Maintaining Internal Consistency of Report for Real-Time OLAP with Layer-Based View

Ziyu Lin[1,⋆], Yongxuan Lai[2], Chen Lin[1], Yi Xie[1], and Quan Zou[1]

[1] Department of Computer Science, Xiamen University, Xiamen, China
[2] Software School, Xiamen University, Xiamen, China
{ziyulin,laiyx,chenlin,csyxie,zouquan}@xmu.edu.cn

Abstract. Maintaining internal consistency of report is an important aspect in the field of real-time data warehouses. OLAP and Query tools were initially designed to operate on top of unchanging, static historical data. In a real-time environment, however, the results they produce are usually negatively influenced by data changes concurrent to query execution, which may result in some internal report inconsistency. In this paper, we propose a new method, called layer-based view approach, to appropriately and effectively maintain report data consistency. The core idea is to prevent the data involved in an OLAP query from being changed through using lock mechanism, and avoid the confliction between read and write operations with the help of layer mechanism. Our approach can effectively deal with report consistency issue, while at the same time avoiding the query contention between read and write operations under real-time OLAP environment.

Keywords: OLAP; report consistency.

1 Introduction

Real-time data warehouses have been receiving more and more attention (e.g. [1,2,3,4,5,6]) during the past few years, which is updated in as close to real time as possible [7]. However, OLAP and Query tools were initially designed to operate on top of static data, and they do not ensure that the data involved is protected from being modified. Therefore, the report result may be negatively influenced by the underlying changing data. In real-time data warehouse environment, this can lead to inconsistent and confusing query results, which is called *internal inconsistency of report* [8].

Take the simple report in Fig. 1 for example. It includes a multi-pass SQL statement made up of many smaller SQL statements. All these SQL statements will sequentially operate on a set of temporary tables. There will be no problem when the data is static, but it is not the case when the underlying data changes while the first temp table is being created. Most database systems (including multi-version databases [9]) will return the data that was current at the point that the query started to run [8]. At 0:01, the INSERT statement into TEMP1 started to run and lasted for four seconds. Then the query to load

⋆ Supported by the Natural Science Foundation of China under Grant No. 61001013 and 61001143, and the Fundamental Research Funds for the Central Universities under Grant No. 2010121066.

X. Du et al. (Eds.): APWeb 2011, LNCS 6612, pp. 143–154, 2011.

```
0 : 00  create table TEMP1{Category_Id LONG, DOLLARSALES DOUBLE}
0 : 01  insert into TEMP1
            select all.[Category_Id] AS Category_Id
               sum (all.[Tot_Dollar_Sales]) AS DOLLARSALES
            from [YR_CATEGORY_SLS] all
            group by all.[Category_Id]
0 : 05  create table TEMP2 (ALLPRODUCTSD DOUBLE)
0 : 06  insert into TEMP2
            select sum((all.[Tot_Dollar_Sales]) AS ALLPRODUCTSD
            from [YR CATEGORY_SLS] all
0 : 08  select distinct pa1.[Category_Id] AS Category_Id,
            all.[Category_Desc] AS Category_Desc,
            all.[DOLLARSALES] AS DOLLARSALES,
            pa1.[DOLLARSALES]/pa2.[ALLPRODUCTSD]) AS DOLLARSALESC
        from [TEMP1] pa1,
           [TEMP2] pa2,
           [LU CATEGORY] all
        where pa1.[Category_Id]=all.[Category_Id]
0 : 09  drop table TEMP1
0 : 10  drop table TEMP2
```

Fig. 1. Sales by category with percent contribution

data into TEMP2 began to run at second 6. This means that TEMP1 will contain data current as of 0:01, but for TEMP2, it will contain data current as of 0:06. Suppose that during those five seconds, a few large sales were registered, they will be included in the total dollar amount contained in TEMP2, but won't be represented in the category-level data that is in TEMP1. So, when the data is brought together in the final SELECT statement, the total in TEMP2 will be larger than the sum of the categories in TEMP1, and then the total percentage number will be less than 100%. Obviously, this will lead to an incorrect report.

