Report

- **Problem:**

  The problem here is to sort a large data which varies from 2GB to 20GB in size and the environment that needs to be used is shared memory multi-threaded environment. Also known as external sort or Tera Sort.

- **Solution**:
  - As we know we cannot use in memory sort for such large data files as there is limited memory that we have, and such large files cannot be accommodated at once.
  - Hence, we use external sort algorithms to deal with such kind of sorting.
  - There are variety of external sort algorithms available like Two-Pass Multiway merge(TPMM) sort, merge sort, quicksort and many more.
  - We take the amount of data according to the available free memory and sort that much data and write it back to the temp file.

- **Methodology**:
  - Programming language used: **JAVA**
  - Algorithm used: **Two Pass Multiway Merge algorithm**.
  - Input:
    1. The input file is generated using gensort.
    2. Following command generates file with 1000 records where each record corresponds to 100 bytes each (98 bytes of data and 2 bytes for "\r\n").
       - gensort -a 1000 sample.txt
       - so the file size will be 1000*100 = 100000 bytes.
    3. Calculate the size of each chunk by appropriately adjusting the available free memory in JVM.
    4. Calculate the number of chunks by using formula:
       - Number of chunks = fileSize / chunkSIze.
    5. Now read the amount of data from the file as per the chunk size.
    6. Sort the data that is read above and store it to a temp file whose size will be equal to chunk size (obviously!)
    7. Keep repeating steps 5 and 6 until the entire file is read.
    8. Once you have sorted chunks available, use Multiway merge, where you pick up the smallest data from all the chunks and write that smallest one to the final output file.
    9. Keep repeating step 8 until all the chunks are read completely.
    10. Now, we have the final output file which is sorted one.

11. To validate the output, we make use of valsort and run the below command to validate whether your output file is sorted or not and also for duplicity check along with checksum.
    - valsort output.txt

- Multithreading is achieved on above steps where multiple threads can divide and sort the chunks to increase efficiency.
- Also, first 10 bytes of each line in the input file are considered as key and the sorting is done based on that part to increase sorting behavior.
- There are two read operations performed and two write operations performed on the disk.

- **Performance Evaluation**:

  - This section will describe the performance of mysort program as compared to linux sort program in shared memory system.
    - Number of threads: **4** (Best Performance achieved after trying several concurrency values)
    - Input file size:
      1. **2GB**
      2. **20GB**

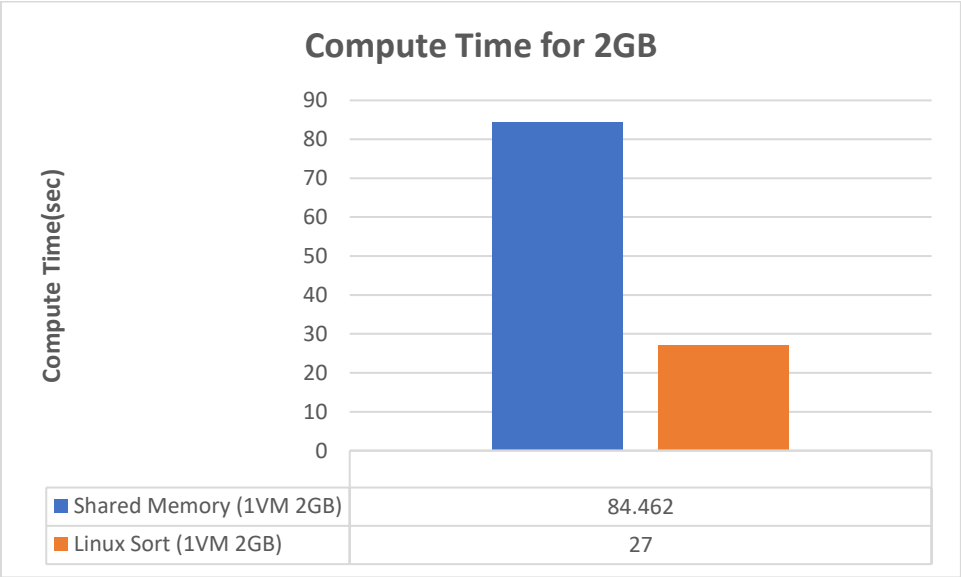
Experiment	Shared Memory (1VM 2GB)	Linux Sort (1VM 2GB)	Shared Memory (1VM 20GB)	Linux Sort (1VM 20GB)
Compute Time (sec)	84.462	27	867.871	395
Data Read (GB)	4	4	40	40
Data Write (GB)	4	4	40	40
I/O Throughput (MB/sec)	94.71715091	296.2962963	92.17959812	202.5316456

