Research of Real-time Data Warehouse Storage Strategy Based on Multi-level Caches

Shao YiChuan¹, Xingjia Yao²

¹ School of Information Engineering  
Shenyang University  
Shenyang China  
² College of New Energy Engineering  
Shenyang University of Technology  
Shenyang China

Abstract

Real-time data warehouse extend the application of traditional data warehouse. It can not only support tactical queries for enterprise but also provide much variable tactical decision support effectively. For these reasons, it is very meaningful to research on the structure of real-time data warehouses. This paper introduced the background of real-time data warehouse and proposed the strategy of real-time data warehouse which is based on double mirror replication mechanism. The strategy is composed of double steps. First we used double mirror replication mechanism to enable continuous loading data in the real-time data warehouse with minimum impact in query execution time. Second we proposed incorporating multi-level caches into the data warehouse structure which is based on real-time partition and gave the process of design and implementation with details. We differentiated between queries with various data freshness requirements, and used multi-level caches to satisfy these different requirements.

© 2012 Published by Elsevier B.V. Selection and/or peer-review under responsibility of Garry Lee  
Open access under CC BY-NC-ND license.  
Keywords: real-time data warehouse; double mirror replication; incorporating multi-level caches

1. Introduction

Recent years, as computer science and information technology developed rapidly, the electronic information data has become more and more important in the enterprise daily management. Effective, accurate and timely data analysis is required imminently by enterprises. Traditional data warehouse commonly just support some analysis and queries on history data while the variety of commerce information can’t be displayed in real-time. Real-time data warehouse extend the application of traditional data warehouse and can support tactical queries for enterprise. Real-time data warehouse
effectively cut the time-lapse and can provide much variable tactical decision support. For these reasons, it is very meaningful to study on the structure of real-time data warehouses.

The conflict between real-time data import and real-time data query will arise in the real-time data warehouse[1]. One of the most difficult parts in the construction of any data warehouse is to import data from different business systems to ECCL. More additional difficulties will be increased if the process becomes real-time. Almost all systems are in operation in batch mode[2]. In these systems, the available data is assumed to exist in a place in a kind of extracted file within a certain definite time schedule. Then, the systems transform and clean the data, and then import it to the data warehouse[3]. The processing exerts a significant influence on the nonresponse period of the data warehouse, for no users can access the data warehouse in the process of import. But with the continuous and real-time import of data, the system could not have any nonresponse periods. The most cumbersome retrieval time period for the data warehouse may be consistent with the time when the data is imported most frequently. Therefore, a fundamental contradiction has been produced between the system and this requirement for continuous update without nonresponse periods[4].

In this article, a data warehouse architecture design based on the real-time storage area has been proposed. Different from traditional data warehouses, the subsystem of this architecture is separated into real-time ETL (Extract/Transform/Loading) and periodic ETL. Periodic ETL refers to periodic batch import of data in data sources, which is finally loaded into the data warehouse. But for real-time ETL, CDC (Change Data Capture) tools directly and automatically capture change data from the data source, which is then loaded into the real-time storage area. Through query, update, deletion and other operations in the real-time storage area, when the system’s trigger conditions are satisfied, the data is then loaded into the data warehouse in batches. The storage part can be divided into real-time data storage area and static data storage area. Real-time data query and storage are implemented based on the real-time data storage area. Static data is just the data warehouse. Static data or historical data query is implemented based on the static data storage area. Update query scheduling components can determine the operation order of query and update through the users’ demand for them. Data quality components could generate a report when showing query results, so that users can know those data the query contents refer to and the freshness of these data. Through query control components, real-time data query can be implemented in the real-time storage area. Compared with the traditional data warehouses, the response time for query is significantly improved. For historical data query, seamless links are needed between the real-time storage area and data warehouse, so as to ensure the accuracy of query. In this article, a replication mechanism based on double mirror partition alternation has been put forward to realize the real-time storage area and satisfy users’ demand for data of different freshness, in order to reduce the response time for query.

2. DOUBLE mirror replication mechanism

Real-time storage area is made up of double mirror partitions based on data warehouse replication mechanism, which are operated alternately. Such as Fig1. This plays a vital role in solving query competitions.
With the continuous and real-time import of data, the query operation from users will be much affected, including its accuracy and query efficiency. Therefore, a conflict between real-time data import and real-time data query will arise in the real-time data warehouse. When data is loaded to the real-time storage area in ETL, it is likely for users to be making many inquires about the real-time storage area. These inquires will be brought into the same statistical result, and then we shall have to consider whether the new data will be brought into the result. If we do not make any improvement, the accuracy of query results with OLAP is usually barely satisfactory, which thus requires us to design a new structure to solve this problem.