Multi-version database is a desirable approach to ensuring read consistency. However, read consistency in multi-version database is only achieved on the level of single-pass SQL statement [9], which means that it can not be used to deal with the internal consistency of report that contains multi-pass SQL statement made up of many smaller SQL statements (see Fig.1). Moreover, multi-version database is not good at dealing with the query contention issue resulting from the real-time update and query. Temporal model (e.g. [10]) is another one of the methods that can be used to solve the report inconsistency issue. However, keeping temporal data warehouses up-to-date is complex [11], and in some cases, the data warehouse may even become blocked due to the query contention issue resulting from performing queries on changing data.

We here propose a new layer-based view approach for appropriately and effectively maintaining report's internal consistency, and at the same time avoiding query contention issue which is a hard work for other available methods. The concepts of layer, view and lock are introduced to effectively control read and write operations upon fact tables. The core idea is that, all the data involved in an OLAP query is read-locked and is not allowed to be updated until the query finishes its reading job, so as to maintain

the internal consistency of report. When the data is read-locked, if there is confliction between read and write operations on certain layer, the write operation will be redirected to another layer to continue its work so as to avoid waiting time and maintain the consistency as well. To achieve this target, we will present in detail the mechanisms for layer generating and deleting, view generating and deleting, and lock applying and confliction resolving. Also the algorithm and an example of our approach will be given here. Compared with the other available methods, the advantage of our method is the avoiding of confliction between read and write operations upon fact tables, which means that there will be no waiting time any more and therefore desirable system performance can be achieved. Also, our method can be easily used in memory database. Unlike most of the multi-version databases, in which multi-version method is deeply integrated with the database systems, our method is completely independent of the type of database, which means that it can be used together with any database product.

The remainder of this paper is organized as follows: Section 2 gives the detailed description of layer-based view approach. Then we present experimental results in Section 3, followed by the discussion of the related work (Section 4). Finally, we give the discussion and conclusion in Section 5.

2 Maintaining Report's Internal Consistency with Layer-Based View Approach

In this section, the frequently used concepts will be defined first, followed by the description of the mechanisms of our approach in detail. Finally, we will give the algorithm and an example of our approach.

2.1 Term and Definition

Definition 1. ***Layer:*** *A layer, denoted by Δ , is a table to store a set of records $\{r_0, r_1, ..., r_n\}$. Layer can be classified into root layer and non-root layer. If, for two layers Δ_A and Δ_B, Δ_B is generated from Δ_A, we say Δ_B is the child of Δ_A, which is denoted by $\Delta_B \succ \Delta_A$, and Δ_A is the parent of Δ_B, which is denoted by $\Delta_A \prec \Delta_B$.*

In real-time data cache, every fact table is corresponded to one root layer. Every non-root layer, which is initially an empty table without any data when generated, has the same table structure as its parent layer.

Definition 2. ***View:*** *Let $L = \{\Delta_0, \Delta_1, ..., \Delta_{m-1}\}$ be a set of layers, and $R = \{r_0, r_1, ..., r_{n-1}\}$ be a set of records, where $r_i \in \Delta_j$, $0 \leq i \leq n-1$ and $0 \leq j \leq m-1$. A view is defined as Γ = (V, ϕ), where $V = \{v_0, v_1, ..., v_{n-1}\}$ is a set of records, $\phi(v_p) = r_q$, $0 \leq p \leq n-1$ and $0 \leq q \leq n-1$. Here we say Γ is composed of $\Delta_0, \Delta_1, ..., \Delta_{n-1}$, and Δ_j is a composing layer of Γ.*

A view defines the mapping between every record of itself and the records of its composing layers, from which OLAP tools get to know where the required data actually locate. There may be many views in the system, but the only view that can be seen by newly arrived OLAP queries is the "*current view*", which is denoted by $\Gamma_{current}$. After OLAP queries get $\Gamma_{current}$, they will use it during the whole reading process, even though the "current view" now may become "old view" in the future.

