Performance Problems of Forecasting Systems [⋆]

Haitang Feng
Supervised by: Nicolas Lumineau and Mohand-Saïd Hacid

Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France
{haitang.feng, nicolas.lumineau, mohand-said.hacid}@liris.cnrs.fr

Abstract. Forecasting systems usually use historical data and specific methods (typically statistical models) to derive decisional information. To be accurate, the volume of historical data is often very large and the calculation and the exploration processes are complex. Anticipeo is one of the forecasting systems which is devoted to the prediction of sales based on collected sales over a "long" period of time. Even if it provides quite a reliable prediction, the query evaluation stays a costly process with high latency.
So far, we have investigated the latency provenance and we have proposed some solutions to improve query response time for the exploration process. Our design principles can be reused in any application context that displays similar requirements.

Keywords: Query optimization, performance, DBMS Tuning, OLAP

1 Introduction

Forecasting systems are based on historical information and use specific methods (typically statistical models) to derive decisional information and present them in an understandable way. These systems are an important issue in an industrial context because strategic decisions partly depend on forecasting. The features of a good forecasting system are: economy, availability, plausibility, simplicity and accuracy[1]. These features require forecasting systems to consider some different design properties from other information systems.
Our work reports on Anticipeo, a forecasting system which uses historical data to derive decisional sales information. Results are displayed in hierarchies and OLAP operations are employed to navigate the results. Performance issues exist both in the calculation stage and the exploration stage. We present these two stages and their problems in detail in the following of this paper.
So far, we have investigated the exploration stage. To solve the problem, we diagnose its latency provenance, provide benchmarks on the actual system and propose some preliminary solutions. The paper is organized as follows : Section 2 presents the problem and our motivation, Section 3 describes the related works

[⋆] Research partially supported by the French Agency ANRT (www.anrt.asso.fr) and Anticipeo (www.anticipeo.com).

and the solutions we consider. In Section 4, we show the experimental results. We conclude and present our future work in Section 5.

2 Problem Statement and Motivation

By comparison with other information systems, the design of forecasting systems should comply with some requirements:

- Limited life duration
  Forecasting calculation is periodically triggered and once the period passed, the forecasts are no longer exploitable. For example, the weather or the traffic forecasts predicted for yesterday are not useful today.
- Non-reusable
  The forecasting work is a global calculation and it is almost impossible to reuse the former results for reducing future forecasting calculation. This means that partial recomputation of precomputed results is not easy.
- Few updates
  As forecasting information result from a computation process, there is often no need for updates. However, in some exceptional cases, the system needs some corrections for some unpredictable situations: e.g., the impact of a car accident for a traffic forecasting system.
- Timeliness
  Forecasting systems should be able to provide just in time responses for users. Decision makers devise corresponding plans of purchasing, productions, logistics, etc., according to sales forecasting. The delay between achieved data entering and new data forecasting should be as short as possible.

Anticipeo is one of the forecasting systems which uses statistical models to derive forecasting sales from historical sales. It helps enterprise decision makers or executives to adapt their business plan to future trends of the market.
Every month, customers provide Anticipeo with their new achieved sales. Anticipeo integrates the data into the system, calculates forecasting data and prepares information for analysis. Customers can then navigate the forecasting sales via user interface. If they want to perform some modifications on the forecast, the visualization generator will be relaunched to produce updated presentations.
Two issues regarding the manipulation of very large tables at the calculation stage and at the visualization stage constitute the main problems that we tackle. To achieve a reliable result[1], the calculation engine treats each sale individually with a corresponding statistical model (e.g., standard model, seasonal model, occasional model, etc.). The treatment relies on 36-month historical data in addition to tens of parameters (e.g., weighting coefficient, serial correlation, seasonal coefficient, etc.) used in statistical models. Even if sales are grouped by month, the number of monthly sales is still high. As a consequence, the calculation stage is quite time-consuming.

[1] Here, reliable stands for good quality and good approximation.

