PA2-Report

Name - Puneet Singh

CWID - A20413330

Problem Set:

We are required to sort files of size 2GB and 20GB. A 20GB file is large enough to fit into memory(RAM). In order to sort these files external sorting algorithm is used , Where we divide these files into smaller chunks and store them as temporary files. These temporary files can be sorted using several sorting techniques . In this implementation we are using merge sort. External sort uses hybrid sort merge strategy i.e in sorting chunks of data are small enough to fit into ram are read, sorted and written out to temporary files . In merge phase these temporary files are merged into single large file.

Methodology:

For implementing External sort on 20 and 2 GB file we are dividing the files into unsorted chunks using two functions createChunks and create unsortedtempfiles. Createchunks() reads lines from the input file , calculated based on selected blocksize.And calls Unsortedtempfile() function which writes these lines to separated unsorted temporary files.

Sort Phase:

After creation of temporary file we are using ExecuterService class to create a pool of threads , these threads execute tasks , which are lined up in the Queue. This Queue is maintaned by Executor Service class. The Executor threads operate asynchronously with the Main thread. Once the executor threads are spawned the main thread continues its execution . Each thread performs sorting operation of individual unsorted temporary file. This segregation is done on the basis of filename. Each individual files have unsorted lines which are read and sorted on the basis of keys . Each Line is of length 100 Bytes, first 10 bytes are considered as Keys and are sorted lexographicaly. Once the keys are sorted, the sorted keys are use to get corresponding remaining part of the line and written into temporary sorted files present in /tmp/sorted/ forder. We intend to merge sorted files once these files are created , Thus we are using CountDownLatch class to wait main thread for completion of execution for all executor threads. Once all the executor threads finish sorting temporary files , main thread starts its execution once again and calls Mergechunks() function.

Merge Phase:

Mergechunks() function uses a PriorityQueue in order to perform a K-way merge. We are using PriorityQueue class provided by java libraries. Priority queue is min heap based Data structure which provides min value among k objects in logK time. The objects stored in the Priority Queue are instances of MinheapNode class , which has an instance for buffer reader which reads from the specified file and a string value which stores the new line read from the buffer reader . Priority Queue uses a Comparator object which compares the object stored in the queue and performs heapify operations internally. PriorityQueue gives an abstraction to the min heap operations.

Verification:

Once the sorted files are merged this merged file is validated with valsort program . which provides all the detail comparison of the sorted and merged output file with input file.

Performance Evaluation :

Experiments	SharedMem 1VM 2GB	Linux Sort 1VM 2GB	Shared Memory 1VM 20GB	Linux Sort 1VM 20 GB
Compute time(sec)	113 sec	23 secs	1114 secs	389 secs
Data Read (GB)	6GB(3 times read)	2 GB	60(3 times read)	40GB
Data Write (GB)	6GB(3 times write)	2GB	60(3 times write)	40 GB
I/o throughput (MB/sec)	106.19(MB/sec)	173.91	107.71	205.65

Compute Time : Denotes the total time taken to perform External sort plus the validation.
Data Read: Data read is the total amount of data read during partitioning or chunks , sorting phase and in merging phase.
Data Write : Data write is the total amount of data written during partitioning or chunks , sorting phase and in merging phase.
I/o throughput : I/O throughput is calculated using the formula :
(Total Data Read +Total Data Write)/ total time taken to compute.

Data Read and Data Write is more for shared memory implementation as there are three phases and in each phase complete file is read from the disk and written to the disk . Thus there are three Read and three write. Two read and Writes cam also be done using split command from linux to split the file into chunks and then use two read and two write to perform sorting and merging phase. For this implementation we are using java program to create chunks as explained above.

Compute time in shared memory depends on several factors i.e, size of chunks and number of threads used for sorting and merging technique. Merging of the files can be done using normal Two way merge or K-way merge.

Data Read and write for Shared memory implementation are considered 6 GB and 60 GB for 2GB and 20 GB respectively. As the programs reads the total data 3 times and writes it back to the disk 3 times. Where as in linsort case , for 2 GB only 2GB read and write can be considered

as complete file can be stored into RAM. For 20 GB data has to be read two times and written 2 times thus total 40 GB data should be considered.

Following are the snapshots of the logs generated for all the experiments:

- Shared Memory (1 VM 2GB) : Shared memory 2 GB experiment usses 10 threads and 50 MB block size. For higher speed we have reduced the blocksize to 50 MB , thus there are total 40 temp files to be merged. Each with 500000 lines. During the multiple experiments , if we increase the block size then compute time increases.

```
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof

real    1m53.348s
user    3m45.552s
sys     0m30.908s
end sorting
Records: 20000000
Checksum: 98923e9cff98ac
Duplicate keys: 0
SUCCESS - all records are in order
```

- Linsort (1VM 2GB)

```
Total time taken for sorting 2 GB file in 4 threads using lin sort is
23
Records: 20000000
Checksum: 98923e9cff98ac
Duplicate keys: 0
SUCCESS - all records are in order
~
```

- Shared Memory (1 VM 20 GB): For 20GB we have taken 40MB blocksize , i.e. there are total 500 files to merge in merge step . Each of them have 400,000 lines to sort. These lines are sorted using merge sort in NlogN time. Increasing the blocksize results in increased compute time and this results into low throughput.

```
start sorting
/input/data-20GB.in

input file size:20000000000
creating chunks
/input/data-20GB.in

writing in file:1

writing in file:2

writing in file:3

writing in file:4

writing in file:5

writing in file:6

writing in file:7



Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof
Eof

real    18m34.246s
user    39m5.680s
sys     2m20.016s
end sorting
srun: Force Terminated job 4065
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
Records: 200000000
Checksum: 5f5cc94518a4203
Duplicate keys: 0
SUCCESS - all records are in order
```

- Linsort (1VM 20 GB)

```
Total time taken for sorting 2 GB file in 4 threads using lin sort is
389
Records: 200000000
Checksum: 5f5cc94518a4203
Duplicate keys: 0
SUCCESS - all records are in order
~
~
~
~
~
~
```

Throughput Comparison: Simple throughput comparison is done using column graph: