Similarity based mixed transaction concurrency control protocol

PAN Yi a, LU Yan-sheng
Department of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, P.R. China
Received 4 April 2005; revised 27 April 2005

Abstract: Due to the various performance requirements and data access restrictions of different types of real-time transactions, concurrency control protocols which had been designed for the systems with single type of transactions are not sufficient for mixed real-time database systems (MRTDBS), where different types of real-time transactions coexist in the systems concurrently. In this paper, a new concurrency control protocol MRTT_CC for mixed real-time transactions is proposed. The new strategy integrates with different concurrency control protocols to meet the deadline requirements of different types of real-time transactions. The data similarity concept is also explored in the new protocol to reduce the blocking time of soft real-time transactions, which increases their chances to meet the deadlines. Simulation experiments show that the new protocol has gained good performance.

Keywords: real-time database; semantic concurrency control; temporary consistency; data similarity

CLC number: TP 311.13; Document code: A

1 Introduction

Real-Time database applications, such as stock trading systems, telecommunication managements, and air traffic management are characterized by large volume of data as well as time constrained data access [1]. The correctness of such applications depends on both execution results and time constraints imposed on transaction commitments, generally in terms of deadline. Based on the different consequences of deadline missing, transactions in a real-time database systems (RTDBS) can be classified into three categories: hard, soft and firm. Hard real-time transactions (HRTs) are those that have serious consequences if they miss their deadlines. Firm real-time transactions (FRTs) do not entail a catastrophe and impart no value to the system if their deadlines are missed, while soft real-time transactions (SRTs) have diminishing value even after their deadlines.

Concurrency control is an important mechanism for synchronizing database accesses to maintain its consistency. An overview of various concurrency control, scheduling and conflict resolution strategies is provided in Ref. [1]. Two possible conflict resolution in RTDBS are discussed: the pessimistic method and the optimistic method. Pessimistic algorithms synchronize the concurrent execution of transactions early in their executions and optimistic algorithms delay the synchronization of transactions until their termination. Representative pessimistic algorithms include HP_2PL, 2PL high priority inheritance and Priority Ceiling Protocol (PCP), and classic optimistic methods include OCC_BC, OCC_Sacrifice and OCC_W50 [2-6]. There is an increasing demand in processing mixed real-time transactions concurrently. For instance, in the cooperative distributed navigation systems, a wide variety of transactions occur. Transactions updating sensor data are periodic hard real-time transactions, while display transactions are done by soft real-time transactions. We call this kind of real-time database system as Mixed Real-time Database Systems (MRTDBS), where different types of real-time transactions co-exist.

Due to the unique characteristics and performance requirements of different types of transactions, the concurrency control protocols designed for a single type of real-time transactions may not fit very well for the MRTDBS [7,8]. Thomas presented a two-level concurrency control framework for mixed real-time transactions concurrency control [9], where a master concurrency controller is proposed to detect the possible inter-class data conflicts, which are resolved by performing a serialization check on the time-stamps of transactions. The data conflicts among transactions in the same class are detected and resolved by
individual schedulers. A reduced ceiling protocol for MRTDBS is presented in Ref. [10]. The schedulability of hard real-time transactions can be improved by bounding the blocking time from soft real-time transactions; and the methodologies, such as the Thomas Write Rule, are employed to reduce the number of aborts for soft real-time transactions due to data conflicts with hard real-time transactions. In Ref. [7], strategies for resolving inter-class data conflicts in MRTDBS are investigated, where data conflicts are classified into inter-class conflicts according to the conflicts between hard and soft real-time transactions. A protocol with dynamic adjustment of serialization for RTDBS with mixed transactions has been presented in Ref. [11] to avoid unnecessary transaction restarts.

