Specification and Management of QoS 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

Real-time applications such as e-commerce, flight control, chemical and nuclear control, and telecommunication are becoming increasingly sophisticated in their data needs, resulting in greater demands for real-time data services. Since the workload of real-time databases (RTDBs), providing real-time data services, cannot be precisely predicted, they can become overloaded and thereby cause temporal violations, resulting in a damage or even a catastrophe. Imprecise computation techniques address this problem and allow graceful degradation during overloads. In this paper, we present a framework consisting of a model for expressing QoS requirements in terms of data and transaction preciseness, an architecture based on feedback control scheduling, and a set of algorithms implementing different policies and behaviors. Our approach gives a robust and controlled behavior of RTDBs, 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 that schedules the transactions using EDF and feedback control.

1 Introduction

Lately the demand for real-time data services has increased and applications used in manufacturing, web-servers, e-commerce etc. are becoming increasingly sophisticated in their data needs. The data normally spans from low-level control data, typically acquired from sensors, to high-level management and business data. In these applications it is desirable to process user requests within their deadlines using fresh data. In 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 individual QoS needs requested by clients and transactions are enforced, and hence, any deviations from the QoS needs should be uniformly distributed among clients to ensure QoS fairness.

In earlier work [2, 3] we presented the algorithms FCS-IC-1, FCS-IC-2, FCS-HEF, and FCS-HEDF for managing QoS using imprecise computation [9] and feedback control scheduling [10, 11]. An important feature of this class of algorithms is their ability to adapt to various workloads and tolerate inaccurate estimates of execution times, and still conform to a given QoS specification. In this paper we give an overview of FCS-IC-1 and FCS-IC-2. These algorithms control the preciseness of the results, given by transactions, by monitoring the system deadline miss percentage and adapt the system such that a QoS specification in terms of miss percentage and data preciseness is satisfied. Furthermore, we present a more complex model of transaction preciseness by presenting the notion of transaction error, together with FCS-HEF and FCS-HEDF. These two algorithms support QoS specifications given in terms of transaction error and also address QoS fairness, i.e., they are designed to minimize the deviation in QoS among admitted transactions. In this paper we unify our earlier work into a common framework consisting of a model for expressing QoS requirements in terms of data and transaction preciseness, an architecture based on feedback control scheduling, and a set of algorithms implementing different policies and behaviors. Furthermore, we report new experimental results that show that the proposed algorithms also adapt to various QoS specifications and transaction sets showing different transaction error characteristics.

[*]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.

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.

The rest of this paper is organized as follows. The detailed problem formulation is given in Section 2. In Section 3, the 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 related work, followed by Section 7, where conclusions and future work are discussed.

2 Problem Formulation

In our database 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. In this work 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 indicates how much the value of a data object d_i stored in the RTDB deviates from the corresponding real-world value, which is given by the latest arrived transaction updating d_i.[1]

The quality of user transactions is adjusted by managing the 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 the data object d_i to be updated by T_j is less or equal to MDE (i.e. $DE_i \leq MDE$). If MDE increases, more update transactions are discarded, degrading the quality of data. Similarly, if MDE decreases, fewer update transactions are discarded, resulting in improved data quality.

The goal of our work is to derive algorithms for adjusting data error 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). A major issue is how to compute MDE, depending on the user transaction quality.

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 [12]. For temporal data we only consider base data, i.e., data that holds 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 length of the absolute validity interval of d_i (denoted AVI_i), i.e. $CurrentTime > TimeStamp_i + AVI_i$. For a data object d_i, let data error, $DE_i = 100 \times \frac{|CurrentValue_i - V_j|}{|CurrentValue_i|}(\%)$, where V_j is the value of the latest arrived transaction updating d_i and $CurrentValue_i$ the current value of d_i.

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. User and update transactions (T_i) are assumed to be composed of one mandatory subtransaction (denoted M_i) and $\#O_i$ optional subtransactions (denoted $O_{i,j}$, where $O_{i,j}$ is the j^{th} optional subtransaction of T_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 [9] to transaction impreciseness. Thus, we have divided transactions into subtransactions according to milestones. A mandatory subtransaction completes when it successfully finishes. The mandatory subtransaction is necessary for an acceptable result and it 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 $O_{i,j-1}$ ($2 \leq j \leq \#O_i$) completes. Hence, there is a precedence relation among the subtransactions as 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.

[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.

