CS553 Programming Assignment 2

David Ghiurco - A20325178
Alexandru Iulian Orhean - A20386581

1. Introduction
2. Shared Memory Implementation
    2.1 Virtual Cluster (1-node i3.large)
    2.2 Virtual Cluster (1-node i3.4xlarge)
3. Apache Hadoop MapReduce Implementation
    3.1 Virtual Cluster (1-node i3.large)
    3.2 Virtual Cluster (1-node i3.4xlarge)
    3.3 Virtual Cluster (8-nodes i3.large)
4. Apache Spark Implementation
    4.1 Virtual Cluster (1-node i3.large)
    4.2 Virtual Cluster (1-node i3.4xlarge)
    4.3 Virtual Cluster (8-nodes i3.large)
5. Performance Evaluation
6. Difficulties encountered
7. Conclusions

1. Introduction

The purpose of this assignment is to implement external sort (larger-than-RAM) applications across a wide range of dataset sizes, hardware platforms, and software frameworks. The experiments described in this report include examination of various external sort implementations across 1-node setups, multi-node setups, small datasets (128 GB), large datasets (1 TB), all performed on top of Amazon EC2.

The steps taken in setting up and performing this assignment involved a learning curve in working with Apache Hadoop, and Apache Spark. A lot of time was spent on debugging configurations of various daemons of these frameworks, and getting all cluster components to orchestrate harmoniously was no trivial task. Programming in these frameworks was rather short, as the software stack is designed to provide convenient tools for use cases such as ours.

Our clusters have the following software stack:

- ➔ Linux Ubuntu 16.04 LTS 64-bit (Standard free AMI on Amazon EC2)
- ➔ Java (OpenJDK) 1.8.0_151
- ➔ Apache Hadoop 2.7.4
- ➔ Apache Spark 2.2.0 (Scala 2.11) built with Hadoop 2.7.4
- ➔ Gradle 4.3.1

Gradle is used to compile the Hadoop and Spark applications. No manual Gradle environment setup is required, as everything is scripted.

Note: No MPI implementation was performed in this assignment

2. Shared Memory Implementation

THe shared memory solution is implemented in the C programming language, with the help of GCC built-in atomic operations and the POSIX threads library. It doesn't have any other dependencies and the application can be easily built through common Linux tools (make and gcc). The shared memory implementation has higher performance, when compared with the alternatives (Hadoop and Spark), and that is due to its design and the elimination of any kind of redundancy or checkpointing. The algorithm uses the concepts behind bucket sort in order to split the initial input file into smaller files (preferably having the size smaller than the RAM limit per thread), grouping the input into known ranges of values. After the "split" phase, each smaller input file is sorted independently in memory, using quick sort (implemented by hand, due to optimization reasons).

The bucket sort algorithm has its own pitfalls, especially with respect to the “split” process, that yields results highly dependant on the uniformity of the random distribution of elements that need to be sorted. In order to partially solve this problem, the implementation keeps track of the resulting split files through the use of a tree, that expands its leaf nodes, in order generate files that would fit into memory. This step creates multiple levels of “split” phases and can be tuned through the **FACTOR** defined value in the source code. For best performance, experimentally it has been observed that a value of 4/8 is required of the **FACTOR** variable.

As a bonus, the application contains auto-logging information, that is displayed at the end, containing the number of read and write operations executed in terms of GB and the time required to execute the application.

2.1 Virtual Cluster (1-node i3.large)

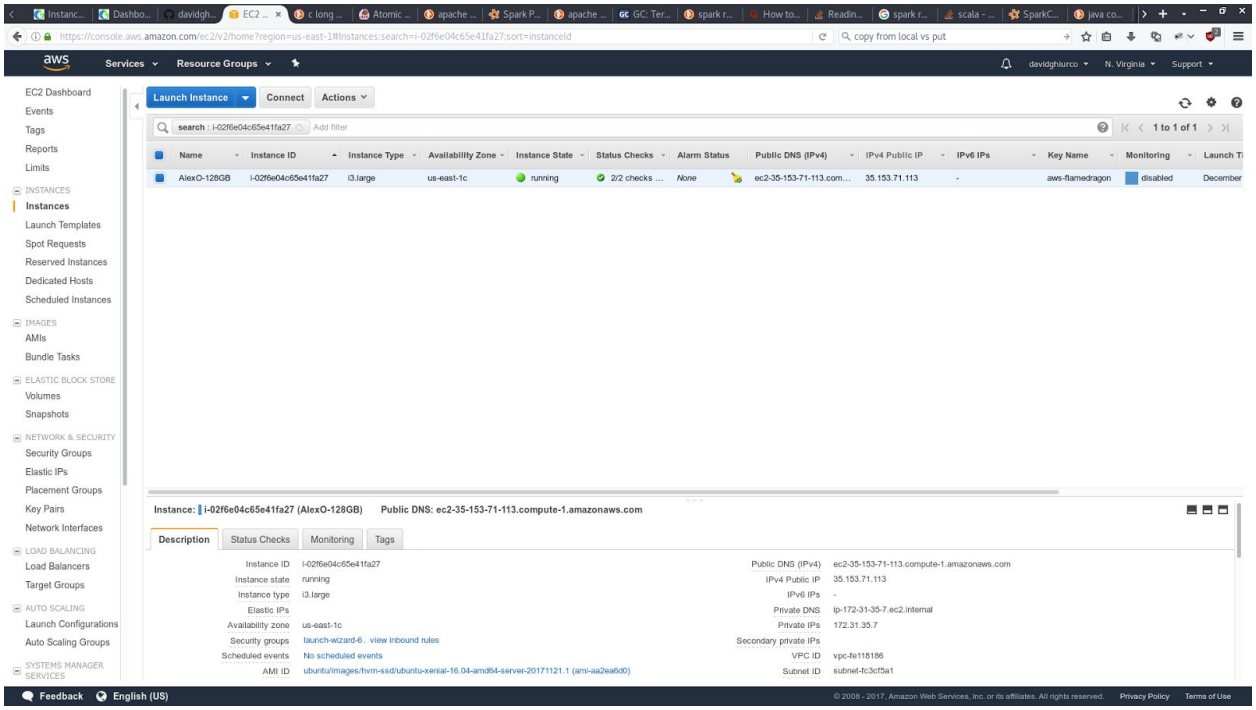


Figure 2.1.a AWS Instance running the 128GB sort experiment.

