NFS setup

```
root@smallnode1:/mnt/project# sudo df -hT
Filesystem                          Type   Size  Used Avail Use% Mounted on
tmpfs                               tmpfs  196M  1.6M  195M   1% /run
/dev/sda3                           ext4    20G   13G  6.6G  65% /
tmpfs                               tmpfs  980M     0  980M   0% /dev/shm
tmpfs                               tmpfs  5.0M  4.0K  5.0M   1% /run/lock
/dev/sda2                           vfat   512M  6.1M  506M   2% /boot/efi
tmpfs                               tmpfs  196M  148K  196M   1% /run/user/1000
192.168.4.166:/exports/project nfs4     24G   14G  9.3G  60% /mnt/project
```

Here at the bottom, we can see nfs successfully on the client and my server is running on 192.168.4.166.

```
sudo: nfs: command not found
root@smallnode2:/home/rakesh# sudo df -hT
Filesystem                          Type   Size  Used Avail Use% Mounted on
tmpfs                               tmpfs  196M  1.5M  195M   1% /run
/dev/sda3                           ext4    22G   14G  7.3G  65% /
tmpfs                               tmpfs  980M     0  980M   0% /dev/shm
tmpfs                               tmpfs  5.0M  4.0K  5.0M   1% /run/lock
/dev/sda2                           vfat   512M  6.1M  506M   2% /boot/efi
tmpfs                               tmpfs  196M  112K  196M   1% /run/user/1000
192.168.4.166:/exports/project nfs4     24G   14G  9.1G  61% /mnt/project
root@smallnode2:/home/rakesh#
```

This is my second node with nfs client running successfully and my server is running on 192.168.4.166.

And done for 3,4,5,6 nodes in the same way

Hadoop Datasets

```
2023-11-15 03:37:55,602 INFO mapreduce.Job:  map 84% reduce 0%
2023-11-15 03:37:56,604 INFO mapreduce.Job:  map 87% reduce 0%
2023-11-15 03:37:57,607 INFO mapreduce.Job:  map 88% reduce 0%
2023-11-15 03:38:01,615 INFO mapreduce.Job:  map 89% reduce 0%
2023-11-15 03:38:02,618 INFO mapreduce.Job:  map 90% reduce 0%
2023-11-15 03:38:17,658 INFO mapreduce.Job:  map 92% reduce 0%
2023-11-15 03:38:21,669 INFO mapreduce.Job:  map 95% reduce 0%
2023-11-15 03:38:22,672 INFO mapreduce.Job:  map 98% reduce 0%
2023-11-15 03:38:23,676 INFO mapreduce.Job:  map 99% reduce 0%
2023-11-15 03:38:27,691 INFO mapreduce.Job:  map 100% reduce 0%
2023-11-15 03:38:43,736 INFO mapreduce.Job:  map 100% reduce 19%
2023-11-15 03:38:49,750 INFO mapreduce.Job:  map 100% reduce 29%
2023-11-15 03:41:32,109 INFO mapreduce.Job:  map 100% reduce 30%
2023-11-15 03:41:38,123 INFO mapreduce.Job:  map 100% reduce 35%
2023-11-15 03:41:44,134 INFO mapreduce.Job:  map 100% reduce 36%
2023-11-15 03:41:50,145 INFO mapreduce.Job:  map 100% reduce 38%
2023-11-15 03:41:56,156 INFO mapreduce.Job:  map 100% reduce 40%
2023-11-15 03:42:02,167 INFO mapreduce.Job:  map 100% reduce 42%
2023-11-15 03:42:08,179 INFO mapreduce.Job:  map 100% reduce 44%
2023-11-15 03:42:14,190 INFO mapreduce.Job:  map 100% reduce 46%
2023-11-15 03:42:20,201 INFO mapreduce.Job:  map 100% reduce 47%
2023-11-15 03:42:26,211 INFO mapreduce.Job:  map 100% reduce 49%
2023-11-15 03:42:32,222 INFO mapreduce.Job:  map 100% reduce 51%
2023-11-15 03:42:38,232 INFO mapreduce.Job:  map 100% reduce 53%
2023-11-15 03:42:44,244 INFO mapreduce.Job:  map 100% reduce 55%
2023-11-15 03:42:50,254 INFO mapreduce.Job:  map 100% reduce 57%
2023-11-15 03:42:56,265 INFO mapreduce.Job:  map 100% reduce 58%
2023-11-15 03:43:02,275 INFO mapreduce.Job:  map 100% reduce 60%
2023-11-15 03:43:08,286 INFO mapreduce.Job:  map 100% reduce 62%
2023-11-15 03:43:14,297 INFO mapreduce.Job:  map 100% reduce 64%
2023-11-15 03:43:20,308 INFO mapreduce.Job:  map 100% reduce 66%
2023-11-15 03:43:26,320 INFO mapreduce.Job:  map 100% reduce 68%
2023-11-15 03:43:32,333 INFO mapreduce.Job:  map 100% reduce 70%
2023-11-15 03:43:38,346 INFO mapreduce.Job:  map 100% reduce 71%
2023-11-15 03:43:44,359 INFO mapreduce.Job:  map 100% reduce 74%
2023-11-15 03:43:50,371 INFO mapreduce.Job:  map 100% reduce 77%
2023-11-15 03:43:56,385 INFO mapreduce.Job:  map 100% reduce 80%
2023-11-15 03:44:02,398 INFO mapreduce.Job:  map 100% reduce 82%
2023-11-15 03:44:08,410 INFO mapreduce.Job:  map 100% reduce 84%
2023-11-15 03:44:14,424 INFO mapreduce.Job:  map 100% reduce 87%
```

