	15946	22656
Data Read (GB) [8xi3.large 1TB]	N/A	1000	525.6	3105
Data Write (GB) [8xi3.large 1TB]	N/A	1000	525.6	3105
I/O Throughput (MB/sec) [8xi3.large 1TB]	N/A	97.87	65.92	274.09
Speedup (weak scale)	2.543	2.982	1.679	3.03
Efficiency (weak scale)	31.7%	37.2%	20.98%	37.8%

We can conclude that performance for 8 node is better as compare to performance of 1 node for share-memory, hadoop, spark and mpi. Among all Spark performance is better and takes less time for sorting large datafile on 1 node and 8 node. According to my result share memory performance is slightly better in some cases as compared to hadoop and spark. But for 8 node spark performance is better as clustering is done in-memory. So Spark performance is better as we increase the node. I can assume that as node goes on increasing we get more better performance for spark. Therefore node like 100 and 1000 we get better performance for spark.

CloudSort represents the Total cost of ownership for external sort. It measure the efficiency of external sort using TCO.

REFRENCE:

https://spark.apache.org/

https://www.edureka.co