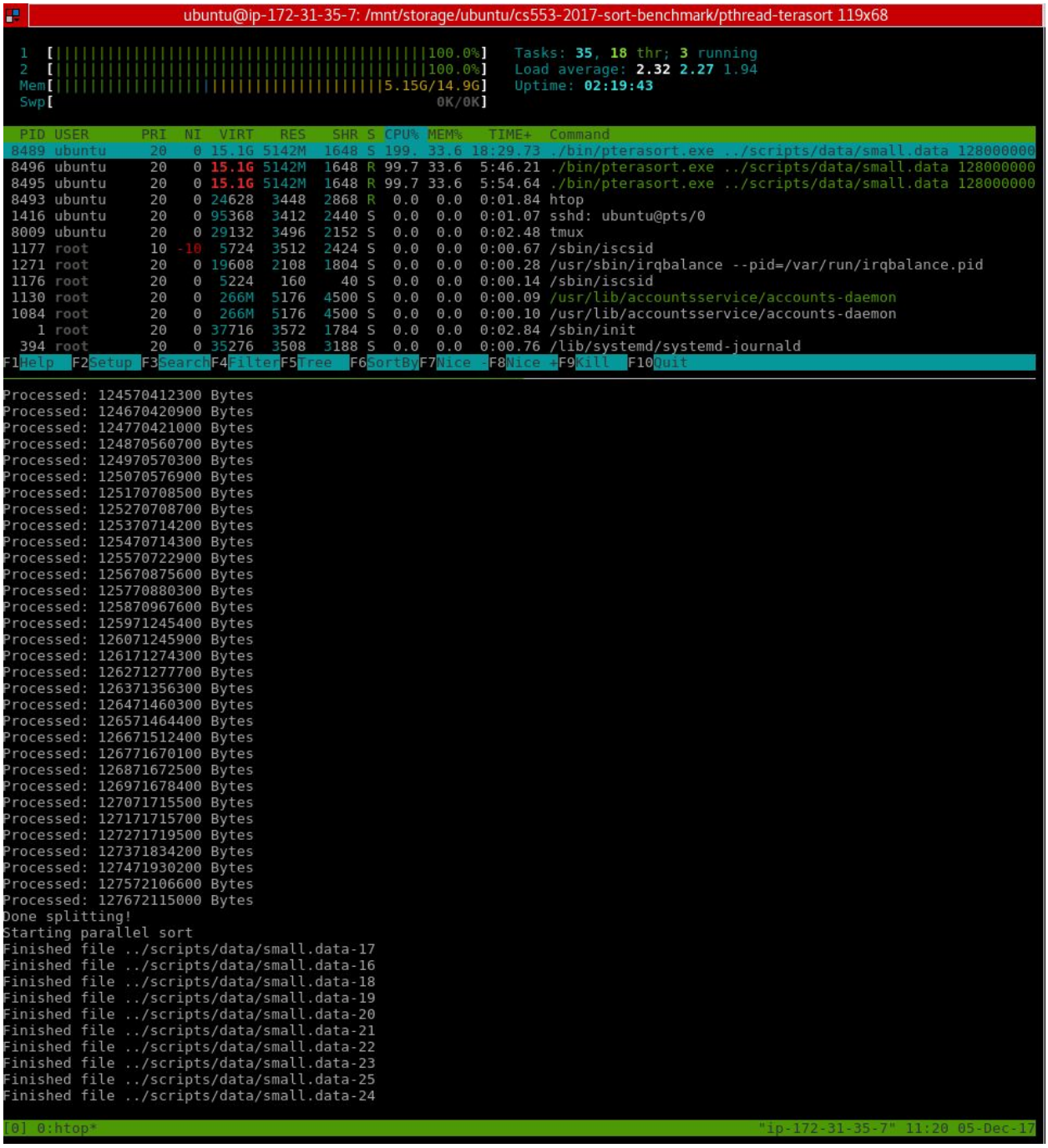


Figure 2.1.b Shared memory application running midway.

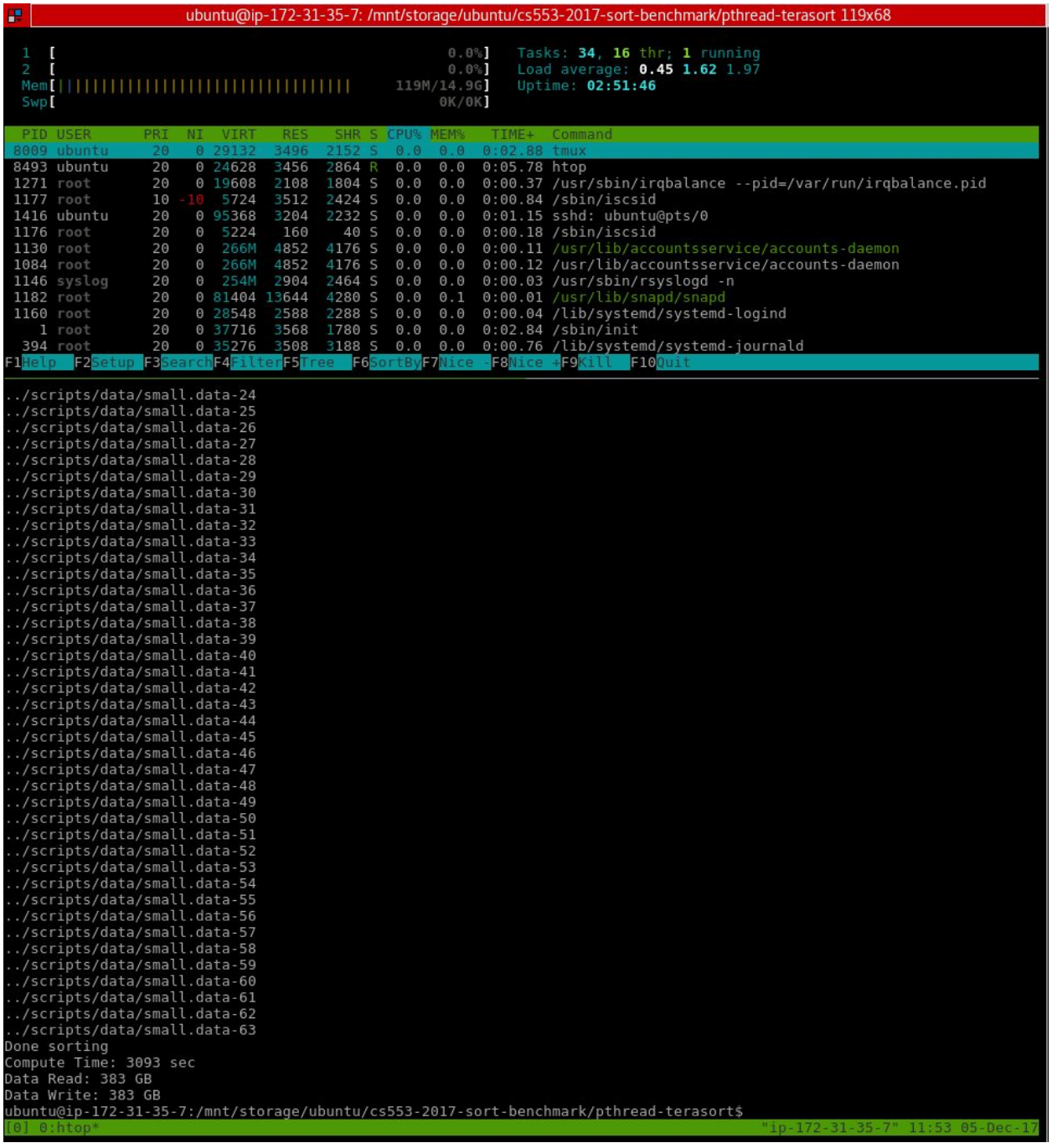


Figure 2.1.c Shared memory application finishing the sort experiment.