The screenshot of mapreduce execution jobs

```
root@8a5bedbca8ad:/code/Hadoop_Code#  hdfs dfs -cat /user/root/output/* | head
2023-11-14 19:46:09,821 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
20 20 20 20 22 4f 21 75 76 65       "O!uve  00000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
20 20 20 2c 4b 34 61 2d 3a 76      ,K4a-:v  00000000000000000000000001B8132  5555EEEE888899994444FFFF1111CCCCEEEE1111EEEE6666FFFF
20 20 20 2e 46 75 44 5c 7d 75      .FuD\}u  00000000000000000000000000797631  5555DDDDBBBB000077772222211112222444400DDDDDDDD99996666
20 20 20 3b 35 59 54 68 63 74      ;5YThct  00000000000000000000000007D3DF5  2222AAAACCCCFFFFAAAA4444455555EEEEE44442222DDDD99992222
20 20 20 3d 32 47 5e 39 7b 2d      =2G^9{-  00000000000000000000000000809EE  5555DDDD1111CCCC9999BBBB0000BBBBCCCCFFFFCCCC44443333
20 20 20 4e 7d 4d 39 3f 73 50      N}M9?sP  00000000000000000000000000429597  5555FFFF000077775555999911111CCCC66669999AAAAEEEE8888
20 20 20 50 30 58 3f 52 73 26      P0X?Rs&  00000000000000000000000000041162E  888833339999FFFF1111CCCC8888CCCC9999EEEEDDDD00003333
20 20 20 5b 58 71 5c 5c 24 25      [Xq\\$%  00000000000000000000000000097A5F0  66666666EEEEDDDD7777FFFF00005555FFFFFFFF888855551111
20 20 20 72 41 6d 51 67 34 76      rAmQg4v  00000000000000000000000008180D3  BBBB111111119999FFFFFFFFFFFFFFFF4444BBBB88884444CCCC
20 20 21 26 29 29 4a 66 33 3b     !&))Jf3;  00000000000000000000000004602E1  4444FFFFCCCC88888888CCCCFFFFCCCCEEEE55556666666666666
cat: Unable to write to output stream.
root@8a5bedbca8ad:/code/Hadoop_Code#  hdfs dfs -cat /user/root/output/* | tail
2023-11-14 19:46:20,103 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
7e 7e 7e 25 41 20 4e 42 5f 74   ~~~%A NB_t  00000000000000000000000003DE5EC  FFFF7777EEEEBBBB4444EEEEEEEE33339999DDDD999900005555
7e 7e 7e 2d 63 2d 43 51 28 3e   ~~~-c-CQ(>  00000000000000000000000000832611  9999FFFF1111777733337777000011114444444440000BBBB6666
7e 7e 7e 38 59 7d 46 71 6c 2a   ~~~8Y}Fql*  00000000000000000000000000742A4C  BBBB1111CCCCEEEE88880000000077773333333DDDD22225555
7e 7e 7e 3e 44 64 3d 51 54 5d   ~~~>Dd=QT]  00000000000000000000000006740F  9999DDDD000055556666CCCC22220000FFFFEEEEDDDDFFFF0000
7e 7e 7e 5d 4a 41 28 7d 6a 24   ~~~]JA(}j$  00000000000000000000000000060BCBE  11112222333344445555999AAAABBBB9999FFFFDDDDBBBB3333
7e 7e 7e 5d 5a 70 2e 23 2f 2b   ~~~]Zp.#/+  00000000000000000000000003B9A5A  CCCC8888EEEEAAAAEEEE33333333777700000FFFFCCCC66667777
7e 7e 7e 5f 6a 51 65 70 69 78   ~~~_jQepix  00000000000000000000000011E5D4  1111999911115555BBBB111100002222EEEE6666BBBB7777DDDD
7e 7e 7e 6e 74 3d 5a 48 5b 4e   ~~~nt=ZH[N  00000000000000000000000000332A13  44441111BBBBBBBB33337777FFFF4444555555553333000CCCC
7e 7e 7e 73 2f 50 71 2c 2d 45   ~~~s/Pq,-E  00000000000000000000000006BE930  2222DDDDDDDD77771111EEEECCCC7777BBBB4444888811111111
7e 7e 7e 7a 62 41 5f 20 54 74   ~~~zbA_ Tt  00000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
root@8a5bedbca8ad:/code/Hadoop_Code# hdfs dfs -rm -r /user/root/output
```

