CS 553 Programing Assignment 2b

Professor: Ioan Raicu

Bhagyashree Bagwe
(A20399761)

1. Problem Statement

In this experiment I have developed sorting application in Hadoop and Spark to be be performed on large datasets of sizes 8GB, 20GB and 80GB. The application reads data from HDFS, sorts it, and then stores the sorted data again in HDFS.

2. Runtime Environment Settings

The sorting experiments are performed at a Proton Cluster accessible at 216.47.142.37

Each Hadoop/HDFS cluster has following configuration:

No of nodes	:	4
No of cores	:	4-cores per node,
Memory	:	8GB RAM and 80GB of SSD storage

More information on proton clusters architecture can be found in following screen-shot:

```
bbagwe@proton:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             8
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 94
Model name:            Intel Core Processor (Skylake)
Stepping:              3
CPU MHz:               2099.996
BogoMIPS:              4199.99
Virtualization:        VT-x
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
L3 cache:              16384K
NUMA node0 CPU(s):     0-7
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb
rdtscp lm constant_tsc rep_good nopl xtopology eagerfpu pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_ti
mer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single rsb_ctxsw retpoline kaiser tpr_shadow vnmi flexpriority ept v
pid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx smap xsaveopt arat
bbagwe@proton:~$
```

3. Methodology

Inputs

- Three datasets of sizes 8GB, 20GB and 800GB are used testing the sorting applications.
- The data is generated using gensort program executable found at http://www.ordinal.com/gensort.html
- Following command is used to generate the data file:
  ./gensort -a no_of_records fileName

Hadoop Sort

- The HadoopSort application uses map/reduce framework to fragment and sort the input values.
- The application starts from a main method which submits a map/reduce job and contains two internal classes Hmapper and HReducer to override the logic of the mapper and reducer methods.
- To understand the Hadoop map reduce sorting algorithm, I have referred following links:
  https://www.tutorialspoint.com/map_reduce/map_reduce_algorithm.htm
  https://data-flair.training/blogs/shuffling-and-sorting-in-hadoop/

Spark Sort

- Spark sort uses Java RDD to perform in-memory sort on input data file.
- It reads input file records in RDD and then maps each row into key-value pair.
- In next steps it sorts the mapped values based on keys. In the end it writes the sorted data to output file.

Output

Three outputs are generated:

- Sorted output → written to temp folder
- Hadoop/Spark output → written to current directory [eg. output.log]
- Slurm output → written to current directory [eg. HadoopSort8GB.log]

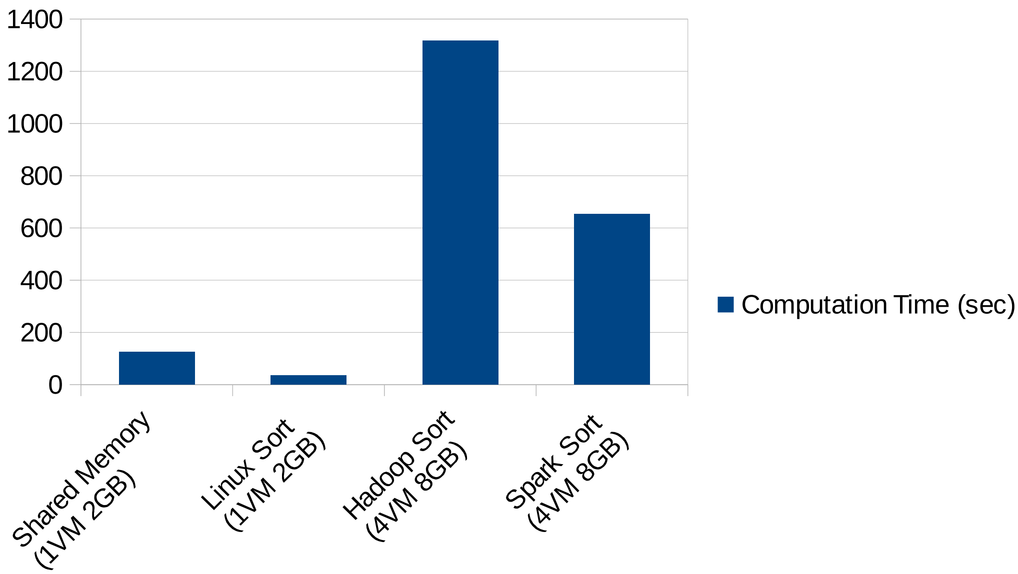
Execution

- The cluster uses Slurm scheduler to deploy Hadoop/Spark jobs on one of the available Hadoop/HDFS clusters. In order to run your applications you will have to implement Slurm job scripts and submit them to Slurm.
- Command is to submit a slurm job: sbatch *slurmFileName*
- Command is to check the status of the slurm job: squeue
- Command to cancel the slurm job: scancel job_id

4. Performance Evaluations

Table 1: Performance evaluation of sort (weak scaling – small dataset)

Experiment	Shared Memory (1VM 2GB)	Linux Sort (1VM 2GB)	Hadoop Sort (4VM 8GB)	Spark Sort (4VM 8GB)
Computation Time (sec)	125	34.98	1317.12	654
Data Read (GB)	7.6	4	8.48	8
Data write (GB)	8.8	4	8.8	8
I/O Throughput (MB/sec)	130	230	12.51	24.46
Speedup	-	-	0.37	0.76
Efficiency(%)	-	-	9.25	19.11



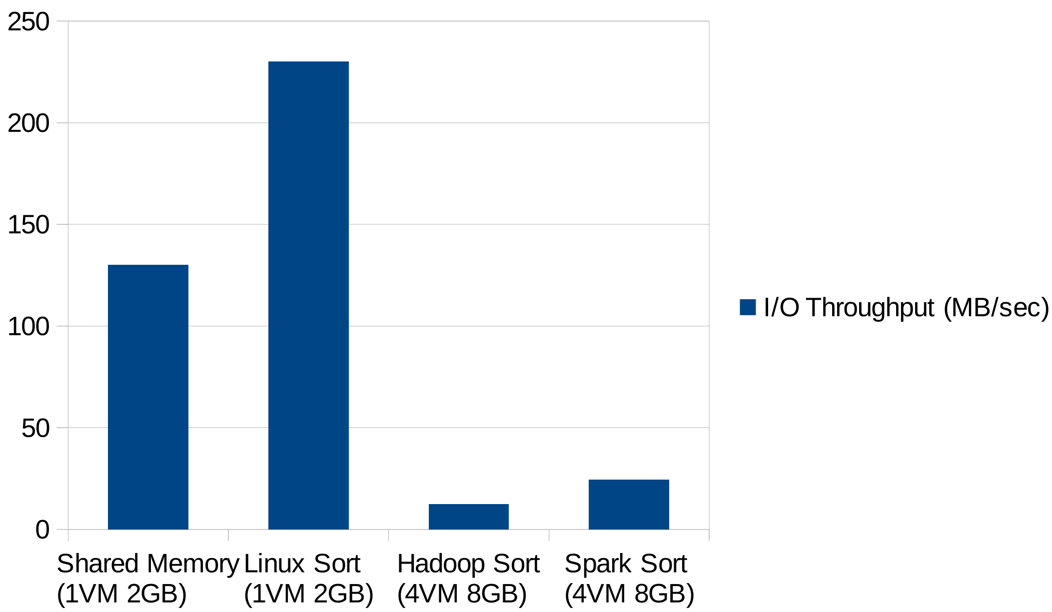
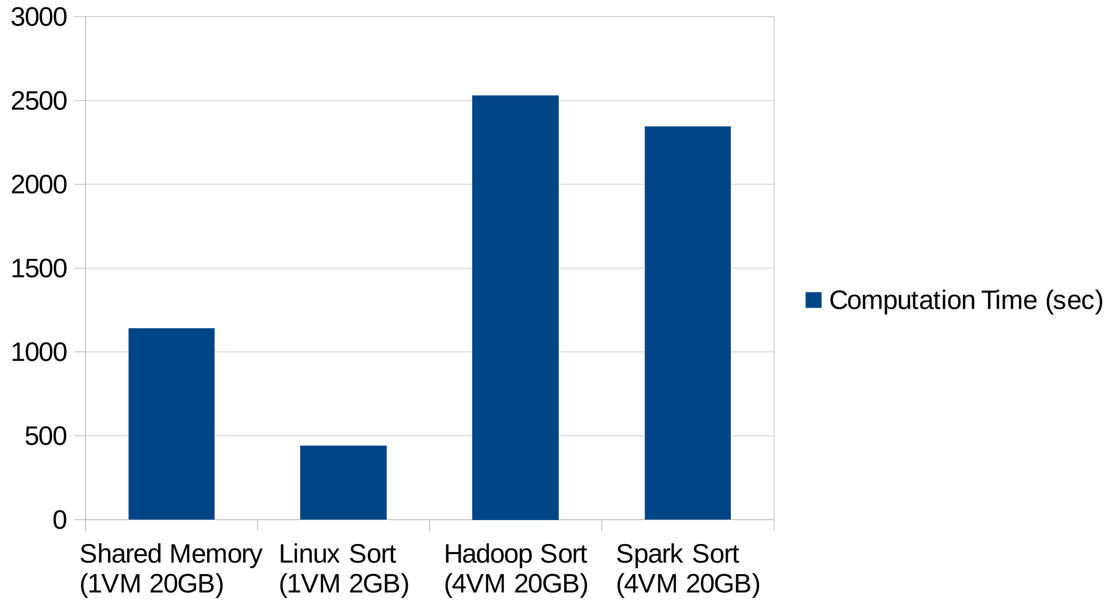