So in this paper, double mirror partitions are constructed in the real-time storage area, whose logic structures and physical structures are all the same. Then, when the records in Mirror Partition 1 are loaded to the data warehouse in batches owing to trigger conditions or operations from the database administrator, the new data in OLTP server will be loaded into Mirror Partition 2. Namely, Mirror Partition 1 is used for query, and Mirror Partition 2 is used for receiving new data, so as to prevent conflicts. Until next time Mirror Partition 2 meets conditions and the records need to be loaded to the data warehouse, the contents in Mirror Partition 1 are depleted, and new data needs to be reloaded into Mirror Partition 1. Query is made in Mirror Partition 2. The two parts of the real-time storage area are working alternately in this way. One partition is used for query, another for receiving ETL update data.

So, the real-time storage area jointly constructed by double mirror partitions can achieve real-time loading of continuous data. This alternate use of Mirror Partition 1 and 2 can not only effectively solve the conflict between update and query, and also meet the query requirement from the real-time storage area when data is loaded in batches.

However, this structure also has certain disadvantages. Although we have reduced operations on the data warehouse by batch loading, due to the mirror partition alternate update mechanism, we cannot get a very good guarantee for the freshness of query results. Thus, based on the double mirror partition alternate replication mechanism, multi-level caching mechanism is proposed in this paper to solve the freshness problem of data.

3. Multi-level caching mechanism

For a real-time data warehouse solution, the most difficult problem is to overcome conflicts and contradictions between user query and system update at any time. Complex selection operation and continuous data insertion behavior have seriously affected the performance of the data warehouse. At present, this problem has been solved through the use of a data warehouse with a larger capacity, external data caching and simplifying the query operation. In a data warehouse, data is divided into two parts, namely static data and dynamic data. Static data can satisfy users’ requirements for query and analysis; dynamic data can achieve real-time property. Once there are updates in the data sources, they will soon be
loaded into the dynamic data of the data warehouse, which then go through the corresponding conversions, so as to meet the demand of real-time property.

The core of multi-level caching mechanism is multi-level caching with a queue and freshness scheduling module. Data will be loaded into the data storage area by queues and finally stored in the real-time storage area. It is the fresh data that can be stored there, so today’s data should be kept in the real-time storage area. Then data there is loaded periodically into the data warehouse in batches. When we position query through the freshness scheduling module, we shall also consider that the recent data should be inquired in the real-time storage area, and historical data should be inquired in the data warehouse. If necessary, effective seamless links are needed at the same time for the contents of multi-level caching, the real-time storage area and the data warehouse. The architecture is shown in Fig 2.

![Fig. 2 The architecture of real-time data warehouse with multi-level cache](image)

Through the query caching mechanism, we can effectively control the influence of a lot of inquires on the system performance, and improve the query speed. The key to increasing query caching is to apply a useful query matching algorithm, with which we can quickly and accurately identify similar inquires, and guarantee the settlement of query conflicts as well as the accuracy of the results.

Since real-time data changes and data query will influence the performance of the data warehouse, in case the real-time property is not reduced, the most simple solution is to try to reduce the complexity of inquires, restrict users to perform very complex inquires, or lower the refresh frequency of real-time data and support complex inquires. For the modeling method of separated real-time data, real-time data and historical data query can be handled separately. That is, when real-time data is inquired, historical data will not be inquired, and vise versa.

Historical data is deposited in a cache pool in result sets, which can improve the query and analysis efficiency with OLAP, so as to provide timely analysis results for users. However, if we ignore the operating characteristic of data cache technology, the performance of query and analysis with OLAP will be declined. For example, due to fast changes of the data items in the buffer, too long or short cache time, or other factors, the response speed from the system to user query and analysis will be reduced. The major indexes for the evaluation of the cache are: buffer capacity, hit rate, access delay, cache replacement algorithm, cache synchronous algorithm, etc. There exists a certain contradiction between hit rate and buffer capacity. The increased buffer capacity will cause a reduced cache hit rate, declined cache efficiency and increased system access delay. So buffer capacity is generally fixed in a certain size. When the buffer capacity is inadequate, a certain cache replacement algorithm is needed to remove those cache objects of less “value” for the system from the cache, but put those objects of more “value” in the cache.
A good replacement strategy, namely cache updating algorithm, plays a decisive role in the efficiency of the cache.

4. The updating algorithm of the Multi-level cache

The updating algorithm of the first-level cache has been described in Algorithm 1. Q1 contains the continuous data from the source system, which are those that will be loaded into the first-level cache. Meanwhile, these data will be inserted into the queue Q2, convenient for data updating in the second-level cache. In this way, data has backups in Q2, and then the data in C1 can be directly removed when it is refreshed, which can reduce operations on the cache.

Algorithm 2 is the updating algorithm of the ith-level cache, where i belongs to [2, n], and its difference from the first-level updating algorithm is: in the first-level cache, data in Q1 is all directly loaded into the cache; for the data in the ith-level cache, it can be loaded into the ith-level cache until necessary preprocessing is made. The preprocessing mainly includes deleting the old version of the data, which aims to guarantee the freshest data storage in the cache, and can reduce operations on the cache.

For a query Q, if it requires at most one hour for the fresh degree, the fresh levels for the caches C1, C2, C3, C4 and C5 are 0 minutes, 10 minutes, 30 minutes, 60 minutes and 240 minutes. We can position this query directly into the fourth-level cache C4, which then responds to this query. In this way, C4 just meets the needs of Q.

Here, Ti is the updating cycle of each queue. Each level of queues update the cache based on different cycles, so as to avoid frequent operations on the cache.

Algorithm 4.1 Update method of level-1 cache

**INPUT**: Input data queue Q1
Input data queue Q2
The dataset C1 in the level-1 Cache

**OUTPUT**: the updated C1 and Q2

1. **while** (Q1 is not empty)
2. d=Q1.pop();
3. integrate d into C1;
4. Q2.put(d);
5. return C1, Q2

Algorithm 4.2 Update method of level-i cache

**INPUT**: Input data queue Qi
Input data queue Qi+1
Updating cycle Ti
The dataset Ci in the level-i Cache

**OUTPUT**: the updated Ci and Qi+1

1. **while** (Ti begins)
2. **while** (qi is not empty)
3. d=qi.pop();
4. put d into set M;
5. **if** (d is a version of d, and F(d)<F(d))
6. Delete d from M;
7. for (each mM)
8. integrate m into Ci;
9. Qi+1.put(m);
10. return Ci, Qi+1
5. Combination of multi-level cache and real-time storage area

The real-time data warehouse has a three-tier storage structure. In order to simplify the operation and improve performance, multi-level cache has the same data structure as the real-time storage area.

In a table with relatively small data volume, it is unnecessary for us to establish such a data structure like indexing or aggregation that is used to optimize the performance of the data warehouse, which will instead increase the operation complexity of multi-level cache. So the multi-level cache has the same data structure as the real-time storage area, which facilitates the cache to be regularly imported to the real-time data storage area. The three-tier storage structure has effectively shared the load, and improved the query response time.

6. Experiments

In the case of a lot of queries at the same time, compare the differences in query performance between the multi-level cache and single cache. Here, we set two query sets of S1 and S2, which are both based on the multi-level cache. They are both composed of the four queries of QS1, QS2, QS3 and QS4, whose proportions are seen in Table 1. In this experiment, 50 records of updating take place per second, and a lot of queries also compete with them in the meantime. So we set 5~100 queries per second in the cache. In Figure 4, the query execution time in different conditions has been shown. SC-S1 indicates that S1 is running in the single cache, and MC-Si indicates that query is running in the multi-level cache. From the result, we can obviously find that the processing time of the multi-level cache is lower than that of the traditional single cache structure. Meanwhile, the reason why S1 is less efficiency than S2 is that this multi-level caching mechanism can position a query to a cache it is fit for. Also, the proportions of the four queries in S1 are not balanced, of which QS1 has accounted for more than 80%. This 80% of queries are allotted to one cache, namely that the same cache has stored most queries in S1. In contrast, for S2, its four queries are equally distributed in four levels of caches, which have effectively shared the query load. So S2’s query execution time is less than S1’s, namely that S2’s efficiency is higher than S1’s. It has also reflected where the multi-level cache has advantages.

Experiment 2  In the case of a lot of updates at the same time, compare the differences in query performance between the multi-level cache and single cache, as shown in Figure 5. The number of queries has all been changed into 100, but the frequencies of updates differ from one another, updating from 10 records per second to 50 records. In different update frequencies, when the multi-level cache and single cache are solving query conflicts, it is still the multi-level cache that has an obvious advantage over the query execution time, which is particularly outstanding when the query frequency is high.

From the experiments on the general real-time storage area and the real-time storage area based on the multi-level cache, we are informed that the real-time storage area solution in this article has a very reliable advantage over the speed in dealing with affairs, and the performance will be better if memory has a larger capacity. We found that at least 8% of the response time would be spent with this method, which was independent of memory and had nothing to do with the memory size. The experimental results also told us that, in the same condition, the larger memory would have less response time, namely rapid response. Meanwhile, for this architecture, with the increasingly frequent occurrence of affairs, the response time extended. With a multiple increase in affairs, the response time growth can still be controlled within 25%, which can be totally accepted by people and satisfy users.
Table 1 Query Sets Composition

<table>
<thead>
<tr>
<th></th>
<th>QS1</th>
<th>QS2</th>
<th>QS3</th>
<th>QS4</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>80%</td>
<td>10%</td>
<td>10%</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>10%</td>
<td>30%</td>
<td>30%</td>
<td>30%</td>
</tr>
</tbody>
</table>

Fig. 3 Query Execution Time Under Different Number of Queries

Fig. 4 Query Execution Time Under Different Updating Frequency

Acknowledgement

In this paper, the multi-level cache method has been used to solve query competitions in the real-time data warehouse. In the experiment of the real-time storage area based on the multi-level caching mechanism, the multi-level cache is apparently much higher efficiency than the single caching mechanism, and its processing time is less than that of the traditional single cache structure. For a large number of queries produced simultaneously, if there are more kinds of queries, the multi-level cache has more obvious advantages. It can locate queries which require different freshness to different levels of caches, so as to avoid too many queries in the same cache, which may save system resources effectively and reduce those unnecessary query operations on the data warehouse.

References


