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 impreciseness, 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.

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 IEEE

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, $DE_i = 100 \times \frac{|CurrentValue_i - V_j|}{|CurrentValue_i|}(\%)$, where V_j 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

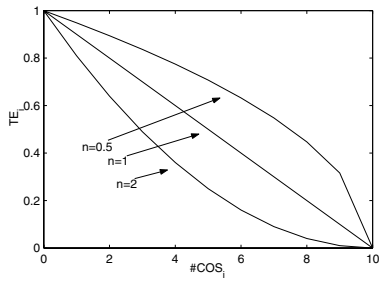

Figure 1. Contribution of $\#COS_i$ to TE_i

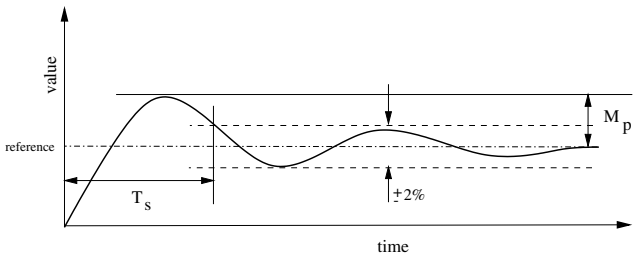

Figure 2. Definition of settling time (T_s) and overshoot (M_p)

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.

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.

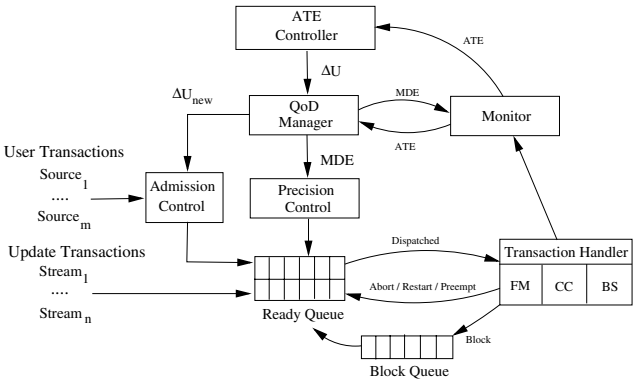

Figure 3. Feedback control scheduling architecture

The DBA can specify a set of target levels or references for ATE and MDE. A QoS requirement can be specified as follows: $ATE_r = 20\%$ (i.e. reference ATE, meaning that we wish $ATE(k)$ to equal ATE_r for all $k > 0$), $MDE_r = 5\%$ (i.e. reference MDE), $T_s \leq 60s$, and $M_p \leq 30\%$. This gives the following transient performance specifications: $ATE \leq ATE_r \times (M_p + 100) = 26\%$, and $MDE \leq MDE_r \times (M_p + 100) = 6.5\%$.

4.2 Feedback Control Scheduling Architecture

In this section we give an overview of the feedback control scheduling architecture. Further, we identify a set of control related variables, i.e. performance references, manipulated variables, and controlled variables.

The general outline of the feedback control scheduling architecture is given in Figure 3. Admitted transactions are placed in the ready queue. The transaction handler manages the execution of the transactions. At each sampling instant, the controlled variable ATE is monitored and fed into the average transaction error controller, which compares the performance reference, ATE_r, with ATE to get the current performance error. Based on the current error the controller computes a change, denoted ΔU, to the total estimated requested utilization. We refer to ΔU as the manipulated variable. Based on ΔU, the QoD manager changes the total estimated requested utilization by adapting the QoD (i.e. adjusting MDE). The precision controller then schedules the update transactions based on MDE. The portion of ΔU not accommodated by the QoD manager, denoted ΔU_{new}, is enforced by admission control.

The transaction handler provides a platform for managing transactions. It consists of a freshness manager (FM), a unit managing the concurrency control (CC), and a basic scheduler (BS). The FM checks the freshness before accessing a data object, using the timestamp and the absolute validity interval of the data. We employ two-phase locking with highest priority (2PL-HP) [1] for concurrency control. 2PL-HP is chosen since it is free from priority inversion and has well-known behavior. We consider three different scheduling algorithms as basic schedulers: (1) Earliest Deadline First (EDF), where transactions are processed in the order determined by increasing absolute deadlines (2) Highest Error First (HEF), where transactions are processed in the order determined by decreasing transaction error, and (3) Highest Error Density First (HEDF), where transactions are processed in the order determined by decreasing transaction error density given by, $TED_i = \frac{TE_i}{AT_i + D_i - CurrentTime}$, where AT_i and D_i denote the arrival time and relative deadline of the transaction T_i, respectively. For all three basic schedulers (EDF, HEF, and HEDF) the mandatory subtransactions have higher priority than the optional subtransactions and, hence, scheduled before them.