It should be notified that the usefulness of the transactions will be greatly affected if their deadlines are missed; therefore the primary performance criterion of the RTDBS is timeliness, not average response or throughput. Compared with an inconsistent database states, meeting transaction deadline is considered to be more critical [13]. Most of the above strategies use serializability correctness criterion for concurrency control in accessing real-time data, which is too restrict for real-time database systems. Our new protocol improves the system performance by formalizing the data similarity notion and adopting the weaker correctness criteria [14], where transactions are allowed to access the similar versions of the same data item concurrently thus to reduce the blocking time of soft real-time transactions and increase the probabilities to meet their deadlines. To the best of our knowledge, this work is the first paper that systematically investigates this issue in MRTDBS.

The rest of the paper is organized as follows. Real-time data characteristics and data similarity notion are discussed in Section 3. Section 4 describes the details of the new MRTT_CC protocol. The simulation model and the experiment results are presented in Section 5. Section 6 is the conclusion.

2 Real-Time Data Characteristics and Data Similarity

Data in a RTDBS can be classified into two categories: normal data and temporary data. The value of the normal data does not change with the passage of time, while the value of the temporary data will become invalid with the passage of time and should be updated timely. Otherwise, the decision based on a data item in the database may be wrong, and potentially disastrous. A temporary data item has multiple versions.

**Definition 1: Absolute Validity Interval (AVI).** The absolute validity interval (AVI) of a temporary data is defined as the maximum age during which the data item is considered stale.

**Definition 2: Temporary Data Item X.** As mentioned before, a temporary data item X has multiple versions $X_1, X_2, ..., X_i$, so a temporary data item $X$ can be defined as a four-items tuple: $X(X_i, V_a(X_i), V_v(X_i), P(X_i))$, where $X_i$ is the value of the $i$-th version of data object $X$; $V_a(X_i)$ is the beginning of the validity interval of $X_i$; $V_v(X_i)$ the end of the validity of $X_i$; and $P(X_i)$, the absolute validity interval of $X_i$ or the period of $X_i$ given by $P(X_i) = [V_a(X_i), V_v(X_i)]$ where $V_a(X_i) < V_v(X_i)$.

**Definition 3: Data logical consistency.** A data item $X$ is logical consistent if and only if it satisfies all the predefined integrity and consistency restriction.

**Definition 4. Data Absolute Consistency.** The $i$-th version of data item $X$ is temporally consistent at time $t$ if and only if

$V_a(X_i) \leq t < V_v(X_i)$

A temporary data item is absolutely consistent if it timely reflects the state of an external object that the data item models, which means each of its version is absolute consistent.

**Definition 5: Data Similarity.** It is often the case that the values of temporary data items in a real-time system are unable to be updated continuously as the update process itself introduces a delay. So, there already exists a discrepancy in the values of the objects in the external environment. By assuming that a small discrepancy is tolerable for most applications and data values that are slightly different in age or in precision are often considered to be the same, the concept of similarity is defined in. Similarity is a binary relation on the domain of a data item, which is reflexive and symmetric, but not necessarily transitive [13]. The concept of similarity can be used to extend the usual transaction concurrency control correctness. In this paper, the $i$-th version and $i+1$-th version of data item $X$ are similar if and only if

$P(X_i) = [V_a(X_i), V_v(X_{i+1})]$.

It is defined that $X_{i+1} \sim X_i$ denotes that the $k$-th
version and $i$-th version of data item $X$ are similar.

3 **Mixed Real-Time Transaction Concurrency Control Protocol (MRTT_CC)**

To simplify the problem, we will mainly focus on hard and soft real-time transactions, and the firm real-time transactions will be integrated in our future research. In MRTDBS, data conflicts are divided into two types: inter-class conflicts and intra-class conflicts, conflicts among different types of transactions belong to inter-class conflicts; conflicts amongst the same kind of transactions belong to intra-class conflicts. The protocol presented in Ref.[10] uses the usual serializability correctness criteria (namely, final-state, view, and conflict serializability) to resolve the conflict between hard real-time transactions and soft real-time transactions. We have noticed that restarting “less important” transactions is a principle adopted by most of the real-time concurrency control protocols to resolve data conflicts and when this rule is applied to an MRTDBS, it is likely that soft real-time transactions are more often restarted by hard real-time transactions. A right concurrency control protocol for the MRTDBS requires to find a reasonable balance between different kinds of real-time transactions. In this paper, by exploring similarity correctness notion for concurrency control, the correctness criteria are extended to view serializability [12]. The blocking time or restart probability of soft real-time transactions can be reduced under the new weakened correctness criteria, which increase their chances to meet their deadline without effecting the concurrent executions of the hard real-time transactions.

