QoS-LS: QoS-based Load Scheduling Algorithm in Real-Time Data Warehouse

Jingang Shi[1, a], Song Guo[1, b], Fangjun Luan[1, c], Limei Sun[1, d]

[1]School of Information and Control Engineering, Shenyang Jianzhu University, Shenyang, 110168

[a]email: shijingang@sjzu.edu.cn, [b]email: guosong@sjzu.edu.cn

[c]email: luanfangjun@sjzu.edu.cn, [d]email: sunlimei@sjzu.edu.cn

Keywords: data warehouse; real-time; QoS; data freshness

Abstract. In real-time data warehouses, data updates are no longer implemented in a periodic way during the idle time, but continuously ongoing. Thus the scheduling of updates and queries becomes a key issue. This paper proposes a load scheduling algorithm based on QoS parameters of query tasks. First, the paper defines some QoS parameters related to queries. Then, according to the specific QoS requirements of queries, the algorithm makes a real-time load scheduling for updates and queries. Finally, some experiments show that the algorithm can adjust the running order of tasks reasonably and use the system resources effectively to provide the faster query response and fresher data to users, according to the specific QoS requirements.

1 Introduction

Data warehouses play important roles in today's information society. But a growing number of business decisions require real-time data support, which raised the real-time data warehouse (RTDWH) [1, 2]. RTDWH uses a continuous real-time method that puts the change data of OLTP system into data warehouses, instead of a periodic loading data method [3]. Because data import is executed during the idle time in the traditional data warehouse, updates and queries do not arrive at the same time. While in RTDWH, data import continues to carry out, so updates and queries are executed at the same time. It will result in two effects. Firstly, there is resource competition for updates and queries. Secondly, it causes data inconsistency while updates and queries are executed in parallel. Therefore, it becomes a key issue how to make a reasonable distribution of system resources, to execute more efficiently updates and queries, and to ensure high data freshness and rapid response to queries to the greatest extent.

There are some algorithms in the traditional scheduling, such as First arrive First execute (FAFE), FAFE Update First (UF), FAFE Query First (QF) [4] and so on. However, they don’t consider the task balance between updates and queries. And they do a uniform treatment for updates and queries, without considering the different requirements for the different queries to users. In this paper, we design a QoS-based updates and queries load scheduling algorithm (QoS-LS), which defines some QoS parameters of queries and does an integrated task scheduling for updates and queries according to the parameters requirements.

The paper is organized as follows. Section 2 introduces the related works on the scheduling technology. Section 3 describes the system architecture of the QoS-LS algorithm and the system model and scheduling algorithm is described in detail in Section 4. Then a running time estimation of the query is described in Section 5. Next, in Section 6, we discuss the experimental setup and present our experimental results. Finally, we conclude the paper.

2 Related Work

Qu [5] proposed a balance scheduling algorithm QUTS in a web database and a two-level structure. But the algorithm is based on web database. And data organization and queries in web database are different from that of data warehouse. So it cannot be directly applied to RTDWH. However our algorithm draws on the two-level scheduling structure of the QUTS algorithm.

Thiele and et al [6] proposed a RTDWH workflow scheduling algorithm WINE. The algorithm is based on partitions of data warehouses and allows users to specify Quality of Service (QoS) and Quality of Data (QoD) of queries. And it adjusts the balance of scheduling execution according to user-specified parameters. But the QoS parameter proposed in the algorithm is an emphasis proportion of queries and updates and is the lack of more practical meaning. We in the literature [7] proposed a priority-based balance scheduling algorithm PBBS. The algorithm took into account the different priorities of the different tasks and ensured that the important tasks are executed first. But the algorithm does also not put forward some effective QoS parameters. The algorithm proposed in this paper draws on the idea of these algorithms and defines quality of service parameters associated with the practical implications, to guide the scheduling based on the parameters.