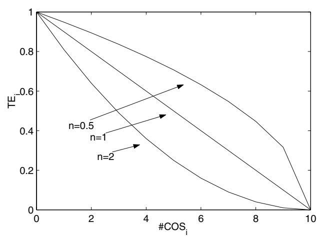

Figure 1. Contribution of $\#COS_i$ to TE_i.

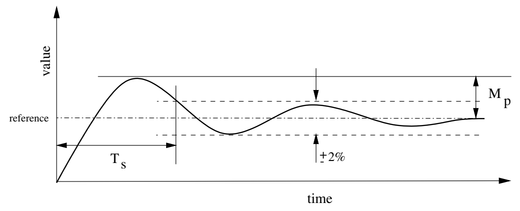

Figure 2. Definition of settling time (T_s) and overshoot (M_p)

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 lower execution times than user transactions.

In some applications it is possible to formally model the preciseness of the answers given by transactions through the use of error functions (e.g., closed systems where the set of transactions that are processed are known in advance). 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 [4]. By choosing n_i we can model and support multiple classes of transactions showing different error characteristics (see Figure 1).

4 Approach

Next we describe our approach for managing the performance of a RTDB in terms of transaction and data quality. First, we start by defining performance metrics and QoS. An overview of the feedback control scheduling architecture is given, followed by issues related to modeling of the architecture and the design of controllers. Finally, we present the algorithms FCS-IC-1, FCS-IC-2, FCS-HEF, and FCS-HEDF.

4.1 Performance Metrics

In this work we adapt both steady-state and transient-state performance metrics [10]. We adapt the following notation of describing discrete variables in the time domain: $A(k)$ refers to the value of the variable A during the time window $[(k-1)W, kW]$, where W is the sampling period and k is the sampling instant.[2] Let $|Terminated(k)|$ be the number of terminated transactions, whereas $|Terminated^M(k)|$ and $|Terminated^O(k)|$ are the number of terminated mandatory and optional subtransactions, respectively. The number of transactions that have missed their deadline is denoted by $|DeadlineMiss(k)|$, and the number mandatory and optional subtransactions that have missed their deadlines is given by $|DeadlineMiss^M(k)|$ and $|DeadlineMiss^O(k)|$, respectively. We exclusively consider transactions admitted to the system.

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

User Transaction Quality Metrics:

- Deadline miss percentage of mandatory user subtransactions is defined as, $M^M(k) = 100 \times \frac{|DeadlineMiss^M(k)|}{|Terminated^M(k)|}(\%)$.
- Deadline miss percentage of optional user subtransactions is defined as, $M^O(k) = 100 \times \frac{|DeadlineMiss^O(k)|}{|Terminated^O(k)|}(\%)$.
- Average transaction error gives the preciseness of the results of user transactions, and it is defined as, $ATE(k) = 100 \times \frac{\sum_{i \in Terminated(k)} TE_i}{|Terminated(k)|}(\%)$.

Data Quality Metric. Maximum data error ($MDE(k)$) is used as a metric for data quality (see section 2).

QoS Fairness Metric. Standard deviation of transaction error ($SDTE$) gives a measure of how much the transaction error of terminated transactions deviates from the average transaction error, and it is defined as, $SDTE(k) = \sqrt{\frac{1}{|Terminated(k)|-1} \sum_{i \in Terminated(k)} (TE_i - ATE(k))^2}$.

System Utilization. We measure system utilization (U) in order to acquire a better understanding of the performance of the algorithms.

Transient-State Performance Metrics. We consider the following transient-state performance metrics (see Figure 2) applied on user transaction quality and data quality performance metrics.

- Overshoot (M_p) is the worst-case system performance in the transient system state and it is given in percentage.
- Settling time (T_s) is the time for the transient overshoot to decay and reach the steady state performance, given by 100% ± 2% of the performance reference. Hence, this is a measure of how fast the system converges towards the desired performance.

4.2 QoS Specification

The QoS specification is given in terms of steady-state and transient-state performance metrics as described in Section 4.1. For each performance metric y we specify its steady-state by a reference y_r, meaning that we want y to stay at y_r at all times, and apply overshoot and settling time to specify its transient-state performance. Quality of data (QoD) is defined 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. We consider the following ways of defining QoS.