The precision controller discards an update transaction writing to a data object d_i having an error less or equal to the maximum data error allowed, i.e. $DE_i \leq MDE$. However, the update transaction is executed if the data error of d_i is greater than MDE. In both cases the time-stamp of d_i is updated.

4.3 System Modeling and Controller Design

We have modeled the controlled system, i.e. RTDB, according to the analytical approach proposed in [12]. The approach has been adapted such that it supports average transaction error. The transfer function of the model describing ATE in terms of ΔU, is given by, $P(z) = \frac{ATE(z)}{\Delta U(z)} = \frac{G_A G_{ATE}}{z-1}$, where G_A represents the extent of worst case actual workload variation in terms of estimated requested workload, and G_{ATE} denote the ATE variation in terms of actual workload. The ATE controller is implemented using PI controllers tuned with root locus [6]. For a more elaborate description, tuning of the model and the ATE controller we refer to [2].

4.4 Algorithms for Data and Transaction Error Management

We use the same QoS management algorithm for FCS-HEF and FCS-HEDF, but use different basic schedulers, e.g. FCS-HEF schedules the transactions with HEF. The following scheme for managing the data and transaction quality is common for FCS-HEF and FCS-HEDF.

Given a certain $\Delta U(k)$, we need to set $MDE(k+1)$ such that the change in utilization due to discarding update transactions correspond to $\Delta U(k)$. Remember that setting $MDE(k+1)$ greater than $MDE(k)$ results in more discarded update transactions and, hence, an increase in gained utilization. Similarly, setting $MDE(k+1)$ less

than $MDE(k)$ results in fewer discarded update transactions and, hence, a decrease in gained utilization. In order to compute $MDE(k+1)$ given a certain $\Delta U(k)$, we use a function $f(\Delta U(k))$ that returns, based on $\Delta U(k)$, the corresponding $MDE(k+1)$. The function f holds the following property. If $\Delta U(k)$ is less than zero, then $MDE(k+1)$ is set such that $MDE(k+1)$ is greater than $MDE(k)$ (i.e. QoD is degraded). Similarly, if $\Delta U(k)$ is greater than zero, then $MDE(k+1)$ is set such that $MDE(k+1)$ is less than $MDE(k)$ (i.e. QoD is upgraded). We will return to the concepts around f in Section 4.5, but the detailed derivation of f can be found in [2].

One of the characteristics of the average transaction error controller is that as long as the average transaction error is below its reference (i.e. $ATE \leq ATE_r$), the controller output ΔU will be positive.[3] Due to the characteristics of f (i.e. $\Delta U(k) > 0 \Rightarrow MDE(k+1) < MDE(k)$), a positive ΔU is interpreted as a QoD upgrade. Consequently, even if the average transaction error is just below its reference, QoD remains high. However, it is desired that the average transaction error, which corresponds to QoT, increases and decreases together with QoD. For this reason, MDE is set not only by considering ΔU, but also according to the current ATE. When ΔU is less than zero (i.e. at an ATE overshoot), MDE is set according to f. However, when ΔU is greater or equal to zero, MDE is set according to the moving average of ATE, computed by $ATE_{MA}(k) = \alpha ATE(k) + (1-\alpha) ATE_{MA}(k-1)$, where α ($0 \leq \alpha \leq 1$) is the forgetting factor. Setting α close to 1 results in a fast adaptation, but will also capture the high-frequency changes of ATE, whereas setting α close to 0, results in a slow but smooth adaptation. When ATE_{MA} is relatively low compared to ATE_r, MDE is set to a low value relative to MDE_r. As ATE_{MA} increases, MDE is increased but not more than $MDE_r \times (M_p + 100)$, since a further increase violates the given QoS specification. The outline of FCS-HEF and FCS-HEDF is given in Figure 4.

4.5 QoD Management

The preciseness of the data is controlled by the QoD manager setting MDE depending on the system behavior. When f is used to compute $MDE(k+1)$ based on $\Delta U(k)$ the following scheme is used.

Rejecting an update results in a decrease in CPU utilization. We define *gained utilization*, $GU(k)$, as the utilization gained due to the result of rejecting one or more updates during period k. $GU(k)$ is formally defined as, $GU(k) = \sum_i \frac{\#RU_i(k)}{\#AU_i(k)} \times EU_i$, where $\#RU_i(k)$ is the number of rejected update transactions T_i generated by $Stream_i$, $\#AU_i(k)$ the number of arrived update transactions T_i, and EU_i is the estimated utilization of the update transactions T_i.

