Management of QoS and Data Freshness in Real-Time Data Warehouses using Feedback Control Scheduling

Issam Hamdi\(^1\), Emna Bouazizi\(^2\), Jamel Feki\(^1\)

\(^1\) MIRACL Laboratory, University of Sfax, Route d’aéroport, Km 3.5, BP. 1088, Sfax 3018, Tunisia
\(^2\) University of Monastir, Route Sallem Bechir, BP. 56, Monastir 5000, Tunisia

haamdiissam@gmail.com, emna.bouazizi@gmail.com, Jamel.Feki@fsegs.rnu.tn

Abstract: Today many organizations use data warehouse for strategic decision making. Today’s real-time business stresses the potential to increasingly process volumes of data at a very high speed for competitiveness reasons. But the data warehouse must often deal with transient usage charges, due to the unpredictability of access to data. The purpose of this work is to maintain the behaviour of the Real-Time Data Warehouse (RTDWH) at a stable condition and reduce the number of transactions responsible for not meeting the deadline. In this paper, we propose an architecture called FCSA-RTDWH (Feedback Control Scheduling Architecture for Real-Time Data Warehouse). The main objectives of this approach are the following: guarantee the data freshness, enhance the deadline miss ratio even in the presence of conflicts and unpredictable workloads and, finally satisfy the requirements of users.

Keywords: —Real-Time Data Warehouse, Feedback Control Scheduling, Quality of service, Data freshness.

1. Introduction

A Data Warehouse Management System (DWMS) extracts data from several distributed data sources, incorporates this data into derived views in the data warehouse and maintains the views under source changes. RTDWH \([8]\) is required because of the lack of real-time update in traditional data warehouse. Current works on RTDWH have focused on speeding up the Extract-Transform-Load (ETL) process \([12,13]\). There were also works on supporting various warehouse maintenance policies, such as immediate (update views whenever the data sources change), deferred (update views only when queried), and periodic \([11]\). However, a common way to deal with a transient overload in a real-time system with the consideration of the real requirements of users is to temporarily discard jobs. This is exactly the problem we are dealing with in this paper.

The main objectives of the proposed approach are the following: guarantee the data freshness and enhance the deadline miss ratio even in the presence of conflicts and unpredictable workloads. This paper is organized as follows: In Section 2 we discuss the related works. Then, in Section 3, we present the system model that we consider. FCSA-RTDWH\(^1\) is introduced in detail in Section 4 and we discuss the benefits of our architecture. We conclude this paper by a discussion of this work and then we present our future works.

2. Related Works

Researching on RTDWH is a hot topic in data warehouse area. Nowadays, there are many new concepts, such as Zero Latency Data Warehousing (ZLDWH) \([4]\), Active Data Warehousing (ADWH) \([7,10,16]\) that are both RTDWH. But few works focus on the quality of service management in RTDWHs which allows users to express their real needs.

Thiele and Al. \([5]\) proposed a workload scheduling WINE in RTDWH. 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. But it does not monitor the system resources and the running status of the update and query queues.

In \([15]\), priority-based balance scheduling algorithm (PBBS) gives different priorities to all tasks according to the response time requirements of queries and the different import levels of the data being updated. However, it sets priorities separately.

Leng and Ai \([1]\) proposed a requirement-based querying and an updating scheduling algorithm (RQUS) which allows users to express their real needs by specifying the acceptable response time delay and the acceptable result staleness when queries are submitted. RQUS has a main lack: it does not deal with the transient overload.

\(^1\) Feedback Control Scheduling Architecture for Real-Time Data Warehouse
3. System Overview

In this paper, we consider an RTDWH that receives write-only update transactions and read-only query transactions and maintains two types of tables: base tables that are sourced directly, and derived tables (views). Transactions are classified into two classes: update transactions and user transactions as explained below:

- **User transactions**, representing user requests, arrive randomly (not periodically) and can only read data. In order to allow users to express their real needs, each query task is provided with two parameters [1], acceptable response time delay $\Delta t_i$ and acceptable result staleness $\Delta s$, when it is submitted by users so that it can satisfy the requirements of users well. We use the absolute deadline (AD) to determine the priorities of tasks. This scheduling algorithm can effectively solve the scheduling problem between short and long queries with deadline.

$$AD(q_i) = t_a(q_i) + \Delta t_i(q_i) + t_e(q_i)$$

Where
- $t_a(q_i)$: the arrival time of the transaction $q_i$
- $t_e(q_i)$: the execution time of $q_i$
- $\Delta t_i(q_i)$: the acceptable response time delay

With AD, the task misses its deadline if the difference between the time that the task is processed completely and its AD is superior to zero.