QoS Specification A. We define quality of transaction (QoT) in terms of deadline miss percentage of optional subtransactions (M^O). QoT decreases as M^O increases (similarly, QoT increases as M^O decreases). The DBA can specify steady-state and transient-state behavior for M^M, M^O, and MDE. A QoS requirement can be specified as the following: $M_r^M = 1\%$ (i.e. reference M^M), $M_r^O = 10\%$, $MDE_r = 2\%$, $U \geq U_l = 80\%$, $T_s \leq 60s$, and $M_p \leq 30\%$. This gives the following transient-state performance specifications: $M^M \leq M_r^M \times (M_p + 100) = 1.3\%$, $M^O \leq 13\%$ and $MDE \leq 2.6\%$. A lower utilization value is given by U_l, meaning that U must be greater or equal to U_l at all times.

QoS Specification B. We define QoT in terms of average transaction error (ATE). QoT decreases as ATE increases (similarly, QoT increases as ATE decreases). The DBA can specify steady-state and transient-state behavior for ATE and MDE. A QoS requirement can be specified as the following: $ATE_r = 20\%$ (i.e. reference ATE), $MDE_r = 5\%$, $T_s \leq 60s$, and $M_p \leq 30\%$. This gives the following transient-state performance specifications: $ATE \leq ATE_r \times (M_p + 100) = 26\%$, and $MDE \leq MDE_r \times (M_p + 100) = 6.5\%$.

4.3 Feedback Control Scheduling Architecture

We employ feedback control scheduling [10, 11] to manage the quality of the service provided by the RTDB. The goal is to control the performance, defined by a set of controlled variables, such that the controlled variables satisfy a given QoS specification. 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. We choose M^M and M^O as controlled variables when the QoS is specified according to QoS specification A, while ATE is the controlled variable when QoS specification B applies. At each sampling instant, the controlled variable(s) is monitored and fed into the QoS controller, which compares the performance reference(s), i.e. M_r^M and M_r^O, or ATE_r, with the controlled variable(s) to get

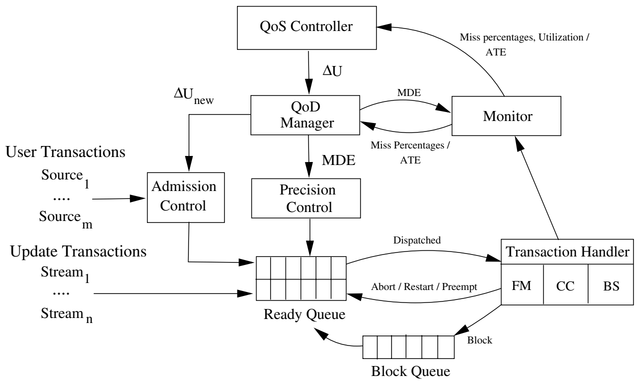

Figure 3. Feedback Control Scheduling Architecture

the current performance error. Based on this 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 returned to the admission controller, which enforces the remaining utilization adjustment.

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 use three different scheduling algorithms as basic schedulers: (i) Earliest Deadline First (EDF), where transactions are processed in the order determined by increasing absolute deadlines (ii) Highest Error First (HEF), where transactions are processed in the order determined by decreasing transaction error, and (iii) 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.[3] 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. $DE_i \leq MDE$.

[3]Note that HEF and HEDF cannot be used in the case when error functions for transactions are not available, as they are error-cognizant and require knowledge of transaction error for each transaction.

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.

We have modeled the controlled system, i.e. the RTDB, according to the analytical approach proposed in [10]. The approach has been adapted such that it supports the imprecise computation model [9]. For deriving and tuning of the model and the feedback controllers, we refer to [2, 3] for details.

4.4 Algorithms for Data and Transaction Quality Management

The algorithms FCS-IC-1 and FCS-IC-2 control QoT by monitoring M^M, M^O, and U, adjusting MDE such that a given QoS specification, according to QoS specification A, is satisfied. Here, we use EDF as a basic scheduler, i.e., transactions are scheduled according to EDF. FCS-HEF and FCS-HEDF are error-cognizant and control QoT by monitoring ATE and adjusting MDE, such that a QoS specification in terms of QoS specification B is satisfied. Furthermore, FCS-HEF and FCS-HEDF are designed to enhance QoS fairness among transactions. We use the same feedback control policy for FCS-HEF and FCS-HEDF, but use different basic schedulers, i.e., FCS-HEF schedules the transactions using HEF and FCS-HEDF schedules the transactions using 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 corresponds 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. 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). Due to space limitation we refer to [2, 3] for the derivation of f.

4.4.1 FCS-IC-1.

FCS-IC-1 employs one utilization and two miss percentage controllers, i.e., one controller to adjust the utilization U according to a lower bound U_l, and two controllers to adjust M^M and M^O. Initially, U is set to a U_l. As long as M^M and M^O are below their references, U is increased by a certain step. As soon as M^M or M^O (or both) are above their references, U is reduced exponentially. This is to prevent a potential deadline miss percentage overshoot due to a too optimistic utilization reference. In addition to this, FCS-IC-1 performs the following.

```
Monitor $M^M(k)$, $M^O(k)$, and $U(k)$
Compute $\Delta U(k)$
**if** ($\Delta U(k) > 0$ and $MDE(k) > 0$) **then**
  Upgrade QoD according to $MDE(k+1) := f(\Delta U(k))$
  Inform AC about the portion of $\Delta U(k)$ not accommodated by QoD upgrade
**else if** ($\Delta U(k) < 0$ and $MDE(k) < MDE_r \times (M_p + 100)$) **then**
  Downgrade QoD according to $MDE(k+1) := f(\Delta U(k))$
  Inform AC about the portion of $\Delta U(k)$ not accommodated by QoD downgrade
**else if** ($\Delta U(k) < 0$ and $MDE(k) = MDE_r \times (M_p + 100)$) **then**
  Reject any incoming transactions
**else**
  Inform the AC of $\Delta U(k)$
**end if**
```

Figure 4. FCS-IC-1

The system monitors the deadline miss percentages and the CPU utilization. At each sampling instant, the CPU utilization adjustment, $\Delta U(k)$, is derived. Based on $\Delta U(k)$ we perform one of the following. If $\Delta U(k)$ is greater than zero, upgrade QoD as much as $\Delta U(k)$ allows. However, when $\Delta U(k)$ is less than zero, degrade QoD according to ΔU, but not greater than the highest allowed MDE (i.e. $MDE_r \times (M_p + 100)$). Degrading the data further would violate the upper limit of MDE given by the QoS specification. In the case when $\Delta U(k)$ is less than zero and MDE equals $MDE_r \times (M_p + 100)$, no QoD adjustment can be issued and, hence, the system has to wait until some of the currently running transactions terminate. An outline of FCS-IC-1 is given in Figure 4.

4.4.2 FCS-IC-2.

In FCS-IC-2, two miss percentage control loops, one for M^M and one for M^O, are used. In the case of FCS-IC-1, the miss percentages may stay lower than their references, since the utilization exponentially decreases every time one of the miss percentages overshoots its reference. Consequently, the specified miss percentage references (i.e. M_r^M and M_r^O) may not be satisfied. In FCS-IC-2, the utilization controller is removed to keep the miss percentages at the specified references.

One of the characteristics of the miss percentage controller is that as long as M^O is below its reference (i.e. $M^O \leq M_r^O$), the controller output ΔU stays positive.[4] Due to the characteristics of f (i.e. $\Delta U(k) > 0 \Rightarrow$

[4]If we have transient oscillations, ΔU, may temporally stay positive (negative) even though the M^O 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, causing a delay before a change to the utilization is requested and has effect.

$MDE(k+1) < MDE(k)$), a positive ΔU is interpreted as a QoD upgrade. Consequently, even if M^O is just below its reference, QoD remains high. We would rather that M^O, which corresponds to QoT, increases and decreases together with QoD given by MDE. For this reason, MDE is set considering both ΔU and M^O. When ΔU is less than zero (i.e. M^O 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 M^O, computed by $M^O_{MA}(k) = \alpha M^O(k) + (1-\alpha) M^O_{MA}(k-1)$, where α ($0 \leq \alpha \leq 1$) is the forgetting factor. Setting α close to 1 results in a fast adaptation, but also captures the high-frequency changes of M^O, whereas setting α close to 0 results in a slow but smooth adaptation. The outline of FCS-IC-2 is given in Figure 5.

```
Monitor $M^M(k)$ and $M^O(k)$
Compute $\Delta U(k)$
**if** ($\Delta U(k) \geq 0$) **then**
  $MDE(k+1) :=$
      $\min(\frac{M^O_{MA}(k)}{M^O_r} MDE_r, MDE_r \times (M_p + 100))$
  **if** ($MDE(k) < MDE(k+1)$) **then**
    Add the utilization gained after QoD degrade to $\Delta U(k)$
  **else**
    Subtract the utilization lost after QoD upgrade from $\Delta U(k)$
  **end if**
  Inform AC of the new $\Delta U(k)$
**else if** ($\Delta U(k) < 0$ and $MDE(k) < MDE_r \times (M_p + 100)$)
**then**
  Downgrade QoD according to $MDE(k+1) := f(\Delta U(k))$
  Inform AC about the portion of $\Delta U(k)$ not accommodated by
  QoD downgrade
**else if** ($\Delta U(k) < 0$ and $MDE(k) = MDE_r \times (M_p + 100)$)
**then**
  Reject any incoming transactions
**else**
  Inform the AC of $\Delta U(k)$
**end if**
```