The above screenshot shows the head and tail of the Hadoop Sort program for 1GB and similarly for 4GB,16GB,32GB

Spark 1GB small instance

Command used :
time spark-submit --class SortGensortData --master local[4] --executor-memory 24G /home/cc/my-spark-project/target/spark-example-1.0-SNAPSHOT.jar 1GB 1GBoutput

And i got output and here is the sorted head and tail

```
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/1GBoutput$ cat part-* > outputfile.txt
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/1GBoutput$ cd ..
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input$ cd 1GBoutput/
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/1GBoutput$ ls
_SUCCESS        part-00001  part-00004  part-00007  part-00010  part-00013  part-00016  part-00019  part-00022
outputfile.txt  part-00002  part-00005  part-00008  part-00011  part-00014  part-00017  part-00020  part-00023
part-00000      part-00003  part-00006  part-00009  part-00012  part-00015  part-00018  part-00021
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/1GBoutput$ cat outputfile.txt | head
    "O!uve  00000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
   ,K4a-:v  00000000000000000000000001B8132  5555EEEE888899994444FFFF1111CCCCEEEE1111EEEE6666FFFF
   .FuD\}u  00000000000000000000000000797631  5555DDDDBBBB000077772222211112222444400DDDDDDDD99996666
   ;5YThct  00000000000000000000000007D3DF5  2222AAAACCCCFFFFAAAA4444455555EEEE44442222DDDD99992222
   =2G^9{-  00000000000000000000000000809EE  5555DDDD1111CCCC9999BBBB0000BBBBCCCCFFFFCCCC44443333
   N}M9?sP  00000000000000000000000000429597  5555FFFF000077775555999911111CCCC66669999AAAAEEEE8888
   P0X?Rs&  00000000000000000000000000041162E  888833339999FFFF1111CCCC8888CCCC9999EEEEDDDD00003333
   [Xq\\$%  00000000000000000000000000097A5F0  66666666EEEEDDDD7777FFFF00005555FFFFFFFF888855551111
   rAmQg4v  00000000000000000000000008180D3  BBBB111111119999FFFFFFFFFFFFFFFF4444BBBB88884444CCCC
  !&))Jf3;  00000000000000000000000004602E1  4444FFFFCCCC88888888CCCCFFFFCCCCEEEE55556666666666666
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/1GBoutput$ cat outputfile.txt | tail
~~~%A NB_t  00000000000000000000000003DE5EC  FFFF7777EEEEBBBB4444EEEEEEEE33339999DDDD999900005555
~~~-c-CQ(>  00000000000000000000000000832611  9999FFFF1111777733337777000011114444444440000BBBB6666
~~~8Y}Fql*  00000000000000000000000000742A4C  BBBB1111CCCCEEEE88880000000077773333333DDDD22225555
~~~>Dd=QT]  00000000000000000000000006740F  9999DDDD000055556666CCCC22220000FFFFEEEEDDDDFFFF0000
~~~]JA(}j$  00000000000000000000000000060BCBE  11112222333344445555999AAAABBBB9999FFFFDDDDBBBB3333
~~~]Zp.#/+  00000000000000000000000003B9A5A  CCCC8888EEEEAAAAEEEE33333333777700000FFFFCCCC66667777
~~~_jQepix  00000000000000000000000011E5D4  1111999911115555BBBB111100002222EEEE6666BBBB7777DDDD
~~~nt=ZH[N  00000000000000000000000000332A13  44441111BBBBBBBB33337777FFFF4444555555553333000CCCC
~~~s/Pq,-E  00000000000000000000000006BE930  2222DDDDDDDD77771111EEEECCCC7777BBBB4444888811111111
~~~zbA_ Tt  00000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/1GBoutput$
```

