Project Report

Goal:

To implement a simplified version of TeraSort utilizing multithreading, instead of distributed execution, simple sampling technique and MapReduce model to sort relatively big amounts of data.

Approach:

First, we will divide the input file to a number of mapper threads, then each mapper will choose a random sample of the data of size considerably smaller than the size of the mapper and sort it using any sorting algorithm. Each sorted sample will be divided to number of sub-samples using cut-points. Cut-points values will be stored in a separate array shared with other mappers. Average of all cut-point values from different mappers will be calculated. Now, a number of empty arrays equal to number of reducers will be created. The mappers' records will be forwarded to those new empty arrays (reducer bins) such that:

- A record T will be forwarded to R[0] iff $T < C[0]$.
- A record T will be forwarded to R[i] iff $C[i-1] < T < C[i]$, for $i>0$ & $i < |C|-1$.
- A record T will be forwarded to R[i+1] iff $T > C[i]$, for $i=|C|-1$.

Analysis:

- Consider we use QuickSort.
- The average case when all the reduces has balanced workloads:
    - Creating the samples is O(S).

● Average case for sorting the sample is O(S log S).

● Average case for sorting a bin is O((n log n)/R).

● Unifying the samples and calculating the cut points is O(S).

● O(S)+O(S log S) + O((n log n)/R)+O(S)

● O(2S)+O(S log S) +O((n log n)/R)

● O(S log S)+O((n log n)/R)

● Since S should not exceed n, and in practice should be chosen to be very small compared to n then the overall time complexity is O((n log n)/R).

● For very large values of R with equivalent number of dedicated processors, R is highly significant.

● Moreover, if we can avail an environment which R is relative to n then the complexity will be O(log n).

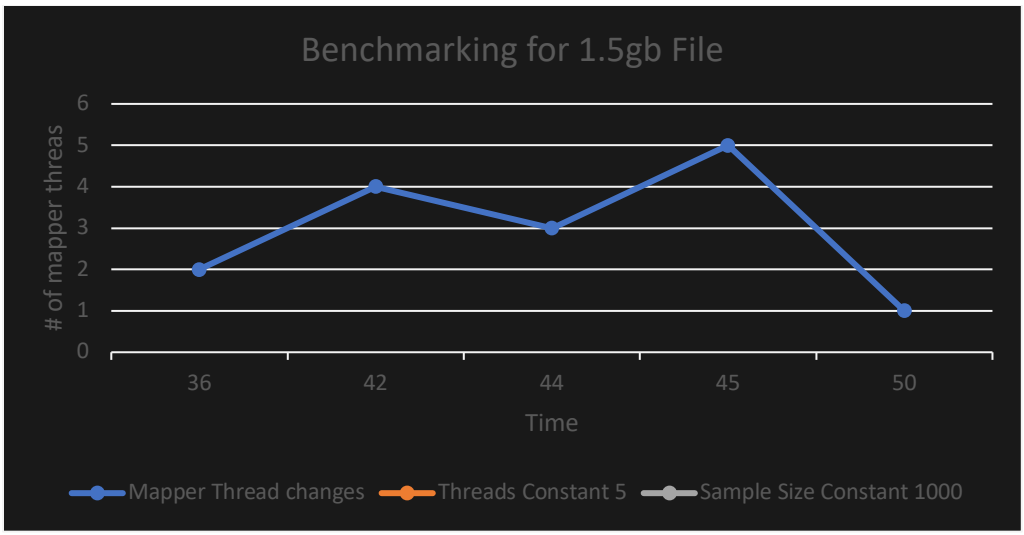
Coding:

