CS422 - Database Systems - Project 2

Baris Sevilmis, Onur Veyisoglu
Sciper Number: 306798, 309320

1 Overview

In database project 2, there are three different tasks to complete, namely RollUp operation, both naive and optimized versions, Thetajoin operation and Locality Sensitive Hashing, LSH, operation. Details about implementations and requested evaluations could be found in following sections.

2 Rollup

Rollup consists of two various implementations, to be more specific naive and optimized versions. Naive method is straightforward, as each of the rollup groups are exploited one by one. Each of the individual resulting element are computed separately. For the optimized version, child parent relations are utilized for more efficient grouping, in other words previous grouping computations are used for the extended grouping computations. Therefore, complexity reduces as dynamic programming similar technique is utilized. In addition, memory requirements are reduced as well, as each individual grouping attribute are not required to be in memory anymore.

Table 1 demonstrates rollup execution times with both of the naive and optimized methods for different input sizes and group lists. All the results are averaged over 3 distinct runs. It is obvious that rollup optimized performs better than naive method in any case even if the difference is in milliseconds for some cases. Nevertheless, optimized version outperforms naive method.

Table 1: Rollup Test Scenarios

ROLLUP *Average of 3 tests per each case* *QUERY: AVG*		*LineOrder_small* *Input_size: 6008* *Output_size: 13517*	*LineOrder_medium* *Input_size:600572* *Output_size:1351145*	*LineOrder_big* *Input_size:6001171* *Output_size:13502343*
Group_list: 0,1,3	*Rollup*	19.1329s	27.14272s	68.3299s
	Rollup Naive	20.5137s	27.3694s	72.8114s
Group_list: 0,1,3,5,7	*Rollup*	23.1146s	37.0277s	79.1091s
	Rollup Naive	22.4832s	35.4575s	84.5223s
Group_list: 0,1,3,5,6,7,9	*Rollup*	26.9011s	34.3690s	151.0791s
	Rollup Naive	38.4707s	35.4085s	224.7641s

3 ThetaJoin

Thetajoin utilizes 2 distinct attribute indices from 2 distinct RDD's, such that RDD's are joined together from these attributes. Main purpose of Thetajoin refers to partition given data among parallel processors, where each partition are to be joined locally and combined after each of these local join operations are completed.
Important thing to notice is to partition data among existing reducers. Optimal partitioning is achieved with an straightforward method, such that both RDD's are divided into equal chunks by using simple index ranges. These index ranges, namely boundaries, are used to determine which value falls into which reducer. Given optimal data partitioning, each reducer should have close amount of jobs instead of overloading a single reducer.
In addition, non qualifying partitions are pruned. As horizontal and vertical boundaries are considered as min-max ranges for chosen portion of data, some of these boundaries do not require comparison, because their comparison are not to be joined. Although, pruning results in additional computational complexity, large joins are to be favored by pruning operations addition to execution time.

Table 2 depicts results of Thetajoin on different input RDDs and different amount of reducers. First set of input RDDs consist of 4000 elements each and have been processed with consecutively 25, 100 & 400 reducers. 3 tests were taken in each case and their average has been taken as the resulting execution times. Although execution times are similar, best result is with 100 reducers. It could be interpreted with the tradeoff of communication cost and job amount for a specific reducer. In case of too many reducers given the size of data, reducers obtain few jobs and finish quickly. However, they have to combine their local results for the complete result and overhead due to communication cost may result in increase of execution time. On the other hand, using few reducers would cause almost no communication overhead but reducers to complete too many jobs locally and therefore increase of execution time.

Second set of tests are done on RDDs of size 50000 each. Reducer amount are chosen as 2500, 10000 and 40000 consecutively. Although, data amount is almost 12 times previous datasets, execution times have been only increased around 3 times. Same arguments about reducer amount could be also mentioned here. To conclude, execution results demonstrate the effectiveness of Thetajoin algorithm directly.

Table 2: Thetajoin execution times

THETAJOIN *Average of 3 tests per each case*	*taxA4K* *tax4BK* *reducers: 25*	*taxA4K* *tax4BK* *reducers:100*	*taxA4K* *tax4BK* *reducers:400*	*taxA50K* *taxB50K* *reducers:2500*	*taxA50K* *taxB50K* *reducers:10000*	*taxA50K* *taxB50K* *reducers:40000*
Atrribute Indices: *(1,1)*	24.0725s	22.6322s	28.1758s	82.0545s	78.1157s	96.6258s

4 Locality Sensitive Hashing(LSH)

In this part, the challenge is to implement Locality Sensitive Hashing scheme to optimize the similarity check of the films.

In order to show the improvement, first, Exact Nearest Neighbor algorithm is applied to our dataset and query points. This method falls behind the LSH since it requires a cartesian product of corpus data and query points and then pairwise Jaccard Similarity check between each film and their key words.

In LSH, costly cartesian operation is avoided by hashing the key words of each film and then only comparing the minimum hash value of each film since it is proved that LSH converges to ExactNN with more iterations. The following hash function is utilized in BaseConstruction method in order to provide randomness:

$$h_a(x) = (ax + b) \bmod p \tag{1}$$

To further improve the time performance, broadcast version of BaseConstruction which avoids unnecessary shuffles of join operation is implemented. With this method, a map of values in corpus dataset is broadcasted and another version of join operation implemented using map operations.