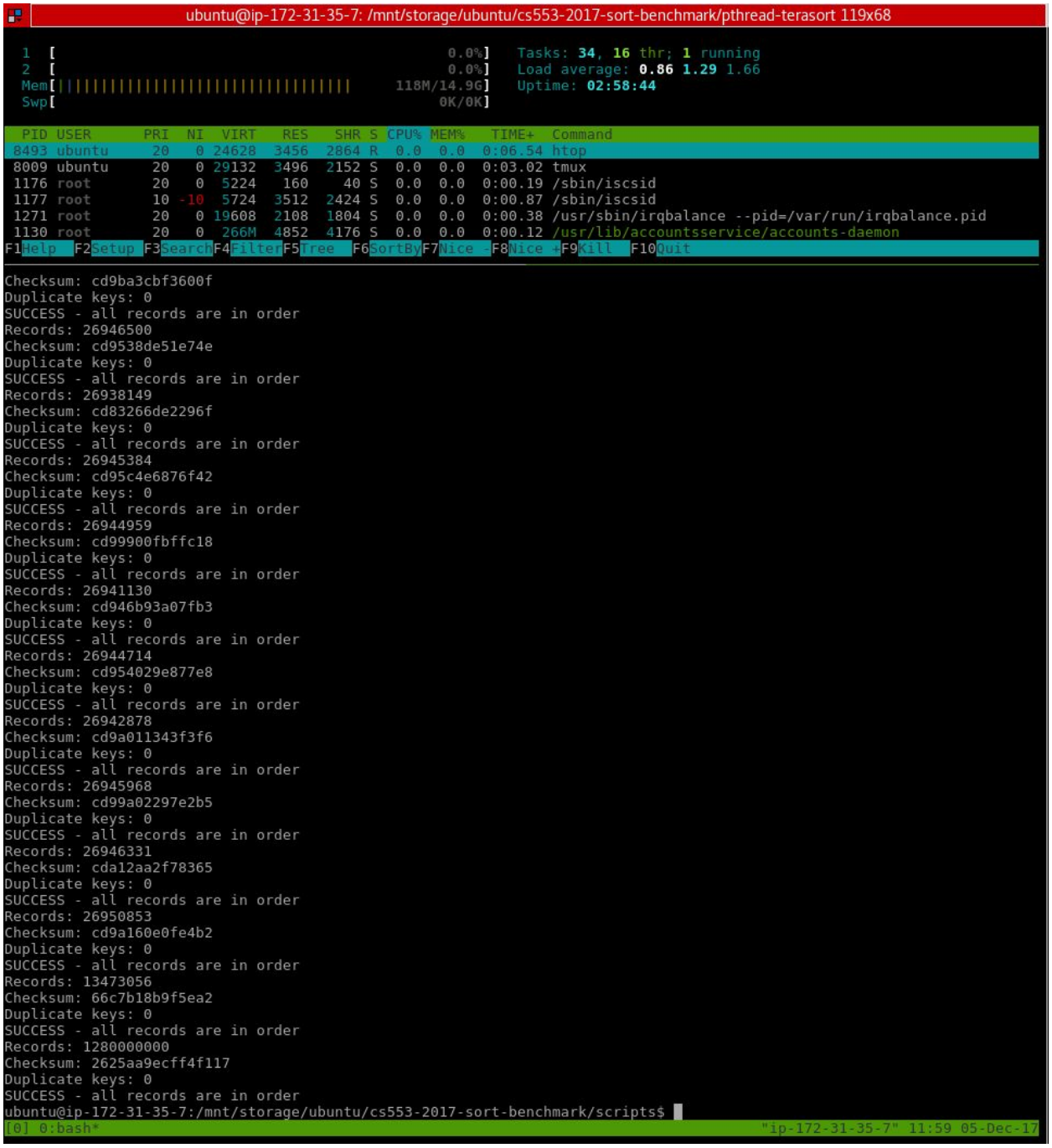


Figure 2.1.d Validation of the shared memory sort results.

2.2 Virtual Cluster (1-node i3.4xlarge)

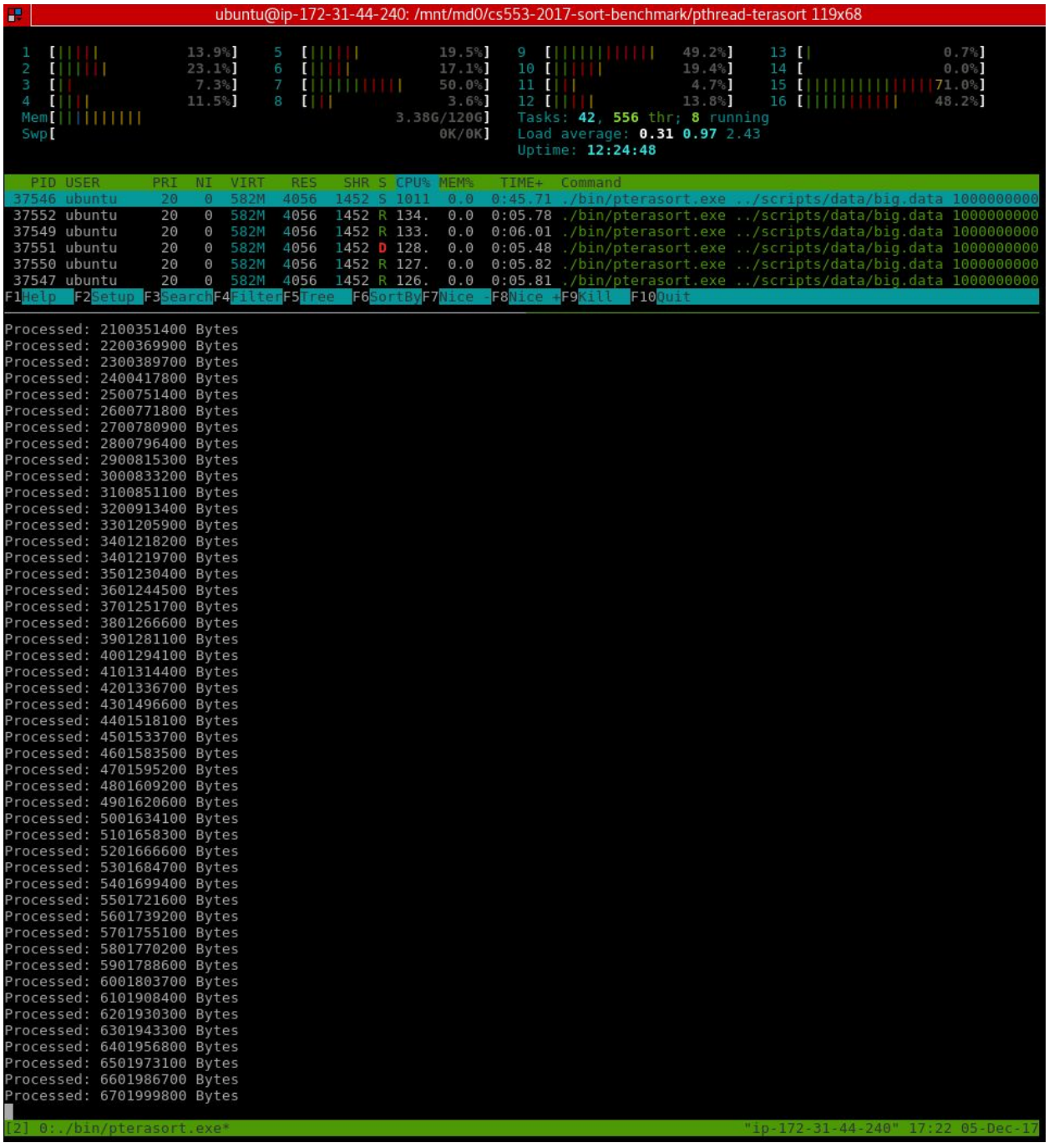


Figure 2.2.a Starting the shared memory application for 1TB.