Spark 4GB Large

Command used :

time spark-submit --class SortGensortData --master local[24] --executor-memory 96G /home/cc/my-spark-project/target/spark-example-1.0-SNAPSHOT.jar 4GB 4GBoutput

```
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input$ cd 4GBoutput/
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/4GBoutput$ ls
_SUCCESS    part-00001  part-00003  part-00005  part-00007  part-00009  part-00011  part-00013  part-00015  part-00017  part-00019  part-00021  part-00023
part-00000  part-00002  part-00004  part-00006  part-00008  part-00010  part-00012  part-00014  part-00016  part-00018  part-00020  part-00022
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/4GBoutput$ cat part-* >output
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/4GBoutput$ cat output |head
    "O!uve  00000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
    ^3CO],  00000000000000000000000158C5C5  5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
   !&S3/]]  00000000000000000000000002145D78  8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
   !,=U#,9  00000000000000000000000019072E3  33332222FFFFBBBB0000FFFFAAAA6666555533333DDDD3333CCCC
   &&!epOj  00000000000000000000000001CBB42E  2222BBBB2222FFFFFFFFFFFFCCCC55556666666677770000333 3
   'AJ;hL;  00000000000000000000000001FBD93E  FFFFFFFF66669999FFFF4444666622223333000 0BBBB33333333
   (VcS$Pz  00000000000000000000000000A9F516  8888AAAA7777BBBB7777AAAA6666DDDDEEEE9999BBBBBBBBBBBB
   ,K4a~:v  00000000000000000000000001B8132  5555EEEE888899994444FFFF1111CCCCEEEE1111EEEE6666FFFF
   .FuD\}u  00000000000000000000000000797631  5555DDDDBBBB000077772222111122224444DDDDDDDD99996666
   2;oEW+S  00000000000000000000000020D0B5F  000011110000000077770000222233334AAAA4444CCCCAAAA0000
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/4GBoutput$ cat output | tail
~~~nt=ZH[N  00000000000000000000000000332A13  44441111BBBBBBBB33337777FFFF4444555555553330000CCCC
~~~o)};EXo  00000000000000000000000001984704  BBBBDDDD55554444FFFF999955553333AAAABBBBEEEE8888DDDD
~~~p2(IR`|  00000000000000000000000001D86C98  CCCCDDDD1111AAAAEEEE888855559999DDDD7777DDDD88889999
~~~ruROgur  00000000000000000000000001E87D8A  BBBBEEEE22222222BBBBAAAA55551111FFFF99995555FFFF7777
~~~s/Pq,-E  00000000000000000000000006BE930  2222DDDDDDDD77771111EEEECCCC7777BBBB4444888811111111
~~~sf&V`wv  00000000000000000000000015B2E94  6666BBBB1111EEEEEEEE5555CCCCDDDD55555555BBBBBBBBDDDD
~~~yKOl:gE  00000000000000000000000002048B4F  CCCC1111444488882222666 6BBBB888855557777EEEEBBBB0000
~~~zbA_ Tt  00000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~zeO^FEg  00000000000000000000000001E06130  4444CCCCBBBB999922228888555588 88CCCCFFFF000011111111
~~~}GxjWHI  00000000000000000000000000CA1345  777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC2222
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/4GBoutput$
```

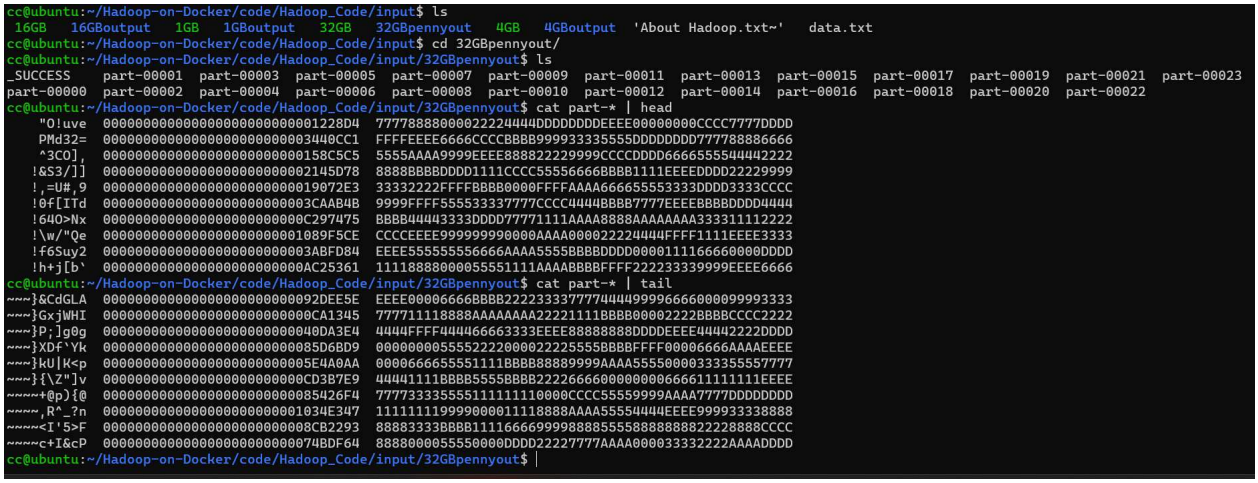
Spark 16GB

time spark-submit --class SortGensortData --master local[24] --executor-memory 96G /home/cc/my-spark-project/target/spark-example-1.0-SNAPSHOT.jar 16GB 16GBoutput