The principle used in MRTT_CC protocol in resolving data conflicts can be summarized in Table 1. It includes three parts: the PCP strategies for conflicts resolving among the hard real-time transactions, the OCC_BC approach for conflicts resolving among the soft real-time transactions and the inter-class conflict resolution strategy MT_CC. The reason of choosing PCP an OCC_BC is because each of them has shown better performance in processing hard real-time transaction and soft real-time transaction concurrency control [15]. When inter-class data conflicts are detected, instead of simply aborting or blocking the soft real-time transactions, the controller will judge whether the conflicts can be eliminated by employing similarity based correctness criteria.

<table>
<thead>
<tr>
<th>Applier/Validating Transaction (SRT only)</th>
<th>HRT Read lock</th>
<th>HRT Write Lock</th>
<th>SRT Read Lock</th>
<th>SRT Write Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holder</td>
<td>√</td>
<td>PCP</td>
<td>√</td>
<td>MT_CC</td>
</tr>
<tr>
<td>HRT Write</td>
<td>PCP</td>
<td>PCP</td>
<td>MT_CC</td>
<td>MT_CC</td>
</tr>
<tr>
<td>SRT Read</td>
<td>√</td>
<td>MT_CC</td>
<td>√</td>
<td>OCC_BC</td>
</tr>
<tr>
<td>SRT Write</td>
<td>MT_CC</td>
<td>MT_CC</td>
<td>OCC_BC</td>
<td>OCC_BC</td>
</tr>
</tbody>
</table>

" √": The lock applier is allowed to set the lock.

In the following, the concurrency control strategies for resolving inter-class conflicts are discussed, assuming that transaction $T_H$ is a hard real-time transaction, transaction $T_S$ is a soft real-time transaction, $T_H/R$ denotes that $T_H$ owns a read lock, $T_H/W$ denotes that $T_H$ owns a write lock, $T_S/R$ denotes that $T_S$ owns a read lock, $T_S/W$ denotes that $T_S$ owns a write lock, $X_f$ denotes the former version of data item $X$, and $X_c$ denotes the current update version of $X$. When a data conflict between $T_H$ and $T_S$ occurs, the concurrency controller will adopt the following options:

```
SWITCH (Transaction that already owns the lock) {
  CASE $T_H/R$: {
    IF $T_S$ applies write lock on data item $X$, THEN
  } 
  CASE $T_H/W$: {
    IF $T_S$ applies read lock on data item $X$, THEN
  } 
T_S$ obtains the read lock
ELSE BLOCK $T_S$
END IF
CASE $T_H/W$: {
  IF $T_S$ applies read lock on data item $X$, THEN
  IF $X_f \not\approx X_c$, THEN
    $T_S$ obtains the write lock
  ELSE $T_S$ obtains the read lock
  END IF
ELSEIF $T_S$ applies write lock on $X$, THEN
  $T_S$ obtains the write lock/* write/write conflicts will be allowed if the hard real-time
```

<table>
<thead>
<tr>
<th>Table 1 MRTT_CC protocol</th>
<th>Applier/Validating Transaction (SRT only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRT Read lock</td>
<td>PCP</td>
</tr>
<tr>
<td>HRT Write lock</td>
<td>MT_CC</td>
</tr>
<tr>
<td>SRT Read lock</td>
<td>OCC_BC</td>
</tr>
<tr>
<td>SRT Write lock</td>
<td>OCC_BC</td>
</tr>
<tr>
<td>Holder</td>
<td>MT_CC</td>
</tr>
</tbody>
</table>

