# Cloud Computing CS553 Spring 20

# Homework 5

# Sort on Single Shared Memory Node

# April 13, 2020


# Team 11

# Shared Memory Sort Design

As part of the shared memory sort, we have implemented the following logic :

We assume that the RAM of the system is at least 8GB, so if we are sorting a file, which is less than 8GB, we can sort that file directly using the in-memory sort, which in our case is Quick Sort. However if a user intends to sort this small file using multiple threads we divide this file in smaller chunks(within the memory), apply quick sort individually and then merge them together into a single file.

If data size goes beyond 8GB, we opt for External sort. The way this works is we divide the data file into smaller temp files based on the number of threads and sort them individually using Quick Sort.

Users can choose the number of threads to be used for faster processing.

We spawn n threads at a time so that all thread handles data of 8GB at a go(Restricting memory usage below 8gb and also improving overall speed leveraging the power of hardware threads which in our case are 48), and this is repeated based on total temp files size. For example, if we have 16 temp files for 16GB we have 8 threads then processing will happen as follows:
8 threads handling 8GB(1GB each) at a go X 2 times = 16Threads

We wait for the spawned thread to complete before spawning a new batch of 8 threads to prevent additional RAM usage.

After this process we have sorted temp files of 1GB each.

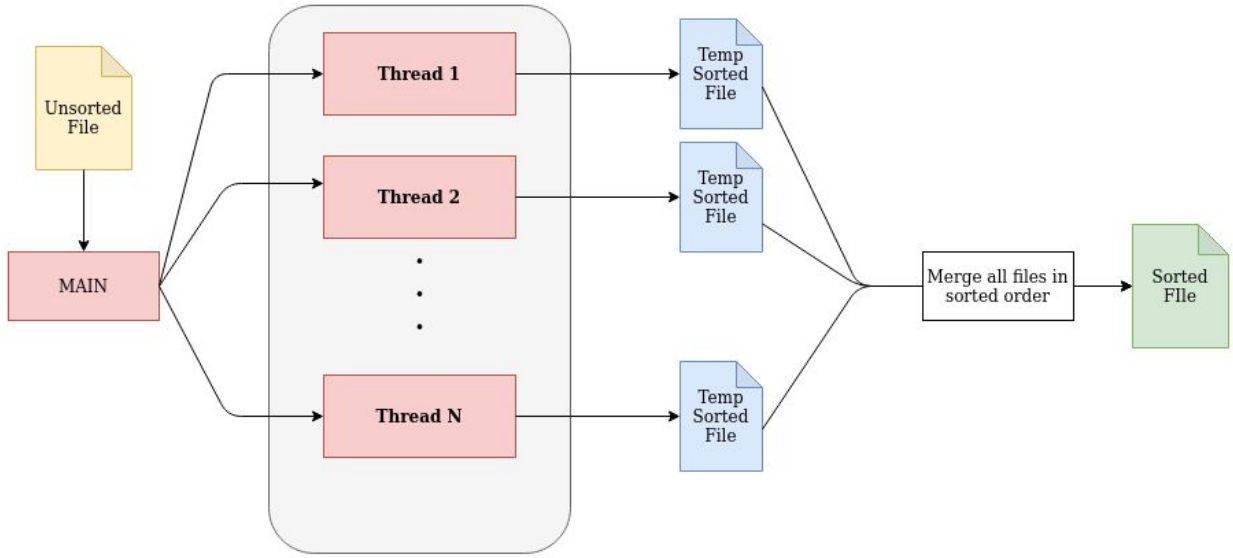


Fig 1. Code flow diagram

As we can see in the fig. Next step is to merge all this smaller temp files into a single file. We use the heap to do this. Initially, we put the 1st line from each file into Min Heap and pop the top element from the heap and write to the **final file**. We keep track of the file of which record/line is popped and put its next line into the heap.We repeat this process until our heap is empty, and thus we have a sorted file.

