Error-Driven QoS Management in Imprecise Real-Time Databases*

Mehdi Amirijoo, Jörgen Hansson
Dept. of Computer and Information Science
Linköping University, Sweden
{meham,jorha}@ida.liu.se

Sang H. Son
Dept. of Computer Science
University of Virginia, Virginia, USA
son@cs.virginia.edu

Abstract

In applications such as web-applications, e-commerce, and engine control, the demand for real-time data services has increased. In these applications, requests have to be processed within their deadlines using fresh data. Since the workload of these systems cannot be precisely predicted, they can become overloaded and as a result, deadline and freshness violations may occur. To address this problem we propose a QoS-sensitive approach based on imprecise computation, applied on transactions and data objects. We propose two algorithms FCS-HEF and FCS-HEDF that give a robust and controlled behavior of RTDBs in terms of transaction and data preciseness, even for transient overloads and with inaccurate run-time estimates of the transactions. Further, performance experiments show that the proposed algorithms outperform a set of baseline algorithms including FCS-EDF, which schedules the transactions using EDF.

1 Introduction

In applications providing real-time data service it is desirable to process user requests within their deadlines using fresh data. In dynamic systems, such as web servers and sensor networks with non-uniform access patterns, the workload of the databases cannot be precisely predicted and, hence, the databases can become overloaded. As a result, deadline misses and freshness violations may occur during the transient overloads. To address this problem we propose a quality of service (QoS) sensitive approach to guarantee a set of requirements on the behavior of the database, even in the presence of unpredictable workloads. Further, for some applications (e.g. web service) it is desirable that the quality of service does not vary significantly from one transaction to another. Here, it is emphasized that the individual QoS needs requested by clients and transactions are enforced and, hence, any deviations from the QoS needs should be uniformly distributed among the clients to ensure QoS fairness.

We employ the notion of imprecise computation [11], where it is possible to trade resource needs for quality of requested service. Imprecise computation has successfully been applied to applications where timeliness is emphasized, e.g. avionics, engine control, image processing, networking, and approximation algorithms for NP-complete problems. We believe that our approach is important to applications that require timely execution of transactions, but where certain degree of imprecision can be tolerated.

In our previous work [3] we presented two algorithms, FCS-IC-1 and FCS-IC-2, for managing QoS using imprecise computation [11, 15, 7, 5] and feedback control scheduling [12, 13, 4]. In this paper we extend our previous work by defining a general model of transaction preciseness, and we present two new scheduling algorithms, FCS-HEF and FCS-HEDF that, in addition to managing QoS, enhance QoS fairness (i.e. decrease the deviation in quality of service among admitted transactions) and provide an improved transient state performance.

We have carried out a set of experiments to evaluate the performance of the proposed algorithms. The studies show that the suggested algorithms give a robust and controlled behavior of RTDBs, in terms of transaction and data preciseness, even for transient overloads and with inaccurate execution time estimates of the transactions. We say that a system is robust if it has good regulation or adaptation in the face of changes in system parameters (e.g. execution time estimation error and applied load), and also has good disturbance rejection, i.e., eliminating the impact of disturbances (in RTDBs disturbances occur due to e.g. concurrency control, resulting in restart or termination of transactions).

The rest of this paper is organized as follows. A problem formulation is given in Section 2. In Section 3, the assumed database model is given. In Section 4, we present our approach and in Section 5, the results of performance evaluations are presented. In Section 6, we give an overview on

*This work was funded, in part by CUGS (the National Graduate School in Computer Science, Sweden), CENIIT (Center for Industrial Information Technology) under contract 01.07, and NSF grant IIS-0208758.

related work, followed by Section 7, where conclusions and future work are discussed.

2 Problem Formulation

In our model, data objects in a RTDB are updated by update transactions, e.g. sensor values, while user transactions represent user requests, e.g. complex read-write operations. The notion of imprecision may be applied at data object and/or user transaction level. The data quality increases as the imprecision of the data objects decreases. Similarly, the quality of user transactions (for brevity referred to as transaction quality) increases as the imprecision of the results produced by user transactions decreases. Hence, we model transaction quality and data quality as orthogonal entities.