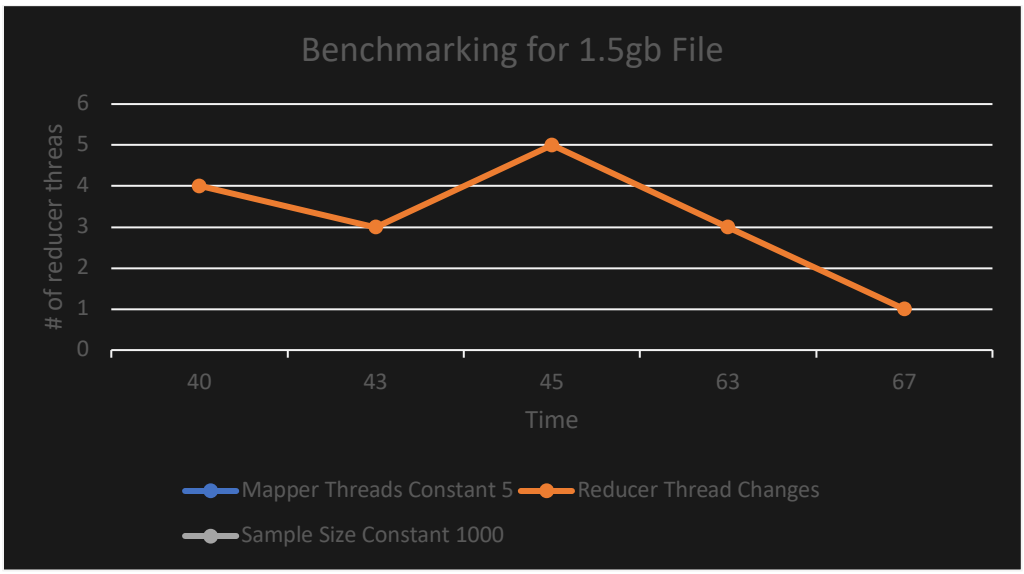
- CommandLineArguments class: Contains mainly of three methods:
    - Parser: that parses the command line entered by the user to extract from it the needed information. Bool method returns TRUE or FALSE indicating success or failure of the method.
    - Execute: instantiating an object of type TeraSort passing to it the parameters needed which were extracted by the parser. Bool method returns TRUE or FALSE indicating success or failure of the method.

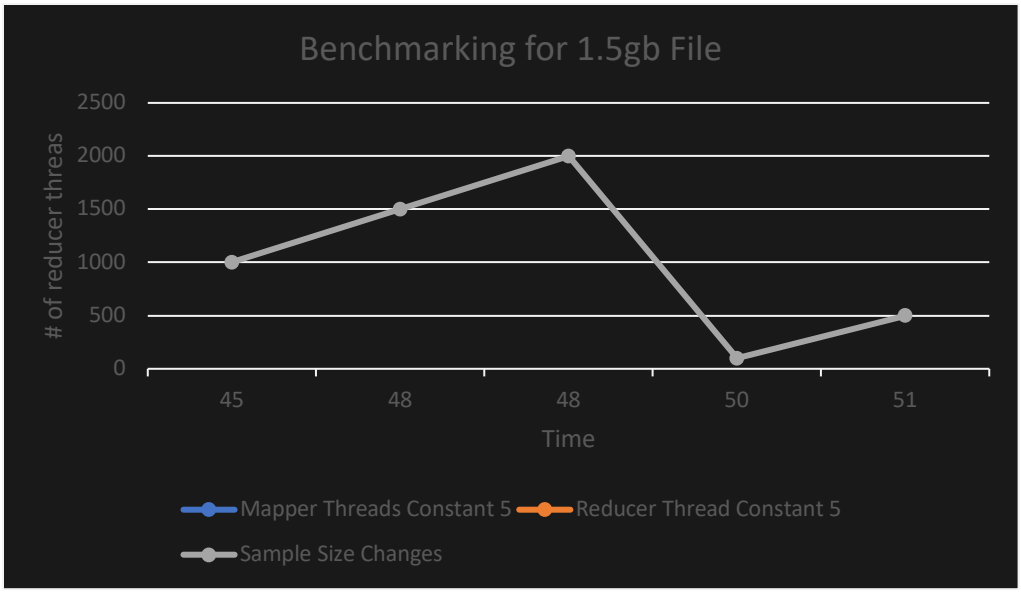
    - getErrorString: a simple getter method to return the private character array error_string.
- Partition class: Contains one private method and three public methods
    - resize: Basically, incrementing the size of partition when needed using realloc. (Private)
    - addItem: Adds a record to the partition. Needs to resize it before doing so. Uses std::mutex.lock to eliminate interruptions.
    - sort: Calls quickSort method of QuickSort class to sort the partitions.
    - dump: writes into the output file after sorting.
- Partitioner class: Contains of five methods
    - addCutPoint: Just adding a cut-point to the partition (sample).
    - calcCutPointsAverage: Calculating the average of the cut-points needed for forwarding the records to the right reducer bin.
    - addItem: Forwards the records to their right reducer bin invoking the method addItem in partition class meanwhile.
    - getPartitionsCount: Returns the number of partitions used
    - getPartition: Returns a specific partition
- Mapper class: The mapping engine. Contains two private methods and five public methods
    - phase1: Dividing the input file into the number of mappers specified by the user. The phase responsible of choosing the samples randomly and sorting them, too. Also, adds cut-points to the samples. (Private)
    - phase2: Forwards the item to its right reducer bin using the method addItem in partitioner class. (Private)