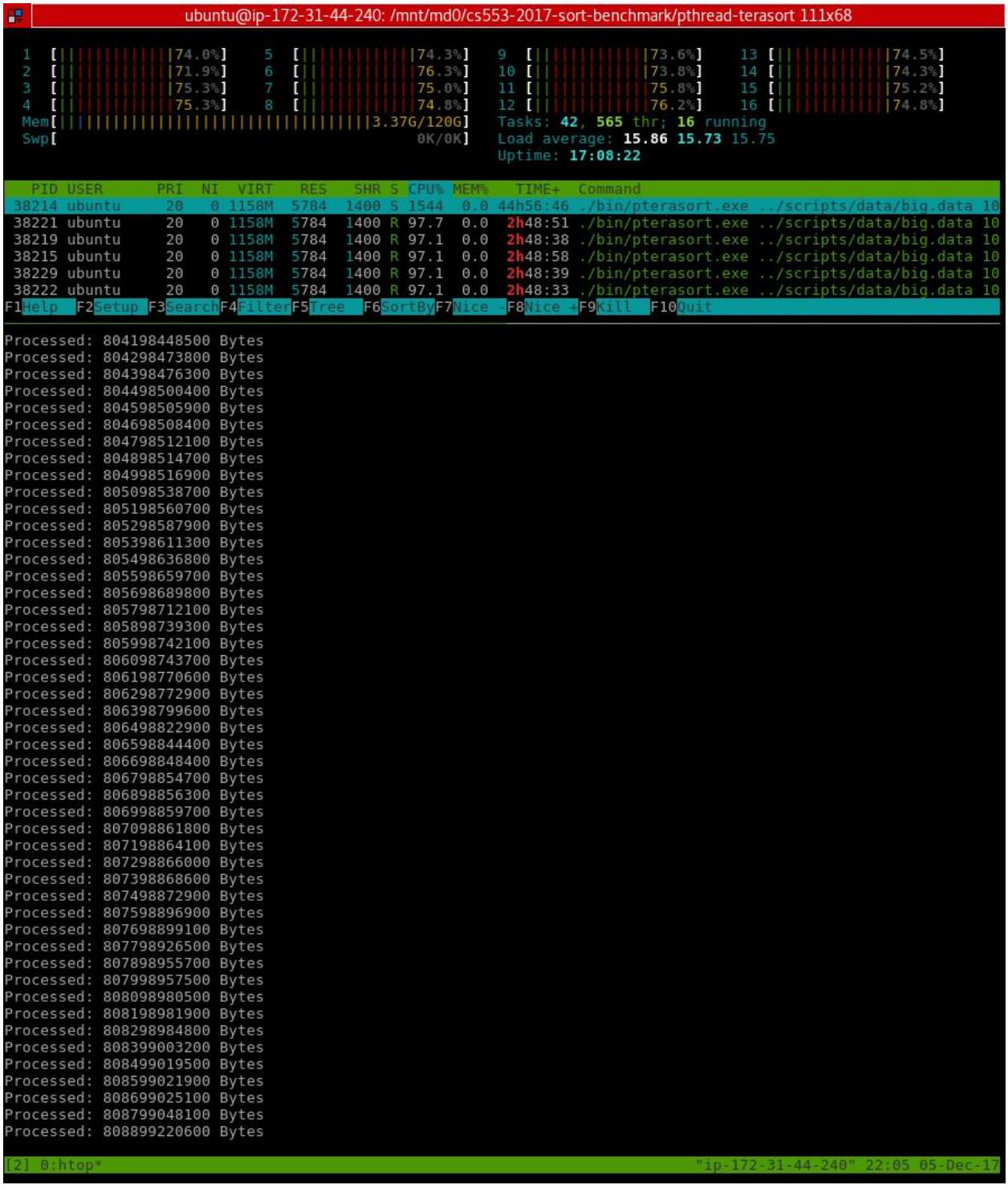


Figure 2.2.b Shared memory application running midway for 1TB.

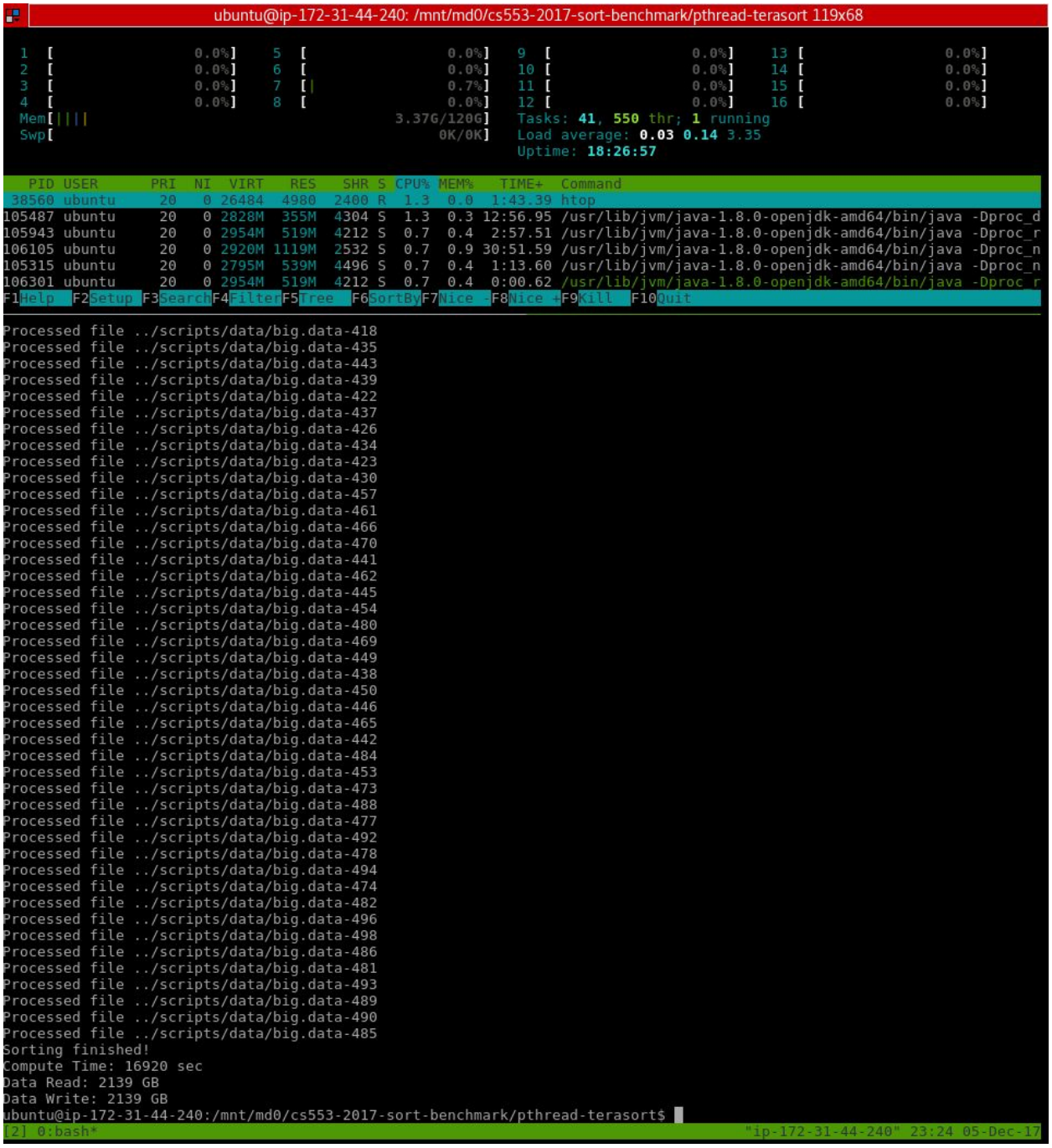
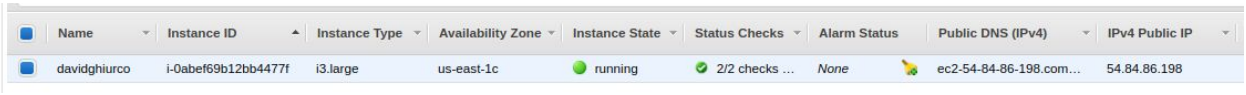


Figure 2.2.c Shared memory application finishing the sort experiment for 1TB.

3. Apache Hadoop MapReduce Implementation

3.1 Virtual Cluster (1-node i3.large)



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
davidghiurco	i-0abef69b12bb4477f	i3.large	us-east-1c	running	2/2 checks ...	None	ec2-54-84-86-198.com...	54.84.86.198

Here is the teragen job that created the 128 GB dataset:

```
17/12/05 05:09:06 INFO mapreduce.Job: Job job_1512448939295_0001 completed successfully
17/12/05 05:09:07 INFO mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=241526
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=170
                HDFS: Number of bytes written=128000000000
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
        Job Counters
                Launched map tasks=2
                Other local map tasks=2
                Total time spent by all maps in occupied slots (ms)=2727749
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=2727749
                Total vcore-milliseconds taken by all map tasks=2727749
                Total megabyte-milliseconds taken by all map tasks=2793214976
        Map-Reduce Framework
                Map input records=1280000000
                Map output records=1280000000
                Input split bytes=170
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=20895
                CPU time spent (ms)=2178840
                Physical memory (bytes) snapshot=559861760
                Virtual memory (bytes) snapshot=3948421120
                Total committed heap usage (bytes)=382205952
        org.apache.hadoop.examples.terasort.TeraGen$Counters
                CHECKSUM=2748814493656638320
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=128000000000
```

