HW6

Luke Logan
Gurunath Reddy

NOTE: We use the following notation throughout the report:

HadoopSort16G-1S means HadoopSort for 16GB dataset, 1 small.instance.
HadoopSort16G-4S means HadoopSort for 16GB dataset, 4 small.instances
HadoopSort16G-1L means HadoopSort for 16GB dataset, 1 large.instance.

Same for LinSort, MySort, and SparkSort.

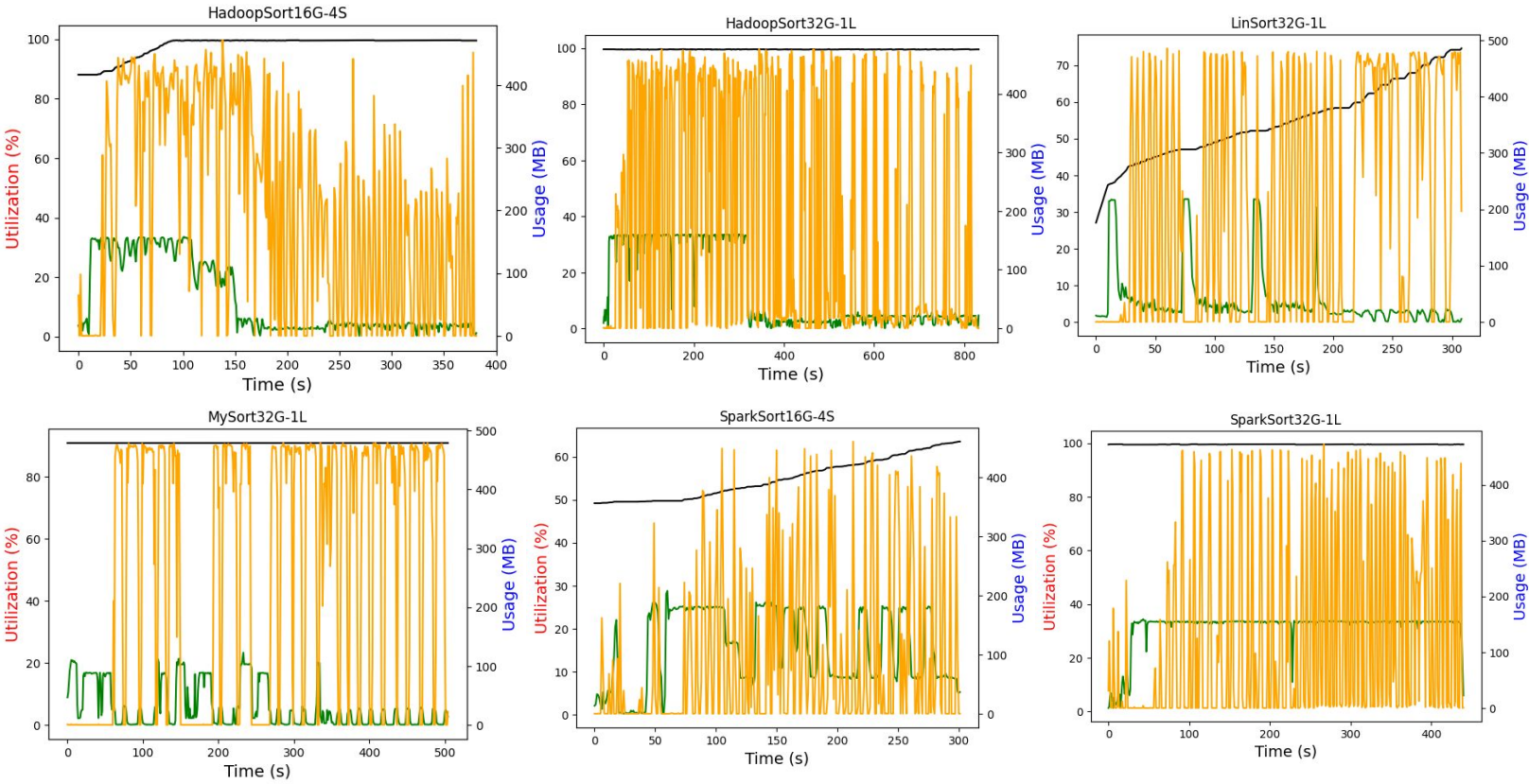

Figure 1: Resource Utilization for Different Sorting Algorithms

Yellow is disk usage (MB).
Green is CPU usage (%).
Black is RAM usage (%).

We collected statistics from the host machine running all virtual machines using sar, which is visualized in these figures. We see that MySort, SparkSort, and HadoopSort are fully utilizing memory, while LinSort takes some time to get there. We also see that LinSort, HadoopSort, and SparkSort exhibit far more spikey behavior than MySort does. We notice in the Hadoop figures, that about halfway through, the CPU utilization turns to nearly 0. This is because we are switching from the map phase to shuffling and reducing, causing more I/O. Furthermore, we see that SparkSort has a high average CPU utilization and high memory utilization consistently through its runtime. Spark stores some intermediate data in memory to perform operations on, so this isn't surprising. However, the average CPU utilization seems to be higher than with LinSort and MySort. This could be due to scheduling overheads, garbage collection, and String manipulation costs that MySort and LinuxSort do not have to deal with. Overall, we see that LinSort has the highest performance of the different workloads for the 1L instance. It writes less data to storage than MySort and has at least the same amount of CPU utilization as MySort.