As illustrated in 4, ExactNN is performing almost same as the LSH methods for small dataset. However, as the size of data increases, LSH schemes outperforms ExactNN. For medium dataset BaseConstruction and BaseConstructionBroadcast shows an equal performance but outperforms ExactNN by working 9 times faster.

For large dataset, BaseConstructionBroadcast runs 3 times faster than BaseConstruction while ExactNN took more than 10 minutes which considered as a timeout.

In terms of recall and precision requirement, one BaseConstructor is used to pass the requirements of query0 as it was required to get a moderate level of precision and recall.

The second query requires a very high precision and a high recall. In order to achieve that a hybrid version of AND and OR Constructors are used. To be spesific, ANDConstruction with 7 ORConstructor children where each ORConstructor has 4 BaseConstructor is implemented.

For the last one, query2, as recall requirement is way more higher, an ORConstruction with 3 BaseConstructors children is used. The precision and recall performance of the methods are demonstrated in Table 3

Table 3: LSH: Precision Recall Scores for Queries: 0, 1, 2

LSH Data: corpus_all	*Precision*	*Recall*	*Approximate Distance to NN*
Query 0:	0.7293	0.8083	0,5992
Query 1:	0.9777	0.6875	0.8036
Query 2:	0.4426	0.9324	0.5615

Table 4: LSH: Execution Time Results for all the Queries

LSH Execution Times Avg. of 3 tests	*Data*	*Base*	*Broadcast*	*ExactNN*
Query 0:	*Small*	14.2873s	13.8819s	18.1788s
Query 1:	*Small*	14.1623s	15.5016s	19.0408s
Query 2:	*Small*	14.0309s	13.8434s	16.8230s
Query 3:	*Medium*	15.5228s	17.3725s	243.0135s
Query 4:	*Medium*	15.5228s	16.1622s	234.4370s
Query 5:	*Medium*	17.6980s	15.9606s	288.5327s
Query 6:	*Large*	68.9945s	20.7287s	*Timeout(>10m)*
Query 7:	*Large*	69.3636	20.5527	*Timeout(>10m)*

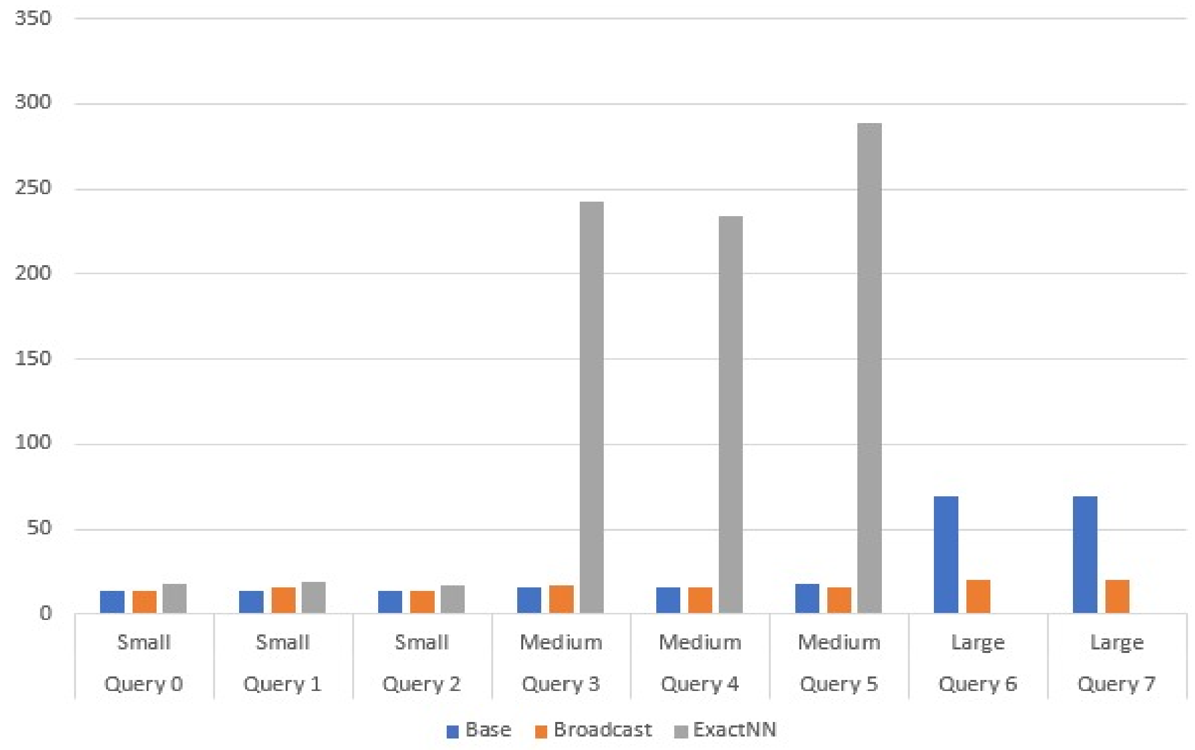

Figure 1: Precision and Recall plots for LSH

In conclusion, considering the performences of all methods, it can be said that for significantly small datasets and small amount of query points, ExactNN method can be used for more accurate results as LSH needs deeper structures with multiple AND and OR constructors with many BaseConstructors to perform as powerful as ExactNN. Yet, for larger datasets and large amount of query points, LSH should be preferred. If the data to work on is so large, a broadcast version should definitely be preferred.