In the following, the concurrency control strategies for resolving inter-class conflicts are discussed, assuming that transaction $T_H$ is a hard real-time transaction, transaction $T_S$ is a soft real-time transaction, $T_H/R$ denotes that $T_H$ owns a read lock, $T_H/W$ denotes that $T_H$ owns a write lock, $T_S/R$ denotes that $T_S$ owns a read lock, $T_S/W$ denotes that $T_S$ owns a write lock, $X_f$ denotes the former version of data item $X$, and $X_c$ denotes the current update version of $X$. When a data conflict between $T_H$ and $T_S$ occurs, the concurrency controller will adopt the following options:

```
SWITCH (Transaction that already owns the lock) {
  CASE $T_H/R$: {
    IF $T_S$ applies write lock on data item $X$, THEN
  } 
  CASE $T_H/W$: {
    IF $T_S$ applies read lock on data item $X$, THEN
  } 
T_S$ obtains the read lock
ELSE BLOCK $T_S$
END IF
CASE $T_H/W$: {
  IF $T_S$ applies read lock on data item $X$, THEN
  IF $X_f \not\approx X_c$, THEN
    $T_S$ obtains the write lock
  ELSE $T_S$ obtains the read lock
  END IF
ELSEIF $T_S$ applies write lock on $X$, THEN
  $T_S$ obtains the write lock/* write/write conflicts will be allowed if the hard real-time
```

```
transactions can commit first*/
END IF }
CASE T_S/R:
IF T_H applies write lock on \(X\) THEN
IF \(X \sim X_e\) THEN
T_H obtains the write lock
ELSE { ABORT T_S
T_H obtains the write lock }
END IF
END IF }
CASE T_S/W:
IF T_H applies read lock on \(X\) THEN
IF \(\Delta \sim \Delta\) THEN
T_H obtains the write lock
ELSE { ABORT T_S
T_H obtains read lock }
END IF
ELSE T_H obtains read lock /* write/write conflicts will be allowed if the hard real-time transactions can commit first*/
END IF }

4 Performance Study

4.1 Mixed Real-time Database Model

As shown in Fig. 1, the MRTDBS model consists of seven components, namely, transaction generators TG, a transaction manager TM, a transaction scheduler TS, a concurrency controller CC (ready queue RQ, block queue BQ), a database manager DM, a resource manager RM and a database DB.

![Fig. 1 the MRTDBS model](image)

There are two types of transaction generators. One is responsible for the generation of the soft real-time transactions, which follows the Poisson distribution. The other is responsible for the generation of the hard real-time transactions, which follows their own periods. Each of the new arriving transaction will be assigned a priority, the priority of the hard real-time transactions is defined according to the rate monotonic algorithm, the priority of the soft real-time transactions is defined according to the EDF strategy [4]. Hard real-time transactions will be assigned higher priorities because they are more critical.

4.2 Simulation Experiments

The goal of the simulation experiments is to compare the performance of the three concurrency control protocols for MRTDBS: MRTT_CC, RCP and extend mixed OCC_BC (MOCC_BC). The general principle of OCC_BC protocol has been extended in our model as in Ref.[10]. In the extend protocol, the execution of the hard real-time transactions will also go through three phases as the same as the soft real-time transactions: execution phrase, validation phase and write phase. When a validating soft real-time transaction finds that there is a conflict between itself and a hard transaction, the soft real-time transactions will be restarted. When a validating hard real-time transaction finds any conflict between itself and other transactions, the other transactions will be restarted.

There are five hard real-time transaction generators with different generation period in our database model. The deadline of a hard real-time transaction is defined to be the generating time of the next transaction from the same generator. The deadline of the soft real-time transactions is defined as follows:

Deadline = \(T_{\text{exp}}(1 + \text{Slack factor})\)