Experiment	Shared Memory / mysort	Linux	Hadoop Sort	Spark Sort
1 small.instance, 1GB *dataset*	0m3.285s	0m8.532s	0m57.022s	1m16.912s
1 small.instance, 4GB *dataset*	0m13.046s	0m42.572s	2m51.480s	2m35.140s
1 small.instance, 16GB *dataset*	3m47.703s	3m18.362s	10m50.524s	10m5.400s
1 large.instance, 1GB *dataset*	0m2.341s	0m7.364s	0m42.593s	0m39.479s
1 large.instance, 4GB *dataset*	0m9.276s	0m26.966s	1m42.552s	1m11.658s
1 large.instance, 16GB *dataset*	0m51.150s	2m47.254s	6m34.378s	4m2.471s
1 large.instance, 32GB *dataset*	8m22.820s	5m9.413s	13m53.999s	7m20.942s
4 small.instances, 1GB *dataset*			0m36.926s	0m52.373s
4 small.instances, 4GB *dataset*			1m34.443s	1m23.832s
4 small.instances, 16GB *dataset*			6m20.510s	5m0.447s

Figure 2: Sorting Times

Which seems to be best at 1 node scale (1 large.instance)?

For different data-sets i.e 1GB, 4GB, 16GB, 32GB between Hadoop and Spark, SparkSort took less time to run than HadoopSort. However, Linux Sort always outperforms SparkSort for each dataset size. Linux Sort is the best for 1 large instance

Is there a difference between 1 small.instance and 1 large.instance?

Yes, all the data-sets are taking less real time for 1 large.instance than 1 small.instance in hadoop and spark.

How about 4 nodes (4 small.instance)? What speedup do you achieve with strong scaling between 1 to 4 nodes?

Strong scaling means we increase the size of the dataset as we increase the amount of concurrency.

When comparing 4GB-1S with 16GB-4S for Spark, we see a speedup of 0.52.
When comparing 4GB-1S with 16GB-4S for Hadoop, we see a speedup of 0.45.

In both cases, we see that the system doesn't scale well with problem size. It's better to sort 4GB of data on one node than it is to sort 16GB of data on 4 nodes. This makes sense considering that each of these write requests are to the same storage device. It is faster to write 4GB of data to a single storage device than it is to write 16GB of data to a single storage device.

What speedup do you achieve with weak scaling between 1 to 4 nodes?

Weak scaling means we increase the amount of concurrency while keeping the dataset size constant.

When comparing 16GB-1S with 16GB-4S for Spark, we see a speedup of 2.08.
When comparing 16GB-1S with 16GB-4S for Hadoop, we see a speedup of 1.69.

How about how many small.instances do you need with Spark to achieve the same level of performance as you did with your shared memory sort?

MySort executed in 3m47.703s (227.703) for the 16B-1S workload.
SparkSort executed in 10m5.400s (605.4s) for the 16GB-1S workload
SparkSort has a speedup of 2.08 every 4 nodes.

227.703 = 605.4 / (2.08 * x/4) -> x = 5.11

We would need 6 small.instances in order to get the same performance as MySort.

Can you draw any conclusions on the performance of the bare-metal instance performance from HW5 compared to the performance of your sort on a large instance through virtualization?

In HW5, Shared Memory Sort was able to sort 4GB of data in 0m20.011s. In this assignment, Shared Memory Sort was able to sort 4GB of data on the large instance in 0m9.276s. Virtualizing storage actually increased performance.

Can you predict which would be best if you had 100 small.instances? How about 1000? Compare your results with those from the Sort Benchmark (http://sortbenchmark.org), specifically the winners in 2013 and 2014.

For Spark, the winners of the contest were able to sort 100TB of data in 1,406 seconds.
For Hadoop, the winners of the contest were able to sort 102.5TB of data in 4,328 seconds.

For our SparkSort, we were able to sort 16GB of data in 10m5.400s (605.4s).
For our HadoopSort, we were able to sort 16GB of data in 10m50.524s (650.524s).

For SparkSort, we have a speedup of 0.52 every 4x increase in problem size and node count.
For HadoopSort, we have a speedup of .45 every 4x increase in problem size and node count.

Thus, by strong scaling, we could sort (100/4)*16 = 400GB of data with 100 small.instances in :

605.4*(1/.52)^(100/4) = 7,620,067,629 seconds

which is ~241 years. This number is even larger for the HadoopSort since it has a lower strong scaling factor. Thus, when compared to the winners of the contest, our results seem pitiful. However, our experiments were executed on a single machine, which explains the poor scaling.

Also, what can you learn from the CloudSort benchmark, a report can be found at (http://sortbenchmark.org/2014_06_CloudSort_v_0_4.pdf).

This benchmark factors in more than just performance, it factors economic cost of using the public cloud as the medium for sorting. It can be used as a measurement of the efficacy of different cloud platforms for I/O intensive workloads.