Bateni and et al [8] proposed an update scheduling algorithm in RTDWH. Through adjusting the execution order of updates, it can improve data freshness of RTDWH. But the algorithm does not consider the conflict and scheduling between updates and queries. Lu and et al [9] introduced a feedback control mechanisms in the real-time system. Jin and et al [10] proposed a fuzzy feedback control real-time scheduling (FFC-RTS) in the real-time system. These algorithms monitor the feedback information of system resources in the scheduling process, to adjust the scheduling. However, in RTDWH, tasks are divided into updates and queries, and the above algorithms do not distinguish task types. In this paper, we make reference to the real-time feedback control in the literatures. Thus we improve the efficiency of tasks and also make full use of system resources.

Qu and Thiele [4, 6] introduced several basic scheduling algorithms in RTDWH. The FAFE algorithm make a uniform consideration for updates and queries and the tasks are executed according to their arrival times. FAFE Query First (QF) algorithm divides the tasks into two queues, the update queue and query queue. And the query queue has a higher priority than the update queue. The tasks in the same queue are executed according to their arrival times. FAFE Update First (UF) algorithm is similar to the QF algorithm. But the update queue has a higher priority than the query queue. In our experiments, we implement the above basic scheduling algorithms and the algorithm proposed in this paper. And we make a performance comparison for all above algorithms.

3 System Architecture

Our real-time data warehouse system architecture is shown in Figure 1. It includes some source database systems (Source DB), some data load tools ETL, a real-time data warehouse (RTDW), some data warehousing applications, and so on. In the real-time data warehouse ETL, the less real-time data are imported using the traditional static batch import (Batch ETL). While the high real-time data are imported continuously, through the change data capture component (CDC) and the real-time ETL (Real-time ETL). Applications of the real-time data warehouse include some query processing, such as OLAP analysis, report application, data mining and so on.

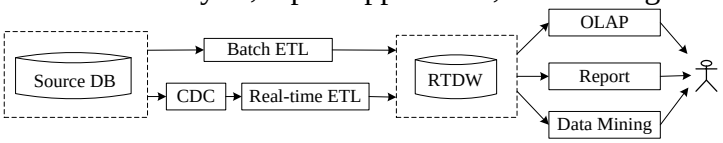

Fig. 1 System Architecture of Real-time Data Warehouse

In Figure 1, because the traditional ETL and queries in the data warehouse will not be executed at the same time, they will not be in conflict. In order to facilitate a discussion of updates and queries scheduling, in this paper we simplify the real-time data warehouse for following three components: (1) the continuous update streams from the data sources; (2) the real-time data warehouse, which receives the real-time updates streams and stores them permanently; (3) some applications on top of the data warehouse, including OLAP analysis and so on.

4 System Model

In this section, we discuss the system model of the scheduling algorithm, including the definition of query QoS parameters, the scheduling framework, the scheduling algorithm and so on.

4.1 Query QoS Parameters

In RTDWH, quality of service requirements for each query is different. Users maybe hope faster response time for some queries and higher data real-time performance for other queries. In order to satisfy the response time and data real-time requirements, the model should allow that users specify quality of service (QoS) parameters for each query. The QoS parameters defined in this paper include the expected response time and the acceptable real-time data delay.

Expected Response Time (*ert*): Suppose a query is issued in time t_0, users want the system to return a query result before a latest time t_1. Thus we define *ert* as the time t_1-t_0.

Acceptable Real-time Delay (*ard*): When a user sends a query q, the dataset that the system uses to answer the query q is dataset *DS*. Thus we define *ard* as the acceptable maximum delay time of the real-time data in dataset *DS*.

4.2 Scheduling Framework

The system framework of load scheduling algorithm in RTDWH is shown in Figure 2. The framework includes an update queue, a query queue, an update and query scheduler and an ODS with partition of the real-time data warehouse.