```
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input$ cd 16GBoutput/
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/16GBoutput$ ls
_SUCCESS    part-00001  part-00003  part-00005  part-00007  part-00009  part-00011  part-00013  part-00015  part-00017  part-00019  part-00021  part-00023
part-00000  part-00002  part-00004  part-00006  part-00008  part-00010  part-00012  part-00014  part-00016  part-00018  part-00020  part-00022
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/16GBoutput$ cat part* >outputfile
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/16GBoutput$ cat outputfile |head
    "O!uve  00000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
    PMd32=  00000000000000000000000034400CC1  FFFFEEEE6666CCCCBBBB9999333355555DDDDDDDD7777888866666
    ^3CO],  00000000000000000000000158C5C5  5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
   !&S3/]]  00000000000000000000000002145D78  8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
   !,=U#,9  00000000000000000000000019072E3  33332222FFFFBBBB0000FFFFAAAA6666555533333DDDD3333CCCC
   !0f[ITd  00000000000000000000000003CAAB4B  9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
   !f6Suy2  00000000000000000000000003ABFD84  EEEE5555555566666AAAA5555BBBBDDDD000011116666000 0DDDD
   "X<XlWW  00000000000000000000000006652BCF  888888887777111144444444411114444666 6BBBBEEEE33330000
   #%NIpq.  00000000000000000000000003B36FB9  11110000333344441111666666666AAAAAAAA00001111CCCCEEEE
   #'^cl'~  00000000000000000000000002EDC5C8  8888AAAA11114444FFFF77773333EEEE44440000FFFF99999999
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/16GBoutput$ cat outputfile | tail
~~~zeO^FEg  00000000000000000000000001E06130  4444CCCCBBBB999922228888555588 88CCCCFFFF000011111111
~~~{{*WEDl  00000000000000000000000097D6633  AAAAAAAAFFFFBBBBDDDDFFFFFFFFCCCC999922228888EEEECCCC
~~~}&CdGLA  00000000000000000000000092DEE5E  EEEE00006666BBBB22223333777774444999966660000999933 33
~~~}GxjWHI  00000000000000000000000000CA1345  777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC2222
~~~}P;]g0g  00000000000000000000000040DA3E4  4444FFFF444466663333EEEE88888888DDDDEEEE44442222DDDD
~~~}XDf`Yk  00000000000000000000000085D6BD9  00000000555522220000222255555BBBBFFFF00006666AAAAEEEE
~~~}kU|K<p  00000000000000000000000005E4A0AA  0000666655551111BBBB88889999AAAA555500003333555577 77
~~~~+@p){@  00000000000000000000000085426F4  77773333555511111111 10000CCCC55559999AAAA7777DDDDDDDD
~~~~<I'5>F  00000000000000000000000008CB2293  88883333BBBB1111666699998888555588888888222288 88CCCC
~~~~c+I&cP  00000000000000000000000074BDF64  88880000555500000DDDD22227777AAAA000033332222AAAADDDD
cc@ubuntu:~/Hadoop-on-Docker/code/Hadoop_Code/input/16GBoutput$
```

Spark 32GB

time spark-submit --class SortGensortData --master local[24] --executor-memory 96G /home/cc/my-spark-project/target/spark-example-1.0-SNAPSHOT.jar 32GB 32GBpennyout



Time in milliseconds

	Hadoop Sort	Spark
Large 1 GB	309295	25339
Large 4 GB	1195824	74839
Large 16 GB	4623041	489121
Large 32 GB	13844934	998032
Small 1 GB	680449	38090
Small 4 GB	3707054	218027
Small 16 GB	12019906	1068332
Small 32 GB	Not feasible	4807494
6 nodes 1 GB	35721	5982
6 nodes 4 GB	143312	22028
6 nodes 16 GB	520348	86274

6 nodes 32 GB	1273792	258481

Performance evaluation :

- All configurations see an increase in Hadoop Sort and Spark processing times as data sizes rise. This makes sense because processing larger datasets takes longer. The pace of rise differs throughout the setups, though.