The Hadoop application was run with 2 Reducer tasks, one for each VCPU:

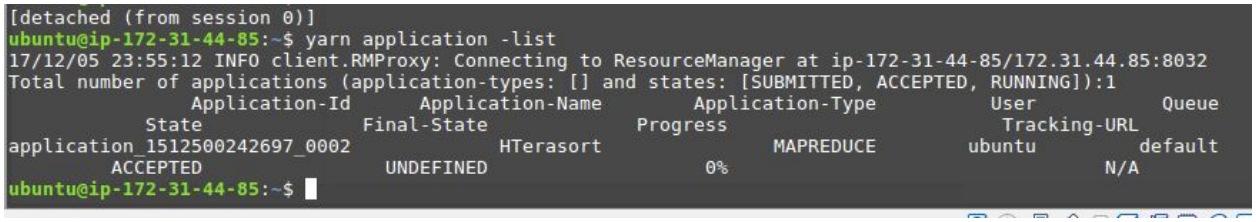
```
ubuntu@ip-172-31-36-148:/mnt/storage/cs553-2017-sort-benchmark$ hadoop jar hadoop-terasort/build/libs/htera.jar $
Dmapred.reduce.tasks=2 /input /output
17/12/05 06:26:15 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-36-148/172.31.36.148:8032
17/12/05 06:26:15 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implemen$
 the Tool interface and execute your application with ToolRunner to remedy this.
17/12/05 06:26:16 INFO input.FileInputFormat: Total input paths to process : 2
Spent 238ms computing base-splits.
Spent 6ms computing TeraScheduler splits.
17/12/05 06:26:16 INFO mapreduce.JobSubmitter: number of splits:954
17/12/05 06:26:17 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1512455154927_0001
17/12/05 06:26:17 INFO impl.YarnClientImpl: Submitted application application_1512455154927_0001
17/12/05 06:26:17 INFO mapreduce.Job: The url to track the job: http://ip-172-31-36-148:8088/proxy/application_1$
12455154927_0001/
17/12/05 06:26:17 INFO mapreduce.Job: Running job: job_1512455154927_0001
17/12/05 06:26:25 INFO mapreduce.Job: Job job_1512455154927_0001 running in uber mode : false
17/12/05 06:26:25 INFO mapreduce.Job:  map 0% reduce 0%
17/12/05 06:27:26 INFO mapreduce.Job:  map 1% reduce 0%
17/12/05 06:28:49 INFO mapreduce.Job:  map 2% reduce 0%
17/12/05 06:30:10 INFO mapreduce.Job:  map 3% reduce 0%
```

Unfortunately, our Hadoop program never terminates on large datasets.
Although sorting was verified on small, test datasets, the job never terminates on the dataset magnitudes required for the assignment.

```
17/12/05 21:12:05 INFO mapreduce.Job:  map 100% reduce 35%
17/12/05 21:12:20 INFO mapreduce.Job:  map 100% reduce 36%
17/12/05 21:12:38 INFO mapreduce.Job:  map 100% reduce 37%
17/12/05 21:13:05 INFO mapreduce.Job:  map 100% reduce 38%
17/12/05 21:13:20 INFO mapreduce.Job:  map 100% reduce 39%
17/12/05 21:13:41 INFO mapreduce.Job:  map 100% reduce 40%
17/12/05 21:14:02 INFO mapreduce.Job:  map 100% reduce 41%
17/12/05 21:14:20 INFO mapreduce.Job:  map 100% reduce 42%
17/12/05 21:14:44 INFO mapreduce.Job:  map 100% reduce 43%
17/12/05 21:15:02 INFO mapreduce.Job:  map 100% reduce 44%
17/12/05 21:15:20 INFO mapreduce.Job:  map 100% reduce 45%
```

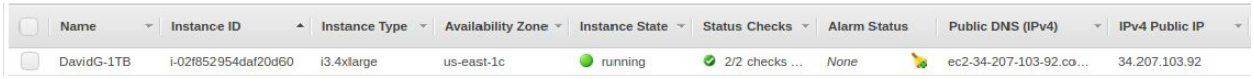
The reduce phase gets stuck at either 33% or 45% (in this case). We suspect this has to do with the infrastructure stack that we are using. Hadoop is not optimized to run on virtual machines (like the simple Ubuntu AMI we are using on Amazon) and as such, Apache recommends either running Hadoop baremetal, or using specialized VM instances (those AMIs are both not free, and also explicitly forbidden by the assignment).

Additionally, the ResourceManager from Yarn indicates that the application is no longer running:



This happens on the other Hadoop deployments/datasets as well. We were unable to debug it.

3.2 Virtual Cluster (1-node i3.4xlarge)



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
DavidG-1TB	i-02f852954daf20d60	i3.4xlarge	us-east-1c	running	2/2 checks ...	None	ec2-34-207-103-92.co...	34.207.103.92

Here is the teragen job that created the 1 TB dataset

```
17/12/05 09:05:07 INFO mapreduce.Job: Job job_1512458895359_0001 completed successfully
17/12/05 09:05:07 INFO mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=241818
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=173
                HDFS: Number of bytes written=1000000000000
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
        Job Counters
                Launched map tasks=2
                Other local map tasks=2
                Total time spent by all maps in occupied slots (ms)=9182381
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=9182381
                Total vcore-milliseconds taken by all map tasks=9182381
                Total megabyte-milliseconds taken by all map tasks=9402758144
        Map-Reduce Framework
                Map input records=10000000000
                Map output records=10000000000
                Input split bytes=173
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=96796
                CPU time spent (ms)=11138570
                Physical memory (bytes) snapshot=688504832
                Virtual memory (bytes) snapshot=4064907264
                Total committed heap usage (bytes)=343408640
        org.apache.hadoop.examples.terasort.TeraGen$Counters
                CHECKSUM=3028416809717741100
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=1000000000000
```

The Hadoop application was run with 16 Reducer tasks, one for each VCPU:

```
ubuntu@ip-172-31-44-240:/mnt/md0/cs553-2017-sort-benchmark$ hadoop jar hadoop-terasort/build/libs/htera.jar -Dmap
red.reduce.tasks=16 /input /output
17/12/05 09:08:30 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-44-240/172.31.44.240:8032
17/12/05 09:08:30 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement
 the Tool interface and execute your application with ToolRunner to remedy this.
17/12/05 09:08:30 INFO input.FileInputFormat: Total input paths to process : 2
Spent 235ms computing base-splits.
Spent 22ms computing TeraScheduler splits.
17/12/05 09:08:30 INFO mapreduce.JobSubmitter: number of splits:7452
17/12/05 09:08:30 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1512458895359_0002
17/12/05 09:08:31 INFO impl.YarnClientImpl: Submitted application application_1512458895359_0002
17/12/05 09:08:31 INFO mapreduce.Job: The url to track the job: http://ip-172-31-44-240:8088/proxy/application_15
12458895359_0002/
17/12/05 09:08:31 INFO mapreduce.Job: Running job: job_1512458895359_0002
17/12/05 09:08:36 INFO mapreduce.Job: Job job_1512458895359_0002 running in uber mode : false
17/12/05 09:08:36 INFO mapreduce.Job:  map 0% reduce 0%
```

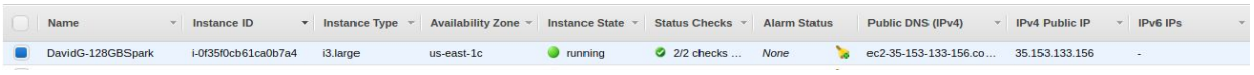
This job run also got stuck at reduce phase 33% and never terminated. We tried debugging it for a couple of days but have come up with nothing. This is the case with all our hadoop jobs in this assignment. Thus we are unable to get final performances results, although we have firm confirmation that the sort application WORKS on small datasets.

3.3 Virtual Cluster (8-nodes i3.large)

We have almost automated the entire multi-node deployment process. We spent a very long time writing the scripts, but we had a lot of trouble getting the system completely loaded. Multiple YARN NodeManager daemons constantly refused to load and we could not get a working cluster even though we got a quota increase for 8 i3.large nodes

4. Apache Spark Implementation

4.1 Virtual Cluster (1-node i3.large)



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs
DavidG-128GBSpark	i-0f35f0cb61ca0b7a4	i3.large	us-east-1c	running	2/2 checks ...	None	ec2-35-153-133-156.co...	35.153.133.156	-

Spark was run this instance with 2 partitions (for 2 VCPUs)

```
17/12/05 21:39:24 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:25 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:26 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:27 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:28 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:29 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:30 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:31 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:32 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:33 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:34 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:35 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:36 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:37 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:38 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:39 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:40 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:41 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:42 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:43 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:44 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:45 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:46 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:47 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:48 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:49 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:50 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:51 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:52 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:53 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:54 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:55 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:56 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:57 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:58 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:39:59 INFO yarn.Client: Application report for application_1512480559181_0004 (state: RUNNING)
17/12/05 21:40:00 INFO yarn.Client: Application report for application_1512480559181_0004 (state: FINISHED)
17/12/05 21:40:00 INFO yarn.Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: 172.31.43.252
         ApplicationMaster RPC port: 0
         queue: default
         start time: 1512504394705
         final status: SUCCEEDED
         tracking URL: http://ip-172-31-43-252:8088/proxy/application_1512480559181_0004/
         user: ubuntu
```

Here is the output of the spark application run on 128 GB

```
ubuntu@ip-172-31-43-252:/mnt/storage/cs553-2017-sort-benchmark$ hadoop fs -ls /output2
Found 2 items
-rw-r--r--   1 ubuntu supergroup 30665318899 2017-12-05 21:40 /output2/part-00000
-rw-r--r--   1 ubuntu supergroup 27283596802 2017-12-05 21:36 /output2/part-00001
```

Again, we have issues with the output format of the flushed sorted output in Spark. The output shown above is actually non-sensical binary when run on the 128 GB dataset.

However, we ran the same Spark application on Chameleon on a small data set and it sorted successfully:

```
cc@flamedragon:~/output$ rm *.sum
cc@flamedragon:~/output$ ./valsort -o 0.sum part-00000
Records: 62879
Checksum: 7b121ea3b17f
Duplicate keys: 0
SUCCESS - all records are in order
cc@flamedragon:~/output$ ./valsort -o 1.sum part-00001
Records: 65121
Checksum: 7ee759b68c09
Duplicate keys: 0
SUCCESS - all records are in order
cc@flamedragon:~/output$ cat 0.sum 1.sum > all.sum
cc@flamedragon:~/output$ ./valsort -s all.sum
Records: 128000
Checksum: f9f9785a3d88
Duplicate keys: 0
SUCCESS - all records are in order
cc@flamedragon:~/output$ S
```

Unfortunately, this means that we do not have conclusive performance data of our Spark application on large datasets.

4.2 Virtual Cluster (1-node i3.4xlarge)

4.3 Virtual Cluster (8-nodes i3.large)

We were unable to get a working multi-node deployment for Spark for the same reasons as above with the Hadoop clusters. We configured Spark to work with YARN, and YARN refused to cooperate on multi-node deployments.

5. Performance Evaluation

The shared memory implementation got some very interesting results overall. In the case of the 1TB dataset/configuration the application generated several segmentation faults caused by the inherent limit of open stream files for one program (approximately 2024) at the same time. In order to quickly fix this problem, files are opened and closed on demand, instead of being open throughout the split phase, thus contributing to a deficiency in performance. The efficiency of the application is pretty low, around at half, in contrast to the read and write throughput supported by these devices (over 600 MB/s sequentially). The I/O system but also the bus were stressed throughout the shared memory execution, and due to un-optimized and unorganized data movement, the application achieved low efficiency. From the scalability point of view, the weak speedup shows that the application scales well on multi-core systems.

Since we managed to run only single node experiments, in this regard the shared memory implementation outperformed Hadoop and Spark, being able to sort 32GB in less than 2 minutes, while the latter were able to do it in tens of minutes and almost ten minutes respectively. On one node the shared memory implementation scales even better than Hadoop and Spark, making use of the shared memory capabilities, without having to merge results or intermediary data, since it was already shared. ON 8 nodes, probably Hadoop and Spark would have the advantage since they support distributed communication and synchronization. Moreover we have observed that the more nodes Hadoop had the faster it ran applications. Experimentally, it took more time to generate 128GB of data on a single node than to generate 1TB of data on 5 nodes.

Experiment (instance/dataset)	Shared Memory TeraSort	Hadoop TeraSort	Spark TeraSort
Compute Time (sec) [1xi3.large 128GB]	3093		
Data Read (GB) [1xi3.large 128GB]	383		
Data Write (GB) [1xi3.large 128GB]	383		
I/O Throughput (MB/sec) [1xi3.large 128GB]	247.66		
Compute Time (sec) [1xi3.4xlarge 1TB]	16920		
Data Read (GB) [1xi3.4xlarge 1TB]	2139		
Data Write (GB) [1xi3.4xlarge 1TB]	2139		
I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]	252.83		
Compute Time (sec) [8xi3.large 1TB]	N/A		

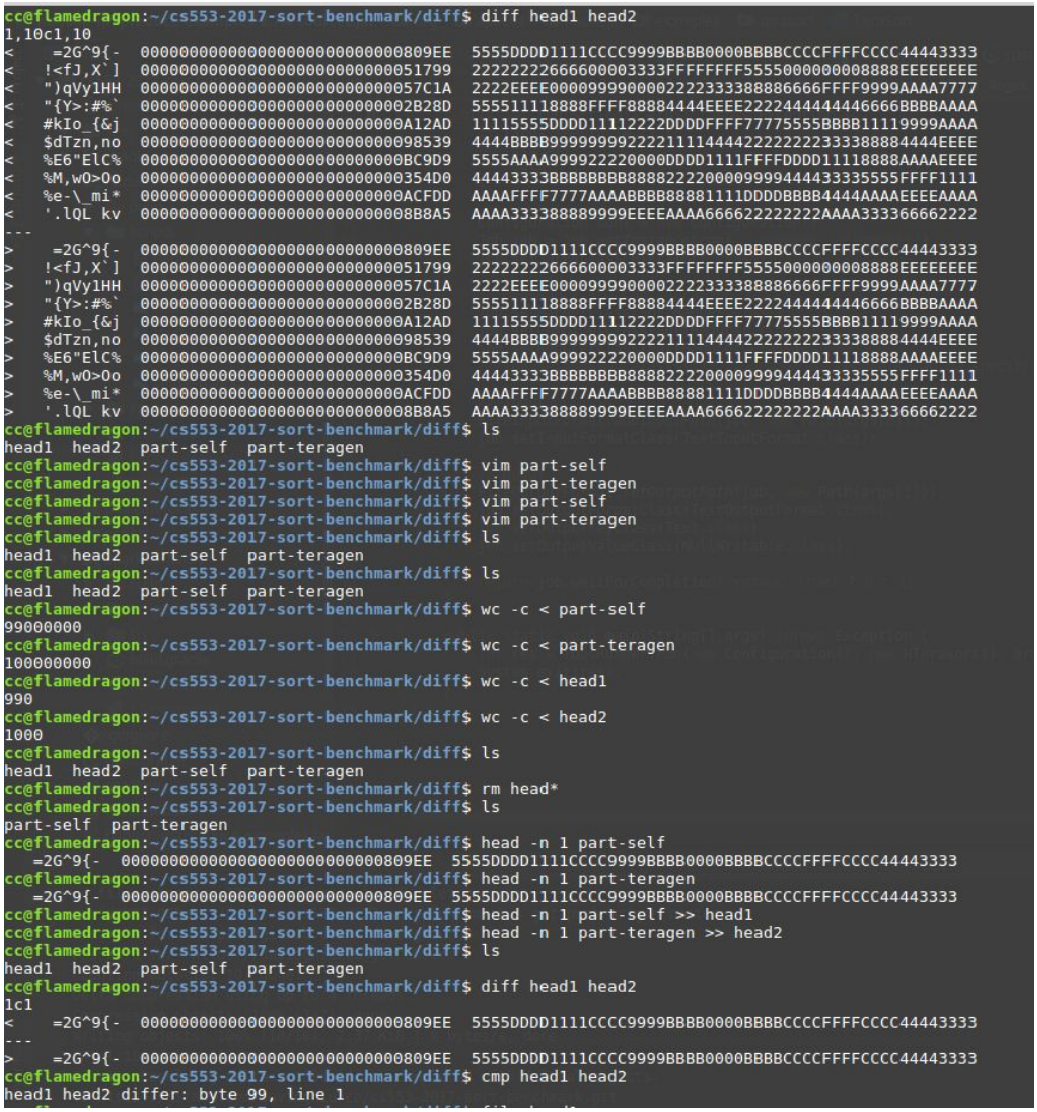
Data Read (GB) [8xi3.large 1TB]	N/A		
Data Write (GB) [8xi3.large 1TB]	N/A		
I/O Throughput (MB/sec) [8xi3.large 1TB]	N/A		
Speedup (weak scale)	1.02		
Efficiency (weak scale)	40-50%		

Table : Performance evaluation of TeraSort

6. Difficulties Encountered

Line Endings

One of us spent a considerate amount of time debugging the sort validator. A quick look at the output of our implementation would suggest that our output was indeed sorted alphabetically However the validator thought otherwise. It took some time to track down the culprit: LF vs CRLF line ending. As an example, below are screenshots of 2 sorted files. One produced by our implementation (which correctly writes files with LF endings, on the Linux platform) and the other is produced by Hadoop TeraGen.



```
cc@flamedragon:~/cs553-2017-sort-benchmark/diff$ file head1
head1: ASCII text
cc@flamedragon:~/cs553-2017-sort-benchmark/diff$ file head2
head2: ASCII text, with CRLF line terminators
cc@flamedragon:~/cs553-2017-sort-benchmark/diff$ '
```

Hadoop TeraGen documentation indicates that this is the format for each row in the generated data:

`<10 bytes key><10 bytes rowid><78 bytes filler>\r\n`

Experimentation determined that this is the case for gensort as well.

We fixed this issue by using the TeraInputFormat & TeraOutputFormat classes that are part of the Apache TeraSort example library. This way, the file formats of our jobs are exactly like our validators (teravalidate and/or valsort) expect. Note, the sort implementation is still our own, and does not have any dependency to TeraSort directly.

These line differences and slight variations in file formats were causing the following vsort validation problems:

```
cc@flamedragon:~/cs553-2017-sort-benchmark/diff$ ./valsort part-tera
Records: 100
Checksum: 365ed3f3e1
Duplicate keys: 0
SUCCESS - all records are in order
cc@flamedragon:~/cs553-2017-sort-benchmark/diff$ ./valsort part-self
sump_pump_fatal_error: pfunc_get_rec: partial record of 8 bytes found at end of input
```

In the end, we ended up not using the Tera formats anymore because we suspected that they were causing our hang issues on Hadoop and Spark. It ended up not mattering, because even after we reverted the changes, the jobs still refuse to complete.

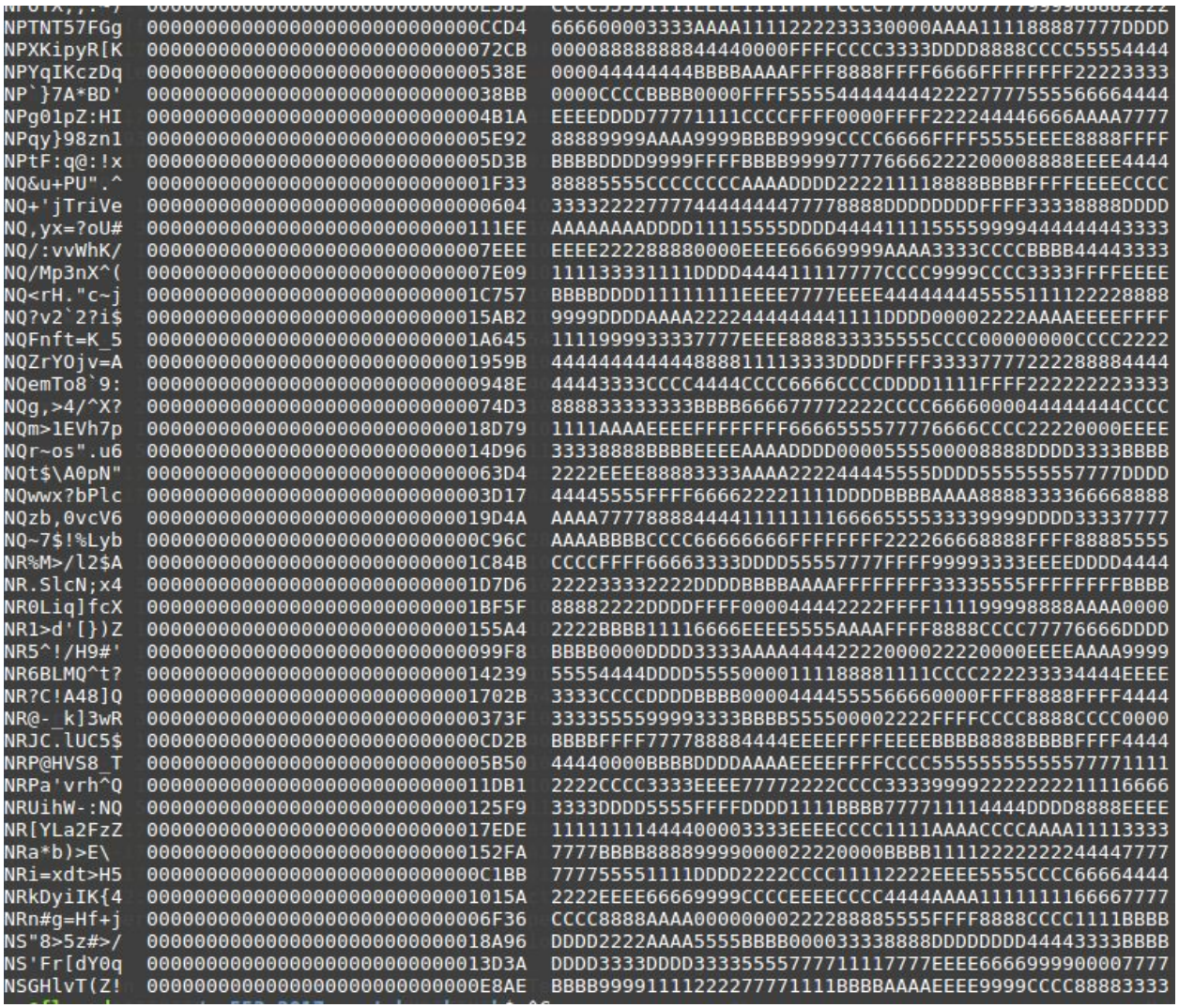
Hadoop & Spark Infrastructure management

Both our Hadoop and Spark implementations were tested on small datasets that would not take a long time to complete. However, when running on large datasets, a lot of errors were encountered, too many to screenshot here.

Multinode deployment was also a nightmare. We couldn't get it to work after full days of trying to configure and tune it to work with out applications (or any applications in general).

Spark sort validation

Our implementation clearly sorts the rows produced by gensort correctly:



This output is from the output directory of the spark job running on top of HDFS.

We tried running the TeraValidate validator, but we get the following error(s):

```
cc@flamedragon:~/cs553-2017-sort-benchmark$ hadoop jar hadoop-mapreduce-examples.jar teravalidate /output /report
17/12/05 12:32:21 INFO client.RMProxy: Connecting to ResourceManager at flamedragon/127.0.1.1:8032
17/12/05 12:32:21 INFO input.FileInputFormat: Total input paths to process : 2
Spent 28ms computing base-splits.
Spent 2ms computing TeraScheduler splits.
17/12/05 12:32:21 INFO mapreduce.JobSubmitter: number of splits:2
17/12/05 12:32:21 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1512463451507_0024
17/12/05 12:32:22 INFO impl.YarnClientImpl: Submitted application application_1512463451507_0024
17/12/05 12:32:22 INFO mapreduce.Job: The url to track the job: http://flamedragon:8088/proxy/application_1512463
451507_0024/
17/12/05 12:32:22 INFO mapreduce.Job: Running job: job_1512463451507_0024
17/12/05 12:32:29 INFO mapreduce.Job: Job job_1512463451507_0024 running in uber mode : false
17/12/05 12:32:29 INFO mapreduce.Job:  map 0% reduce 0%
17/12/05 12:32:38 INFO mapreduce.Job:  map 100% reduce 0%
17/12/05 12:32:38 INFO mapreduce.Job: Task Id : attempt_1512463451507_0024_m_000000_0, Status : FAILED
Error: java.io.EOFException: read past eof
      at org.apache.hadoop.examples.terasort.TeraInputFormat$TeraRecordReader.nextKeyValue(TeraInputFormat.java:2
61)
      at org.apache.hadoop.mapred.MapTask$NewTrackingRecordReader.nextKeyValue(MapTask.java:556)
      at org.apache.hadoop.mapreduce.task.MapContextImpl.nextKeyValue(MapContextImpl.java:80)
      at org.apache.hadoop.mapreduce.lib.map.WrappedMapper$Context.nextKeyValue(WrappedMapper.java:91)
      at org.apache.hadoop.mapreduce.Mapper.run(Mapper.java:145)
      at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:787)
      at org.apache.hadoop.mapred.MapTask.run(MapTask.java:341)
      at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
      at java.security.AccessController.doPrivileged(Native Method)
      at javax.security.auth.Subject.doAs(Subject.java:422)
      at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1746)
      at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)

Container killed by the ApplicationMaster.
Container killed on request. Exit code is 143
Container exited with a non-zero exit code 143

17/12/05 12:32:38 INFO mapreduce.Job: Task Id : attempt_1512463451507_0024_m_000001_0, Status : FAILED
Error: java.io.EOFException: read past eof
      at org.apache.hadoop.examples.terasort.TeraInputFormat$TeraRecordReader.nextKeyValue(TeraInputFormat.java:2
61)
      at org.apache.hadoop.mapred.MapTask$NewTrackingRecordReader.nextKeyValue(MapTask.java:556)
      at org.apache.hadoop.mapreduce.task.MapContextImpl.nextKeyValue(MapContextImpl.java:80)
      at org.apache.hadoop.mapreduce.lib.map.WrappedMapper$Context.nextKeyValue(WrappedMapper.java:91)
      at org.apache.hadoop.mapreduce.Mapper.run(Mapper.java:145)
      at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:787)
      at org.apache.hadoop.mapred.MapTask.run(MapTask.java:341)
      at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:164)
      at java.security.AccessController.doPrivileged(Native Method)
      at javax.security.auth.Subject.doAs(Subject.java:422)
      at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1746)
      at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:158)
```

The error has something to do with the file output format. We have been unable to debug this problem. We are certain that our sort is correct, but there is a very hard-to-debug problem with our output format and we are unsure how to debug it within Spark.

Therefore, we decided to take the more manual route of generating local data using gensort, copying it to HDFS, run the application, pull down the output directory back to the local filesystem, and run a validator there.

7. Conclusions

What conclusions can you draw? Which seems to be best at 1 node scale? How about 8 nodes? Can you predict which would be best at 100 node scale? How about 1000 node scales? Compare your results with those from the Sort Benchmark [9], specifically the winners in 2013 and 2014 who used Hadoop and Spark. Also, what can you learn from the CloudSort benchmark, a report can be found at [10].

2013, 1.42 TB/min

Hadoop

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

Their scaled results (adjusted for the number of nodes) is: 102.5 TB / 4328 seconds / 2100 nodes = 102500 GB / 4328 seconds / 2100 nodes = 0.1128 GB / s / node

2014, 4.27 TB/min

Apache Spark

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

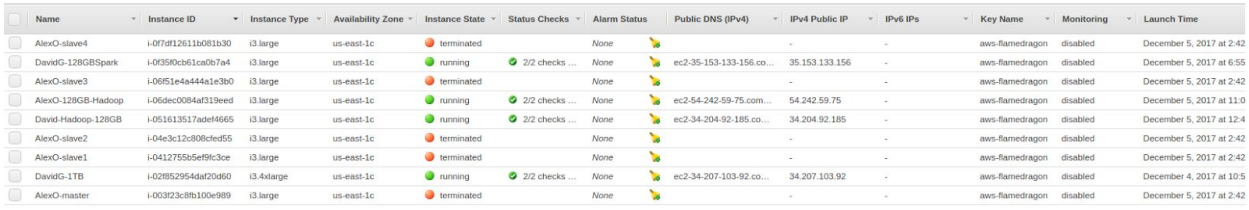
Their scaled results (adjusted for the number of nodes) is: 100 TB / 1406 seconds / 207 nodes = 100000 / 1406 seconds / 207 nodes = 0.3436 GB / s / node

As far as CloudSort is concerned, the report indicates that it is a benchmark used to test the cost of public clouds for computation on a fixed dataset size (e.g. 100 TB). The reason external sort is used is twofold: 1) It's easy to implement and 2) it is representative of many real world workloads, since it stresses various parts of the compute ecosystem: CPU, as well as OS, storage and network IO. The applications are simple and easy to port to other systems when

they get upgraded. The importance of CloudSort is also that it can be used to highlight the deficiencies in IO-intensive workloads in the virtualized cloud, which generally perform better on in-house data centers.

Screenshots

Here is the entirety of the instances we used on AWS throughout this assignment. Some have been terminated to save costs. But this is was the entirety of our ecosystem:



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name	Monitoring	Launch Time
AlexO-slave4	i-0f7df12611b081b30	i3.large	us-east-1c	terminated		None		-	-	aws-flamedragon	disabled	December 5, 2017 at 2:42
DavidG-128GBSpark	i-0f35f0cb61ca0b7a4	i3.large	us-east-1c	running	2/2 checks ...	None	ec2-35-153-133-156.co...	35.153.133.156	-	aws-flamedragon	disabled	December 5, 2017 at 6:55
AlexO-slave3	i-06f51e4a444a1e3b0	i3.large	us-east-1c	terminated		None		-	-	aws-flamedragon	disabled	December 5, 2017 at 2:42
AlexO-128GB-Hadoop	i-06dec0084af319eed	i3.large	us-east-1c	running	2/2 checks ...	None	ec2-54-242-59-75.com...	54.242.59.75	-	aws-flamedragon	disabled	December 5, 2017 at 11:0
David-Hadoop-128GB	i-051613517adef4665	i3.large	us-east-1c	running	2/2 checks ...	None	ec2-34-204-92-185.co...	34.204.92.185	-	aws-flamedragon	disabled	December 5, 2017 at 12:4
AlexO-slave2	i-04e3c12c808cfed55	i3.large	us-east-1c	terminated		None		-	-	aws-flamedragon	disabled	December 5, 2017 at 2:42
AlexO-slave1	i-0412755b5ef9fc3ce	i3.large	us-east-1c	terminated		None		-	-	aws-flamedragon	disabled	December 5, 2017 at 2:42
DavidG-1TB	i-02f852954daf20d60	i3.4xlarge	us-east-1c	running	2/2 checks ...	None	ec2-34-207-103-92.co...	34.207.103.92	-	aws-flamedragon	disabled	December 4, 2017 at 10:5
AlexO-master	i-003f23c8fb100e989	i3.large	us-east-1c	terminated		None		-	-	aws-flamedragon	disabled	December 5, 2017 at 2:42