[3] If we have transient oscillations, ΔU may temporally stay positive (negative) even though the ATE has changed from being below (above) the reference to be above (below) the reference value. This is due to the integral operation, i.e., due to earlier summation of errors, which represents the history and therefore cause a delay before a change to the utilization is requested and has effect.

```
Monitor ATE(k)
Compute ΔU(k)
if (ΔU(k) ≥ 0) then
   MDE(k+1) =
      min(ATE_MA(k)/ATE_r MDE_r, MDE_r × (M_p + 100))
   if (MDE(k) < MDE(k+1)) then
      Add the utilization gained after QoD degrade to
      ΔU(k)
   else
      Subtract the utilization lost after QoD upgrade from
      ΔU(k)
   end if
   Inform AC of the new ΔU(k)
else if (ΔU(k) < 0 and
         MDE(k) < MDE_r × (M_p + 100)) then
   Downgrade QoD according to
      MDE(k+1) = f(ΔU(k))
   Inform AC about the portion of ΔU(k) not accommo-
   dated by QoD downgrade
else
   {i.e. ΔU(k) < 0 and
         MDE(k) = MDE_r × (M_p + 100)}
   Reject any incoming transactions
end if
```

Figure 4. QoS management algorithm of FCS-HEF and FCS-HEDF

In our approach, we profile the system and measure GU for different MDEs and linearize the relationship between these two, i.e. $MDE = \mu \times GU$. Further, since RTDBs are dynamic systems in that the behavior of the system and environment is changing, the relation between GU and MDE is adjusted on-line. This is done by measuring $GU(k)$ for a given $MDE(k)$ during each sampling period and updating μ. Having the relationship between GU and MDE, we introduce the help function

$$h(\Delta U(k)) = \min \left(\begin{array}{c} \mu \times (GU(k) - \Delta U(k)), \\ MDE_r \times (M_p + 100) \end{array} \right).$$

Since MDE is not allowed to overshoot more than $MDE_r \times (M_p + 100)$ we use the min operator to enforce this requirement. Further, since MDE by definition cannot be less than 0, we apply the max operator on h and obtain,

$$MDE(k+1) = f(\Delta U(k)) = \max(h(\Delta U(k)), 0).$$

5 Performance Evaluation

5.1 Experimental Goals

The main objective of the experiments is to show whether the presented algorithms can provide QoS guarantees according to a QoS specification. We have for this reason studied and evaluated the behavior of the algorithms according to a set of performance metrics:

- Load ($Load$). Computational systems may show different behaviors for different loads, especially when the system is overloaded. For this reason, we measure the performance when applying different loads to the system.
- Execution Time Estimation Error ($EstErr$). Often exact execution time estimates of transactions are not known. To study how runtime error affects the algorithms, we measure the performance considering different execution time estimation errors.

5.2 Simulation Setup

The simulated workload consists of update and user transactions that access data and perform virtual arithmetic/logical operations on the data. Update transactions occupy approximately 50% of the workload.

In our experiments, one simulation run lasts for 10 minutes of simulated time. For all the performance data, we have taken the average of 10 simulation runs and derived 95% confidence intervals. We assume the following QoS specification: $ATE_r = 20\%$, $MDE_r = 5\%$, $T_s \leq 60s$, and $M_p \leq 30\%$. The workload model of the update and user transactions are described as follows. We use the following notation where the attribute X_i refers to the transaction T_i, and $X_i[t_i]$ is associated with the subtransaction t_i.

Data and Update Transactions. The DB holds 1000 temporal data objects (d_i) where each data object is updated by a stream ($Stream_i$, $1 \leq i \leq 1000$). The period (P_i) is uniformly distributed in the range $(100ms, 50s)$ (i.e. $U : (100ms, 50s)$) and estimated execution time (EET_i) is given by $U : (1ms, 8ms)$. The average update value (AV_i) of each $Stream_i$ is given by $U : (0, 100)$. Upon a periodic generation of an update, $Stream_i$ gives the update an actual execution time given by the normal distribution $N : (EET_i, \sqrt{EET_i})$ and a value (V_i) according to $N : (AV_i, AV_i \times VarFactor)$, where $VarFactor$ is uniformly distributed in (0,1). The deadline is set to $AT_i + P_i$.