Table 1: Performance evaluation of TeraSort

Inference:

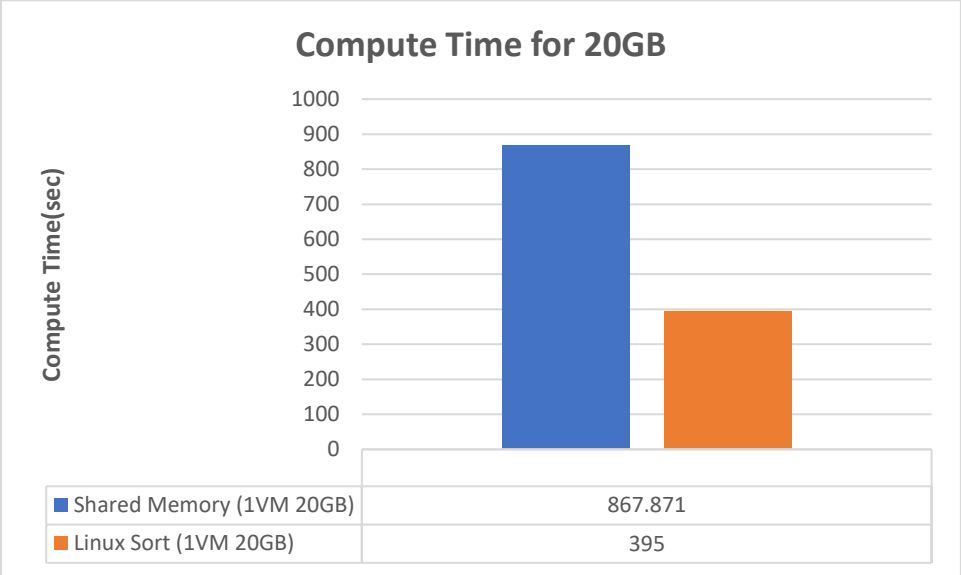
- From the above table, we can observe that compute time by linux sort is comparatively lower as compared to Shared Memory sort for both 2GB and 20GB.
- The number of reads is equal for shared memory and linux sort programs.
- Also, the number of writes is equal for linux sort and shared memory programs.
- The throughput of linux sort is comparatively higher as compared to shared memory sort.
- The throughput decreases as the file size increases for both Shared Memory sort and Linux sort.
- Both linux sort and shared memory sort performs same number of read and write operations.

- **Graphical Visualization**:



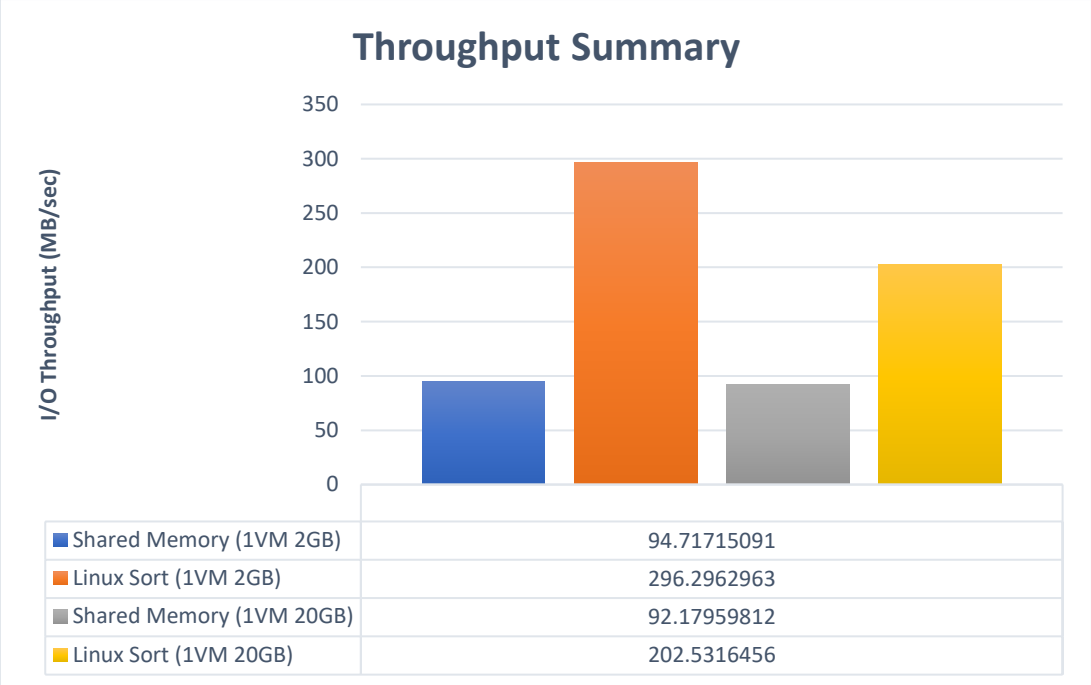
Compute Time: Shared Memory vs Linux Sort(2GB)

- ✓ The above graph represents the comparison between shared memory sort and linux sort for 2GB dataset.
- ✓ As we can clearly observe that linux sort takes very less time as compared to Shared Memory set.
- ✓ One of the reason linux sort is more faster the program is entirely multi-threaded based on the number of processors where the program is being executed.



Compute Time: Shared Memory vs Linux Sort(20GB)

- ✓ The above graph shows comparison between Shared Memory vs Linux sort for 20GB dataset.
- ✓ It represents that shared memory sort takes much more time as compared to Linux sort.
- ✓ Linux sort code base is highly multi-threaded whereas he shared-memory program is partially multi-threaded.



Throughput Summary: Linux Sort vs Shared Memory (2GB and 20GB)

- ✓ The above graph summarizes the Throughput of Linux Sort vs Shared Memory sort for various data input sizes.
- ✓ Linux sort on 2GB is the most fastest out of all the experiments that were executed.
- ✓ Shared Memory has throughputs which are quite near to each other.
- ✓ It also shows that as we increase the data size, the throughput decreases.

Output:

1. **MySort2GB**:

```
-rw-rw-r-- 1 rambani rambani     559 Apr 11 19:32 mysort20GB.log
-rw-rw-r-- 1 rambani rambani     550 Apr 11 19:35 mysort2GB.log
rambani@neutron:~/cs553-pa2a$ cat mysort2GB.log
File Size:2000000000 MemorySize: 250000000 Concurrency: 4
Available Free Memory: 125930704
Number of Chunks: 8 Chunk Size: 250000000
Splitting and sorting input file /input/data-2GB.in into chunks
Total time to sort and create chunks: 71.037
Number of temp files created: 8
Starting merging of sorted files...
Files are merged
Total time for external sort: 84.462 seconds
--------------------------------Valsort Results---------------------------------
Records: 20000000
Checksum: 98923e9cff98ac
Duplicate keys: 0
SUCCESS - all records are in order
rambani@neutron:~/cs553-pa2a$
```

2. **MySort20GB:**

```
-rw-rw-r-- 1 rambani rambani     550 Apr 11 20:35 mysort2GB.log
-rw-rw-r-- 1 rambani rambani     558 Apr 11 20:51 mysort20GB.log
rambani@neutron:~/cs553-pa2a$ cat mysort20GB.log
File Size:20000000000 MemorySize: 250000000 Concurrency: 4
Available Free Memory: 125930704
Number of Chunks: 80 Chunk Size: 250000000
Splitting and sorting input file /input/data-20GB.in into chunks
Total time to sort and create chunks: 636.806
Number of temp files created: 80
Starting merging of sorted files...
Files are merged
Total time for external sort: 867.871 seconds
--------------------------------Valsort Results---------------------------------
Records: 200000000
Checksum: 5f5cc94518a4203
Duplicate keys: 0
SUCCESS - all records are in order
rambani@neutron:~/cs553-pa2a$
```

3. Linsort2GB:

```
rambani@neutron:~/cs553-pa2a$ cat linsort2GB.slurm
#!/bin/bash

#SBATCH --job-name="linsort2GB"
#SBATCH --output="linsort2GB.log"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=compute

START_TIME=$SECONDS
LC_ALL=c sort /input/data-2GB.in > /tmp/data-2GB.out
ELAPSED_TIME=$(($SECONDS - $START_TIME))

valsort /tmp/data-2GB.out

echo "Total time for linux sort: $ELAPSED_TIME seconds" >> /exports/home/rambani/cs553-pa2a/linsort2GB.log

rambani@neutron:~/cs553-pa2a$ cat linsort2GB.log
Records: 20000000
Checksum: 98923e9cff98ac
Duplicate keys: 0
SUCCESS - all records are in order
Total time for linux sort: 27 seconds
rambani@neutron:~/cs553-pa2a$
```

4. Linsort20GB:

```
rambani@neutron:~/cs553-pa2a$ cat linsort20GB.slurm
#!/bin/bash

#SBATCH --job-name="linsort20GB"
#SBATCH --output="linsort20GB.log"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=compute

START_TIME=$SECONDS
LC_ALL=c sort /input/data-20GB.in > /tmp/data-20GB.out
ELAPSED_TIME=$(($SECONDS - $START_TIME))

valsort /tmp/data-20GB.out

echo "Total time for linux sort: $ELAPSED_TIME seconds" >> /exports/home/rambani/cs553-pa2a/linsort20GB.log

rambani@neutron:~/cs553-pa2a$ cat linsort20GB.log
Records: 200000000
Checksum: 5f5cc94518a4203
Duplicate keys: 0
SUCCESS - all records are in order
Total time for linux sort: 395 seconds
rambani@neutron:~/cs553-pa2a$
```

- **Challenges**:
    - The most important problem I faced was when my code was working all fine on window machine and valsort gave correct output but when I executed the code on linux, it gave me following error:
      "sump pump fatal error: pfunc_get_rec: partial record of 90 bytes found at end of input"
      This was resolved by adding "\r\n" at EOL instead of "\n".
    - The main error that I faced was with FileReader which gave "Too Many Files Open" error on linux. I had to ensure that file descriptors are properly closed so that files were not left in open mode.
    - While multi-threading, I often got "OutOfMemory" error due to limited heap size of JVM. So, when we execute the program on multiple threads, I had to divide the available memory at that time among the number of threads.
    - Also, while executing program in parallel, I also faced IndexOutOfBoundException due to index going out of the specified limit while accessing string.
    - Another problem that was faced is that the slurm jobs were getting timed out in case of 20GB operations, I had to make my code more efficient in order to overcome this difficulty.

- **Conclusion**:
    - Disk is the bottle neck for the sorting as read and write to a file takes the maximum amount of time.
    - Also, the number of threads are very important in deciding the concurrency levels.
    - If the thread count is not properly managed it may lead to degrading performance.
    - Here, I have selected 4 as the total number of threads because each machine has 4 cores on it.
    - Also, as the file size increases the time taken increases and the throughput decreases in all the scenario (here, 2GB and 20GB).
    - Linux sort gives better performance as compared to shared memory sort in this case because linux sort program is written in highly concurrent environment whereas the shared memory code is only partially multithreaded.