Figure 5. FCS-IC-2

4.4.3 FCS-HEF and FCS-HEDF

FCS-HEF and FCS-HEDF are extensions to FCS-IC-2, but where QoT is measured in terms of ATE, instead of M^O. Hence, we replace the miss percentage control loops for a single average transaction error control loop. Here, MDE is adjusted based on the control signal ΔU and the moving average of ATE, given by $ATE_{MA}(k) = \alpha ATE(k) + (1-\alpha) ATE_{MA}(k-1)$. Due to space limitation we do not provide full algorithm descriptions for FCS-HEF and FCS-HEDF but refer instead to Figure 5 where M^O_{MA} is replaced with ATE_{MA}.

5 Performance Evaluation

5.1 Experimental Goals

The main objective of the experiments is to determine if 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 by varying a set of parameters:

- 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.
- QoS specifications. It is important that an algorithm can manage different QoS specifications. Here we compare the results of the presented algorithms with regards to different QoS specifications.
- Transaction error functions. The characteristics of the error functions depend on the actual application. For this reason, we evaluate the performance of the algorithms with regard to different transaction sets showing different transaction error characteristics.

5.2 Simulation Setup

The simulated workload consists of update and user transactions, which 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. 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 transaction T_i, and $X_i[t_i]$ is associated with subtransaction t_i (where $t_i \in \{M_i, O_{i,1}, \ldots, O_{i,\#O_i}\}$). We analyze $ATE(k)$, $MDE(k)$, $SDTE$, and CPU Utilization (below referred to as utilization) U.

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 of update transactions (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 $ArrivalTime_i + P_i$.

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 times of mandatory and optional ($EET_i[t_i]$) subtransactions are given by $U : (5ms, 15ms)$. The estimation error $EstErr$ is used to introduce an 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 of a transaction is set to $ArrivalTime_i + EET_i \times SlackFactor$. The slack factor is uniformly distributed according to $U : (20, 40)$. The inter-arrival time is exponentially distributed with the mean inter-arrival time set to $EET_i \times SlackFactor$.

We have considered two different transaction sets having different transaction error characteristics. In the first set, referred to as $TSet1$, 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). In the second set, referred to as $TSet2$, 50% of the transactions have an error order of 0.5, 30% have error order of 1, 15% have error order 2, and 5% have error order 5.

In our experiments we use the following QoS specification: $QoSSpecA = \{M_r^M = 1\%, M_r^O = 10\%, MDE_r = 2.5\%, T_s \leq 60s, M_p \leq 30\%, U_l = 80\%\}$, $QoSSpecB_1 = \{ATE_r = 20\%, MDE_r = 5\%, T_s \leq 60s, M_p \leq 30\%\}$, $QoSSpecB_2 = \{ATE_r = 10\%, MDE_r = 10\%, T_s \leq 60s, M_p \leq 30\%\}$.

5.3 Baselines

To the best of our knowledge, there has been no earlier work on techniques for managing data impreciseness and transaction impreciseness, satisfying QoS or QoD requirements. For this reason, we have developed three baseline algorithms, FCS-EDF, Baseline-1, and Baseline-2, to study the impact of the workload on the system. We compare the behavior of FCS-HEF and FCS-HEDF with FCS-EDF, which is similar to FCS-HEF and FCS-HEDF, but where EDF is used as a basic scheduler. We choose EDF since it is optimal (in minimizing deadline misses) and has well-known behavior. The algorithm outline of Baseline-1 and Baseline-2 is given below. Depending on the given QoS specification, let Υ be either M^O or ATE.

Baseline-1. If Υ (i.e. M^O or ATE) is greater than its reference, the utilization has to be lowered, which is achieved by discarding more update transactions, i.e. increasing MDE. MDE is set according to $MDE(k+1) = \min(\frac{\Upsilon(k)}{\Upsilon_r} MDE_r, MDE_r \times (M_p + 100))$. A simple AC 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. To prevent a potential overshoot, we increase MDE as soon as Υ is greater than zero. If $\Upsilon(k)$ is greater than zero, $MDE(k)$ increases 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 $\Upsilon(k)$ is equal to zero, $MDE(k)$ decreases stepwise until zero is reached (i.e. $MDE(k+1) = \max(MDE(k) - MDE_{step}, 0)$). The same AC as in Baseline-1 is used.