User Transactions. Each $Source_i$ generates a transaction T_i, consisting of one mandatory subtransaction, M_i, and $\#O_i$ ($1 \leq \#O_i \leq 10$) optional subtransaction(s), $O_{i,j}$ ($1 \leq j \leq \#O_i$). $\#O_i$ is uniformly distributed between 1 and 10. The estimated (average) execution time of subtransactions ($EET_i[t_i]$) is given by $U : (5ms, 15ms)$. The estimation error $EstErr$ is used to introduce execution time estimation error in the average execution time given by $AET_i[t_i] = (1 + EstErr) \times EET_i[t_i]$. Further, upon generation of a transaction, $Source_i$ associates an actual execution time to each subtransaction t_i, which is given by $N : (AET_i[t_i], \sqrt{AET_i[t_i]})$. The deadline is set to $AT_i + EET_i \times SlackFactor$. The slack factor is uniformly distributed according to $U : (20, 40)$. Transactions are evenly distributed in four classes representing error function orders of 0.5, 1, 2, and 5 (e.g. 25% of the transactions have an error order of 1).

5.3 Baselines

To the best of our knowledge, there has been no earlier work on techniques for both managing data impreciseness and transaction impreciseness, satisfying QoS or QoD requirements. For this reason, we have developed two baseline algorithms, Baseline-1 and Baseline-2, to study the impact of the workload on the system. We also compare the behavior of FCS-HEF and FCS-HEDF with FCS-EDF since EDF is optimal (in minimizing deadline misses) and has well-known behavior. The baselines are given below.

Baseline-1. If ATE is greater than its reference, the utilization is lowered by increasing MDE. Here, the preciseness of the data is adjusted based on the relative average transaction error. MDE is set according to $MDE(k+1) = \min(\frac{ATE(k)}{ATE_r} MDE_r, MDE_r \times (M_p + 100))$. A simple admission control is applied, where a transaction (T_i) is admitted if the estimated utilization of admitted subtransactions and EET_i is less or equal to 80%.

Baseline-2. In order to prevent a potential overshoot, we increase MDE as soon as ATE is greater than zero. If $ATE(k)$ becomes greater than zero, we increase $MDE(k)$ stepwise until $MDE_r \times (M_p + 100)$ is reached (i.e. $MDE(k+1) = \min(MDE(k) + MDE_{step}, MDE_r \times (M_p + 100))$). If $ATE(k)$ is equal to zero, we decrease $MDE(k)$ stepwise until zero is reached (i.e. $MDE(k+1) = \max(MDE(k) - MDE_{step}, 0)$). The same admission control as in Baseline-1 is used.

FCS-EDF. FCS-EDF is similar to FCS-HEF and FCS-HEDF in that it is based on the algorithm given in Figure 4, but where EDF is used as a basic scheduler whereas FCS-HEF and FCS-HEDF use HEF and HEDF, respectively. We compare the performance of FCS-HEF and FCS-HEDF with FCS-EDF, since the behavior of EDF is well-known.

5.4 Performance Metrics

To measure QoS fairness among transactions, we introduce Standard Deviation of Transaction Error, $SDTE(k)$, which gives a measure of how much the

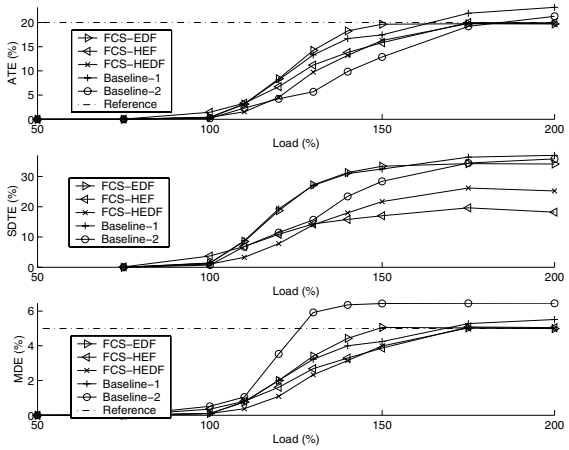

Figure 5. Average Performance: $EstErr = 0$

transaction error of terminated transactions deviates from the average transaction error. $SDTE(k)$ in the k^{th} sampling instant is given by, $SDTE(k) = \sqrt{\frac{1}{|Terminated(k)|-1} \sum_{i \in Terminated(k)} (TE_i - ATE(k))^2}$.

In our simulations we analyze ATE, MDE, utilization (denoted U), and $SDTE$.

5.5 Experiment 1: Results of Varying Load

The setup of the experiment is given below, followed by the presentation of the results. Figure 5 shows ATE, $SDTE$, and MDE.

Experimental setup. We apply loads from 50% to 200%. The execution time estimation error is set to zero (i.e. $EstErr = 0$).

Tardy Mandatory Subtransactions. We have not observed any deadline misses for any of the five algorithms, which is consistent with our requirement of completing all mandatory subtransactions.

Average Transaction Error. The confidence intervals for all algorithms are within $\pm 2.1\%$. For Baseline-1 and Baseline-2, the average transaction error (ATE) increases as the load increases, violating the reference, ATE_r, at loads exceeding 175%. Baseline-2 produces low ATE for loads 100-175%. This is due to the high MDE, and since many update transactions are discarded more resources can be allocated to user transactions. In the case of FCS-EDF, ATE reaches the reference at 150% applied load. For FCS-HEF and FCS-HEDF, ATE reaches the reference at 175%. All FCS algorithms provide a robust performance since ATE is kept at the specified reference during overloads. It is worth mentioning that ATE for FCS-EDF is higher than the other algorithms. As given by our simulation setup for user transactions (Section 5.2), we can see that about 50% of the transactions apply error functions where the order of the error functions are greater than one (i.e. $n_i > 1$). Hence, for a great portion of the transactions the error decays significantly at the completion of the initial optional subtransactions and less as more subtransactions are completed. During overloads where transactions cannot be precisely scheduled, it is more feasible to distribute the resources to transactions such that only the first optional subtransactions are completed, as completing the last subtransactions does not decrease the transaction error as much. This is done under HEF scheduling since resources are allocated to transactions with the highest error and given that approximately 50% of the transactions have error functions with n_i greater than one, the average transaction error is minimized. EDF, however, allocates resources to transactions with no regard to the actual decrease in error when completing a subtransaction.

Standard Deviation of Transaction Error. The confidence intervals for all algorithms are within $\pm 1.92\%$. $SDTE$ is higher for FCS-HEF than the other algorithms when the system is underutilized (load less than 100%). This is due to the relatively high ATE for loads less than 100%. For all algorithms, $SDTE$ increases as load and ATE increase. At 200% load, the corresponding $SDTE$ for FCS-EDF, FCS-HEF, and FCS-HEDF is 34.1%, 25.2% and 18.2%, respectively. Consequently, deviation of transaction error is minimized when HEF scheduling is used. It is worth mentioning that under HEF scheduling $SDTE$ is less as resources are allocated with regards to transaction errors, while under EDF scheduling resources are distributed with regards to deadlines. Under EDF scheduling, two transactions with deadlines near each other may receive different amounts of resources and, hence, they may terminate with a significant deviation in transaction error, while in the same case, under HEF scheduling the resources are divided such that both transactions terminate with transaction errors close to each other. We would like to point out that there is a lower bound for $SDTE$ as transaction error functions are discrete and, hence, transaction errors decrease in steps. This becomes more evident the fewer subtransactions a transaction has as the steps become greater.

Average MDE. The confidence intervals for all algorithms are within $\pm 0.5\%$. The average MDE for Baseline-1 and Baseline-2 violates the reference MDE set to 5% at applied loads of 175% and 130%, respectively. In contrast, in the case of FCS-EDF, FCS-HEF, and FCS-HEDF, MDE is at the reference during overloads.

Average Utilization. For all approaches, the utilization is above 95% for loads between 100-200% showing the high performance of the algorithms.

The experiments shown that FCS-HEF, FCS-HEDF, and FCS-EDF are robust against varying applied loads. Moreover, FCS-HEF outperforms the other algorithms with regards to QoS fairness of admitted transactions.