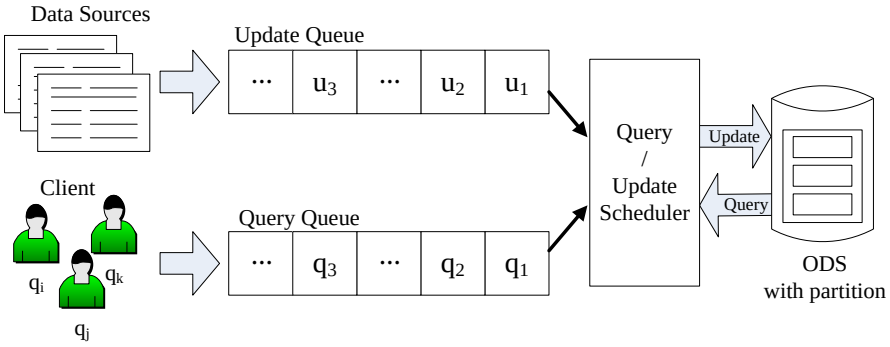

Fig. 2 Framework of load scheduling algorithm

Although updates and queries are both tasks in RTDWH, but there are still a lot of differences between them, so we put updates and queries into two different queues.

4.3 Scheduling Parameters of Queries

A query q is enriched by a two-tuple <*qos*, *qod*>. Where, *qos* represents the quality of service and query response speed requirements, *qod* represents the data quality and data freshness requirements. And *qos*, $qod \in [0, 1]$ and $qos + qod = 1$. However, different from the literature [5], the *qos* and *qod* of this paper do not need to be specified by users, but they are calculated dynamically based on the expected response time and the acceptable real-time data delay. And the calculation method is as follows.

Query Remaining Time (*qrt*): In order to complete a query task before its deadline, the query task has a latest start execution time. We define *qrt* as the difference of the latest start execution time and the current system time and the calculation method as shown in Formula 1, where t_a is the task arrival time, *ert* is the expected query response time, t_e is the execution time of the query, t_c is the current system time.

$$qrt(q) = t_a + ert - t_e - t_c \tag{1}$$

Partition Data Freshness F(p_i): If there are no update tasks that are not yet executed in a partition p_i, We define F(p_i) as the current system time; otherwise it is the timestamp of the latest data of the partition p_i.

Query Stale Distance (*qsd*): Suppose a query task is executed now, we define *qsd* as the differences of the current system time t_c, the data freshness of all data partition that are accessed by the query and the acceptable real-time data delay *ard* of the query. The calculation method is shown in Formula 2.

$$qsd(q) = t_c - \min_{p_i \in q}(F(p_i)) - ard \tag{2}$$

Then, we define *qos* and *qod* as a function of *qrt* and *qsd*, as shown in Formula 3, 4.

$$qos(q) = \tfrac{1}{2} - \tfrac{1}{\pi}\arctan(qrt(q) + qsd(q)) \tag{3}$$

$$qod(q) = \frac{1}{2} + \frac{1}{\pi}\arctan(qrt(q) + qsd(q)) \quad (4)$$

The meaning of *qos* and *qod* is as follows. When a query task is likely to exceed its deadline, its *qos* value is greater and it should be executed as soon as possible. When the data freshness of the query task does not meet user requirements, its *qod* value is greater and update tasks associated with the query should be executed as soon as possible. And $qos, qod \in [0, 1]$ and $qos + qod = 1$.

4.4 Scheduling Algorithm

The scheduling algorithm can be divided into two stages. The first stage decides whether the system enters a query mode or an update mode based on the overall requirements of the system. The second stage decides the execution strategy of query tasks or update tasks.

In the whole system, we can calculate a total *qos* and *qod* requirement of all queries, represented by $\sum qos$ and $\sum qod$. Then, according to the total requirement, the model decides the system is in the update mode or the query mode. If $\sum qos > \sum qod$, the system enters into the query mode; otherwise, the system enters into the update mode.

If it is the query mode, the model executes a query task from the query task queue. And all query tasks are sorted according to the *qos* value of the tasks.

If it is the update mode, the model executes an update task from the update task queue. And all update tasks are sorted according to the weight *w(u)*, the calculation method as shown in Formula 5.

$$w(u) = \sum_{\forall q, |P_q \cap Pu|=1} \frac{qod}{1 + pos_q} \quad (5)$$

Where, pos_q is the position of the query *q* in the query queue. The meaning of the weight *w(u)* is as follows. If the *qod* of all query tasks associated with the update is higher, or the position of the query tasks is closer to the queue head, the update should be executed as soon as possible.

The load scheduling algorithm is described as follows.

(1) A query task is inserted into the query queue according to the *qos* value and an update task is inserted into the update queue according to the *w(u)* value.

(2) Calculate <*qos*, *qod*> of all query tasks and the total $\sum qos$, $\sum qod$. If $\sum qos > \sum qod$, go to 3); otherwise, go to 4).

(3) Query mode: The task in the query queue head is executed and removed. Then Recalculate <*qos*, *qod*> of the remaining tasks and adjust the *w(u)* value of the update tasks. Then go to 2).

(4) Update mode: The task in the update queue head is executed and removed. Then go to 2).

5 Query Running Time Estimation

We need to calculate the estimated completion time of tasks to achieve that the closest deadline tasks are executed first. So we need to estimate the query running time. In this paper, we estimate the time according to the historical running time of queries.

Algorithm 1. EstimateQueryRunningTime

```
1: if the query Q is exist in the query history then
2:     get the top-n latest running times of the query in the history;
3:     compute the weighted average of the top-n latest running times as t_avq;
4: else if other queries on the same data table are exist in the query history then
5:     get the latest running times of all queries on the same data table;
6:     compute the weighted average of all above latest running times as t_avq;
7: else
8:     get the latest running times of all queries in the query history;
9:     compute the weighted average of all above latest running times as t_avq;
10: end if
11: return t_avq;
```

We estimate the running time as follow. For a given query, if the system has recorded a running time of the same query at past, we return the time as the estimated running time of the query. If the system does not record the running time of the same query at past, we look for other queries that are on the same data table and refer their running times to estimate the running time. And if there are no the above-mentioned historical running times, we refer the running times of all historical queries to

estimate the running time. The realization of the idea is shown in Algorithm 1.

In the second and third cases, the running time is estimated according to the history of queries on the same data table or all queries in RTDWH. So the estimated running time will not be very accurate. However, when the system is running for a long time, the model has basically collected the running information of all queries. And eventually it will be induced to the first case, namely, there is the history of the same query. Thus the estimated running time will be accurate.

The query histories are updated as follow. When a query is completed, the model records the running time. If it is not exist in the history, the running time is added directly; otherwise, the model keeps the top-n latest running times of the query and the earlier information will be removed.

6 Experiments

In this section, we conduct a comparative analysis between the QoS-LS algorithm and other algorithms mentioned, such as FAFE, UF and QF.

6.1 Experimental Setup

The experiments carried out on a 3.16 GHZ Pentium 4 PC with 4GB RAM under Windows XP and Oracle 9i DBMS. We implemented the RUQS-CD algorithm as well as other algorithms using J2EE. The TPC-DS toolkit is employed to generate test data, with a scale factor of 1, i.e., 19,557,335 tuples. We generate all queries using qgen2 of TPC-DS. The update dataset is generated using dbgen2 of TPC-DS.

6.2 Performance Comparison

In this section, we make a performance comparison among all algorithms, including data freshness, query response time and deadline missing ratio *DMR*.

(1) Deadline Missing Ratio (DMR)

It is defined as the number of the unfinished tasks until the query deadline divides the total number of query tasks.

$$DMR = \frac{|Q_{dm}|}{|Q_e|} \tag{6}$$

(2) Query Response Time (*QRS*)

It is defined as the sum of the waiting times of all executed query tasks divides the total number of queries that have been executed.

(3) Query Data Freshness

Data freshness is calculated as shown in Formula 7, which $N_u(p_i)$ indicates the number of updates that have not been executed in the partition p_i. Because a query may access multiple partitions of RTDWH, the total data freshness of the query is taken as the minimum value of data freshness in each partition. Then the data freshness of the whole system is equal to the average of the data freshness of all queries.

$$Q_{fresh}(q) = \min_{p_i \in P_q}\left(\frac{1}{1+N_u(p_i)}\right) \qquad Q_{fresh}(W) = \frac{1}{|W_q|}\sum_{q_i \in W_q} Q_{fresh}(q_i) \tag{7}$$

Table 1 Algorithmic performance comparison

	Data freshness	Query response time (s)	DMR
QoS-LS	0.96	2.35	5%
FAFE	0.58	2.23	46%
QF	0.06	0.25	98%
UF	1	4.43	0

The experiment shows that the comprehensive performance of the QoS-LS algorithm is the best. However, some aspects are not as good as other algorithms. And the experimental results are analyzed as follows:

Compared with the FAFE algorithm, the query response speed of FAFE and QoS-LS is almost the same. However, since the FAFE algorithm does not have a good control over the update mechanism, the performance of the data freshness and the deadline miss rate is very poor.

The query response of QF is the fastest. But it is too much to pursue the response speed of the

query without updating the data in time, resulting in data freshness and DMR is far from the user requirements. Almost all of the data are not fresh, and all update tasks are not completed before the deadline. For the UF algorithm, it is too much to pursue the data freshness, so DMR is not as good as the UF algorithm. But the query response time of UF is too large, is almost 2 times of QoS-LS. So the comprehensive performance is not as good as the QoS-LS algorithm proposed in this paper.

To sum up, the QoS-LS algorithm has a good resource allocation in queries and updates load scheduling of RTDWH, which can meet the requirements of the user's query response speed and data freshness.

7 Conclusions

In this paper, we propose a QoS-based updates and queries scheduling algorithm in RTDWH and describe the algorithm framework and idea in detail. First, we propose some QoS parameters related to the query, and then we make a real-time scheduling for updates and queries according to the QoS requirements. Finally, the experimental results show that the algorithm can reasonably adjust the execution order of tasks, effectively use system resources and provide the faster response and the higher real-time data according to the QoS requirements. However, the estimation model of query running time is not very perfect. In future works, we will further improve the estimation model.

Acknowledgement

This work is supported by National Natural Science Foundation of China (No. 61602323).

References

[1] C. White. Intelligent business strategies: Real-time data warehousing heats up [C]. DM Review, 2002.

[2] Michael Haisten. Real time data warehouse: The next stage in data warehouse evolution [C]. DM Review, 2003.

[3] S. Brobst, J. Rarey. The five stages of an active data warehouse evolution [J]. Teradata Magazine, 2001, 3(1):38-44.

[4] H. Qu, A. Labrinidis. Preference-aware query and update scheduling in web-databases [C]. ICDE 2007: 356-365.

[5] H. Qu, A. Labrinidis, D. Mosse. UNIT: User-centric transaction management in web-database systems [C]. ICDE 2006:1-10.

[6] M. Thiele, U. Fischer, W. Lehner. Partition-based workload scheduling in living data warehouse environments [C]. DOLAP 2007: 57-64.

[7] J. G. Shi, Y. B. Bao, F. L. Leng, G. Yu. Priority-based balance scheduling in real-time data warehouse [C]. HIS, 2009:301-306.

[8] M. Bateni, L. Golab, M. Hajiaghayi, H. Karloff. Scheduling to minimize staleness and stretch in real-time data warehouses [C]. SPAA, 2009: 29-38.

[9] C. Lu, J.A. Stankvoic, G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms [J]. Journal of Real-Time Systems, 2002, 23(1-2):85~126.

[10] H. Jin, H.A. Wang, Y. Fu. A fuzzy feedback control real-time scheduling algorithm [J]. Journal of Software, 2004, 15(06): 791-798.