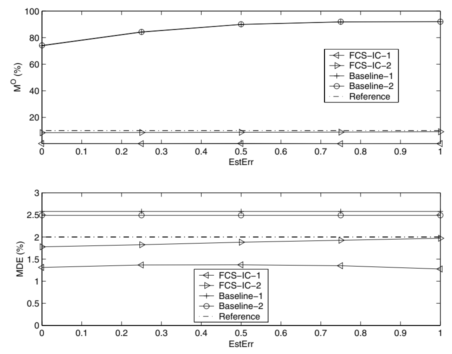

Figure 6. Average Performance: $Load = 200\%$, $QoSSpecA$

5.4 Experiment 1: Results of Varying EstErr

The setup of the experiment is given below, followed by the presentation of the results.

Experimental setup. We apply 200% load and measure M^O, ATE, $SDTE$, MDE, and U. The execution time estimation error is varied according to $EstErr$ = 0.00, 0.25, 0.50, 0.75, and 1.00. For FCS-IC-1 and FCS-IC-2 we use QoS specification $QoSSpecA$, while for FCS-HEF and FCS-HEDF QoS specification $QoSSpecB_1$ and transaction set $TSet1$ holds. Figure 6 shows the performance of FCS-IC-1 and FCS-IC-2, and Figure 7 shows the performance of FCS-HEF, FCS-HEDF, and FCS-EDF. Dash-dotted lines indicate references.

Results. For all algorithms the confidence intervals are within $\pm 3.5\%$ for M^O, ATE, and $SDTE$, and within 0.3% for MDE. For all algorithms and baselines the utilization has been observed to be above 95%. As we can see for Baseline-1 and Baseline-2, the controlled variables (i.e. M^O and ATE) and MDE change for varying $EstErr$. However, FCS-IC-1, FCS-IC-2, FCS-HEF, and FCS-HEDF provide a robust control of the performance as M^O, ATE, and MDE do not change considerably for varying execution time estimation errors. Further, we see that FCS-HEF provides a lower $SDTE$ than the other algorithms. Note that FCS-IC-1 provides a much lower M^O than M_r^O. This is due to the properties of the utilization controller, where

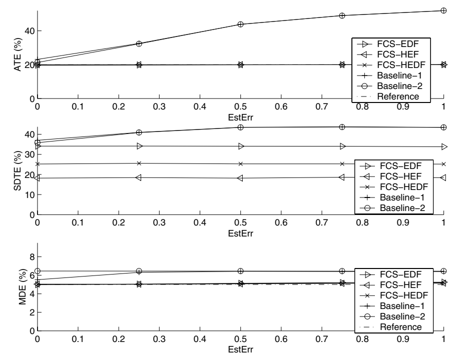

Figure 7. Average Performance: $Load = 200\%$, $QoSSpecB_1$, $TSet1$

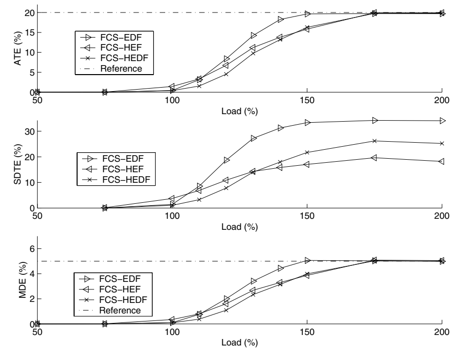

Figure 8. Average performance: $EstErr$ **= 0,** $QoSSpecB_1$, $TSet1$

the utilization decreases exponentially every time M^O overshoots its reference, yielding an overall lower utilization, and consequently, a lower M^O compared to M_r^O. From above we can conclude that FCS-IC-1, FCS-IC-2, FCS-HEF, and FCS-HEDF are insensitive to changes to execution time estimation and, hence, they can easily adapt when accurate run-time estimates are not known.

5.5 Experiment 2: Varying QoS Specification

The setup of the experiment is given below, followed by the presentation of the results.

Experimental setup. We apply loads from 50% to 200% and measure ATE, $SDTE$, and MDE. The execution time estimation error is set to zero (i.e. $EstErr = 0$). Transaction set $TSet1$ is used. Figure 8 shows the performance of FCS-HEF, FCS-HEDF, and FCS-EDF given $QoSSpecB_1$, while Figure 9 shows the performance of the algorithms under $QoSSpecB_2$. Dash-dotted lines indicate references.