Regarding the navigation part, results are displayed in form of hierarchies. To show the sales trend, we display both 36-month historical data and 24-month predictive data. This means 60-month aggregations should be generated for hierarchies when displaying. In addition, we need some meta-data to understand/interpret the displayed data. As the number of sales could be very high, the tables used to display the results can easily reach gigabyte scale.
In the first time, we mainly consider the optimization regarding the second issue, that is, the navigation part.

2.1 Features of Anticipeo Data

Sales are organized in hierarchies according to the needs of end users. Anticipeo uses a relational database to store data. The main problem in this case can thus be summarized as a visualization issue of a multidimensional decision support application using a relational database.
In this application, three dimensions are defined: *customer* dimension, *product* dimension and *time* dimension. Since the sales are grouped by month, the time dimension is implicit. Only two dimensions are explored: customer dimension and product dimension. Each of these two dimensions is composed of several hierarchies such as geographical distribution and purchasing store organization for the customer dimension. Regarding the navigation of sales, a materialized view containing all display or relation information about customer, product, hierarchy, purchase date and sales volume is created (called $MV_{example}$ in the rest of this paper). Actually, we have three materialized views for three metrics: turnover, quantity and price. They serve to guarantee a quick data access and avoid frequent joins between tables containing this information.

2.2 Typical Queries

There are three kinds of operations that users perform : simple querying on one hierarchical tree, cross-hierarchy querying and data updates. In the following, we talk about these three operations with the most significant queries linked to them.

Simple Querying
We name this type of operation “Simple querying” because the queries require directly the visualization of sales on only one level of a hierarchical tree even if this level can be no matter which level of no matter which hierarchical tree from no matter which dimension. Usually, in an analytical system, we present an element’s information with its direct successors for further exploration. So this operation of simple retrieving can be considered as the retrieving of a required element with the corresponding elements from the inferior level.
The principal query of this operation constructs aggregations using “GROUP BY”. Because this “Simple querying” operation is the most frequent one exercised by users, the response time should be immediate. All levels of hierarchical trees are pre-computed and materialized.

Cross-hierarchy Querying
The second type of operations is the cross-hierarchy querying. These queries refer to information retrieval from two hierarchies: one of them is from the customer dimension and the other one is from the product dimension. The time dimension is implicit in this operation.
Technically, the principal query constructs aggregations for a certain level of one of the dimensions using "GROUP BY" and the other dimension acts as the filter condition in this query.
Unlike the precedent operation, we cannot pre-compute all the possible combinations. Another solution should be used to accelerate this operation.

Data updating
Previous two types of operations have many similar points with classical data warehouses. But this third operation is something unusual for data warehouses. The sales forecasting system allows users to modify future projections to simulate future scenarios, which means, data stored in the pre-computed data cube will be modified. This is against the main characteristic according to the original design of data warehouse: non-volatile, "Data in the data warehouse are never over-written or deleted once committed, the data are static, read-only, and retained for future reporting"[7]. However, this is a real need for decision makers to prepare for different future situations by using forecasting systems.
When users of the application query the data cube, they may decide to modify a future projection. They can do this operation on any level of any hierarchy (even in the cross-hierarchy mode). Once the modification is executed, the result with updated data should be presented to users for their analysis and eventually further simulating modifications. Technically, when users modify a data value, the modification is proportionally distributed to corresponding elements on the elemental sales level. Levels of different hierarchies aggregated by using these sales must be updated consequently, including the level where users have performed modifications. Then the level is displayed immediately following the modification to users. During this operation, there are two important queries: update of basic sales and displaying level reconstruction.

- Updating of basic sales
  This query updates the elemental sales by which the aggregations of the modified level are built. The update work is more time-consuming for it concerns the write operation on the disk and in particular, when indexes have to be updated at the same time.
- Reconstruction of corresponding level
  Once an element is modified on the elemental sales level, all other levels of the hierarchies need to be updated because they are aggregations using the modified element. The rule after updates is that all superior levels are destructed if an elemental sale is modified no matter whether the elements are composed by the modified element. Then, we reconstruct the level.
  The query of reconstruction is the same as the "simple querying" or the "cross-hierarchy querying" depending on mode before the modification.