Starting with data impreciseness, for a data object stored in the RTDB and representing a real-world variable, we can allow a certain degree of deviation compared to the real-world value. If such deviation can be tolerated, arriving updates may be discarded during transient overloads. In order to measure data quality we introduce the notion of *data error* (denoted DE_i), which gives an indication of how much the value of a data object d_i stored in the RTDB deviates from the corresponding real-world value, given by the latest arrived transaction updating d_i.[1]

The quality of user transactions is adjusted by manipulating data error, which is done by considering an upper bound for the data error given by the *maximum data error* (denoted MDE). An update transaction T_j is discarded if the data error of a data object d_i to be updated by T_j is less or equal to MDE (i.e. $DE_i \leq MDE$). Increasing MDE implies that more update transactions are discarded, degrading the quality of data. Similarly, decreasing MDE implies that fewer update transactions are discarded, resulting in a greater data quality.

Moreover, we introduce the notion of transaction error (denoted TE_i), inherited from the imprecise computation model [11], to measure the quality of a transaction, T_i. Here, the quality of the result given by a transaction depends on the processing time allocated to the transaction. The transaction returns more precise results (i.e. lower TE_i) as it receives more processing time.

The goal of our work is to derive algorithms for manipulating MDE, such that the data quality and the transaction quality satisfy a given QoS specification and the deviation of transaction quality among admitted transactions is minimized (i.e. QoS fairness is enforced).

[1] Note that the latest arrived transaction updating d_i may have been discarded and, hence, d_i may hold the value of an earlier update transaction.

3 Data and Transaction Model

We consider a main memory database model, where there is one CPU as the main processing element. In our data model, data objects can be classified into two classes, temporal and non-temporal [14]. For temporal data we only consider base data, i.e. data that hold the view of the real-world and are updated by sensors. A base data object d_i is considered temporally inconsistent or stale if the current time is later than the timestamp of d_i followed by the absolute validity interval of d_i (denoted AVI_i), i.e. $CurrentTime > TimeStamp_i + AVI_i$.

For a base data object d_i, let data error be defined as, $DE_i = 100 \times \frac{|CurrentValue_i - V_i|}{|CurrentValue_i|}(\%)$, where V_i is the value of the latest arrived transaction updating d_i, and $CurrentValue_i$ is the current value of d_i. Note that we apply the notion of impreciseness on base data objects only and, hence, data errors do not propagate, i.e., they cannot affect the accuracy of other data objects and transactions.

Transactions are classified either as update transactions or user transactions. Update transactions arrive periodically and may only write to base data objects. User transactions arrive aperiodically and may read temporal and read-/write non-temporal data. A user or update transaction T_i is composed of one mandatory subtransaction (denoted M_i) and $\#O_i$ optional subtransactions (denoted $O_{i,j}$, where $1 \leq j \leq \#O_i$). For the remainder of the paper, we let $t_i \in \{M_i, O_{i,1}, \ldots, O_{i,\#O_i}\}$ denote a subtransaction of T_i.

We use the milestone approach [11] to transaction impreciseness. Thus, we have divided transactions into subtransactions according to milestones. A mandatory subtransaction is completed when it is completed in a traditional sense. The mandatory subtransaction is necessary for an acceptable result and must be computed to completion before the transaction deadline. Optional subtransactions are processed if there is enough time or resources available. While it is assumed that all subtransactions of a transaction T_i arrive at the same time, the first optional subtransaction, i.e. $O_{i,1}$, becomes ready for execution when the mandatory subtransaction is completed. In general, an optional subtransaction $O_{i,j}$ becomes ready for execution when $O_{i,j-1}$ (where $2 \leq j \leq \#O_i$) completes. Hence, there is a precedence relation given by $M_i \prec O_{i,1} \prec O_{i,2} \prec \ldots \prec O_{i,\#O_i}$.

We set the deadline of all subtransactions t_i to the deadline of T_i. A subtransaction is terminated if it is completed or has missed its deadline. A transaction is terminated when its last optional subtransaction completes or one of its subtransactions misses its deadline. In the latter case, all subtransactions that are not yet completed are terminated as well. If a user transaction is terminated when its last optional subtransaction is complete, then the corresponding transaction error is zero and we say that the transaction is

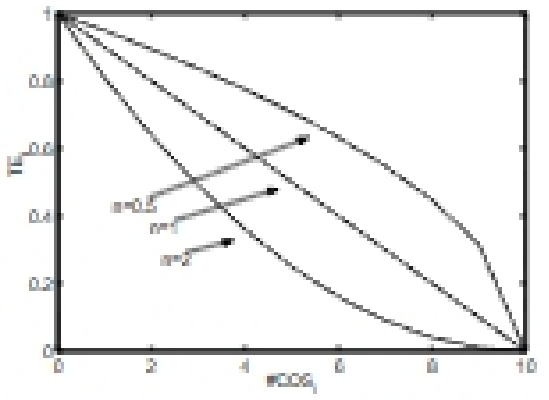

Figure 1. Contribution of $\#COS_i$ to TE_i

precisely scheduled.