Table 2: Performance evaluation of sort (strong scaling – large dataset)

Experiment	Shared Memory (1VM 20GB)	Linux Sort (1VM 2GB)	Hadoop Sort (4VM 20GB)	Spark Sort (4VM 20GB)
Computation Time (sec)	1141.25	439.024	2529.45	2343
Data Read (GB)	116	40	20	20
Data write (GB)	128	40	22	20
I/O Throughput (MB/sec)	210	180	16.60	17.07
Speedup	-	-	0.4511	0.4869
Efficiency(%)	-	-	11.25	11.17



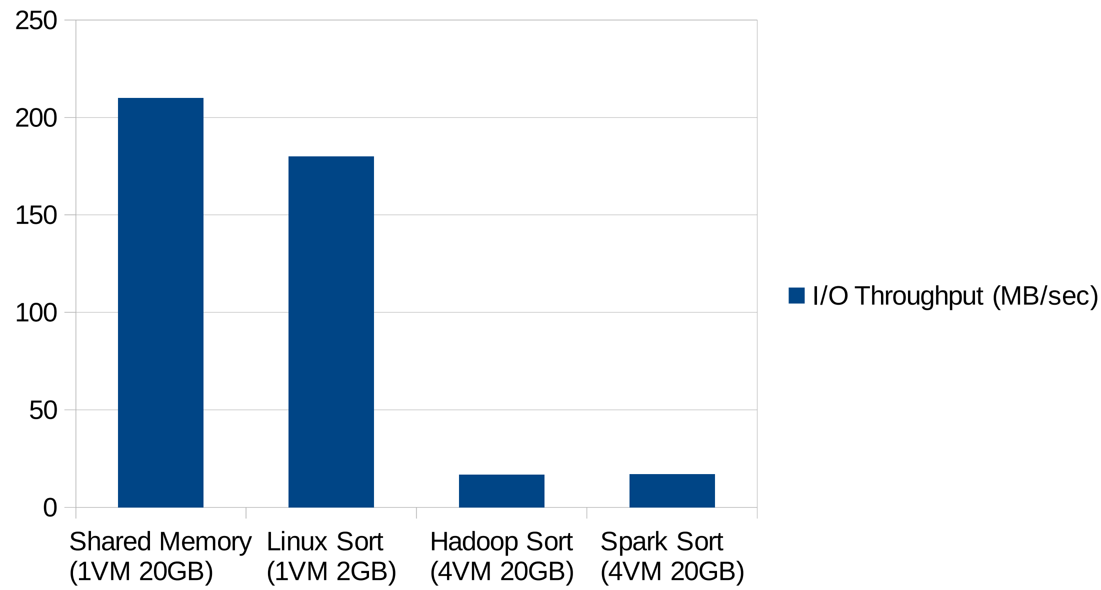
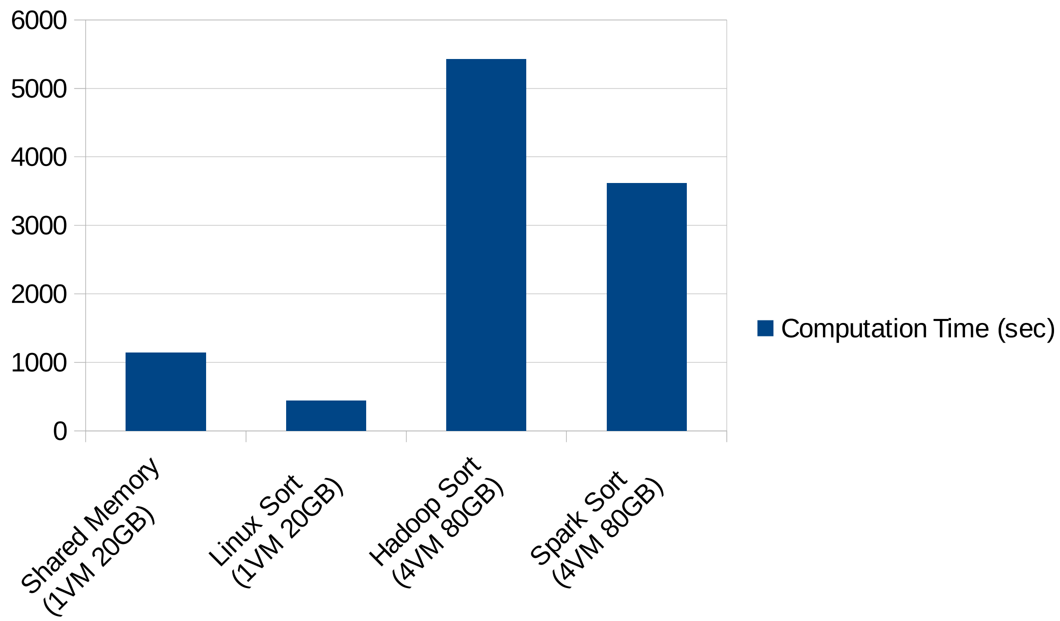
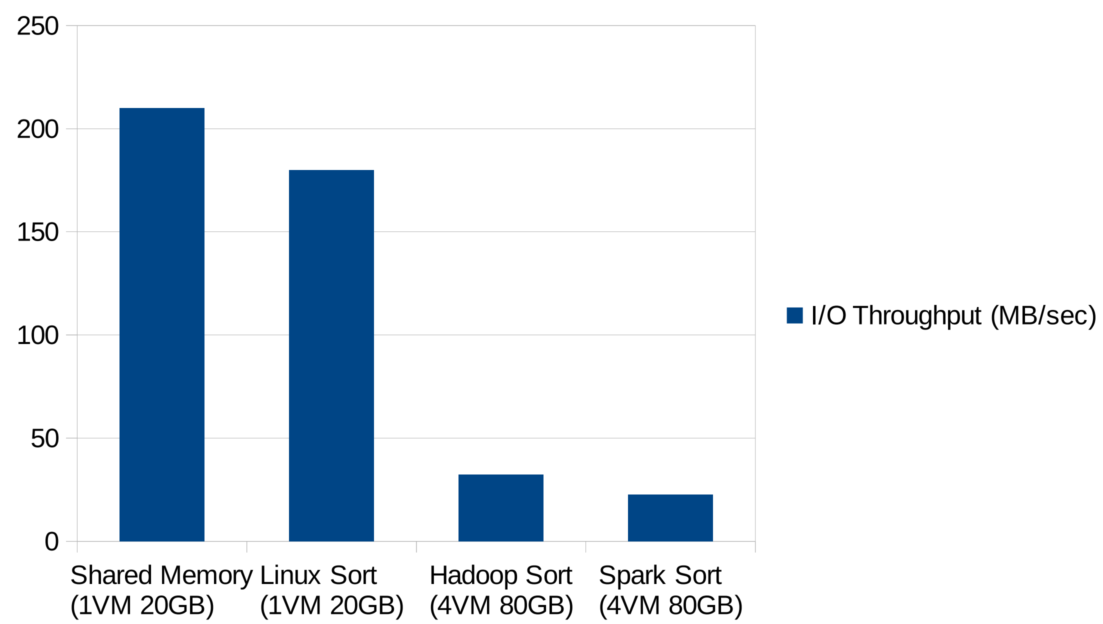