**Design Tradeoffs**

1) The viable number of threads is a tradeoff as we can only choose threads in the power of 2 i:e 1, 2, 4, 16 and 32. We can also choose 40 also as it perfectly divides the 8GB chunk and no loss of data is seen. If any number other than above mentioned is chosen, then there is possible data loss as the number won't divide the 8GB perfectly.
2) Limiting the RAM to 8GB is also one of the tradeoffs. How much RAM to be limited, is not done dynamically.

**Problem :-** The main problem is to sort the large data which can not fit the RAM. In these kinds of cases the external sort can be used to divide, sort and merge. First the data is divided into chunks and then chunks are sorted separately and finally merged to the main file.

**Methodology**

The methodology is simple: for smaller files less than 4GB the file is sorted as it is Using RAM, For files larger than 4GB it is the necessity to divide the according to the amount of RAM to be used at a given time. We are dividing the data such that 4GB data is sorted at any given time into temporary files. The temporary files can vary according to the number of threads given. For eg: if the number of threads are 8 then there would be 4-chunks incase of 16 GB file and each chunk has 8 files of 500 MB each. For 64 GB the data is divided into 16 chunks(4GB each) and each has files of 500 MB size. Once each chunk's files are created and sorted, all the files are merged into the main file and the temporary files are deleted.

**Runtime Environment settings**

| | |
|---|---|
| NODE TYPE | compute_skyake |
| #CPUs | 2 |
| NUMBER OF THREADS | 48 |
| PROCESSOR MODEL | Intel Xeon @2.60Ghz |
| STORAGE VENDOR AND MODEL | Samsung (MZ 7KM240HMHQ 0D3) |
| RAM | 192 |
| OPERATING SYSTEM | Ubuntu 18.04 |

**Results:**

| Exp | Shared Memory (1GB) | Linux Sort (1GB) | Shared Memory (4GB) | Linux Sort (4GB) | Shared Memory (16GB) | Linux Sort (16GB) | Shared Memory (64GB) | Linux Sort (64GB) |
|---|---|---|---|---|---|---|---|---|
| **# threads** | 8 | 32 | 8 | 16 | 32 | 16 | 32 | 32 |
| **Sort Approach** | In-memory | In-memory | In-memory | In-memory | External | External | External | External |
| **Sort Algorithm** | Quick Sort | Quick Sort | Quick Sort | Quick Sort | Quick Sort | Quick Sort | Quick Sort | Quick Sort |
| **Data Read (GB)** | 1.01 | 0.98 | 3.94 | 3.9 | 15.65 | 15.62 | 62.52 | 62.49 |
| **Data Write (GB)** | 0.98 | 0.95 | 3.73 | 7.52 | 31.09 | 31.25 | 124.36 | 170.95 |
| **Sort Time (Sec)** | 14 | 10 | 54 | 51 | 192 | 219 | 818 | 1084 |
| **Overall I/O Throughput (MB/sec)** | 141.59 | 174.03 | 141.96 | 212.11 | 243.37 | 214.03 | 228.47 | 199.47 |
| **Overall CPU Utilization (%)** | 96.5 | 84 | 99.06 | 86.56 | 98.2 | 86.5 | 99.41 | 87.04 |
| **Avg Memory Utilization (GB)** | 1.84 | 1.67 | 8 | 5.49 | 2.16 | 7.44 | 2.32 | 7.83 |

**Difference in performance:**

**1GB File:**

| | Shared Memory | Linux sort |
|---|---|---|
| **Sort Time(sec)** | 14 | 10 |
| **Overall I/O Throughput (MB/s)** | 141.59 | 174.03 |
| **Overall Cpu (%)** | 96.5 | 84 |
| **Average Memory Utilization(GB)** | 1.84 | 1.67 |

As seen from the table there is not much difference in the performance. The shared memory sort is slower as compared to Linux Sort. Our cpu usage is 96.5% percent and average memory utilization is also of 1.84 which is less than that of linux sort.

**4GB File:**

| | Shared Memory | Linux sort |
|---|---|---|
| **Sort Time(sec)** | 54 | 51 |
| **Overall I/O Throughput (MB/s)** | 141.96 | 212.11 |
| **Overall Cpu (%)** | 99.06 | 86.56 |
| **Average Memory Utilization(GB)** | 8 | 5.49 |

As seen from the table results are almost similar. Overall CPU and Memory utilization is higher in our case. However overall I/O throughput in case of Linux sort far better. So, they tend to balance each other and we get similar performance.

**16GB File:**

| | Shared Memory | Linux sort |
|---|---|---|
| **Sort Time(sec)** | 192 | 219 |
| **Overall I/O Throughput (MB/s)** | 243.37 | 214 |
| **Overall Cpu (%)** | 98.2 | 86.5 |
| **Average Memory Utilization(GB)** | 2.16 | 7.44 |

Shared memory sort performs faster than Linux sort for two reasons: 1) Overall CPU utilization is ~100 2) overall I/O throughput is more than that of Linux sort.

**64GB File:**

| | Shared Memory | Linux sort |
|---|---|---|
| **Sort Time(sec)** | 818 | 1084 |
| **Overall I/O Throughput (MB/s)** | 228.47 | 199.47 |
| **Overall Cpu (%)** | 99.41 | 87.04 |
| **Average Memory Utilization(GB)** | 2.32 | 7.83 |

Shared memory sort performs faster than Linux sort for two reasons: 1) Overall CPU utilization is ~100 2) overall I/O throughput is more than that of Linux sort.

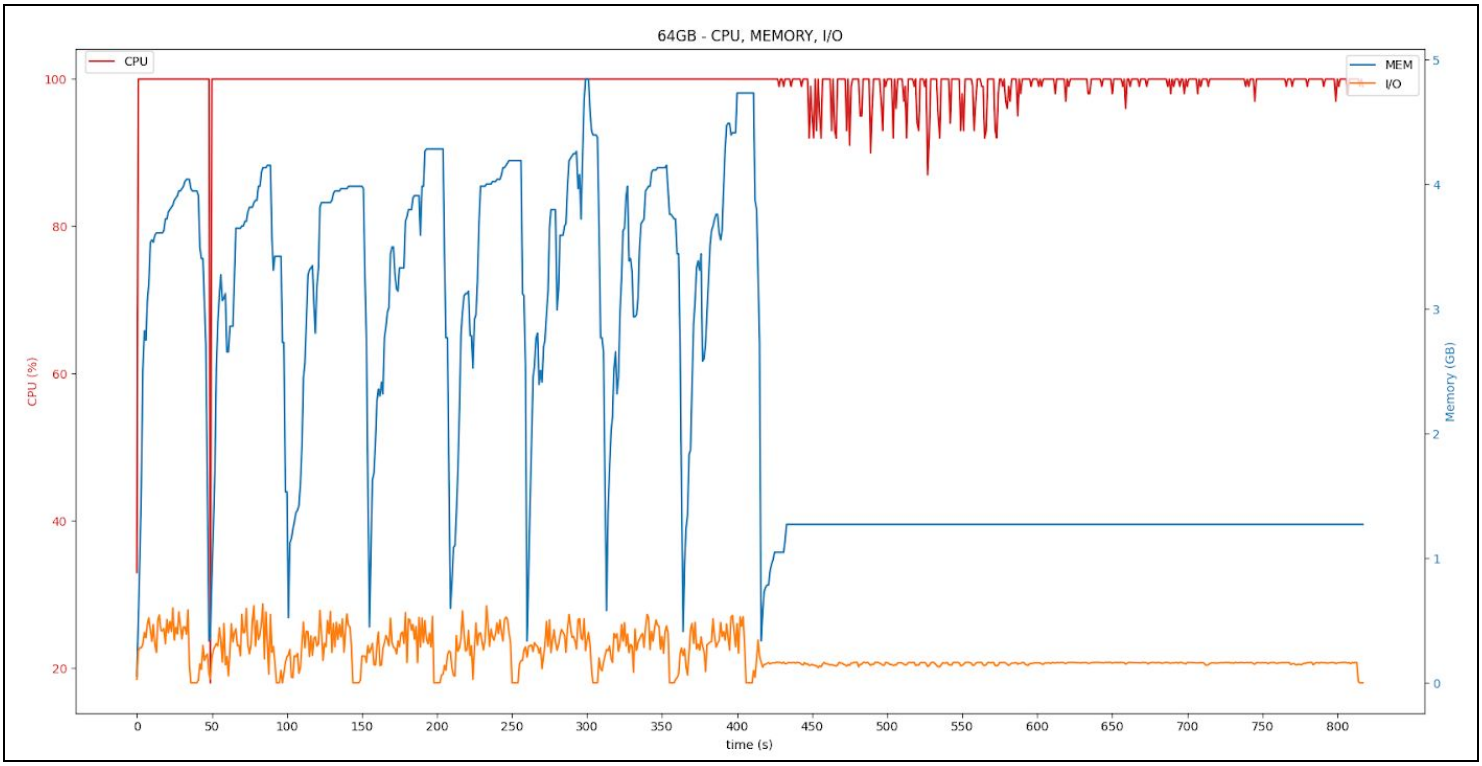


**Figure 2. CPU vs I/O vs Memory utilization Shared Memory**

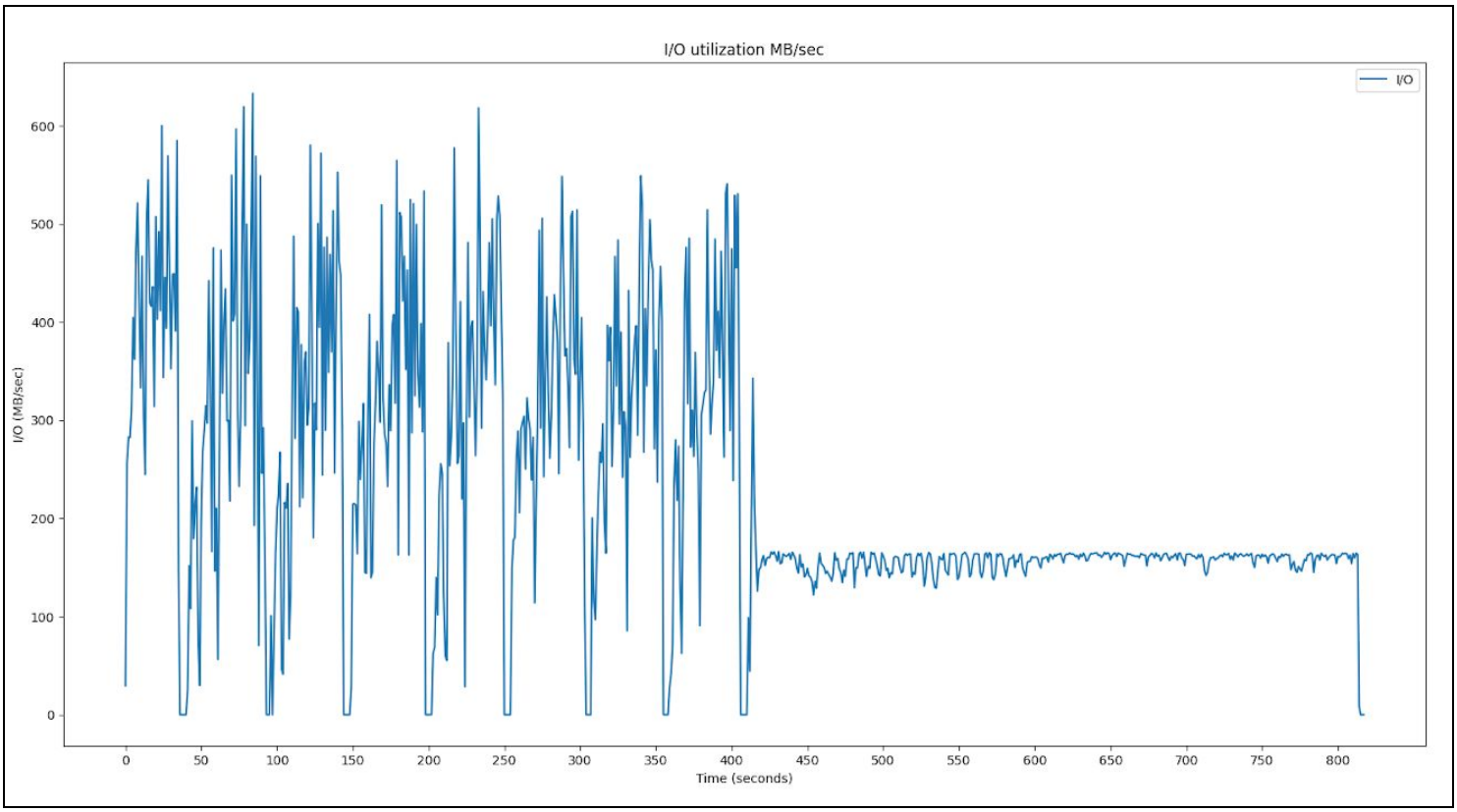


**Fig 3. Overall I/O utilization Shared Memory**

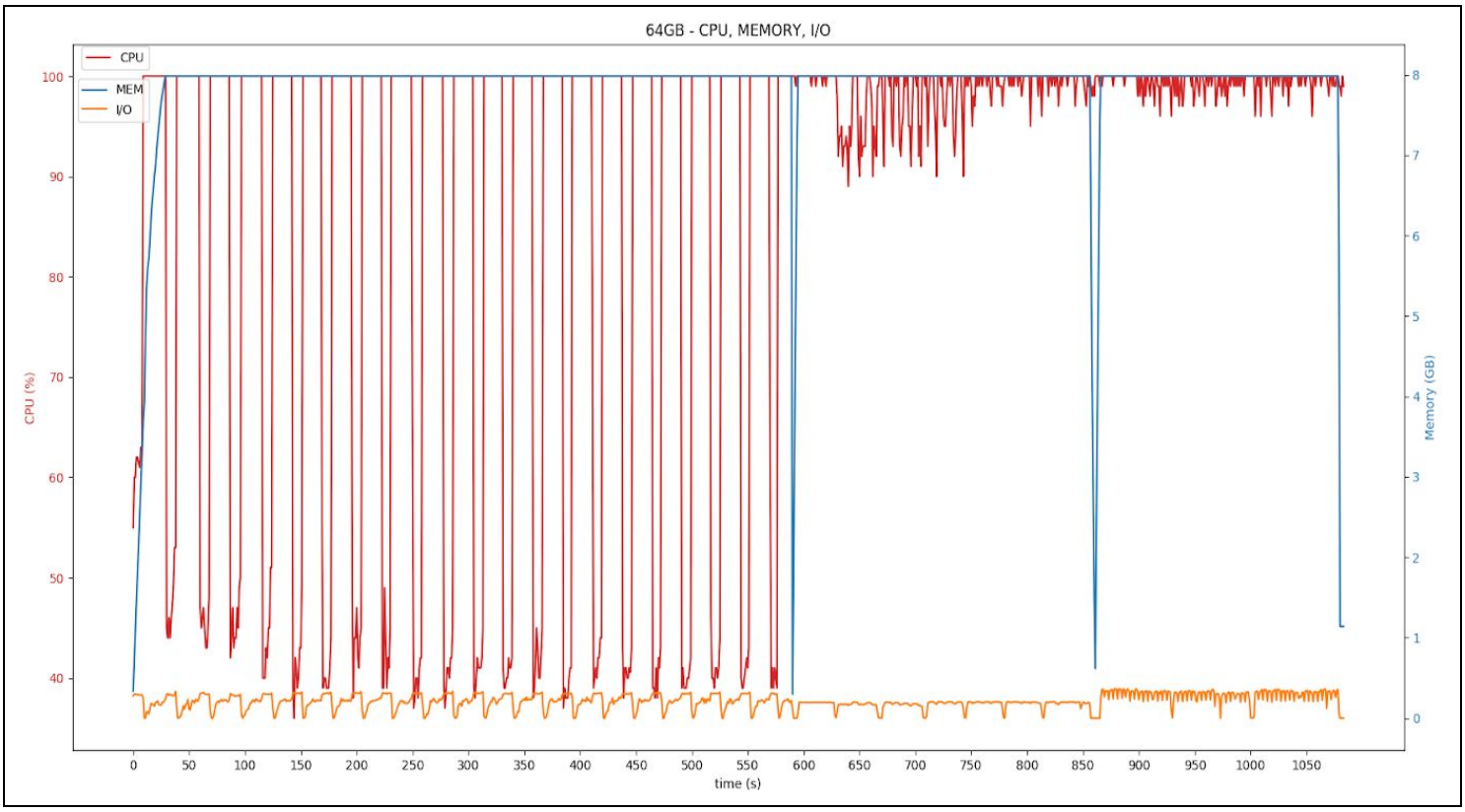