Results. For all algorithms, the confidence intervals of ATE and $SDTE$ are within $\pm1.9\%$, while the same figure for MDE is $\pm0.8\%$. As we can see, ATE and MDE grow

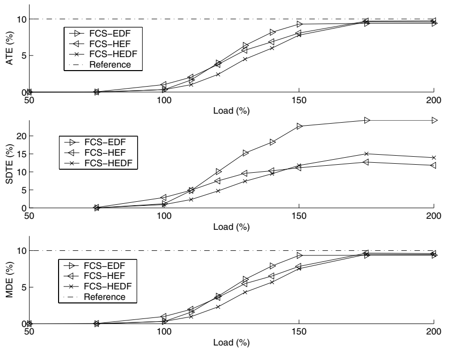

Figure 9. Average performance: $EstErr$ **= 0,**$QoSSpecB_2$, $TSet1$

towards their references as the applied load increases, consequently satisfying the QoS specifications. Thus, we have shown that the proposed algorithms can support different QoS specifications.[5]

It can be observed that the difference in $SDTE$ between FCS-HEF and FCS-HEDF is smaller for $QoSSpecB_2$ than $QoSSpecB_1$. We notice that when ATE is lower than approximately 10%, FCS-HEDF performs better than FCS-HEF with regard to lowering $SDTE$. For $QoSSpecB_2$, the reference ATE_r is set to 10%, resulting in a lower ATE (compared to the ATE generated for $QoSSpecB_1$) and, hence, the difference in $SDTE$ becomes smaller.

5.6 Experiment 3: Effects of Varying Order of Transaction Error Functions

In experiment 2 we evaluate the algorithms using the transaction set $TSet1$. Below we compare the results obtained from experiment 2 using a different set of transactions, $TSet2$. This is to evaluate the performance of FCS-HEF and FCS-HEDF with regard to different sets of transactions having different transaction error characteristics.

Experimental setup. We apply loads from 50% to 200% and measure ATE, $SDTE$, and MDE. The execution time estimation error is set to zero (i.e. $EstErr = 0$). QoS specification $QoSSpecB_1$ and transaction set $TSet2$ are used. Figure 10 shows the performance of FCS-HEF, FCS-HEDF, and FCS-EDF. Dash-dotted lines indicate references.

Results. The confidence intervals of ATE and $SDTE$ for all algorithms are within $\pm1.7\%$, while for MDE the same figure is $\pm0.4\%$. For FCS-EDF, FCS-HEF, and FCS-HEDF we can see that ATE and MDE satisfy the given QoS specification as they are consistent with the references during high applied loads.

Comparing to Figure 8, it can be observed that ATE for

[5]We do not present the results of the Baseline-1 and Baseline-2, since they have showed poor results in earlier experiments.

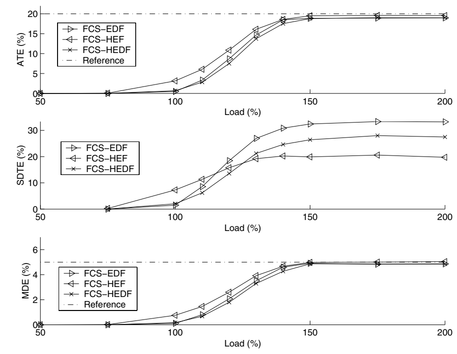

Figure 10. Average performance: $EstErr$ = 0, $QoSSpecB_1$, $TSet2$

FCS-HEF is higher than the other algorithms. Given a set transactions where the error order of each transaction is less than one (i.e. $n < 1$), ATE is lower under EDF scheduling than under HEF scheduling. Consider two transactions, T_1 and T_2, that have error functions with $n = 0.5$ (see Figure 1) and where the difference in deadlines is very small (imagine that the difference is infinitely small). Assume that it takes one time unit to complete an optional subtransaction and that there are 10 time units to the deadline of both transactions, hence, we can only execute and complete 10 subtransactions before their deadline. The question is how to schedule the subtransactions such that the average transaction error is minimized. In the case of HEF, we assign 5 time units to each transaction (i.e. completing 5 optional subtransactions), giving an average transaction error of approximately 0.75. In the case of EDF we assign 10 time units to the transaction with the earlier deadline and zero time units to the second, giving an average error of 0.5. Thus the average error becomes lower under EDF scheduling. Studying $TSet2$, we see that about 50% of the transactions have error functions with orders less than 1 (i.e. $n < 1$), hence, here ATE becomes lower under EDF scheduling than under HEF scheduling.