- **Update transactions** update the values of real-time data (sensor data) in order to reflect the real world status. They are executed periodically and have only to write new sensor data. Each task loads new data into a table. So tables are updated repeatedly over time. If the period is unknown or unpredictable, we let the user choose a period when the warehouse should check for new data. Data update is no longer implemented in the batched and periodic way during the idle time of data warehouses but it is continuously on going. For each task $T_i$, the deadline of $T_i$ is estimated to be $(r_i + P_i)$ where $r_i$ is its release time and $P_i$ is its period [3].

The priority of an update task is related to two parameters [1]: The acceptable result staleness and the locations of the queries that are affected by the update task. The priorities of update tasks are calculated according to the partition unit. Update tasks on a partition have the same priority and are scheduled in the increasing order of the arrival timestamps. Therefore, update tasks are scheduled in a decreasing order of priorities and in an increasing order of arrival timestamps. The priority of a partition $p$ is calculated as follows [1]:

$$P(p) = \sum_{q \in Q_p} \frac{M_i}{loc_q} + 1$$

Where, $Q_p$ is the set of query tasks accessing the partition $p$, $loc_q$ is the location of a query task $q_i$ which belongs to $Q_p$ and $M_i$ is calculated as follows:

$$M_i = \begin{cases} S(p_j) - \Delta s(q_i) & \text{if } S(p_j) > \Delta s(q_i) \\ 0 & \text{otherwise} \end{cases}$$

Where $S(p_j)$ is the staleness of the partition $p_j$ (We will discuss the staleness of the partition in more details in Section 3.1).

3.1. Staleness result

The staleness is computed only from those unapplied updates that are already in the system at the time the user runs the query; this is because the user does not expect to see “future results” [5]. In RTDWH, the data staleness is calculated according to the partition unit. The staleness of a partition $p_i$ is defined as the difference between the maximum arrival timestamp of each unapplied update on $p_i$ and the freshness of $p_i$[1]:

$$S(p_i) = \max(t_a(u_j),p_i) - F(p_i)$$

Where $F(p_i)$ is the freshness of the partition $p_i$ as the maximum timestamp of all records in $p_i$ and $\max(t_a(u_j),p_i)$ is the maximum arrival timestamp of each unapplied update on $p_i$. Because a query is only concerned about the update tasks that arrive before it, the staleness of a partition $p_i$ related to a query $q_i$ is changed as follows:

$$S(p_i,q) = \max_{t_a(u_j) \leq t_a(p_i)}(t_a(u_j),p_i) - F(p_i)$$

For a query task $q$, the staleness $S(q)$ is the maximum staleness of all partitions $p_i$ that affect query results:

$$S(q) = \max_{p_i \in Q_p} (S(p_i,q))$$

3.2 The quality of service in RTDWH

The QoS can be seen as a metric that permits to measure the overall system performance. Indeed QoS is a collective measure of the service level provided to the customer. It is characterized by different performance criteria that include basic availability, error rate, response time and the rate of successful transactions before deadlines.

In [1], the QoS introduces two concepts: Quality of Transactions (QoT) and Quality of Data (QoD). They are defined as follows:
QoT: denotes the ratio of the number of query tasks that miss their deadlines to the number of query tasks that have already been executed.

QoD: is the ratio of the number of queries for which the result staleness is unacceptable to the number of queries that have already been executed.

4. A Feedback Control Scheduling Architecture for Real-Time Data Warehouse: FCSA-RTDWH

In this section, we will present the design of FCSA-RTDWH that provides data services with QoS guarantees for RTDWH.

To guarantee the desired deadline miss ratio even in the presence of unpredictable workloads, FCSA-RTDWH exploits a feedback control loop that we will explain in Section 4.2. As shown in Fig. 1, the architecture consists of an admission control, a precision control, a handler, a view manager and a monitor. Admitted transactions are placed in the ready queue. The transaction handler manages the execution of the transactions. The system performance statistics are collected periodically by the monitor. Below we briefly describe each component.

- Admission control: Used to avoid the overloading of the system. It is based on an estimated CPU utilization and the target utilization set point. At each sampling period, the target utilization parameter is set by the local controller. The estimated execution time of an admitted transaction is credited to the estimated CPU utilization. Transactions are rejected if the estimated CPU utilization is higher than the target utilization set by the controller. It is based on the calculated operation load and quality service parameters specified by the data warehouse administrator.

- Precision control: A data version is related to a timestamp indicating the latest observation of this data item in the real world. We define a Data Error (DE) that represents the derivation between the last value and the updated value. The upper bound is given by the maximum data error (denoted MDE). The controller can separate the update transactions when the data update is sufficiently representative of the real world considering the value of MDE: it determines whether the transaction wishes to update a real-time data can be discarded (DE<MDE) or not (DE>MDE). Further, the precision controller decreases the response time of user transactions with access to fresh data by discarding any unnecessary freshness.

- Execution mode (EM): Determines the execution mode of the present system according to the requirements of users. Specifically, the system is in the state of the update or query. Leng et al [1] introduced two parameters: (i) the response delay ratio (Rrl) that denotes the sum of ratios between real response time delay trl and acceptable response time delay ∆trl, and (ii) the result staleness ratio (Rs) that is the sum of ratios between the real result staleness s and the acceptable result staleness ∆s. If Rs ≤ Rrl the system mode is query, the query scheduler will run. It schedules the head query task to execute because the query queue is sorted when a new query task arrives. Otherwise (Rs > Rrl), the system mode is updated, the update scheduler will run. Moreover, Rs and Rrl are recalculated whenever an update or query task is processed completely.

- Monitor: It measures the system performance by inspecting an execution of transactions (number of transactions completed, abandoned, who missed maturity...). The measured values are part of the feedback control loop that helps to stabilize the system.
• Handler: The incoming transactions are dispatched and processed by the transaction handler. It consists of a concurrency controller (CC), a freshness manager (FM), and a scheduler (SC).
  o Scheduler: It allows scheduling user transactions and update transactions according to their priority. Within each queue, transactions are scheduled using scheduling algorithms.
  o Concurrency controller: Data warehouse consistency can be maintained using concurrency control protocols. In [6], two-versions no locking (2VNL) algorithm is proposed for enhancing data availability by allowing user transactions and a maintenance transaction to be executed concurrently. 2VNL achieves the concurrency by offering two alternative data versions to user transactions. In this sense, authors in [6] proposed not only 2VNL but also an nVNL algorithm which maintains more than two database versions. However, nVNL algorithm suffers from a limitation: it is not easy to estimate the sufficient number of required versions. Moreover, the cost of versions management increases as the number of versions increases. For this reason, we propose to use the SCC (Speculative Concurrency Control) [19] that is one of the first concurrency control protocols of real-time transactions. It uses the duplication of read transaction when a conflict is detected to resolve conflicts as read-write and write-read conflicts.
  o Freshness manager: It checks the freshness of the data that will be accessed by the transaction.
• Miss ratio and utilization controllers: This component allows adjusting the QoS parameters based on the values determined by the monitor and the reference parameters determined by the data warehouse administrator.
• View manager: It selects a materialized view and manages data warehouse maintenance under both concurrent data updates and schema changes using a maintenance process.
  o View selection (VS): A dynamic view selection algorithm has appeared in the literature like DynaMat [17]. These approaches focus on the performance of refresh materialized views. But they have not completely addressed other performance issues for query requests such as query response time. In this context, we present DynaSeV (Dynamic Selection of materialized Views), an algorithm that dynamically materializes views (cf. Section 4.1).
  o Maintenance process (MP): Such algorithms for maintaining a data warehouse under data source updates are called view maintenance algorithms. There were also some works on maintaining materialized view definitions under schema changes, and on adapting its view extent under schema changes. TxnWrap [18] is the first work to put the existing view maintenance algorithms into the context of a serializability theory, to remove concurrency concerns from them. In the context of maintaining a data warehouse in parallel, PVM [14] addresses the problem of concurrent data update detection in a parallel execution mode and the variant data warehouse commit an order problem.