**Figure 4: CPU vs I/O vs Memory utilization Linux Sort**

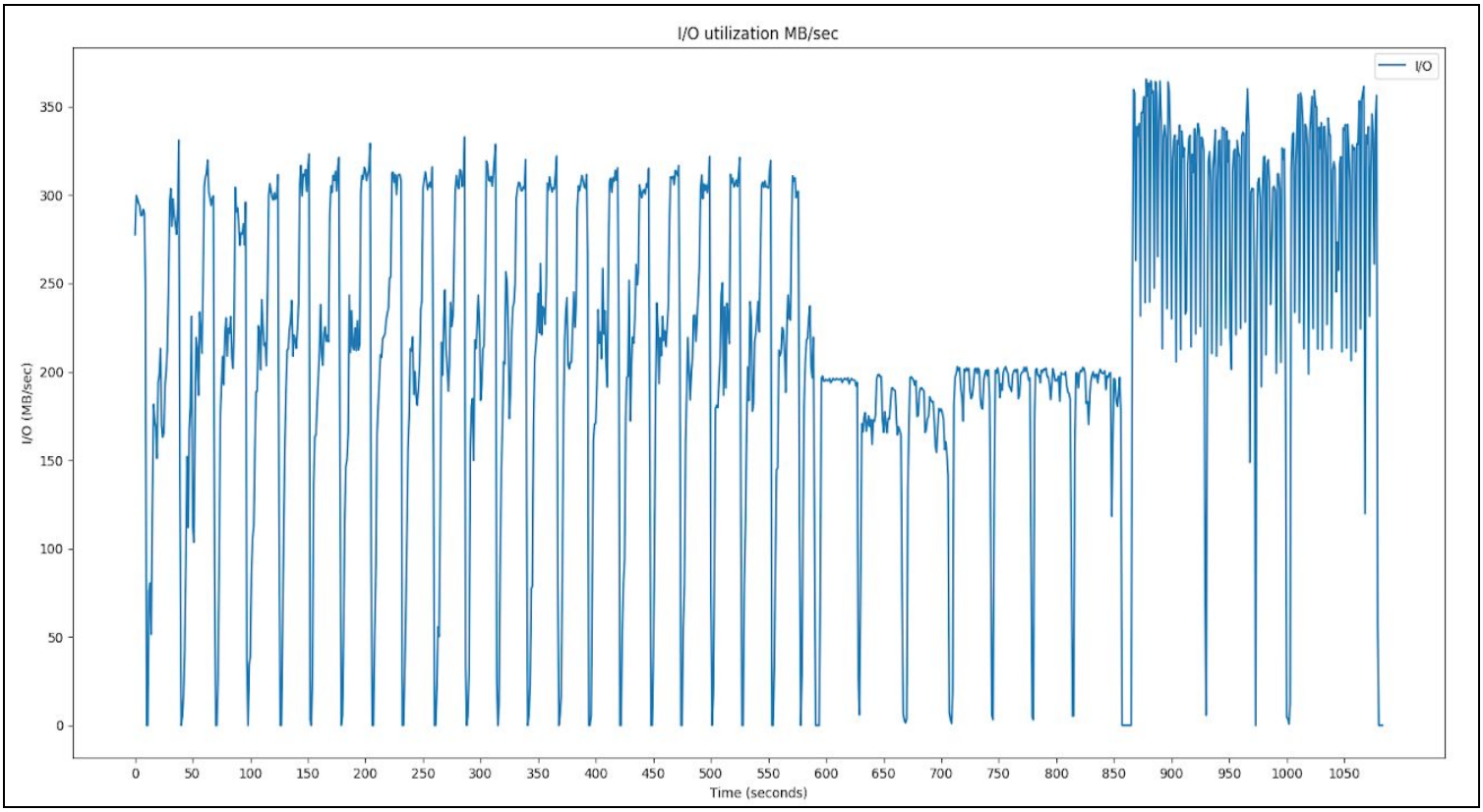


**Figure 5: Overall I/O utilization Linux Sort**

**Valsort logs:-**

**Valsort on Linux sort**

1GB:-
Records: 10000000
Checksum: 4c48a881c779d5
Duplicate keys: 0
SUCCESS - all records are in order

4GB:-
Records: 40000000
Checksum: 1312774ebf75c93
Duplicate keys: 0
SUCCESS - all records are in order

16GB:-
Records: 160000000
Checksum: 4c4a5084cc6403c
Duplicate keys: 0
SUCCESS - all records are in order

64GB:
Records: 640000000
Checksum: 1312cc6bda0f2ac4
Duplicate keys: 0
SUCCESS - all records are in order

**Valsort on Shared Memory sort**
1GB:
Records: 10000000
Checksum: 4c48a881c779d5
Duplicate keys: 0
SUCCESS - all records are in order

4GB:
Records: 40000000
Checksum: 1312774ebf75c93
Duplicate keys: 0
SUCCESS - all records are in order

16GB:
Records: 160000000
Checksum: 4c4a5084cc6403c
Duplicate keys: 0

SUCCESS - all records are in order

64GB:
Records: 640000000
Checksum: 1312cc6bda0f2ac4
Duplicate keys: 0
SUCCESS - all records are in order

**Linux memory logs**

**Plotting from file: 1GB_linux_sort_data_32T**
---
Exact sort time: 10 seconds
I/O utilization: 174.03 MB/sec
CPU utilization: 88.00%
Memory utilization: 1.67 GB
Data Read: 0.95
Data Write: 0.98
---

**Plotting from file: 4GB_linux_sort_data_16T**
Process id: 41028
Threads: 16
---
Exact sort time: 51 seconds
I/O utilization: 212.11 MB/sec
CPU utilization: 86.56%
Memory utilization: 5.49 GB
Data Read: 3.9
Data Write: 7.52
---

**Plotting from file: 16GB_linux_sort_data_16T**
Process id: 47998
Threads: 16
---
Exact sort time: 96 seconds
I/O utilization: 188.44 MB/sec
CPU utilization: 80.04%
Memory utilization: 6.99 GB
Data Read: 15.62
Data Write: 31.25
---

**Plotting from file: 64GB_linux_sort_data_32T**
Process id: 8191
Threads: 32
---
Exact sort time: 1084 seconds
I/O utilization: 215.35 MB/sec
CPU utilization: 89.36%
Memory utilization: 7.82 GB
Data Read: 62.49
Data Write: 170.95
---

**Shared sort logs**

**Plotting from file: 1GB_our_sort_data_8T**
Process id: 39890
Threads: 8
---
Exact sort time: 14 seconds
I/O utilization: 141.59 MB/sec
CPU utilization: 96.50%
Memory utilization: 1.84 GB
Data Read: 1.01
Data Write: 0.98
---

**Plotting from file: 4GB_our_sort_data_8T**
Process id: 41948
Threads: 8
---
Exact sort time: 54 seconds
I/O utilization: 141.96 MB/sec
CPU utilization: 99.06%
Memory utilization: 6.47 GB
Data Read: 3.94
Data Write: 3.73
---

**Plotting from file: 16GB_our_sort_data_32T**
Process id: 2933
Threads: 32
---
Exact sort time: 192 seconds
I/O utilization: 243.47 MB/sec
CPU utilization: 98.20%
Memory utilization: 2.16 GB
Data Read: 15.65
Data Write: 31.09
---

**Plotting from file: log_data/our_logs/64GB_our_sort_data_32T**
Process id: 23679
Threads: 32
---
Exact sort time: 818 seconds
I/O utilization: 228.47 MB/sec
CPU utilization: 99.41%
Memory utilization: 2.32 GB
Data Write: 124.36 GB
Data Read: 62.52 GB
---