Project part I

Part I consists of four tasks which are as follows.

1. Reading .dat files and loading the data into tables.
This is a onetime process. .dat files are read in java, are parsed into data fields e.g. queryid, queryText in case of queries. Then using jdbc connection these values are inserted into the database with the help of prepared statements.
Running time for this task – Approximately 15 seconds for 30000 queries.

2. Computation part related to cosine similarity, bid and quality score.
This part performs the foundation work needed to execute the algorithms. This computation is specific to a query and it remains same irrespective of the algorithm. Hence this is executed only once for a query before executing any algorithm.
This part is coded in stored procedures written in PL/SQL. It consists of following tasks.
    - queryText is spilt into tokens with space as a delimiter and inserted into a table against queryId. These tokens are important because each token can be a keyword bid by some advertiser. (Henceforth in the report, token and keyword will be used interchangeably.)
    - All the advertisers who bid for at least one keyword for a particular query are found out.
    - For a query advertiser pair, below tasks are performed.
        - Cosine similarity between query tokens and keywords of an advertiser is calculated.
        - Total bid by advertiser for that query is calculated.
        - Quality score is calculated using cosine similarity (calculated above) and ctc (available as input).

Combined running time for task 2 and 3 is mentioned below task 3.

3. Execution of following six algorithms – Greedy First Price, Greedy Second Price, Balance First Price, Balance Second Price, Generalized First Price, Generalized Second Price.
This part is coded in stored procedures written in PL/SQL.
    - Using the algorithm, ranks of all the advertisers are calculated for a particular query. Data needed to execute algorithms e.g. quality score, bid etc. is available from the previous task. Depending upon the rank, first n advertisers are selected. (n is taken as input from system.in related table)
    - Only those advertisers who come in the rank and who are required to pay as per their ctc value, pay for the advertisement. To do that, their balance is updated.
    - For first price algorithms, updating the balance after bidding is straightforward, since advertiser pays own bid.
    - But for second price algorithms, advertiser may pay the second valid bid if it is present. To find out second valid bid, before executing any algorithm, valid advertisers are found for a particular query.

This helps to avoid keeping a track of advertisers which may come into rank during execution and get their balance updated but who were valid before.

- All algorithms are executed sequentially for one query and output is written into output tables. Every task has a separate output table.

(Combined Running time for task 2 and task 3 - Approximately 13000 seconds i.e. 216 minutes for 30000 queries.)

4. Writing the results into the files.
   This part writes the data from output tables to output files. This part is executed once outputs for all the queries is generated.
   This part is coded in java.
   Running time for this task – Approximately 1.8 seconds for 30000 queries.

Project part II

Project part II has same components as that of project I. But implementation of few components is changed in part II. To sum it briefly

1. Use of sqlldr in input task.
2. Combining execution of algorithm and writing data into output tables into task 3.
3. Few optimizations in the stored procedures.

Details related to implementation and improved performance are mentioned in the Performance and Optimizations section.

Performance and Optimizations

Overall performance improvement was approximately 19.90% since total execution time was reduced from 216 to 173 minutes for 30000 queries.

1. sqlldr is used to improve the performance of loading the data.

   Significant performance improvement is seen because of this. Previously, program took around 15 seconds to load all the input data (for 30000 queries). After using sqlldr, program takes around 4 seconds to load the data for the same amount of queries.

2. Execution of algorithm and writing data into output tables is combined.
   Previously, after the execution of algorithms for all queries, required data fields were fetched from respective tables and output tables were populated.

   But this was creating unnecessary lookups and joins into some tables which had large number of records. Upon realizing that once an algorithm is executed for a query, output for it can be written to output table in the same pass, these tasks were combined. This saved the lookups into largely populated tables and hence helped to improve the performance. After performing this modification, tasks 2 and 3 which previously took around 216 minutes to execute for 30000 queries can be executed in around 193 minutes.

3. Removing non key fields from queries where aggregation operators were used.

   e.g. "SELECT COUNT (bid) FROM QUERYADVERTISERMAPPING WHERE queryid = q1 AND advertiserid = a1". This table maps an advertiser to a specific query. Since "bid" is a non-key field, according to query explain plan, database was referring to the table where referring to index could have sufficed the purpose.

   Hence the query was modified as "SELECT COUNT (*) FROM QUERYADVERTISERMAPPING WHERE queryid = q1 AND advertiserid = a1". After doing this, according to explain plan database only referred index.

   Performance improvement was seen upto some extent because of this change. Such queries were used in all the algorithms and hence performance of all algorithms is improved. An algorithm with modified query now takes 0.050 seconds for one query which previously took around 0.077 seconds.

4. Non key fields were also removed from queries where they were not needed.

   This change did not help increasing the performance in all the cases, because even though not necessary non-key fields were removed, if few non-key fields were still present in SELECT clause, then database still referred to index as well as table.

Difficulties faced and learning

1. After making a set of changes, I had to rerun the entire program again and again to check whether I am getting the expected performance and also to check that I did not disturb the correctness of it while improving the performance. And since this has to be checked with large amount of queries, there was a lot of delay in running next set of changes.
2. I could learn how database engine decides which component e.g. table or index to refer according to the query parameters specified.
3. I could learn about many techniques of tuning the queries e.g. avoiding use of unnecessary group by, avoiding use of functions inside query. Since I did not make much use of these, I could not try them out from performance perspective, but I could learn those techniques.