For update transactions we assume that there are no optional subtransactions (i.e. $\#O_i = 0$). Each update transaction consists only of a single mandatory subtransaction, since updates do not use complex logical or numerical operations and, hence, normally have a lower execution time than user transactions.

For a transaction T_i, we use an error function to approximate its corresponding transaction error given by, $TE_i(\#COS_i) = \left(1 - \frac{\#COS_i}{\#O_i}\right)^{n_i}$, where n_i is the order of the error function and $\#COS_i$ denotes the number of completed optional subtransactions. This error function is similar to the one presented in [5]. By choosing n_i we can model and support multiple classes of transactions showing different error characteristics (see Figure 1). For example, it has been shown that anytime algorithms used in AI exhibit error characteristics where n_i is greater than one [17].

A summary of transaction model attributes and their abbreviations is given in Table 1.

Table 1. Abbreviation of transaction attributes

Attribute	Description
AET_i	average execution time of T_i
AIT_i	average inter-arrival time of T_i
AU_i	average utilization of T_i
AT_i	arrival time of T_i
D_i	relative deadline of T_i
EET_i	estimated average execution time of T_i
EIT_i	estimated inter-arrival time of T_i
EU_i	estimated utilization of T_i
P_i	period of T_i
TE_i	transaction error of T_i
V_i	update value

4 Approach

Below we describe our approach for managing the performance of a RTDB in terms of transaction and data quality. First, we start by defining QoS and how it can be specified. An overview of the feedback control scheduling architecture is given, followed by issues related to modeling of the architecture and design of controllers. Finally, we present the algorithms FCS-HEF and FCS-HEDF.

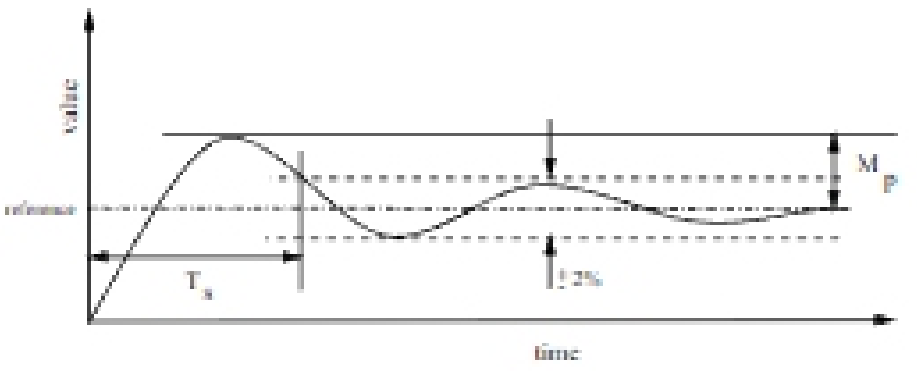

Figure 2. Definition of settling time (T_s) and overshoot (M_p)

4.1 Performance Metrics and QoS specification

In our approach, the database administrator (DBA) can explicitly specify the required database QoS, defining the desired behavior of the database. In this work we adapt both steady-state and transient-state performance metrics [12] as follows:

- Average Transaction Error, denoted $ATE(k)$. A DBA can specify the desired average transaction error of admitted user transactions. The average transaction error gives the preciseness of the results of user transactions, and defined as, $ATE(k) = \frac{\sum_{i \in Terminated(k)} TE_i}{|Terminated(k)|}$, during period k and where $Terminated(k)$ denotes the set of terminated transactions and $|Terminated(k)|$ the number of terminated transactions.[2]
- Maximum Data Error, denoted $MDE(k)$, gives the maximum data error tolerated for the data objects (as described in Section 2) during period k.
- Overshoot, denoted M_p, is the worst-case system performance in the transient system state (see Figure 2) and it is given in percentage. The overshoot is applied to ATE and MDE.
- Settling time, denoted T_s, is the time for the transient overshoot to decay and reach the steady state performance (see Figure 2) and, hence, it is a measure of system adaptability.

We define *Quality of Data* (denoted QoD) in terms of MDE. An increase in QoD refers to a decrease in MDE. In contrast, a decrease in QoD refers to an increase in MDE. Similarly, we define *Quality of Transaction* (denoted QoT) in terms of ATE. QoT increases as ATE decreases, while QoT decreases as ATE increases.

[2] For the rest of this paper, we sometimes drop k where the notion of time is not important.