- In terms of processing speed, Spark routinely beats Hadoop Sort for all data volumes and setups. This is probably because Hadoop Sort's disk-based processing is less efficient than Spark's in-memory data processing capabilities.
- When compared to the individual big and small instances, the 6-node cluster considerably boosts Hadoop Sort and Spark performance. This demonstrates the advantages of distributed computing in parallel.
- For Hadoop Sort and Spark, the large instance—which has more cores, RAM, and disk space—performs better than the tiny instance. This demonstrates how crucial it is to have more computer power for jobs requiring a lot of data.
- The Hadoop Sort on a tiny instance for the 32 GB dataset is tagged as "Not feasible," meaning that the small instance does not have the memory or disk capacity to handle such huge datasets efficiently.
- The 6-node cluster outperforms single instances in terms of performance, however as data size grows, the improvement ratio falls. For example, for smaller datasets, the time decrease from tiny to 6-node setups is more pronounced than for the 32 GB dataset. This points to a potential scaling limit brought on by factors including disk I/O constraints in bigger datasets, network overhead, and complicated data management.
- The absolute times demonstrate that processing times for huge datasets may be extremely high, even with a 6-node cluster (e.g., over a million milliseconds for 32 GB data in Hadoop Sort). Nevertheless, the cluster architecture offers a significant speed improvement over the single-node instances.

For 32GB dataset:

Large Instance, Hadoop Sort, 32 GB: Disk I/O Throughput = 2.37 MB/sec.
Large Instance, Spark, 32 GB: Disk I/O Throughput = 32.85 MB/sec.
6 Small Instances, Hadoop Sort, 32 GB: Disk I/O Throughput = 25.73 MB/sec.
6 Small Instances, Spark, 32 GB: Disk I/O Throughput = 126.82 MB/sec.

Q and A evaluations

- The absolute times demonstrate that, even with a 6-node cluster, processing times for big datasets may still be somewhat high (for example, more than a million milliseconds for 32 GB of data in Hadoop Sort). The cluster arrangement, however, offers a significant speed improvement above the single-node instances.

- Both Hadoop Sort and Spark require more processing time as data volume grows. But compared to Hadoop Sort, Spark's processing time increase is less noticeable, suggesting higher scalability.
- Although the huge instance performs well, bigger data quantities make its limits clear, especially for Hadoop Sort.
- Performance is greatly enhanced by the 6-node cluster's dispersed processing. This is demonstrated via Hadoop Sort and Spark, demonstrating the advantages of parallel processing, particularly for bigger data sets.
- The increase from one to six tiny instances shows robust and positive scalability, particularly for Spark. But the scaling is nonlinear, suggesting that distributed processing has overhead costs.
- The 6-node cluster performs better at weak scaling than a single big instance, even though both configurations scale with higher data volumes.
- Both Hadoop Sort and Spark run well on the huge instance, with Spark exhibiting notable temporal savings.
- The 6-node small instance cluster is more appropriate, especially for Spark, which manages big datasets much more effectively.
- In single and multi-node deployments, Spark processes huge datasets more efficiently than Hadoop Sort.In single and multi-node deployments, Spark processes huge datasets more efficiently than Hadoop Sort.