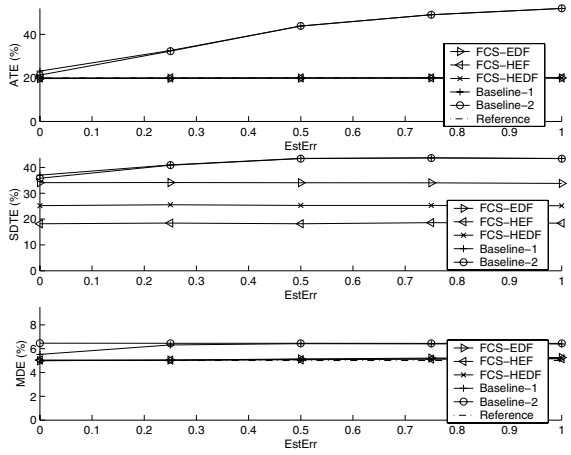

Figure 6. Average Performance: $Load = 200\%$

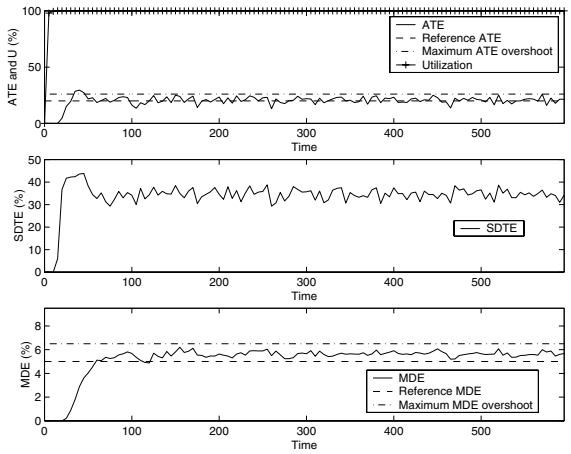

Figure 7. Transient Performance for FCS-EDF. $EstErr = 1.0$**,** $Load = 200\%$

5.6 Experiment 2: Results of Varying EstErr

The setup of the experiment is given below, followed by the presentation of the results. Figure 6 shows ATE, $SDTE$, and MDE.

Experimental setup. We apply 200% load. The execution time estimation error is varied according to $EstErr$ = 0.00, 0.25, 0.50, 0.75, and 1.00.

Tardy Mandatory Subtransactions. As in Experiment 1, no mandatory subtransactions have missed their deadline.

Average Transaction Error. The confidence intervals for all algorithms are within $\pm 3.5\%$. As expected, Baseline-1 and Baseline-2 do not satisfy the QoS specification. In fact, ATE increases as $EstErr$ increases, reaching a value close to 52% for both algorithms. As we can see, FCS-EDF, FCS-HEF, and FCS-HEDF are insensitive against varying $EstErr$.

Standard Deviation of Transaction Error. The confidence intervals for all algorithms are within $\pm 1.5\%$. For Baseline-1 and Baseline-2 $SDTE$ increases as ATE and $EstErr$ increase, reaching 43.4% at $EstErr$ equals 0.5. In the case of FCS-EDF, FCS-HEF, and FCS-HEDF, $SDTE$ does not change since the corresponding ATE is invariant.

Average MDE. The confidence intervals for all algorithms are within $\pm 0.3\%$. Baseline-1 and Baseline-2 violate the specified MDE reference. For FCS-EDF, FCS-HEF, and FCS-HEDF, average MDE does not change considerably for different $EstErr$, peaking 5.26% when $EstErr$ is set to one.

From these results we observe that FCS-HEF, FCS-HEDF, and FCS-EDF are insensitive to changes to execution time estimation errors as they manage to satisfy the given QoS specification for varying $EstErr$.

5.7 Experiment 3: Transient Performance

Studying the average performance is often not enough when dealing with dynamic systems. Therefore we study the transient performance of the proposed algorithms. Figures 7, 8, and 9 show the transient behavior of FCS-EDF, FCS-HEF, and FCS-HEDF. The dash-dotted line indicates maximum overshoot, whereas the dashed lines represent the reference.

Experimental setup. $Load$ is set to 200% and $EstErr$ set to one.

Results. For all algorithms ATE overshoots decay faster than 60s, which are less than the settling time requirement given in the QoS specification.

The highest overshoot for FCS-EDF has been noted to 35.9% at time 40. For FCS-HEF, the highest overshoot was noted to 30.7% at time 40 (the second highest overshoot was noted to 26.1% at time 555) and finally, the highest overshoot for FCS-HEDF was observed to be 32.5% at time 45. As we can see, the algorithms do not satisfy the overshoot requirements given in the QoS specification (i.e. $ATE \leq 26\%$). It is worth mentioning that data conflicts, aborts or restarts of transactions and inaccurate run-time estimates contribute to disturbances in a RTDB, complicating the control of ATE (note that we have set $EstErr$ to one).

As we can see from Figures 7-9, FCS-HEF produces less oscillations in ATE and consequently $SDTE$ and MDE. Further, FCS-HEF is less prone to overshoot. Under EDF scheduling, newly admitted transactions are placed further down in the ready queue, since they are less likely to have earlier deadlines than transaction admitted earlier. This means that an actual change to the controlled variable, i.e. ATE, is not noticed until the newly admitted transactions are executing. However, this delay is removed un-

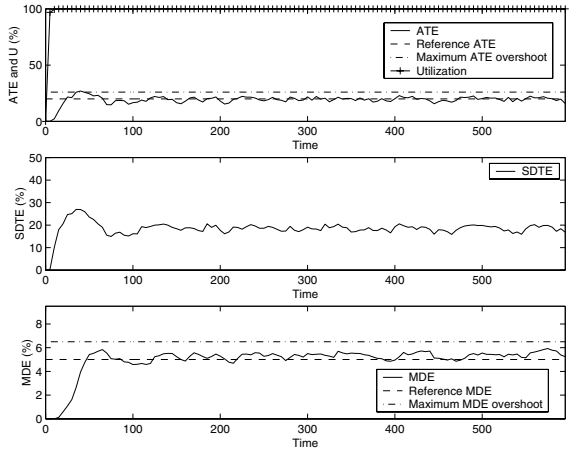

Figure 8. Transient Performance for FCS-HEF. $EstErr = 1.0$, $Load = 200\%$

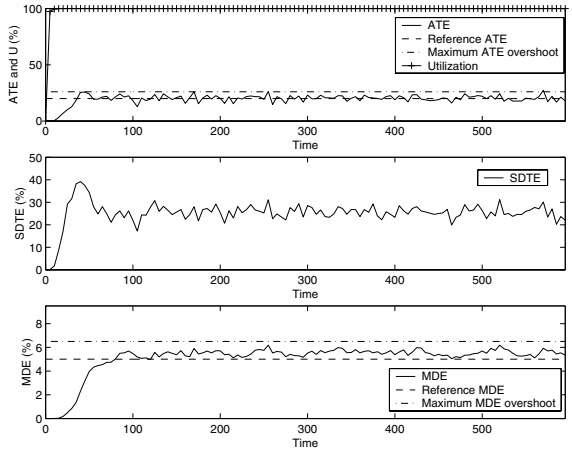

Figure 9. Transient Performance for FCS-HEDF. $EstErr = 1.0$, $Load = 200\%$

der HEF scheduling, since newly arrived transactions are more likely to have higher priority than the old transactions (since they have greater TE) and, consequently, a transient overload/underload is observed earlier. Hence, under HEF scheduling the controlled variable is more responsive to changes in the manipulated variable. Now, from feedback control theory we know that delays in systems (low responsiveness of controlled variables) promote oscillations and may even introduce instability [6]. Given this, we can conclude that under EDF scheduling we should observe more oscillations in ATE than compared with HEF scheduling.

5.8 Summary of Results and Discussions

Our experiments show that the algorithms FCS-HEF and FCS-HEDF are robust against load variations and inaccurate execution time estimations as ATE and MDE have been consistent with their references for varying load and varying execution time estimation error. The proposed algorithms outperform the baseline algorithms and FCS-EDF and can manage the given QoS specifications well. We have carried out other types of experiments [2], but have not presented them due to space limitation (e.g. performance evaluation of the algorithms for transactions sets having different transaction error characteristics and evaluation of the algorithms with respect to different QoS specifications). The results of the additional experiments are consistent with the results presented in this paper, i.e., they show the robustness and feasibility of FCS-HEF and FCS-HEDF.

It was shown that FCS-HEDF produces a lower ATE compared to the other algorithms. Also, it was observed that FCS-HEF provides a lower $SDTE$ compared to other algorithms, lowering the deviation of transaction error among terminated transactions. This property is feasible in applications where QoS fairness among transactions is emphasized. Finally, FCS-HEF showed a better transient behavior compared to FCS-EDF and FCS-HEDF, as it produced fewer and smaller ATE overshoots.

We conclude that FCS-HEF should be used in applications where QoS fairness among transactions is important, but also where overshoots must be prevented, i.e. the worst-case performance has to comply with the specification. However, FCS-HEDF is particularly feasible in cases where low ATE is desired since under HEDF scheduling, ATE is smaller compared to HEF and EDF scheduling.

6 Related Work

There has been several algorithms proposed addressing imprecise scheduling problems [11, 15, 7, 5]. These algorithms require the knowledge of accurate processing times of the tasks, which is often not available in RTDBs. Further, they focus on maximizing or minimizing a performance metric (e.g. total error). The latter cannot be applied to our problem, since in our case we want to control a set of performance metrics such that they converge towards a set of references given by a QoS specification.

Feedback control scheduling has been receiving special attention in the past few years [12, 13, 4]. Lu et al. have presented a feedback control scheduling framework where they propose three algorithms for managing the miss percentage and/or utilization [12]. In comparison to the proposed approaches here, they do not address the problem of maximizing QoS fairness among admitted tasks. In the work by Parekh et al., the length of a queue of remote procedure calls (RPCs) arriving at a server is controlled [13]. Changing the periodicity of a set of tasks in response to load variations has been suggested in [4]. In contrast to FCS-HEF and FCS-HEDF, aperiodic tasks are not considered in their

model.

