Cloud Computing
PA 2 (Part-A)
External Sort on Single Shared Memory Node with Multithreading

Name: Sharanheer Choudhari (A20398615)

Goal:

To implement an external sorting application which can sort data that is large enough to fit in the main memory. For this experiment we are trying to sort 2GB and 20GB file.

Methodology:

- Following methodology was used to implement external sorting
- Divide the large file into smaller chunks of size which can fit in the main memory.
- Using in place sorting algorithm (QuickSort) sort the individual smaller chunks using multithreading.
- Merge all the individual sorted file back into a single sorted file using k-way merge.

Implementation:

- For this experiment, I designed and wrote external sorting application in Java using multithreading to achieve faster performance.
- The external sorting application was tested on a cluster which had 4-cores, 8GB Main Memory, and 80GB SSD.
- To sort 2GB, I used quick sort algorithm to sort the data and as the size is small enough to fit in the main memory there was no external sorting used.
- To sort 20GB, since the size is to large to fit in the main memory, external sorting was used where in I divided the file in 5GB chunks and sorted them individually using multithreading and then using 4-way merge combined the individual files.
- The k-way merge is implemented using the Heap data structure to compare and find the smallest element among all the individual files.
- The external sorting took 2 passes i.e. 2 reads and 2 writes.
    - First reading different chunks and sorting them individually and then write it to separate files.
    - Second read is performed to read all the chunks and merge then and write back the final output.
- The output of all the experiments were validated using valsort.

Results:

Experiment	Shared Memory (1VM 2GB)	Linux Sort (1VM 2GB)	Shared Memory (1VM 20GB)	Linux Sort (1VM 20GB)
Compute Time (sec)	142	30	731	393
Data Read (GB)	2	2	40	40
Data Write (GB)	2	2	40	40
I/O Throughput (MB/sec)	28.84	136.53	112.065	208.44

Analysis:

Shared memory (2GB):
It took 142 seconds to perform the sorting of 2GB and since the size of the data set is small than the main memory, it took 1read of 2GB and 1write of 2GB.

Shared memory (20GB):
It took 731 seconds to perform the sorting of 20GB and as the size of the data set was to large to fit in the main memory, external sorting was used which took 2 reads and 2 writes. First reading 20GB to sort individual files and write it back to individual files and then read the individual files again to merge them together and write back to the single output file.

Linux Sort (2GB):
Linsort also make best use of the available memory and therefore it takes 1 read and 1 write operation to perform the sorting for 2GB data set.

Linux Sort (20GB):
It took 393 seconds to perform the sorting of 20GB dataset and as per the documentation, linux sort uses k-way merge to perform external sorting which gives 2 reads and 2 writes.

Throughput calculation:

I/O Throughput (MB/sec) = (Data Read (MB) + Data Write (MB)) / Compute Time (Sec)

Conclusion:
Performed external sorting on different dataset (2GB, 20GB) by implementing external sort program in java and using the linsort application. Also, verified the results using valsort and compared the results.

Output Snapshot:

```
schoudhari@compute-15:~$ ./linsort2GB.slurm
Records: 20000000
Checksum: 98923e9cff98ac
Duplicate keys: 0
SUCCESS - all records are in order
schoudhari@compute-15:~$ cat linsort2GB.log
Time taken to sort 2GB using linsort 30-0 seconds
```

```
schoudhari@compute-15:~$ ./linsort20GB.slurm


Records: 200000000
Checksum: 5f5cc94518a4203
Duplicate keys: 0
SUCCESS - all records are in order
schoudhari@compute-15:~$
```

References:

https://en.wikipedia.org/wiki/External_sorting
https://en.wikipedia.org/wiki/Quicksort
https://en.wikipedia.org/wiki/K-way_merge_algorithm
http://www.ordinal.com/gensort.html
https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#lineSeparator--
https://dzone.com/articles/javas-fork-join-framework