    - execute: Directs the mapper to do either phase1 or phase2
    - start: Invokes execute on each mapper
    - setThread: Simple modifier assigning a thread to each mapper, so that all mappers run in parallel.
    - waitForRunToFinish: Invoking join method on the mapper thread to wait for other mappers to finish.
    - randGen: Random integer generator used in sampling.
- QuickSort class: Sorts the samples. Has two private methods and one public method.
    - partition: Decides the pivot needed for applying the quick sort algorithm. Applies comparisons, too.
    - Insertion sort: Applies insertion sort algorithm for sorting relatively small data segments.
    - quickSort: A recursive method to sort the samples using quick sort algorithm. Also, decides when to use insertion sort instead.
- Reducer class: The engine for the reducing phase. Contains four methods:
    - execute: Invoking sort method in partition class to sort …
    - start: Invokes execute method on each reducer.
    - setThread: Assigns a thread for each reducer, so they can work in parallel.
    - waitForRunToFinish: Invokes join method on thread to wait for other reducers to finish.
- TeraSortItem class: Contains a struct of data record, two parametrized constructors other than the default, several simple operators and nine methods:
    - key: returns the key of the data record, based on which the sorting is applied.

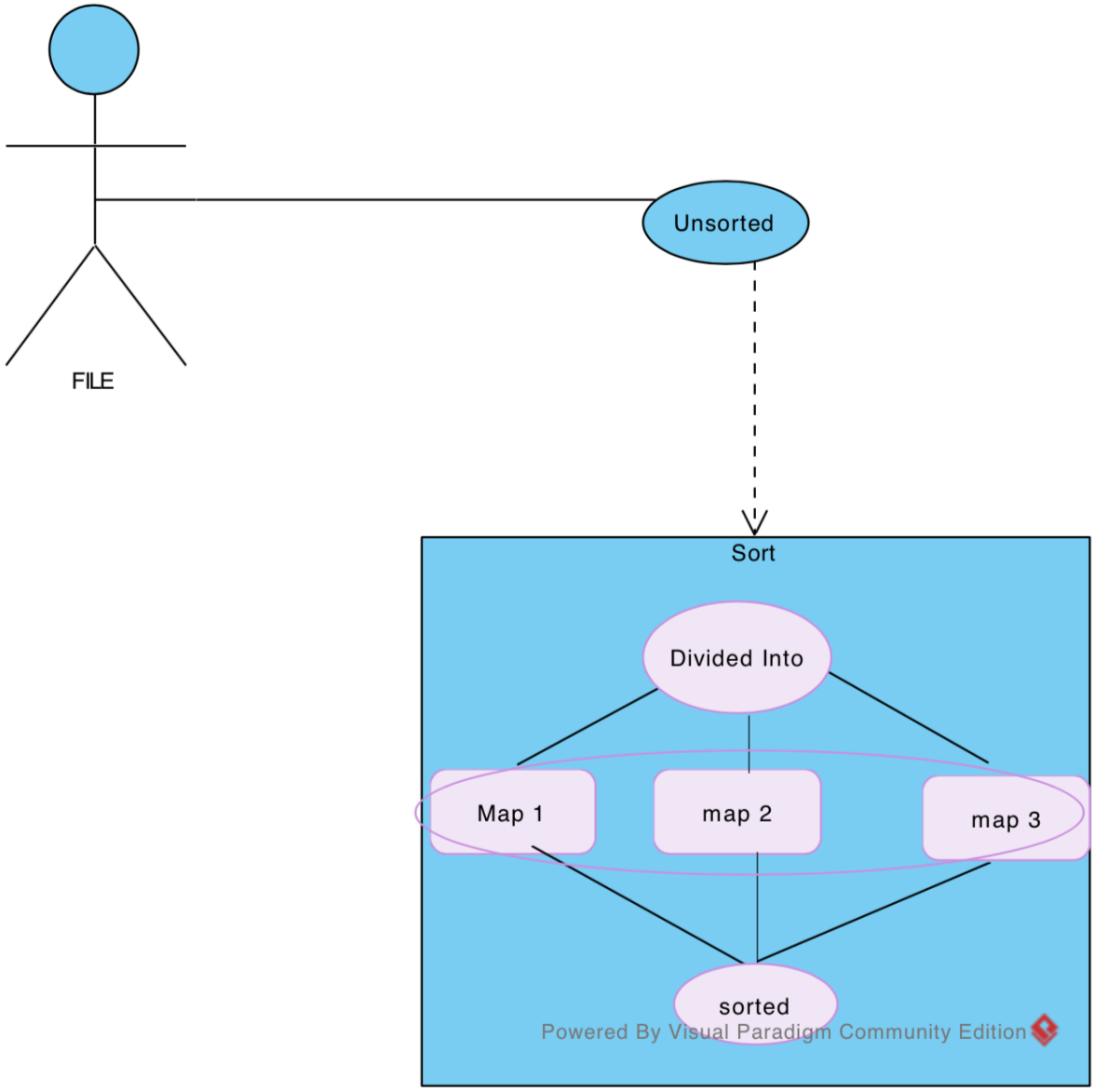
    - swap: Basically, swaps two records.
    - hash: …
    - getTeraItem: Simple getter returning the data record.
    - setb1: A modifier creating a barrier within the data record.
    - getb1: A getter returning the barrier set before.
    - addToKeyValue: Increment the key value with a certain amount.
    - getKeyValue: Returns the key value.
    - keyValueAverage: calculating the average of the key values.
- TeraSort class: Contains one method:
    - execute: Generates the mapper threads and invokes them to start executing. Calculates cut-points average and start executing again. Generates the reducer threads and invokes them to start executing. Finally, dumps the sorted data into the output file.
- The main: Will save the current time to start variable then parse the command line argument and invoke execute method in TeraSort class which will start the program. If execution failed, it will print error. If succeeded, it will print the time needed to complete the program.