Table 3: Performance evaluation of sort (weak scaling – large dataset)

Experiment	Shared Memory (1VM 20GB)	Linux Sort (1VM 20GB)	Hadoop Sort (4VM 80GB)	Spark Sort (4VM 80GB)
Computation Time (sec)	1141.25	439.024	5429	3621
Data Read (GB)	116	40	80	80
Data write (GB)	128	40	88	80
I/O Throughput (MB/sec)	210	180	32.31	22.63
Speedup	-	-	0.840	1.26
Efficiency(%)	-	-	21	31.5





The log files are generated in following format:
Hadoop sort 8GB log

```
bbagwe@proton: ~/cs553-pa2b
                Data-local map tasks=95
                Rack-local map tasks=26
                Total time spent by all maps in occupied slots (ms)=2437470
                Total time spent by all reduces in occupied slots (ms)=408624
                Total time spent by all map tasks (ms)=2437470
                Total time spent by all reduce tasks (ms)=408624
                Total vcore-milliseconds taken by all map tasks=2437470
                Total vcore-milliseconds taken by all reduce tasks=408624
                Total megabyte-milliseconds taken by all map tasks=2495969280
                Total megabyte-milliseconds taken by all reduce tasks=418430976
        Map-Reduce Framework
                Map input records=80000000
                Map output records=80000000
                Map output bytes=8800000000
                Map output materialized bytes=8960000720
                Input split bytes=13440
                Combine input records=80000000
                Combine output records=80000000
                Reduce input groups=80000000
                Reduce shuffle bytes=8960000720
                Reduce input records=80000000
                Reduce output records=80000000
                Spilled Records=334623496
                Shuffled Maps =120
                Failed Shuffles=0
                Merged Map outputs=120
                GC time elapsed (ms)=34922
                CPU time spent (ms)=1182890
                Physical memory (bytes) snapshot=35375116288
                Virtual memory (bytes) snapshot=238729756672
                Total committed heap usage (bytes)=24343740416
        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=8000487424
        File Output Format Counters
                Bytes Written=8800000000
bbagwe@proton:~/cs553-pa2b$
```