3 Optimization

Our purpose is to get better performance for forecasting systems similar to our case. In the tuning, we should take into consideration not only the part of visualization, but also the updates in data warehouses.

3.1 Hardware and Operating System

The first question we examine is whether our application is working in the appropriate environment.
The main technical characteristics of the server are: two Intel Quad core Xeon-based 2.4GHz, 16GB RAM and one SAS disk of 500GB. The operating system is a 64-bit Linux Debian system using EXT3 file system. So, our focus in this audit is to inspect the hardware for three criteria: CPU, memory and I/O.

3.2 DBMS Configuration

The application has some different characteristics compared to most of existing web services. The most significant one is that there are only a few users logged in the application (because users are often decision makers of an enterprise), but every user can launch resource-consuming queries to the database. Some "blind" tuning has been done based on existing experimental results on different web services[12]. Since *InnoDB* is the main storage engine used, the main MySQL system variables (see [9] for variable definition) manipulated are *innodb_buffer_pool_size*, *innodb_log_file_size*, *query_cache_size*, *innodb_flush_log_at_trx_commit*, *key_buffer_size*, etc. The actual configuration sets these variables to 8GB, 800MB, 64MB, 0, and 512MB, respectively. Additional benchmarking will be discussed in the following to determine whether the adopted configuration is an efficient one and if not, we will propose an optimal configuration.

3.3 Additional Materialized views

The visualization of achieved and forecasting results in dimensions concerned this application is a typical Data Warehouse problem using relational databases. In this domain, one of the most used solutions is to select useful intermediate results and store them as materialized views. Many approaches have been proposed for the selection of materialized views.
The main idea is to use the greedy approaches[4, 6, 10]. These solutions preprocess the most beneficial intermediate results in a limited-space hypothesis to avoid complex computations so as to enhance data access. Extensions of these solutions consider also the maintenance cost[2] or large scale databases[5, 8], or else make the set of materialized views dynamic according to the workload[3]. They have already been proved to bring significant improvements to data access. So we would like to test the impact of these solutions in our case. For the benchmark, we choose to implement the basic Greedy Algorithm.

3.4 Database Design

This application works on a large materialized view for result visualization. The advantage is obvious: we can omit time-consuming joins over tables containing millions of rows. However, it creates other problems, e.g., for queries which need to make several joins on this view to derive aggregations for superior levels. Our idea is to find a medium solution that can both avoid costly joins on different tables and reduce the time of self join of this table.
The star schema or snowflake schema[11] (which is the normalized version of a star schema) could be a solution. We can break this view into two separate ones, otherwise, we create two materialized views instead of one: one view contains only the hierarchical information and the other one contains all remaining information.

4 Experimental Results

All the following evaluations were carried out in the environment presented in Section 3.1 and 3.2.

4.1 Observations on Existing Application

Some experiments have been conducted so to estimate the execution time of the application for both the calculation and the navigation processes. Several different size databases are compared.
The result shows that the calculation time and the query evaluation time is quasi polynomial in respect to the sales number. This conclusion helps us to estimate the execution of a new database if we know the number of sales that the company deals with. The previous results also show the brake of the enterprise: in the case of a 55-GB database, it takes more than 18 seconds to answer a user query. If the application considers larger databases, the response time can hardly be acceptable by the user.

4.2 Diagnosis of Latency Provenance

Two diagnoses are performed in this part. We first conduct an analysis on time distribution. For the mathematical calculation, time spent on the application server represents 30% of the total time and the remaining 70% is used to access the database. During the navigation, the part of the time spent on the application server represents less than 10% of the total time.
In a second time, we are interested in better knowing the system behavior. We use SAR, one of Linux performance monitoring tools to collect and analyze system activity information. CPU is idle during on average 89.42% of time. Memory is normally used without swapping activities (0% during all the process time). Disk I/O also shows a normal activity with rare occurrences of device saturation according to the average number of read and written sectors.

The diagnosis shows that the latency observed on the application is not a material problem, neither a program level issue. The performance issues of the database may be the source of the performance problem.