Definition 3. ***Area:*** *Let $R = \{r_0, r_1, ..., r_n\}$ be the set of records that a layer Δ (or a view Γ) contains. An area, denoted by δ, is a subset of R. Here we say the area δ is in the layer Δ (or in the view Γ), which is denoted by $\delta \subseteq \Delta$ (or $\delta \subseteq \Gamma$). Suppose there are $\delta, \delta_0, \delta_1, ..., \delta_{n-1}$, where $\delta \subseteq \Gamma$, $\delta_i \subseteq \Delta_j$, $0 \leq i \leq n-1$, $0 \leq j \leq m-1$ and m is the number of layers. If, by the function ϕ of Γ, the records in δ are mapped to the records in $\delta_0, \delta_1, ..., \delta_{n-1}$, we say that δ is composed of $\delta_0, \delta_1, ..., \delta_{n-1}$, and δ_j is a composing area of δ.*

In order to better understand the concepts of layer, area and view, we give an example in Fig.2. As can be seen in Fig.2, View1 is composed of three layers, i.e. Layer1, Layer2 and Layer3. The three records in View1, which are what OLAP tools can see, are actually located in the three different layers. This is similar to the layer technology used in painting software (e.g. PhotoShop), where a photo is composed of many layers, and what we can see is the result of combining the objects in different layers together. This also explains why a table is called a layer in our paper. In Fig.2, there are four areas, where Area1 $\subseteq$ Layer1, Area2 $\subseteq$ Layer2, Area3 $\subseteq$ Layer3 and Area4 $\subseteq$ View1. Area1 contains one record, Area2 contains three records, Area3 contains two records and Area4 contains three records.

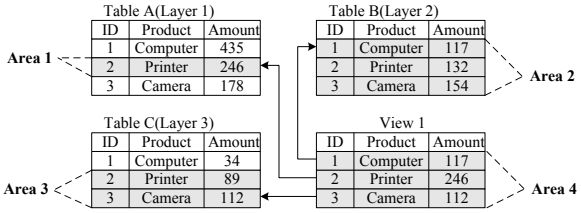

Fig. 2. An example of layer, area and view

Definition 4. ***Lock:*** *A lock Ω is used to lock certain area δ so as to control the read and write operations upon δ, where $\delta \subseteq \Gamma$ or $\delta \subseteq \Delta$. $\Omega_{on}(\delta)$ and $\Omega_{off}(\delta)$ mean placing locks on and removing locks from δ respectively.*

Lock includes "*read lock*" and "*write lock*" (see Table 1), and read lock has higher priority than write lock. The former is assigned by system to the OLAP tools to protect the target area being read from being updated, and the latter is used to inform the other operations that the target area is being updated by write operation, or else it will probably lead to the occurrence of inconsistency. Table 2 gives the compatibility relationship between read lock and write lock. Also a lock can be a virtual lock or an actual lock. A virtual lock is put on an area in a view, while the target object of an actual lock is an area in a layer.

2.2 Lock Mechanism

Lock mechanism is responsible for the jobs such as lock applying, lock translating and lock confliction resolving, so as to effectively control the read and write operations upon the changing data to maintain the internal consistency of report.

Table 1. Different types of locks

lock	virtual lock	read lock
		write lock
	actual lock	read lock
		write lock

Table 2. Lock compatibility matrix

	read lock	write lock
read lock	compatible	incompatible
write lock	incompatible	incompatible

Lock applying. Only virtual lock can be applied by OLAP tools and data loading and updating tools, for what these tools can see in the first place are views instead of layers.

- *Read lock applying*. The purpose of read lock is to declare the "occupation" of specific area by OLAP tools, which means the loading tools have no right to updating the "occupied" area. Since read lock has higher priority than write lock, read lock applying process never fails. For a multi-pass SQL statement, its read lock application is submitted as a whole to the system.
- *Write lock applying*. Write lock is used by updating operation to express the intention of updating certain area. Write lock has lower priority than read lock, and therefore its request may not be satisfied all the time. For a transactional updating statement, its write lock application is submitted as a whole to the system.

Definition 5. ***Lock transforming:*** *Suppose* δ *is composed of* $\delta_0, \delta_1, ..., \delta_{n-1}$*, where* $\delta \subseteq \Gamma$*,* $\delta_i \subseteq \Delta_j$*,* $0 \leq i \leq n-1$*,* $0 \leq j \leq m-1$ *and* m *is the number of layers. Lock transforming is the process of transforming the virtual lock on* δ *into actual locks on* $\delta_0, \delta_1, ..., \delta_{n-1}$*, which is denoted by* $\Omega_{on}(\delta) \rightarrow \Omega_{on}(\delta_0) \cup \Omega_{on}(\delta_1) \cup ... \cup \Omega_{on}(\delta_{n-1})$.

A view is composed of one or more layers, and therefore there is a need to transform the lock on it into one or more locks on layers. The following is an example to explain the transformation mechanism between virtual lock and actual lock.