Spark 8GB log

```
bbagwe@proton: ~/cs553-pa2b
bbagwe@proton:~/cs553-pa2b$ cat SparkSort8GB.log

2018-05-01 02:33:27 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where appl
icable
2018-05-01 02:33:27 INFO  SparkContext:54 - Running Spark version 2.3.0
2018-05-01 02:33:27 INFO  SparkContext:54 - Submitted application: SparkSort
2018-05-01 02:33:27 INFO  SecurityManager:54 - Changing view acls to: bbagwe
2018-05-01 02:33:27 INFO  SecurityManager:54 - Changing modify acls to: bbagwe
2018-05-01 02:33:27 INFO  SecurityManager:54 - Changing view acls groups to:
2018-05-01 02:33:27 INFO  SecurityManager:54 - Changing modify acls groups to:
2018-05-01 02:33:27 INFO  SecurityManager:54 - SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(bb
agwe); groups with view permissions: Set(); users  with modify permissions: Set(bbagwe); groups with modify permissions: Set()
2018-05-01 02:33:28 INFO  Utils:54 - Successfully started service 'sparkDriver' on port 43508.
2018-05-01 02:33:28 INFO  SparkEnv:54 - Registering MapOutputTracker
2018-05-01 02:33:28 INFO  SparkEnv:54 - Registering BlockManagerMaster
2018-05-01 02:33:28 INFO  BlockManagerMasterEndpoint:54 - Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
2018-05-01 02:33:28 INFO  BlockManagerMasterEndpoint:54 - BlockManagerMasterEndpoint up
2018-05-01 02:33:28 INFO  DiskBlockManager:54 - Created local directory at /tmp/blockmgr-a1d0f010-b282-40b2-ae80-1e656c71af2d
2018-05-01 02:33:28 INFO  MemoryStore:54 - MemoryStore started with capacity 366.3 MB
2018-05-01 02:33:28 INFO  SparkEnv:54 - Registering OutputCommitCoordinator
2018-05-01 02:33:28 INFO  log:192 - Logging initialized @3430ms
2018-05-01 02:33:28 INFO  Server:346 - jetty-9.3.z-SNAPSHOT
2018-05-01 02:33:28 INFO  Server:414 - Started @3577ms
2018-05-01 02:33:28 INFO  AbstractConnector:278 - Started ServerConnector@200606de{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2018-05-01 02:33:28 INFO  Utils:54 - Successfully started service 'SparkUI' on port 4040.
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@20312893{/jobs,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@36ac8a63{/jobs/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@4d9d1b69{/jobs/job,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@251f7d26{/jobs/job/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@77b21474{/stages,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@52d10fb8{/stages/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@41c07648{/stages/stage,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@22680f52{/stages/stage/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@60d84f61{/stages/pool,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@39c11e6c{/stages/pool/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@324dcd31{/storage,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@503d56b5{/storage/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@72bca894{/storage/rdd,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@433ffad1{/storage/rdd/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@1fc793c2{/environment,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@2575f671{/environment/json,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@329a1243{/executors,null,AVAILABLE,@Spark}
2018-05-01 02:33:28 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@ecf9fb3{/executors/json,null,AVAILABLE,@Spark}
```

5. What conclusions can you draw?

We can draw following conclusions:

- We can conclude that hadoop and spark scale well on distributed system and giver better throughput than shared memory.
- Spark always gives better performance in terms of throughput than hadoop due its in-memory computations which minimizes disk access of the algorithm
- If we increase data size, both spark and hadoop scale well

6. Which seems to be best at 1 node scale?

Spark performs best at 1 node scale. Spark does in-memory sort and does not write intermediate data to HDFS.

7. How about 4 nodes?

Even for 4 nodes Spark is better solution. Sparks in-memory computation is its USP. Since time access is biggest concern, spark saves a lot of time by avoiding intermediate disk accesses.

8. Can you predict which would be best at 100 node scale? How about 1000 node scales?

As mentioned above spark would be a better choice over Hadoop for 100 as well as 1000 node scale.

9. Compare your results with those from the Sort Benchmark?

- Hadoop Winner in 2013:
  102.5 TB in 4,328 seconds
  2100 nodes x (2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks)
  Thomas Graves [Yahoo! Inc.]
  Considering configuration of our cluster to sort using Thomas's benchmark he would need approximately ~ 120000 minutes

- Spark winner in 2014:
  100 TB in 1,406 seconds
  207 Amazon EC2 i2.8xlarge nodes x (32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD)
  Reynold Xin, Parviz Deyhim, Xiangrui Meng,
  Ali Ghodsi, Matei Zaharia Databricks
  Considering configuration of our cluster to sort using Thomas's benchmark he would need approximately ~ 170000 minutes

10. what can you learn from the CloudSort benchmark?

This benchmark provides us with the minimum cost for sorting the data on public cloud. It considers a record size of 100 bytes with 10 bytes of key. This benchmark helps in improving memory intensive applications. In summary we can say cloudsort provides minimum cost sort on public cloud.