Cs426 Project 2

Alba Mustafaj

November 13, 2017

Implementation

Main

In main.c the data is taken firsty from the argv at indexes 1-4, then both query and document files are read into arrays. For query, the size is already taken from argv[1]. Then for storing the data in the array the function readQuery (implemented in utils.c) is called. For making the 2D array for storing the dictionary, firstly the function getNrOfRows is called which returns the number of lines of the dictionary. Then, 2 arrays are created, 1 for storing the ids: `int idArr[nrOfRows]` and the other for dictionary: `int docsArr [nrOfRows][nrOfCols]`.
I take into consideration if the nr of rows is not divisible by the nr of processors, so I assign the remainder to the master as well. After calculating the number of rows corresponding to each process (*int elementsToEachProcess = nrOfRows / processorsNr;*) and that for master (*elementsToEachProcess+extraforMaster*), the sending process from master starts. The following are sent to each process:

- k
- D (how many ints in a line)
- query array
- chunk of documents array
- id array
- rows to each process

Each process (including master) applies the simmilarity function to the chunk of documents Array it is responsible for, then calls the sorting function to sort them in a descending order. For sorting I decided to chose merge sort since it is the most efficent because it is **nlogn** (quick sort would be an alternative too) mergeSort function takes the array where simmilarity values are stored as well as the id array, so when the comparisons are done, the consistency with the id is maintained. Concretely, in the merge function, temp left and right arrays are used for sorting the similarity values and also left and right arrays are kept for ids. **The id's are not sorted, but whenever a swapping is made for the similarity value, the same swap is made for the id as well, so the corresponding id for a certain file with that simmilarity value is preserved correctly.** I also

take into consideration that number of top documents can't be bigger than the total number of documents which goes to each process. After sorting is done, only the k most simmilar values are stored in myvals array and their ids in myids array. After that each process calls kreduce function (explained in the section below) which provides the topk array with the ids of top k values in descending order.

kreduce function

In the kreduce function I firstly calculated the final array size (the array which will hold k values from each processor) `int arraySize = processorsNr * k`. Then I created 2 arrays with that size, one for IDs and the other for simmilarity values. I put 2 conditions in the function:

- If process ID is not 0 : Each process will send myids and myvals array it has to the master process.
- If process ID is 0: Fristly will put the values it has in myvals in the final array starting from index 0 to k-1(inclusive). The same is done for myids, they are put in the same position in finalIds array. Then, in a for loop bounded by the number of processors, the master receives 2 arrays of size k (myvals and myids)from each process. I control the index to put these chunks in the finalArray and finalIds array by keeping an int *space* which is initially k (because indexes 1 to k-1 are occupied by myvals of master) and is updated in each iteration of the loop (`space = space+k; //avoid collapsing indexes`).Then mergeSort function is called taking as parameters the finalArray, finalIds array, 0 and arraySize-1 which are leftmost and rightmost indexes. After the recursive sorting process finishes we are provided with a finalArray of simmilarity values oredered in a descending order. After this, we populate the topk array with the k first elements in the finalIds array, because, as mentioned before, the ids are swapped as well in the sorting process, so any mismatch between the id of the file and the simmilarity value of it is avoided.

Results

Interpretations

In the graphs below, the numbers in the x axis represent the number of processors. The results differ for different k, file size and processor. By Figure 2, the time elapsed for finding 1000 first simmilar documents is higher

than that for 20 and 200. It can be attributed to the time needed querying, sorting, merging and puting the values into topk array. Thus, as k increases and file size remains the same, the time elapsed increases as well. From the graphs can also be deduced that as the number of processors increases, the time increases as well. Being that the time is very small, in this result can contribute the software errors as well as the fact that there is a higher number of sending and receiving events as the number of processes increases(thus taking more time. In Figure 3, the number of ints in a line is 2 and a simmilar behaviour to that of Figure 1 is observed. The ups and downs can be attributed to the system's latency errors as well being that the numbers are relatively small. All in all, as the number of top documents required increases, the time elapsed increases as well

Time \ processes	2	4	5	8	10
Sequential time	0.007187	0.011648	0.010476	0.01512	0.01515
Parallel Time	0.005394	0.006833	0.007778	0.007889	0.008
Total Time	0.01258	0.01848	0.01825	0.01940	0.0262

Figure 1: When D = 4

From the table above it is clear that sequential time is greater than the paralel one. The sequential process was all about getting the args and setting up the arrays. Having a higher time elapsed in the sequential part than that of the paralel process can be due to opening the files, storing the data in the array, and calculating the parameters to be used in the parallel process.

The graph in Figure 4 is a special one as time elapsed for different D(nr of ints in a line) is depicted. It is clear that for a bigger file size time elapsed is higher. This is due to query process, there is more data to query and eventually leads to a greater number of computations as power and addition which give the simmilarity value. To conclude, k, D and p are 3 parameters which affect the perfomance and it is clear by looking at the table and graphs. Increasing the number of processes may result in higher time elapsed due to higher number of communications. This cannot be generalized as in this case the time required for the task is small, however in another assignment when the communicating time is less than the task, it can be neglected. Increasing the number of top k values also results in a higher time elapsed, as more data is passed in the arrays (myvals and myids). These arrays are later passed by send and receive between master and other processors as well as sorted again in the kreduce function, thus tak-

ing more time. Lastly, increasing the nr of ints in the line (but keeping the number of processors and the top k values the same) results in higher time elapsed as there is more data to process, compare, swap, send and receive.

Graphs

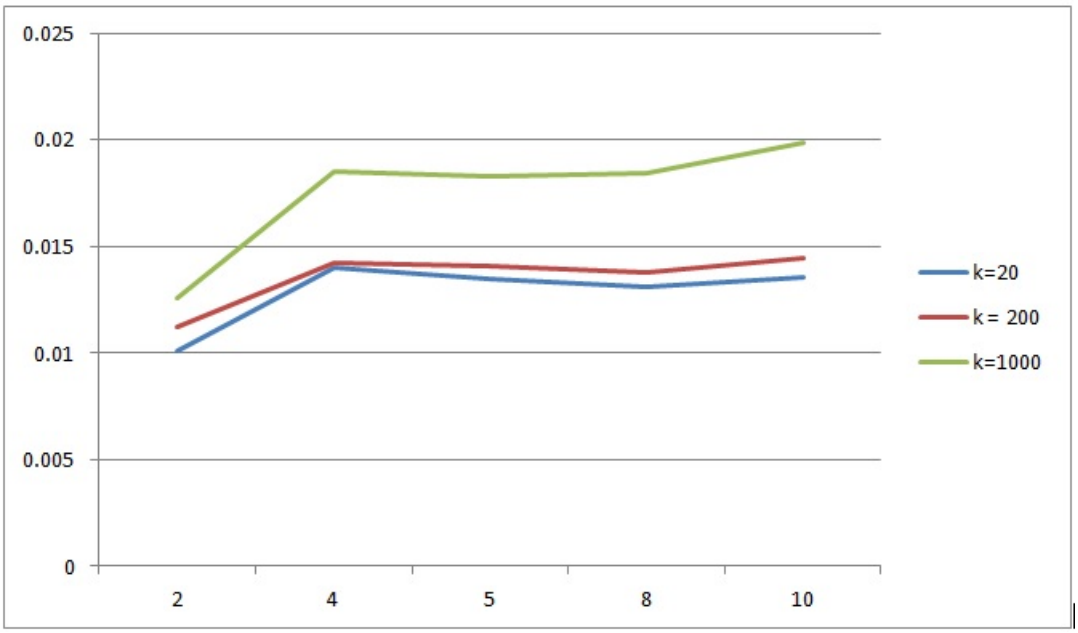

Figure 2: D = 4

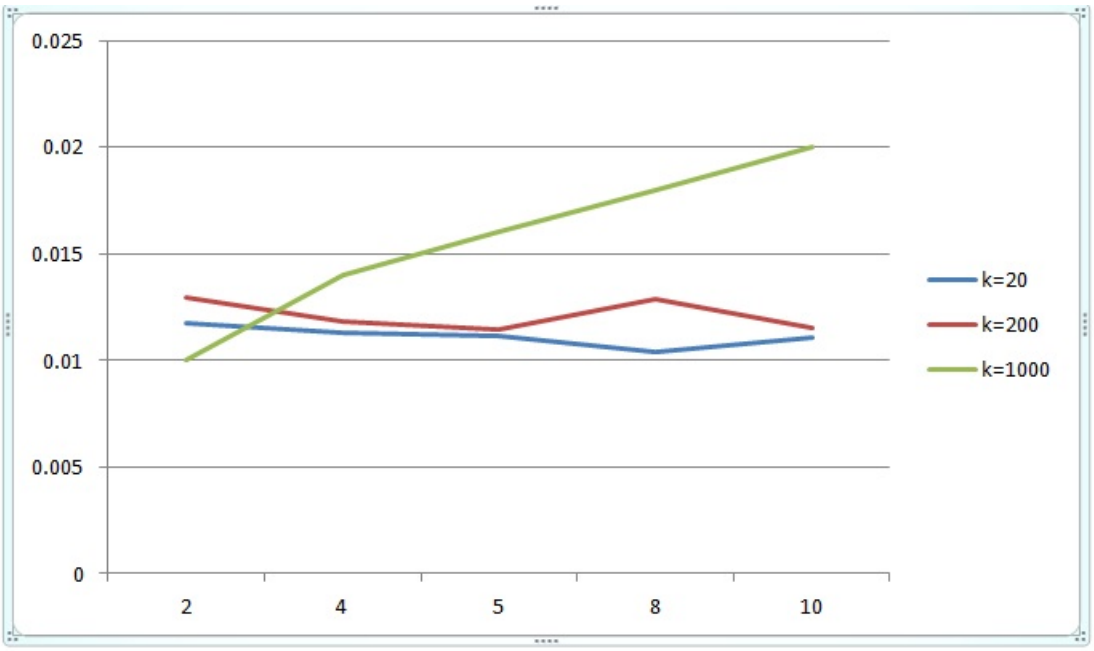

Figure 3: D = 2

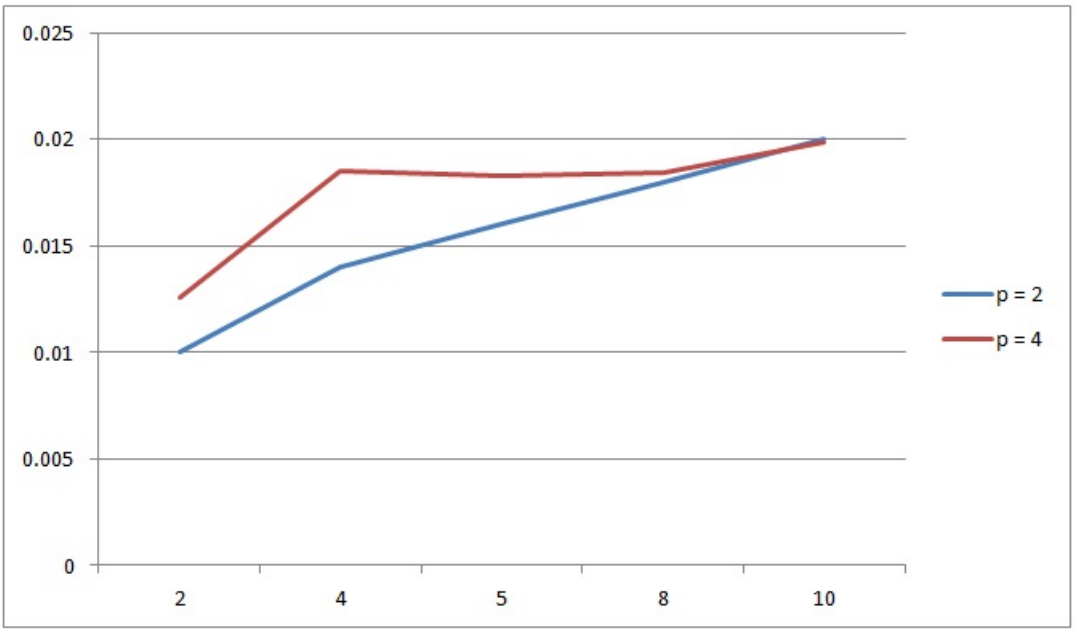

Figure 4: D = 2 vs D = 4