4.1. DynaSeV (Dynamic Selection of materialized Views)

In order to allow users to control the selection process of materialized views, the administrator defines the followings parameters:

- \( S_{\text{max}} \): A maximum size to optimize the storage space \( S \) of materialized views.
- Two parameters which are taken as reference for selection of views to be materialized: a threshold of runtime execution \( E_{\text{min}} \) and a minimum threshold for the priority of transaction \( p_{\text{min}} \).
- \( p_{vi} \): A user-defined priority for each table \( V_i \); higher values of \( p_{vi} \) indicate higher priorities [2].

Algorithm 1 DynaSeV (Dynamic Selection of materialized Views)

T: user transaction that accesses n tables \( V_1..V_n \) (read-only access).

\( E_T \): the execution time of \( T \).

\( H \leftarrow \{ V_i : \text{where } p_{vi} > p_{\text{min}} \text{ and } i \in [1;;n] \} \): the set of tables accessed by the transaction \( T \) where \( p_{vi} > p_{\text{min}} \). Let \( L \) be the length of this set \( H \).

A priority \( p_T \) is associated for each transaction \( T \) according to accessed tables to indicate the importance level.

\[ p_T \leftarrow \frac{L}{n} \]

When the transaction \( T \) is finished:
Begin
if \( p_T > p_{\min} \) and \( E_T > E_{\min} \) then
  if \( S < S_{\max} \)
  Store this transaction result (a materialized view)
  else
  Clean materialized views: we remove the unless used views in a period of time.
  Store this transaction result (a materialized view)
endif
End

DynaSev selects views from results of incoming queries. It uses the notion of transaction importance \( p_T \) to select the most important views which are very costly (see Fig.2). A transaction is considered costly if its execution time exceeds the threshold \( E_{\min} \) set by the administrator whereas it is considered important if it has an access to the tables that have priorities greater than \( p_{\min} \). Therefore, the size of the materialized views would grow monotonically overtime. To optimize the storage space \( S \) of materialized views, the administrator fixes a maximum size \( S_{\max} \). When the size of the materialized views exceeds \( S_{\max} \), DynaSev removes the least used views over a period of time.

4.2. Feedback control loop

A utilization controller may saturate at 100\%. In contrast, a miss ratio controller can saturate when the real-time system is underutilized (0 miss ratio as a result). Each control loop generates a control signal, called requested CPU utilization adjustment \( \Delta U \), to achieve the target miss ratio (or target utilization) (see Fig.3). Further, the feedback control scheduling and admission control can prevent a severe overload by dynamically adjusting the CPU utilization based on the current system behaviour and avoiding excessive admissions of user transactions to the system. Using the feedback loop allows to stabilize the system during the instability periods [9]. It is based on observation and auto-adaptation principles. The observation principle consists of observing the results obtained by the system and checking if the current QoS observed is consistent with the QoS initially required. The auto-adaptation consists of the system to adapt the results according to the QoS needed by the clients. In this way, the feedback loop ensures the stability of the system.

![Figure 1. DynaSev’s architecture.](image)

![Figure 2. Miss Ratio/Utilization Controllers.](image)

4.3. Benefits of FCSA-RTDWH

FCSA-RTDWH is a good solution for QoS guarantees. It allows to limit the deadline miss ratio and to support freshness for the data accessed by timely transactions (even in the presence of unpredictable workloads). So the data used by transactions are fresh. FCSA-RTDWH permits also to guarantee the quality of data (precision and freshness) and the quality of transaction which are enhanced by alleviating the risk of transaction miss deadline, and therefore it enhances the QoS. FCSA-RTDWH can perform very well in this context of changing user requirements due to the adjustability.

5. Conclusion and Future Work

In this paper, we discussed the problem of load changes in data warehouses. We proposed a new approach called FCSA-RTDWH to maintain the behaviour of the RTDWH in a stable condition and satisfy the requirements of users. We used a precision
control and admission control to control the incoming transactions according to CPU utilization and the concept of the maximum data error. We also introduced the notion of transaction importance $p_t$ to select the most important views which are very costly.

The advantage of our approach is to allow users to express their real needs for their queries and to deal with the transient overload to better manage the RTDWH performance. A simulator is currently being implemented to describe the impact of our scheduling approach.

This article gives some avenues of research based on this approach. We plan to extend this work in several different ways. We will consider other aspects to study different components of the feedback control scheduling architecture for the QoS management in RTDWH.

References