Labrinidis et al. introduced the notion of QoD in the context of web servers [10]. Their proposed update scheduling policy of cached web pages can significantly improve data freshness compared to FIFO scheduling. Kang et al., used a feedback control scheduling architecture to balance the load of user and update transactions [9]. In our previous work, we presented two algorithms for managing QoS based on feedback control scheduling and imprecise computation [3], where QoS was defined in terms of deadline miss percentage of subtransactions.

The correctness of answers to databases queries can be traded off to enhance timeliness. The query processors, APPROXIMATE [16] and CASE-DB [8] are examples of such databases where approximate answers to queries can be produced within certain deadlines. However, in both approaches, impreciseness has been applied to only transactions and, hence, data impreciseness has not been addressed. Further, they have not addressed the notion of QoS. In our work, we have introduced impreciseness at data object level and considered QoS in terms of transactions and data impreciseness.

7 Conclusions and Future Work

In this paper we have argued for the need of increased adaptability of applications providing real-time data services. To address this problem we have proposed a QoS-sensitive approach based on feedback control scheduling and imprecise computation applied on transactions and data objects. Imprecise computation techniques have shown to be useful in many areas where timely processing of tasks or services is emphasized. In this work, we combine the advantages from feedback control scheduling and imprecise computation techniques, forming a framework consisting of a model for expressing QoS requirements, an architecture, and a set of algorithms. We have developed 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 runtime estimates of the transactions. The proposed algorithms outperform the baseline algorithms and FCS-EDF and can manage the given QoS specifications well.

For our future work, we will model the relationship between data error and transaction error and extend our model to support service differentiation.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Transactions on Database System*, 17:513–560, 1992.

[2] M. Amirijoo, J. Hansson, and S. H. Son. Error-driven QoS managment in imprecise real-time databases. Technical report, Department of Computer Science, University of Linköping, 2002. www.ida.liu.se/∼rtslab/publications.

[3] M. Amirijoo, J. Hansson, and S. H. Son. Algorithms for managing QoS for real-time data services using imprecise computation. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications*, 2003.

[4] G. C. Buttazzo and L. Abeni. Adaptive workload managment through elastic scheduling. *Journal of Real-time Systems*, 23(1/2), July/September 2002. Special Issue on Control-Theoretical Approaches to Real-Time Computing.

[5] J. Chung and J. W. S. Liu. Algorithms for scheduling periodic jobs to minimize average error. In *Real-Time Systems Symposium*, pages 142–151, 1988.

[6] G. F. Franklin, J. D. Powell, and M. Workman. *Feedback Control of Dynamic Systems*. Addison-Wesley, third edition, 1998.

[7] J. Hansson, M. Thuresson, and S. H. Son. Imprecise task scheduling and overload managment using OR-ULD. In *Proceedings of the 7th Conference in Real-Time Computing Systems and Applications*, pages 307–314. IEEE Computer Press, 2000.

[8] W. Hou, G. Ozsoyoglu, and B. K. Taneja. Processing aggregate relational queries with hard time constraints. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 68–77. ACM Press, 1989.

[9] K. Kang, S. H. Son, J. A. Stankovic, and T. F. Abdelzaher. A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases. 14th Euromicro Conference on Real-time Systems, June 19-21 2002.

[10] A. Labrinidis and N. Roussopoulos. Update propagation strategies for improving the quality of data on the web. *The VLDB Journal*, pages 391–400, 2001.

[11] J. W. S. Liu, K. Lin, W. Shin, and A. C.-S. Yu. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5), May 1991.

[12] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-time Systems*, 23(1/2), July/September 2002. Special Issue on Control-Theoretical Approaches to Real-Time Computing.

[13] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance managment. *Journal of Real-time Systems*, 23(1/2), July/September 2002. Special Issue on Control-Theoretical Approaches to Real-Time Computing.

[14] K. Ramamritham. Real-time databases. *International Journal of Distributed and Parallel Databases*, (1), 1993.

[15] W. K. Shih and J. W. S. Liu. Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error. *IEEE Transactions on Computers*, 44(3):466–471, 1995.

[16] S. V. Vrbsky and J. W. S. Liu. APPROXIMATE - a query processor that produces monotonically improving approximate answers. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):1056–1068, December 1993.

[17] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.