where \(T_{\text{exp}}\) is the expected execution time of the soft real-time transaction, slack factor is a random variable uniformly distributed between the slack bound. In RTDBS, the major performance measure is the miss rate (MR), which is defined as the number of deadline missing transactions over the total transaction number. In our model two miss rates are defined: \(MR_H\) and \(MR_S\). The former indicates the probability of hard real-time transaction missing deadlines and the latter identifies the probability of soft real-time transaction missing deadlines. We assume that the experiments model is conducted over a main memory database system, so we do not consider the disk processing. The parameters and their baseline settings are summarized in Table 2.

The selected results obtained from our simulation experiments are shown in Fig. 2 and Fig. 3 respectively, under different workloads of the soft real-time transactions and at a hard real-time transaction workload respectively equal to 15% (HRT generation periods are 80 s, 40 s, 25 s, 20 s, 16 s) and 30% (HRT
periods are half shortened) utilization of the system. The simulator is implemented in OpNet, which is a simulation tool for network simulation.

From the observation of Fig. 2, when the transaction workload is low, the MR_S is lower under mixed OCC_BC, while as the increase of soft real-time transaction arrival rate, the performance of MRTT_CC is better than both RCP and OCC_BC due to exploring of data similarity notion and adopting weaker correctness criteria. It helps to reduce unnecessary soft real-time transaction restart and blocking. The MR_H is maintains to be zero under both MRTT_CC and RCP. On the contrary, when system work load become higher, MR_H under mixed OCC_BC rises. The reason is that the validation of a transaction has to be performed without delay, therefore when a SRT goes through the validation phrase, it may block the execution of other HRTs. The results shown in Fig. 3 is similar to that of Fig. 2, the MR_S of MRTT_CC is lower than RCP and mixed OCC_BC when the system workloads of both soft and hard real-time transactions are heavy.

Table 2  Parameters and their baseline settings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
<th>Baseline value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBsize</td>
<td>Database Size</td>
<td>400 pages</td>
</tr>
<tr>
<td>N_HRT</td>
<td>Number of HRT Generator</td>
<td>5</td>
</tr>
<tr>
<td>$\lambda_{SRT}$</td>
<td>Mean arrival times of SRT</td>
<td>1/s</td>
</tr>
<tr>
<td>MTS_HRT</td>
<td>HRT Length</td>
<td>5 op.</td>
</tr>
<tr>
<td>MTS_SRT</td>
<td>SRT Length</td>
<td>(5 to 10) op. uniform distribution</td>
</tr>
<tr>
<td>$P_{op}$</td>
<td>Probability of write operation</td>
<td>50%</td>
</tr>
<tr>
<td>$S_{LT}$</td>
<td>Slack range of SRT</td>
<td>1.5 to 2.5</td>
</tr>
<tr>
<td>CPUcl</td>
<td>CPU time for checking a lock</td>
<td>0.1 ms</td>
</tr>
<tr>
<td>CPUsm</td>
<td>CPU time for checking data similarity</td>
<td>0.05 ms</td>
</tr>
</tbody>
</table>

Fig. 2  MRS and MRH when HRT workload is 15%

Fig. 3  MRS and MRH when HRT workload is 30%

5 Conclusions

In recent years, issues on the concurrency control of mixed transactions in RTDBS have received growing attention, as it is more practical for many real-time applications. We call this kind of RTDBS as Mixed Real-time Database Systems. The real-time concurrency control protocols designed for a single
type of real-time transactions may not fit very well for MRTDBS, due to the unique characteristics and performance requirements of different types of transactions. In this paper, we propose a new concurrency control protocol MRTT_CC for MRTDBS. In our new protocol, different strategies are adopted to resolve the data conflicts between different types of transactions and the data similarity definition is used to release the serializability constraint. A simulation model has been implemented to compare the performance of MRTT_CC protocol and other two existing protocols RCP and OCC_BC. The experiment results show that with the similarity correctness notion, a better improvement in system performance can be obtained.

References