Further, for loads less than 110% we can see that FCS-HEF produces higher $SDTE$ than the other algorithms. This is due to the high ATE during the same load interval. However, $SDTE$ for FCS-HEF is lower for the other algorithms for loads above 130% as the difference in ATE becomes smaller between the algorithms.

Comparing the performance of the algorithms given in Figure 8 and Figure 10, we conclude that FCS-HEF and FCS-HEDF are robust against varying applied load and varying transaction error characteristics.

5.7 Summary of Results and Discussion

Our experiments show that FCS-IC-1, FCS-IC-2, FCS-HEF, and FCS-HEDF are robust against inaccurate execution time estimations as M^O, ATE, and MDE remain unaffected for varying execution time estimation errors. Also, FCS-HEF and FCS-HEDF can adapt to different QoS specifications and various transaction sets showing different transaction error characteristics. The proposed algorithms outperform the baseline algorithms and FCS-EDF and they can manage a given QoS specifications well. We have carried out other types of experiments [2, 3] for the algorithms given in this paper, where we evaluate the performance of the algorithms with regard to transient-state behavior, i.e., overshoot and settling time. The results of the additional experiments show that the FCS-IC-1, FCS-IC-2, FCS-HEF, and FCS-HEDF are able to handle transient overloads and control the magnitude and settling time of the overshoots. In particular, we have observed that FCS-IC-1 and FCS-HEF efficiently suppress overshoots and, hence, perform better during transient-states than FCS-IC-2, FCS-HEDF, and FCS-EDF.

FCS-IC-1 can manage to provide near zero M^O and it is able to efficiently suppress potential overshoots, but does not fully satisfy the QoS specification with regard to the reference. FCS-IC-2, on the other hand, provides an M^O near its reference, M_r^O. Thus FCS-IC-1 should be applied to RTDBs where overshoots cannot be tolerated, but where consistency between the controlled variables and their references is relaxed, i.e., we do not require the system to produce the desired miss percentages and MDE. The experiments show that FCS-IC-2 is particularly useful when consistency between the controlled variables and their references is emphasized, but where overshoots are accepted.

Moreover, it was showed that FCS-HEF in general provides a lower $SDTE$ than FCS-HEDF and the baselines. We conclude that FCS-HEF should be used in applications where QoS fairness among transactions is important, but also where the performance of the RTDB during transient-state must not violate given QoS specifications.

6 Related Work

There has been several algorithms proposed addressing imprecise scheduling problems [9, 5, 4]. 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). These optimization problems 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.

Lu et al. have presented a feedback control scheduling framework where they propose three algorithms for managing the miss percentage and/or utilization [10]. However 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 [11] using automatic control.

Labrinidis et al. introduced the notion of QoD in the context of web-servers [8]. 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 [7]. In our previous work, we presented a set of algorithms for managing QoS based on feedback control scheduling and imprecise computation [2, 3], where QoS was defined in terms of transaction and data preciseness.

The correctness of answers to databases queries can be traded off to enhance timeliness. The database query processors, APPROXIMATE [13] and CASE-DB [6] are examples of such databases where approximate answers to queries can be produced within certain deadlines.

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. The expressive power of our approach allows a DBA to specify the desired QoS with regard to steady-state and transient-state, capturing the dynamics of a RTDB. FCS-IC-1 and FCS-IC-2 address QoS specifications given in terms of deadline miss percentage of optional subtransactions, while FCS-HEF and FCS-HEDF address specifications based on the notion of transaction error. Given a QoS specification, the four algorithms FCS-IC-1, FCS-IC-2, FCS-HEF, and FCS-HEDF 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. The proposed algorithms outperform the baseline algorithms and FCS-EDF, where transactions are scheduled with EDF and feedback control, and can manage the given QoS specifications well. This is a significant improvement over current techniques for specifying and satisfying QoS requirements.

For our future work, we will model the relationship between data error and transactions error, expressing transaction error in terms of completed optional subtransactions and the data error of the data objects accessed by a transaction. Further, we also intend to extend our work to manage the notion of service differentiation and derived data management.

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. 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 (RTCSA)*, 2003.

[3] M. Amirijoo, J. Hansson, and S. H. Son. Error-driven QoS management in imprecise real-time databases. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2003.

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

[5] 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 (RTCSA)*, pages 307–314. IEEE Computer Press, 2000.

[6] 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.

[7] 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 (ECRTS), June 19-21 2002.

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

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

[10] 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.

[11] 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.

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

[13] 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.