HW5

Luke Logan, Gurunath Reddy

4/13/2020

1 Report

The MySort program is able to sort a file in-place. Ie, no additional storage is required by this program. The user can customize the buffer size and the number of threads used. Our algorithm is a combination of even-odd sort, mergesort, and quicksort. It has a storage and memory complexity of O(1) and has a runtime complexity of O(n*lg(n)). This program works for any file size.

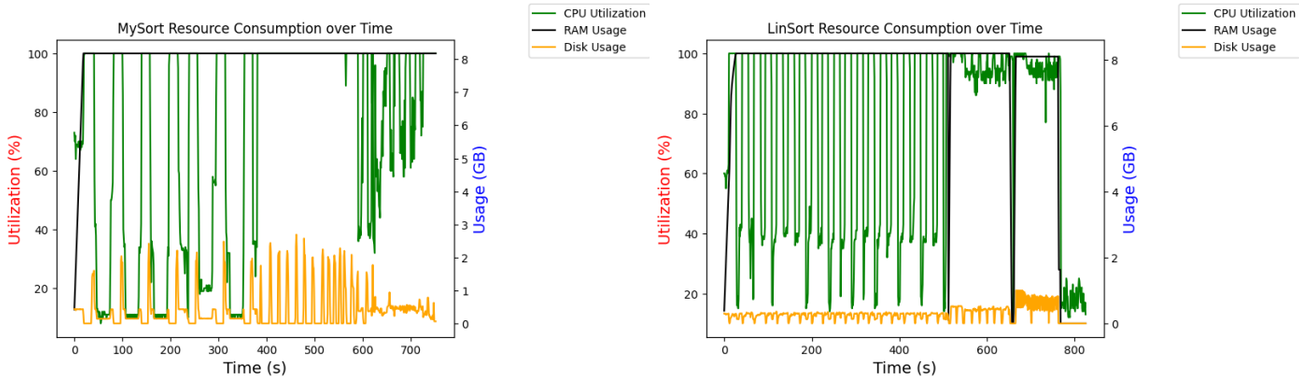
In order to benchmark this program, we used gensort to create datasets of varying sizes and then sorted the datasets using Linux Sort. We clear the disk cache before performing each sorting operation. We use pidstat to collect cpu, memory, and disk statistics during the runtime of the sorting programs. We use the Linux time program to record the amount of time the sorting program takes (which has an accuracy of ms). We store the output of time and pidstat in a log file and then parse it with Python. Before running the large benchmarks, we tested the performance of Linux Sort and MySort for various thread counts. We used datasets of size 1GB and buffers of size 100MB (in order to force external sort). We found that the number of threads that was optimal in both cases was 16 for this workload, and thus we used 16 threads for our test cases.

	Linux Sort (1GB)	MySort (1GB)	Linux Sort (4GB)	MySort (4GB)	Linux Sort (16GB)	MySort (16GB)	Linux Sort (64GB)	MySort (64GB)
Number of Threads	16	16	16	16	16	16	16	16
Sorting Algorithm	mergesort	quicksort	mergesort	quicksort	mergesort	quicksort	mergesort	quicksort
Sorting Approach	in_memory	in_memory	in_memory	in_memory	external	external	external	external
Data Read (GB)	0.97	0.95	3.9	3.9	15.62	15.63	62.49	62.5
Data Written (GB)	0.98	0.4	7.81	3.7	31.25	37.31	176.62	235.56
Sort Time (s)	11.021	5.07	50.02	20.011	200.038	166.019	827.074	754.075
Overall Throughput (MB/s)	176.92	267.91	234.24	379.83	234.31	318.82	289.10	395.26
Memory Utilization (GB)	1.31	0.81	4.28	2.98	6.19	7.72	7.35	8.08
CPU Utilization (%)	75.9	84.75	75.24	84.37	70.16	79.5	75.28	72.02

Table 1: Table

From Table 1, we see that MySort outperforms Linux Sort in the in-memory tests. For the 1GB and 4GB datasets, this is likely due to the performance of quicksort over mergesort. We see that the CPU is being about 10% more utilized in MySort as opposed to Linux Sort. However, for the 1GB test, we also see that the amount of data written according to pidstat is .4GB, which is about half of what Linux sort writes for this case. This is measurement error. The MySort program only runs for 5 seconds, and thus pidstat only reported 4 measurements, leaving the data being written in the last second out of the calculation. For the case of in-memory sort for MySort, our program must write at least as much as it has read, so this is not the reason for the performance boost.

From Table 1, we see that MySort outperforms Linux Sort in the external tests as well. We see that for both the 16GB and 64GB cases, the MySort program writes more data to SSD; and yet, it still achieves better performance. This is because our algorithm makes better use of storage and CPU. When looking at Figures 1a and 1b, we notice that the CPU has much less spikey behavior in MySort when compared to Linux Sort. Furthermore, we see that MySort reads in much more data per second than Linux Sort since the orange spikes in 1a are consistently higher than those in 1b. Linux Sort appears to make multiple small reads from SSD, resulting in significant latency penalties that cause the CPU to stall. In other words, our program experiences less delay due to small reads. MySort will read in 8GB of data in parallel and then perform sorting when all of the data has been loaded in. Since we read the 8GB of data in parallel all at once, our sorting algorithm experiences fewer delays, causing a net performance boost.



(a) (b)

Figure 1: Resource Utilization