In Fig. 3, View1 is composed of three layers, i.e. Layer1, Layer2 and Layer3. While what OLAP tools can see is View1, with the help of view definition, they will finally be "guided" to the three layers where the data actually locate. This also takes place for write operations. Now suppose that Lock0 is imposed on View1, which can be either a read lock or a write lock. Through the analysis of view definition, Lock0 is finally translated into three locks with Lock1 on the second record of Layer1, Lock2 on the first record of Layer2 and Lock3 on the third record of Layer3.

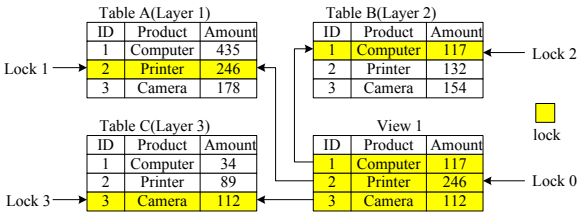

Fig. 3. The transformation of virtual lock into actual lock

Definition 6. ***Lock confliction*** *If, according to the lock compatibility matrix in Table 2, two locks $\Omega_{on}(\delta)$ and $\Omega'_{on}(\delta)$ are incompatible, we say that there is lock confliction on δ.*

Whenever there is lock confliction on δ ($\delta \subseteq \Delta$), the write operation has to be redirected to another layer Δ_{child}, which is a child layer of Δ. If Δ_{child} does not exist, it will be generated automatically by system through the mechanism of layer generating. This process is called *lock confliction resolving*, during which, one important aspect is to maintain the transactional consistency of the undergoing write operation.

2.3 Layer and View Mechanism

Layer generating: The layer generating process is activated whenever the write operation needs to be completed in another layer due to the lock confliction, and at the same time there is not one layer available for it. The newly generated layer has the same table structure as its parent layer, but is initially empty.

Layer merging and deleting: In order to achieve better system performance, there is also a need to merge and delete layers under certain condition. Whenever the preset threshold is met, the layer merging process begins.

View generating: Whenever there is updating against the definition of a view, a new view containing the newest definition will be generated above the old one with the latter unchanged, and then the new view becomes $\Gamma_{current}$. The reason for the generation of a new view when the definition of the current view is changed, is out of the consideration of avoiding the confliction between the read operation of OLAP tools and the write operation of updating against $\Gamma_{current}$.

View deleting: The process of view deleting begins when certain view is no longer used by OLAP tools, or it will lead to the depletion of system resources, because new views are generated constantly along the time. Sometimes the deleting process of views is also accompanied with the merging of layers.

2.4 The Algorithm for Layer-Based View Approach

Based on the mechanisms described above, we can now implement the layer-based view approach. Fig. 4 shows the main algorithm for this method. The 5th and 12th lines are executed concurrently instead of sequentially with the help of multi-thread technology. In other words, the read process and the confliction resolving process go at the same time. And for every layer, the confliction resolving process can also execute concurrently. The read operation will read the right now available data first so as to avoid the waiting time, and then read the previously write-locked but currently available data. Even though the transactional updating has to be assured to be finished when the undergoing write operation needs to be redirected to another layer during the confliction resolving process, the overall waiting time of read operation is still very little and usually can be neglected due to the adoption of multi-thread technology as is described above.

In the 7th line, if the undergoing write operation is a part of transactional updating, it can not be stopped until the transaction is finished. In the 8th and 21th lines, as far as

Input: an SQL statement S
the current view $\Gamma_{current}$
Output: execution result of S

1. scan the statement to get the value of δ;
   /* δ is the area in $\Gamma_{current}$ that S is requried to lock;*/
2. **if** S is a read operation
3. $\Omega_{on}(\delta)$;
4. $\Omega_{on}(\delta) \rightarrow \Omega_{on}(\delta_0) \cup \Omega_{on}(\delta_1) \cup ... \cup \Omega_{on}(\delta_{n-1})$;
5. **for** $(i = 0; i < n; i++)$
6. **if** (there is lock confliction on δ_i) /*$\delta_i \subseteq \Delta$*/
7. stop the undergoing write operation rightly;
8. find another right layer Δ_{child} for the undergoing write operation;
   /* Δ_{child} is a child layer of Δ; */
9. continue to do the suspended write operation on Δ_{child};
10. generate a new view Γ_{new} to record the layer information;
11. $\Gamma_{current}$ = Γ_{new};
12. read the data on the target layers into a temporary table T;
13. **for** $(i = 0; i < n; i++)$
14. $\Omega_{off}(\delta_i)$;
15. **return** T;
16. **else** /* S is a write operation*/
17. $\Omega_{on}(\delta)$;
18. $\Omega_{on}(\delta) \rightarrow \Omega_{on}(\delta_0) \cup \Omega_{on}(\delta_1) \cup ... \cup \Omega_{on}(\delta_{n-1})$;
19. **for** $(i = 0; i < n; i++)$
20. **if** (there is lock confliction on δ_i) /*$\delta_i \subseteq \Delta$*/
21. find another right layer Δ_{child} for the write operation of S;
22. do the write operation of S on Δ_{child};
23. generate a new view Γ_{new} to record the layer information;
24. $\Gamma_{current}$ = Γ_{new};
25. **else**
26. do the write operation of S on Δ ;
27. **for** $(i = 0; i < n; i++)$
28. $\Omega_{off}(\delta_i)$;
29. **return** success information;

Fig. 4. The algorithm for layer-based view approach

Δ_{child} is concerned, it may be the existing child layer of Δ or a newly generated child layer of Δ according to the different conditions, and it has the same table structure as Δ.

2.5 An Example of Layer-Based View Approach

In order to better understand how our layer-based view approach works, we here give an example.

As can be seen in Fig.5, at time T_1, there is only one layer, i.e. Layer1 in the system. View1 ($\Gamma_{current}$) is composed of Layer1, and is also what OLAP tools can see at time T_1. We suppose that a query Q_1 has already read-locked all the records (i.e. v_1, v_2 and v_3) of View1 before time T_1, and it will not release its read locks until time T_3.

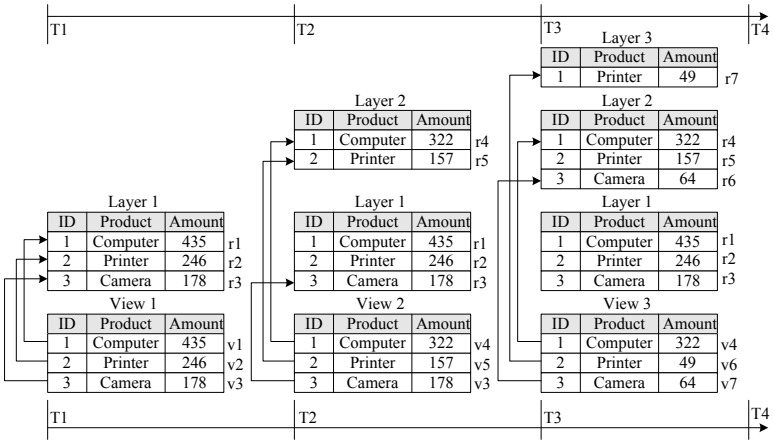

Fig. 5. The generating process of layers and views

According to the knowledge about the lock transformation, the virtual read locks on v_1, v_2 and v_3 of View1, are transformed into the actual read locks on r_1, r_2 and r_3 of Layer1. So r_1, r_2 and r_3 of Layer1 are all read-locked. Then, at time T_1, an update operation U_1 arrives at the system, expecting to update the records v_1 and v_2 of View1. Before starting the update work, U_1 must apply to the system for the virtual write locks on v_1 and v_2. The virtual write locks will then be transformed into the actual write locks on r_1 and r_2. However, r_1 and r_2 are already read-locked by Q_1, and also in our method, read lock has higher priority to write lock, so the write lock application of U_1 fails due to the lock confliction. But U_1 will not wait for the read locks to be released, and it will be redirected to Layer2, the child layer of Layer1, to continue its job. In other words, U_1 will write the update results (r_4 and r_5) into Layer2. Layer2 is automatically generated by the system to accommodate the write operation U_1. After U_1 completes its work, the system will generate a new view, i.e. View2, to reflect the most recent data, and then View2 becomes $\Gamma_{current}$. From now on, View2 is what OLAP tools can see, but View1 will not be deleted until Q_1 finishes its work.

Then, at time T_2, another query Q_2 arrives at the system, and needs to read v_4, v_5 and v_3. Q_2 first applies virtual read locks on v_4, v_5 and v_3 of View 2, which will be transformed into three actual locks, i.e. the actual read locks on r_4 and r_5 of Layer2, and the actual read lock on r_3 of Layer1. Read lock application will never fail, so Q_2 starts to read the locked records, and here we suppose that it will not release its read locks until the time T_3. When Q_2 is undergoing its read work, another update operation U_2 arrives at the system, and expects to update the records v_5 and v_3 of View2. Its virtual write locks on v_5 and v_3 of View2 will be transformed into actual write locks, i.e. the lock on r_5 of Layer2 and the lock on r_3 of Layer1. Because r_3 of Layer1 is still read-locked, lock confliction occurs, and the system will redirect U_2 to Layer2, the child layer of Layer1, to do the update against r_3. Similarly, update operation against r_5 will also be redirected to Layer3, the child layer of Layer2. After U_2 finishes its work, the system will generate a new view, i.e. View3, to present the most recent data, and

then View3 becomes $\Gamma_{current}$. From now on, View 3 is what OLAP tools can see, but View2 will not be deleted until Q_2 completes its work.

At time T_3, both Q_1 and Q_2 finish their reading job, and release their read locks. During the whole reading process, data involved in Q_1 (r_1,r_2 and r_3) and that involved in Q_2 (r_4,r_5 and r_3) are never changed, so the internal consistency of report is well maintained. Also, when the reading work is undergoing, the update operations of U_1 and U_2 are not blocked, instead they perform their jobs smoothly and successfully.

3 Empirical Study

Now we report the performance evaluation of our method. The algorithms were implemented with C++. All the experiments are conducted on 4*2.4GHz CPU (double core), 32G memory HP Proliant DL585 Server running Windows Server 2003 and Oracle 10g (for operational system and data warehouse) and Oracle TimesTen In-Memory Database (for real-time data cache).

We use TPC benchmark TPC-H to get the required datasets in our experiment. DBGEN, a tool provided by TPC, is used here to generate the required datasets to populate the database in the data source. We take real-time data cache [8] running Oracle TimesTen In-Memory Database to store all the real-time data. The external data cache database is generally modeled identically to the data warehouse, but typically contains only the tables that are real-time. Also through JIM or RJIM system [8], we can seamlessly combine the real-time data in the data cache and historical data in the data warehouse. With the help of Streams Components provided by Oracle 10g, it is easy to capture the change data in the data source and send them to the destination queue, from which they are dequeued to be integrated into the data caches.

Performance ratio. In this experiment, we will show that our method can not only maintain report internal consistency, but also effectively avoid the contention between read and write operations so as to achieve desirable performance for both update and query. In order to show the influence of read and write operations upon system performance, we change the contention ratio (denoted by r) between these two kinds of

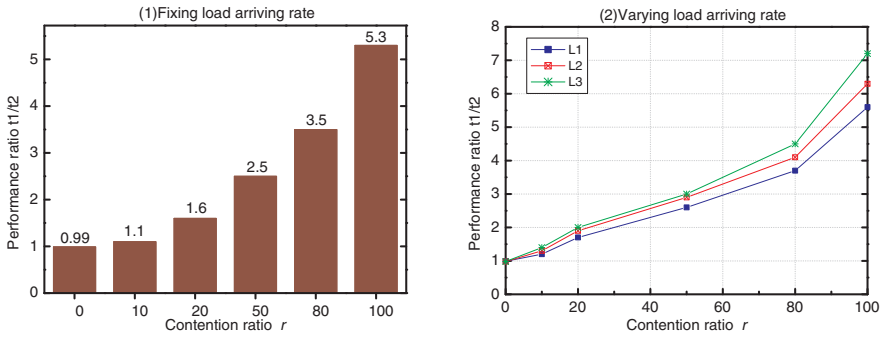

Fig. 6. Performance ratio. (1) Fix the load arriving rate, and change the value of contention ratio r from 0 to 100%. (2) Change the load arriving rate, and for each type of load L_1, L_2 and L_3, change the value of contention ratio r from 0 to 100%.

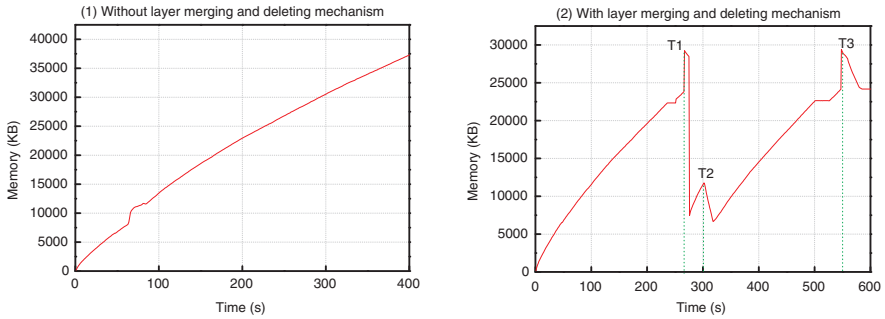

Fig. 7. Memory usage. (1) When there is no layer merging and deleting mechanism, the memory space is to be exhausted. (2) When there exists layer merging and deleting mechanism, the memory can be recycled.

operations in the load. Fig. 6 (1) shows how the performance ratio t_1/t_2 changes when varying the value of r from 0 to 100%, where t_1 denotes the total running time for the given update and query load L when not taking layer based view approach, and t_2 the total running time for L when taking our method. In Fig. 6 (1), we can see that when r equals 0, which means there is no read and write contention, the value of t_1/t_2 is 0.99. In another word, under such circumstance, it will bring negative benefits, though only a few, when taking layer based view approach. This is due to the additional cost for layer management. However, benefits from our method will become more and more evident when the contention ratio r increases. As can be seen in Fig. 6 (1), the performance ratio is 5.3 when r equals 100%.

We also test our method under three different kinds of workloads, i.e., L_1, L_2 and L_3. These workloads contain the same sequence of update and query operations with the same contention ratio, but feeding the system at different rates. The update and query operations in L_1 arrive at the slowest rate, and those in L_3 at the fastest rate. Fig. 6 (2) shows that our method can achieve much better performance ratio (the same as the definition above, i.e., t_1/t_2) when arriving rate is faster. Since there will be much more contention between read and write operations within a given time window when increasing the arriving rate, it can be concluded that our method can effectively deal with the query contention issue besides maintaining report internal consistency.

Memory usage. In this experiment, we will show that our method can effectively manage memory usage through the mechanism of layer merging and layer deleting. Fig. 7 (1) shows the memory usage when there is no layer merging and deleting mechanism. In such case, more and more memory is used to support the continuously generated layers. Since there is no layer dropping mechanism, those layers without any use in the future still reside in memory, which wastes much memory space and is to exhaust the limited memory resource in the end. In contrast, the memory resource can be recycled (see Fig. 7 (2)) when there exists layer merging and deleting mechanism. In Fig. 7 (2), there are three turning point T_1, T_2 and T_3 in the memory usage curve. At T_1 and T_3, the memory usage amounts to a predefined value M, layer merging process starts, which merges different layers into one if condition is met and drops many useless layers to

release the memory. At T_2, a lot of DELETE operations occurs in the system, which leads to the start of layer dropping process and much memory is recycled.

4 Related Work

Report's internal consistency is an important issue for real-time OLAP. Temporal model (e.g. [10]) can be used to solve the problem, which enables analytical processing of detailed data based on a stable state at a specific time [11]. While research in temporal databases has grown immensely in recent years, only a few DWH research projects paid attention to such problem as temporal model. Until the work done by Bobert et al. in [10], most of the previous research has been concentrated on performance issues, rather than higher-level issues, such as conceptual modeling [12]. In [10], an approach is presented to model conceptual time consistency problems, in which all states that were not yet known by the system at specific point in time are consistently ignored, and thus enables timely consistent analyses.

Temporal model addresses the issue of supporting temporal information efficiently in data warehousing systems, however, keeping temporal data warehouses up-to-date is complex [11]. Usually, it is more appropriate to be used to deal with the temporal consistency problem brought by late-arriving data, than to be used to solve the problem of report's internal consistency resulting from the continuous data integrating in the real-time data warehouse environment.

Another desirable way is to use an external real-time data cache, and at the same time without compromising report's internal consistency, data latency, or the user experience [8]. However, there are problems with the method even with the help of JIM system. The most obvious one is that, in the real-time data warehouses environment, real-time data cache is continuously updated, and therefore the "read" operation of getting a snapshot for OLAP tools will undoubtedly conflict with the "write" operation of data updating. If there are many concurrent users in the system, such confliction is to deteriorate system performance greatly. Such problem can not get resolved simply through adding more hardware to the system.

Multi-version database is a desirable method to maintain read consistency [9]. To some extent, our method is similar to multi-version database. However, there are still great differences between them. For example, our method can provide read consistency for multi-pass SQL statement, but it is hard for multi-version database. Also, our method is independent of DBMS and can be used in memory database. However, multi-version method is usually integrated into DBMS products, and can not be used in memory databases in some cases. Furthermore, our method can effectively deal with query contention issue besides maintaining report internal consistency, while multi-version database can not.

There are also some other methods available now, such as using a near real-time approach and risk mitigation for true real-time [8]. In [13], a new method is presented, which raises the level of abstraction for the use of replicated and cached data by allowing applications to state their data currency and consistency requirements explicitly and having the system take responsibility for producing results that meet those requirements.

5 Discussion and Conclusion

In this paper, we have revisited the issue of maintaining internal consistency of report for real-time OLAP. We propose a new method, called layer-based view, to effectively maintain report's internal consistency in real-time data warehouses environment. Important concepts such as layer, view and lock are defined, and the related mechanism, especially the layer generating mechanism, are discussed in detail. The advantages provided by layer-based view approach include no confliction between read and write operations, achievement of report's internal consistency and faster response time for OLAP queries.

Important future research directions in this field will be the appropriate definition of threshold for layer merging process, and the application of the theory into the field of read-time data warehouses in business environment.

References

1. Tho, M.N., Tjoa, A.M.: Zero-Latency Data Warehousing for Heterogeneous Data Sources and Continuous Data Streams. In: 5th International Conference on Information Integrationand Web-based Applications Services, pp. 55–64. Austrian Computer Society, Vienna (2003)
2. Thalhammer, T., Schrefl, M.: Realizing active data warehouses with off-the-shelf database technology. Software-Practice & Experience 32(12), 1193–1222 (2002)
3. Chen, L., Rahayu, J.W., Taniar, D.: Towards Near Real-Time Data Warehousing. In: 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 1150–1157. IEEE Computer Society, New York (2010)
4. Santos, R.J., Bernardino, J.: Real-time Data Warehouse Loading Methodology. In: 12th International Database Engineering and Applications Symposium, pp. 49–58. ACM, New York (2008)
5. Bateni, M., Golab, L., Hajiaghayi, M.T., Karloff, H.J.: Scheduling to Minimize Staleness and Stretch in Real-time Data Warehouses. In: 21st Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 29–38. ACM, New York (2009)
6. Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N.: Supporting Streaming Updates in an Active Data Warehouse. In: 23rd International Conference on Data Engineering, pp. 476–485. IEEE Computer Society, New York (2007)
7. Conn, S.S.: OLTP and OLAP Data Integration: a Review of Feasible Implementation Methods and Architectures for Real Time Data Analysis. In: Proceedings of SoutheastCon 2005, pp. 515–520. IEEE Computer Society, New York (2005)
8. Langseth, J.: Real-Time Data Warehousing: Challenges and Solutions, `http://www.dssresources.com`
9. Korth, H.F.: Database System Concepts, 3rd edn. McGraw-Hill, New York (1999)
10. Bruckner, R.M., Tjoa, A.M.: Capturing Delays and Valid Times in Data Warehouses-Towards Timely Consistent Analyses. Journal of Intelligent Information Systems 19(2), 169–190 (2002)
11. Bruckner, R.M., Tjoa, A.M.: Managing Time Consistency for Active Data Warehouse Environments. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2001. LNCS, vol. 2114, pp. 254–263. Springer, Heidelberg (2001)
12. Pedersen, T.B., Jensen, C.S., Dyreson, C.E.: A Foundation for Capturing and Querying Complex Multidimensional Data. Information Systems 26(5), 383–423 (2001)
13. Guo, H., Larson, P.A., Ramakrishnan, R., Goldstein, J.: Relaxed currency and consistency: how to say "good enough" in SQL. In: ACM SIGMOD 2004 International Conference on Management of Data, pp. 815–826. ACM, New York (2004)