4.3 DBMS configuration

We set values above and below the current settings of main variables presented in 3.2 to see whether the current ones are optimized. The result is, for most of variables, the current value is already the optimal one and only two variables *innodb_buffer_pool_size* and *innodb_log_file_size* could do better if defined as 12GB and 1GB, respectively.
We run an integral test with new DBMS settings on the whole system and we get a better performance of 7.32% with *innodb_log_file_size* set to 1GB. The current value of *innodb_buffer_pool_size* is already the most optimal one for a machine that serves at the same time as a database and a web server. So, the system is running with an almost optimal configuration.

4.4 Selection of Materialized Views

The classical greedy algorithm has been implemented on a 55-GB database with some adaptation. We have noticed that the cardinality of nodes (# tuples of views) is highly skewed among the nodes in the lattice. So, instead of limiting the number of selected views, we limit the number of tuples to be materialized. The result reveals a significant interest in precomputing materialized view candidates for this application. With only one quarter of materialized tuples, 279940 rows precisely, approximately 87.91% performance improvement can be achieved.
Unfortunately, MySQL, the DBMS used by Anticipeo to implement and manage forecasting data, supports neither materialized views nor automatic query rewriting by materialized views. Our results show the possibility of performance improvement if we had used another DBMS that supports automatic query rewriting. We could cite the Oracle relational database system.

4.5 Database Schema Modification

We trace queries whose execution time is more than 2 seconds while users perform actions on the interface. We obtain four query types on all the slow queries. One instance of every query type is evaluated on both the actual data schema and the new data schema. We have an interesting improvement: an average gain of 8.083%, 4.034%, 17.126% and 12.088% respectively for the four query types. With regard to the response time of a consultation based on precomputed data, there is some degradation because an extra join has to be added to put the hierarchical information and the rest together. However, the evaluation time is still on the scale of tens of milliseconds. So, this newly created difference could be ignored for a web service.

5 Conclusion and Future Work

We have presented four main optimization approaches that we consider in order to improve the performance of a forecasting system: hardware, DBMS configuration, DB design and selection of materialized views. In our context, we consider a real world application, “Anticipeo”, and we prove that with our solutions, the system presents a better performance. Further research work will include the index improvement and the parallelization of databases.
Besides, we have not looked into the calculation performance improvement so far. As said in the introduction, former forecasting results are not used for future calculations. This means, at time t_i, the database DB is at state S_i. Once new forecasting information at time $t_{i+\delta}$ has been derived, the database DB changes to $S_{i+\delta}$, which is completely different from S_i. There could be some repeating intermediate results during the calculation and these intermediate results should be kept and used for future needs. We are investigating this issue.

References

1. T. K. BAN. Features of a good forecasting system, Jan. 2011. http://tankahban.blogspot.com/2011/01/features-of-good-forecasting-system.html.
2. E. Baralis, S. Paraboschi, and E. Teniente. Materialized views selection in a multidimensional database. In *VLDB*, pages 156–165, 1997.
3. S. Biren, R. Karthik, and R. Vijay. A hybrid approach for data warehouse view selection. *International Journal of Data Warehousing and Mining(IJDWM)*, 2:1–37, 2006.
4. H. Gupta. Selection of views to materialize in a data warehouse. In *ICDT*, pages 98–112, 1997.
5. H. Gupta and I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *ICDT*, pages 453–470, 1999.
6. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD Conference*, pages 205–216, 1996.
7. W. H. Inmon. *Building the Data Warehouse (3rd Edition)*. Wiley, 2005.
8. Y. Kotidis and N. Roussopoulos. Dynamat: A dynamic view management system for data warehouses. In *SIGMOD Conference*, pages 371–382, 1999.
9. Oracle. Server system variables, Feb. 2011. http://dev.mysql.com/doc/refman/5.0/en/server-system-variables.html.
10. A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *VLDB*, pages 488–499, 1998.
11. P. Vassiliadis and T. K. Sellis. A survey of logical models for olap databases. *SIGMOD Record*, 28(4):64–69, 1999.
12. P. Zaitsev. Innodb performance optimization basics, Nov. 2007. http://www.mysqlperformanceblog.com/2007/11/01/innodb-performance-optimization-basics.