Benchmarking for 1.5gb File
# of mapper threas
0
1
2
3
4
5
6
36
42
44
45
50
Time
Mapper Thread changes
Threads Constant 5
Sample Size Constant 1000

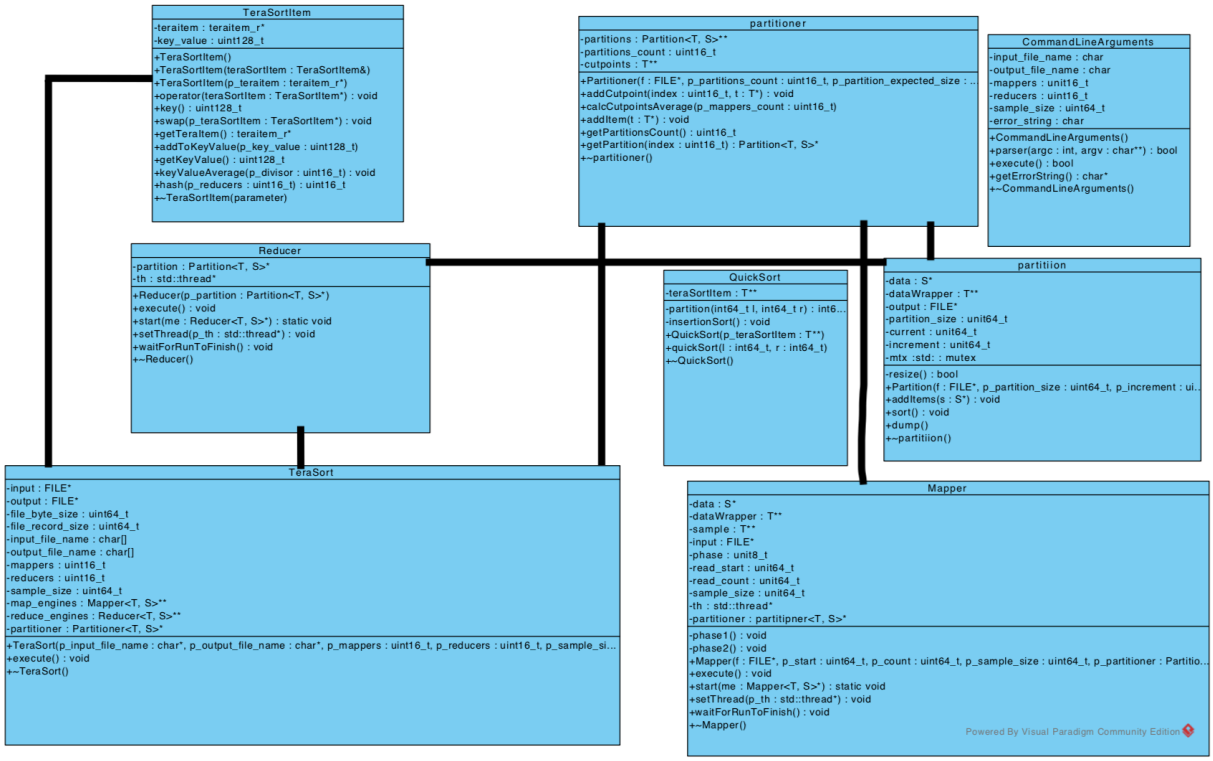

Benchmarking for 1.5gb File
# of reducer threas
0
1
2
3
4
5
6
40
43
45
63
67
Time
Mapper Threads Constant 5
Reducer Thread Changes
Sample Size Constant 1000


Benchmarking for 1.5gb File
# of reducer threas
0
500
1000
1500
2000
2500
45
48
48
50
51
Time
Mapper Threads Constant 5
Reducer Thread Constant 5
Sample Size Changes

Use Case Diagram:



Class Diagram:



Sequence Diagram: