Algorithms for Managing QoS for Real-Time Data Services Using Imprecise Computation*

Mehdi Amirijoo[1], Jörgen Hansson[1], Sang H. Son[2]

[1] Department of Computer Science, Linköping University, Sweden
{meham,jorha}@ida.liu.se
[2] Department of Computer Science, University of Virginia, Virginia, USA
son@cs.virginia.edu

Abstract. Lately the demand for real-time data services has increased in applications where it is desirable to process user requests within their deadlines using fresh data. The real-time data services are usually provided by a real-time database (RTDB). Here, since the workload of the RTDBs cannot be precisely predicted, RTDBs can become overloaded. As a result, deadline misses and freshness violations may occur. To address this problem we propose a QoS-sensitive approach to guarantee a set of requirements on the behavior of RTDBs. Our approach is based on imprecise computation, applied on both data and transactions. We propose two algorithms to dynamically balance the workload and the quality of the data and transactions. Performance evaluations show that our algorithms give a robust and controlled behavior of RTDBs, in terms of transaction and data quality, even for transient overloads and with inaccurate run-time estimates of the transactions.

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 used span 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 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 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. Our scheme is important to applications where timely execution of transactions is emphasized, but where it is not possible to have accurate analysis of arrival patterns and execution times.

* 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. ©RTCSA 2003.

Our approach is based on imprecise computation [9], where it is possible to trade off resource needs for quality of requested service. This has successfully been applied to applications where timeliness is emphasized, e.g., avionics, engine control, image processing [4, 11], networking [12], and approximation algorithms for NP-complete problems [18]. In our work, the notion of impreciseness is applied on both data and transactions, and the goal is to satisfy a QoS specification, in terms of data and transaction impreciseness, giving the desired quality of the provided service. We propose two dynamic balancing algorithms, FCS-IC-1 and FCS-IC-2, to balance the quality of the data and the transactions. Main challenges include unpredictability of workload, in terms of unknown arrival patters and inaccurate execution time estimates, but also effective balancing between transaction and data quality. To solve this issue, we apply feedback control scheduling [10] to provide robustness under these conditions.

The suggested algorithms, FCS-IC-1 and FCS-IC-2, are designed such that the behavior of a RTDB can be controlled, even in the presence of load variation and inaccurate execution time estimates. We have carried out a set of experiments to evaluate the performance of the algorithms. In the simulation studies we have applied a wide range of workload and run-time estimates to model potential unpredictabilities. The studies show that FCS-IC-1 and FCS-IC-2 give a robust and controlled behavior of RTDBs, in terms of transaction and data quality, even for transient overloads and when we have inaccurate run-time estimates of the transactions. This has been shown by comparing the performance against selected baseline algorithms.

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 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 is applied at data object and user transaction level. The data quality increases as the imprecision of the data objects decreases. Similarly, the quality of user transactions increases as the imprecision of the results produced by user transactions decreases. Note that quality of user transactions is related to quality of data. Since user transactions access and read data objects, decreasing the quality of data may lead to a decrease in the quality of user transactions. However, in this work we model user transaction quality and data quality as orthogonal entities and, hence, quality of data and quality of user transactions are considered to be independent. In the future, we will extend our model to capture more advanced relations between user transaction quality and data quality.

In practice, a database administrator (DBA) specifies a desired QoS level in terms of steady-state and transient-state behavior of data and user transaction quality. The goal is to adapt the behavior of the RTDB such that the given QoS specification is satisfied. This is achieved by balancing the workload among update and user transactions. In general, lowering the user transaction workload leads to increased resources available for update transactions, resulting in an increase in data quality. Similarly, lowering the update transaction workload results in an increase in user transaction quality.

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 and if such deviation can be tolerated, arriving updates may be discarded. In order to measure data quality we introduce the notion of *data error*. Let d_i denote an arbitrary data object and T_j a transaction updating d_i. The data error, denoted DE_i, of a data object d_i is defined as a function of the current value (denoted $CurrentValue_i$) of d_i and the update value (denoted $UpdateValue_j$) of the latest arrived update transaction, T_j, i.e. $DE_i = \Phi(CurrentValue_i, UpdateValue_j)$. The data error of a data object gives an indication of how much the value stored in the RTDB deviates from the corresponding value in the real-world, given by the latest arrived update transaction.

The workload of updates is adjusted by manipulating the data error, which is done by considering an upper bound for the deviation between the values of the data objects stored in the RTDB and the corresponding values in the real-world. The upper bound is given by the *maximum data error* (denoted MDE) and is set based on a set of performance variables giving the current state of the RTDB (e.g. quality of user transactions). The data error is adjusted by the following criteria. An update transaction (T_j) is discarded if the data error of the data object (d_i) that is to be updated by T_j is less or equal to MDE (i.e. $DE_i \leq MDE$). In contrast, an update transaction is executed and committed if the corresponding DE_i is greater than MDE.

If MDE increases, more update transactions are discarded as we tolerate greater data error, hence, lower data quality. Similarly, if MDE decreases, fewer update transactions are rejected, resulting in a lower data error, and consequently, greater data quality. The goal of our work is to derive algorithms for adjusting data error, such that the data and the user transaction quality satisfy a given QoS specification. A major issue is how to compute MDE, depending on the user transaction quality.

3 Data and Transaction Model

3.1 Database Model

We consider a firm RTDB model, in which tardy transactions, i.e., transactions that have missed their deadlines, add no value to the system and therefore are aborted. We consider a main memory database model, where there is one CPU as the main processing element.

3.2 Data Model and Data Management

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. $CurrentTime > TimeStamp_i + AVI_i$.

Define the the data error of a data object d_i as,

$$DE_i = 100 \times \frac{|CurrentValue_i - UpdateValue_j|}{|CurrentValue_i|} (\%)$$

where $UpdateValue_j$ is the value of the latest arrived transaction updating d_i.

3.3 Transaction Model

Transactions are classified either as update transactions or user transactions. Update transactions arrive periodically and may only write to temporal data objects (i.e. base data objects). User transactions arrive aperiodically and may read temporal and read/write non-temporal data. The inter-arrival time of user transactions is exponentially distributed.

User and update transactions (T_i) are assumed to be composed of one *mandatory subtransaction* (M_i) and $\#O_i$ *optional subtransactions* (denoted $O_{i,j}$, where $1 \leq j \leq \#O_i$). For the remainder of the paper, 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 is completed when it is completed in a traditional sense. The mandatory subtransaction gives an acceptable result and it is desired to complete the mandatory subtransaction before the transaction deadline. The optional subtransactions depend on the mandatory subtransaction and may be processed if there is enough time or resources available. While it is assumed that all subtransactions (t_i) arrive at the same time as the parent transaction (T_i), the first optional subtransaction (i.e. $O_{i,1}$) becomes ready for execution when the mandatory subtransaction completes. 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,

$$M_i \prec O_{i,1} \prec O_{i,2} \prec \ldots \prec O_{i,\#O_i}.$$

A transaction is completed once its mandatory subtransaction is completed. We set the deadline of all subtransactions to the deadline of the parent transaction. A subtransaction is terminated if it is completed or has missed its deadline. A transaction (T_i) is terminated when its last optional subtransaction (i.e.

$O_{i,\#O_i}$) is completed or one of its subtransactions has missed its deadline. In the latter case, all subtransactions that are not completed are terminated as well.

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

In our transaction model, the estimated average utilization of the transactions is known. However, the average or the actual utilization is not known. Hence, a feature in our model is that it models systems in unpredictable environments where the actual CPU utilization of transactions is time-varying and unknown to the scheduler.

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 a 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-IC-1 and FCS-IC-2.

4.1 Performance Metrics and QoS specification

In our approach, the 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. The metrics are as follows:

– *Deadline Miss Percentage of Mandatory User Subtransactions* (M^M). In a QoS specification the DBA can specify the deadline miss percentage of mandatory subtransactions given by,

$$M^M = 100 \times \frac{\#DeadlineMiss^M}{\#Terminated^M} (\%)$$

where $\#DeadlineMiss^M$ denotes the number of mandatory subtransactions that have missed their deadline, and $\#Terminated^M$ is the number of terminated mandatory subtransactions. We exclusively consider user transactions admitted to the system.

– *Deadline Miss Percentage of Optional User Subtransactions* (M^O). M^O is the percentage of optional subtransactions that have missed their deadline. M^O is defined by,

$$M^O = 100 \times \frac{\#DeadlineMiss^O}{\#Terminated^O} (\%)$$

where $\#DeadlineMiss^O$ denotes the number of optional subtransactions that have missed their deadline, and $\#Terminated^O$ is the number of terminated optional subtransactions. We exclusively consider user transactions admitted to the system.

- *Maximum Data Error* (MDE). This metric gives the maximum data error tolerated for the data objects, as described in Section 2.
- *Overshoot* (M_p) is the worst-case system performance in the transient-state (see Figure 1) and it is given as a percentage. The overshoot is applied to M^O, M^M, and MDE.
- *Settling time* (T_s) is the time for the transient overshoot to decay and reach the steady-state performance (see Figure 1).
- *Utilization* (U). In a QoS specification the DBA can specify a lower bound for the utilization of the system.

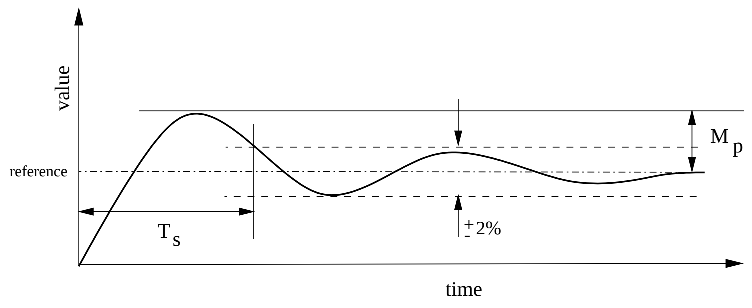

Fig. 1. Definition of settling time (T_s) and overshoot (M_p)

We define *Quality of Data* (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. We measure user transaction quality in terms of deadline miss percentage of optional subtransactions, i.e. M^O. This is feasible in the case when optional subtransactions contribute equally to the final result.

The DBA can specify a set of target levels or references 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), $M_r^O = 10\%$ (i.e. reference M^O), $MDE_r = 2\%$ (i.e. reference MDE), $U \geq 80\%$, $T_s \leq 60s$, and $M_p \leq 30\%$. This gives the following transient performance specifications: $M^M \leq M_r^M \times (M_p + 100) = 1.3\%$, $M^O \leq 13\%$, and $MDE \leq 2.6\%$.

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 2. Admitted transactions are placed in the ready queue. The transaction handler manages the execution of the transactions. At each sampling instant, the controlled variables, miss percentages and utilization, are monitored and fed into the miss percentage and utilization controllers, which compare the performance references, M_r^M, M_r^O, and U_r, with the corresponding controlled variables to get the current performance errors. Based on these the controllers compute 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 control, which enforces the remaining utilization adjustment.

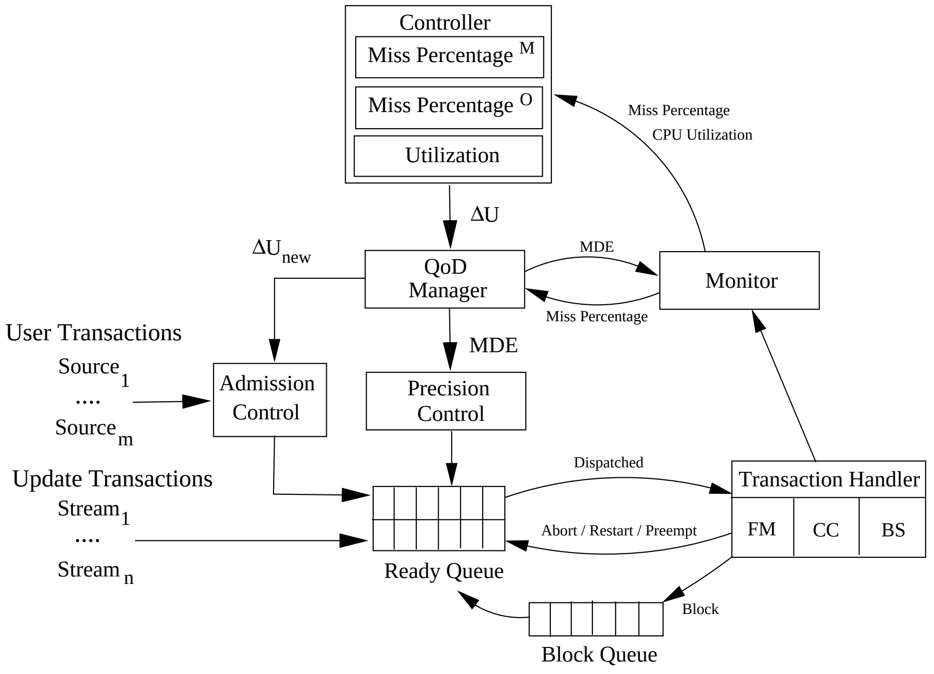

Fig. 2. Feedback control scheduling architecture

The streams ($Stream_i$) generate update transactions, whereas user transactions are generated and submitted by sources ($Source_i$).

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. If a user transaction is accessing a stale data object and the transaction deadline is later than the next update arrival, the transaction is blocked. It is then made ready when the corresponding update commits. However, if the transaction deadline is earlier than next update arrival, the stale data object is used. We use earliest deadline fist (EDF) as a basic scheduler to schedule user transactions. Conceptually, transactions are scheduled in a multi-level queue system. Update transactions and mandatory user subtransactions are placed in the highest priority queue, whereas optional user subtransactions are placed in a lower priority queue. We employ two-phase locking with highest priority (2PL-HP) [1] for concurrency control, where a conflict is resolved by allowing the transaction with the highest priority to lock the data object. 2PL-HP is chosen since it is free from priority inversion and has well-known behavior.

Admission control is applied to control the flow of transactions into the database. When a new transaction is submitted to the database, the admission controller (AC) decides whether or not it can be admitted to the system.

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 [10]. The approach has been adapted such that it supports mandatory and optional subtransactions. For derivation and tuning of the model we refer to [2].

We employ two feedback control scheduling policies, called FC-M and FC-UM [10], to control user transaction quality in the presence of unpredictable workload and inaccurate execution time estimates. Depending on the algorithm used, we apply different feedback control scheduling policies. FCS-IC-1 uses the FC-UM policy, while FCS-IC-2 employs FC-M.

FC-M uses a miss percentage control loop to control the system miss percentage with regards to a reference. Here, separate control loops are used for mandatory and optional subtransactions. Miss percentages of mandatory and optional subtransactions, M^M and M^O, are monitored and controlled with regards to the specified references, i.e. M_r^M and M_r^O.

FC-UM, on the other hand, employs utilization and miss percentage controllers. This has the advantage that the DBA can simply set the utilization reference to a value that causes the desired deadline miss percentage in the nominal case (e.g. based on profiling), and set the miss percentage references (M_r^M and M_r^O) according to the application requirements. For all controllers, the control signal ΔU is computed to achieve the target miss percentage given by the references.

We have extended FC-UM in a way that the reference utilization, denoted U_r, is constantly updated online. The utilization reference is dynamically updated according to a linear increase and exponential decrease scheme. Initially, U_r is set to an initial value. As long as the utilization controller has the control (i.e. the miss percentages are below their references), the utilization reference is increased by a certain step. As soon as one of the miss percentage controllers takes over (i.e. miss percentage above the reference), U_r is reduced exponentially. This is to prevent a potential deadline miss percentage overshoot due to an too optimistic utilization reference. Note that this approach is self-adapting and does not require any knowledge about the underlying workload model.

We have adapted and tuned the feedback controllers, but we do not include these details in this paper due to space limitations. The interested reader is referred to [2].

4.4 Algorithm Specification

We present two algorithms for managing data and user transaction impreciseness. Both are based on adjusting the utilization and the miss percentages using feedback control. The utilization adjustment is enforced partially by adjusting the QoD, which requires setting MDE according to the utilization adjustment (ΔU), as described in Section 4.2. 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.

Given a certain $\Delta U(k)$, we need to set $MDE(k+1)$ such that the utilization (or resources) gained when 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.

FCS-IC-1. FCS-IC-1 (Feedback Control Scheduling Imprecise Computation 1) is based on the extended FC-UM policy (as described in Section 4.3). By using an adaptive scheme where the utilization reference is constantly updated, the utilization yielding the target miss percentage can be approximated. The exponential utilization reduction used with FC-UM decreases the risk for a potential miss percentage overshoot. In addition to this, FCS-IC-1 performs the following.

The system monitors the deadline miss percentages and the CPU utilization. At each sampling period, 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 the data according to ΔU, but not beyond 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 equal to $MDE_r \times (M_p + 100)$, 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 3.

```
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 transaction
**else**
  Inform the AC of $\Delta U(k)$
**end if**
```

Fig. 3. FCS-IC-1

FCS-IC-2. In FCS-IC-2, the FC-M policy is used (as opposed to FCS-IC-1, where FC-UM is applied). In the case of FCS-IC-1, the miss percentages may stay lower than their references, since the utilization is exponentially decreased 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 controllers is that as long as the miss percentages are below their references (i.e. $M^M \leq M_r^M$ and $M^O \leq M_r^O$), the controller output ΔU will be positive.[3] Due to the characteristics of f (i.e. $\Delta U(k) < 0 \Rightarrow MDE(k+1) > MDE(k)$ and $\Delta U(k) > 0 \Rightarrow MDE(k+1) < MDE(k)$), a positive ΔU is interpreted as a QoD upgrade. Consequently, even if the miss percentages are just below their references, QoD remains high. We would rather that the miss percentage of optional subtransactions (M^O), which corresponds to user transaction quality, increases and decreases together with data quality (MDE). For this reason, in FCS-IC-2, the QoD manager is extended

[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.

such that MDE is set not only by considering ΔU, but also according to the current transaction quality given by M^O. When ΔU is less than zero (miss percentage 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. The moving average of M^O is 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 [16]. Setting α close to 1 results in a fast adaptation, but will also capture the high-frequency changes of M^O, whereas setting α close to 0, results in a slow but smooth adaptation. The latter results in the data quality varying with the transaction quality. When M^O_{MA} is relatively low compared to M^O_r, MDE is set to a low value relative to MDE_r. As M^O_{MA} increases, MDE increases but to a maximum value of $MDE_r \times (M_p + 100)$. A further increase violates the QoS specification. The algorithm outline is given in Figure 4.

```
Monitor $M^M(k)$ and $M^O(k)$
Compute $\Delta U(k)$
**if** $(\Delta U(k) \geq 0)$ **then**
  Adjust $MDE(k+1)$ according to
        $MDE(k+1) := \min(\frac{M^O_{MA}(k)}{M^O_r} MDE_r, MDE_r \times (M_p + 100))$
  **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**
  {i.e. $\Delta U(k) < 0$ and $MDE(k) = MDE_r \times (M_p + 100)$}
  Reject any incoming transaction
**end if**
```

Fig. 4. FCS-IC-2

4.5 QoD Management

The preciseness of the data is controlled by the QoD manager which sets $MDE(k)$ depending on the system behavior. When f is used to compute $MDE(k+1)$ based on $\Delta U(k)$ (as in FCS-IC-1 and some cases in FCS-IC-2) 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 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.

An important issue is how to set $MDE(k+1)$ given a certain $\Delta U(k)$. Basically, we want to set $MDE(k+1)$ such that,

$$GU(k+1) = \begin{cases} GU(k) - \Delta U(k), & \Delta U(k) < GU(k), \\ 0, & \Delta U(k) \geq GU(k). \end{cases}$$

This requires that we can predict $GU(k+1)$ induced by $MDE(k+1)$. Note that given $MDE(k+1)$ we can only estimate the corresponding $GU(k+1)$ since our problem is of probabilistic nature. For this mentioned reason, we introduce the notion of *predicted gained utilization*,

$$PGU = g(MDE)$$

where given an MDE, the corresponding GU can be predicted. We derive g based on system profiling, where we measure GU for different MDEs. The function g is then derived by linearizing the relationship between GU and MDE. By taking the inverse of g,

$$MDE = g^{-1}(PGU) = \mu \times PGU \tag{1}$$

we can compute a $MDE(k+1)$ based on a $PGU(k+1)$ where,

$$PGU(k+1) = \begin{cases} GU(k) - \Delta U(k), & \Delta U(k) < GU(k), \\ 0, & \Delta U(k) \geq GU(k). \end{cases} \tag{2}$$

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. Note that on-line profiling also has the advantage of requiring less accurate parameters computed from off-line analysis.

By applying Equation (1) and (2), we compute $MDE(k+1)$ according to the following,

$$\begin{aligned} MDE(k+1) &= f(\Delta U(k)) = \\ &= \min(\mu \times PGU(k+1), MDE_r \times (M_p + 100)). \end{aligned}$$

Since MDE is not allowed to overshoot more than $MDE_r \times (M_p + 100)$, we use a *min* operator to guarantee this.

5 Performance Evaluation

In this section a detailed description of the performed experiments is given. The goal and the background of the experiments are discussed, and finally the results are presented.

5.1 Experimental Goals

The main objective of the experiments is to show whether the presented algorithms can provide guarantees based on a QoS specification. We have for this reason studied and evaluated the behavior of the algorithms according to a set of performance metrics. The performance evaluation is undertaken by a set of simulation experiments, where a set of parameters have been varied. These are:

- 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, which access data and perform virtual arithmetic/logical operations on the data. Update transactions occupy approximately 50% of the workload. Note that the load applied to the database is based on submitted user and update transactions and the tested approaches may reduce the applied load by applying admission control.

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 interval, denoted as vertical lines in the figures. The following QoS specification is used: $M_r^M = 1\%$, $M_r^O = 10\%$, $MDE_r = 2\%$, $U \geq 80\%$, $T_s \leq 60s$, and $M_p \leq 30\%$.

We use the following notation where the metric X_i refers to the transaction T_i, while $X_i[t_i]$ is associated with the subtransaction t_i (where $t_i \in \{M_i, O_{i,1}, \ldots, O_{i,\#O_i}\}$).

Data and Update Transactions. The simulated 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 (AET_i) given by the normal distribution $N : (EET_i, \sqrt{EET_i})$

and a value ($UpdateValue_i$) according to $N : (AV_i, AV_i \times VarFactor)$, where $VarFactor$ is uniformly distributed in (0,1). The deadline is set according to $D_i = ArrivalTime_i + P_i$.

User Transactions. Each $Source_i$ generates a transaction T_i, consisting of one mandatory subtransaction and $\#O_i$ ($1 \leq \#O_i \leq 3$) optional subtransaction(s) ($1 \leq j \leq \#O_i$). $\#O_i$ is uniformly distributed between 1 and 3.

The estimated (average) execution time of the subtransactions ($EET_i[t_i]$) is given by $U : (10ms, 20ms)$. 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, which is given by $N : (AET_i[t_i], \sqrt{AET_i[t_i]})$. The deadline is set according to $D_i = ArrivalTime_i + EET_i \times SlackFactor$. The slack factor is uniformly distributed according to $U : (20, 40)$.

It is assumed that the number of data accesses ($\#DA_i[t_i]$) for each subtransaction is proportional to $EET_i[t_i]$. Hence, longer subtransactions access more data. Upon a transaction generation, $Source_i$ associates an actual number of data accesses given by $N : (\#DA_i[t_i], \sqrt{\#DA_i[t_i]})$ to each subtransaction of T_i. The data set accessed by a transaction is partitioned among the subtransactions such that the partitions are mutually disjoint. However, the data sets accessed by transactions may overlap.

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. Previous work within imprecise computing applied to tasks 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. For this reason, we have developed two baseline algorithms, Baseline-1 and Baseline-2. We use the baselines to study the impact of the workload on the system. Here, we can establish the efficiency of FCS-IC-1 and FCS-IC-2 by comparing the operational envelope of the algorithms, i.e., we can compare the resistance to failure of the algorithms with regard to applied load and/or run-time estimation errors. The baselines are given below.

Baseline-1. The preciseness of the data is adjusted based on the relative miss percentage of optional subtransactions. Conceptually, MDE increases as M^O increases. MDE is set according to $MDE(k+1) = \min(\frac{M^O(k)}{M^O_r} MDE_r, MDE_r \times (M_p + 100))$. A simple AC is applied, where a transaction (T_i) is admitted if the estimated utilization of admitted transactions and T_i is less or equal to 80%.

Baseline-2. In Baseline-1, a significant change in MDE may introduce oscillations in miss percentages. Baseline-2 is similar to Baseline-1, but here MDE is increased and decreased stepwise. The outline of the algorithm is as follows. If $M^O(k)$ is greater than zero, increase $MDE(k)$ by a step (MDE_{step}) 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 $M^O(k)$ is equal to zero, decrease $MDE(k)$ by a step (MDE_{step}) until zero is reached (i.e. $MDE(k+1) = \max(MDE(k) - MDE_{step}, 0)$). The same AC as in Baseline-1 is used here.

5.4 Results of Varying Load

The setup of the experiment is given below, followed by the presentation of the results. Figure 5 shows the average M^O and MDE.

Experimental setup. We measure M^M, M^O, MDE, and U. The experiment setup is as follows. We apply loads from 50% to 200%. The execution time estimation error is set to zero (i.e. $EstErr = 0$).

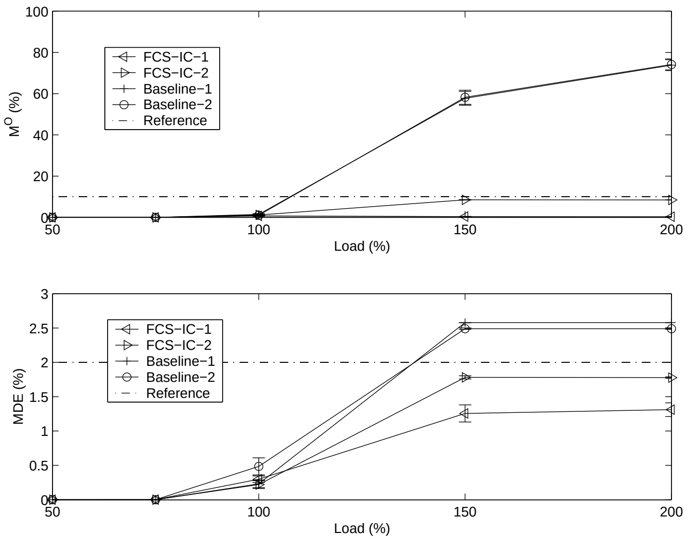

Fig. 5. Average performance for $Load$ = 50, 75, 100, 150, and 200%, $EstErr = 0$

Average Miss Percentage of Mandatory Subtransactions. Miss percentage of mandatory subtransactions (M^M) has been observed to be zero[4] for all four algorithms and, therefore, this has not been included in Figure 5. The specified miss percentage reference (M_r^M), have been set to 1% and this is not satisfied. This is due to higher priority of mandatory subtransactions compared

[4] We have not observed any deadline misses.

to optional subtransactions. According to our investigations, the miss percentage of mandatory subtransactions start increasing when the miss percentage of optional subtransactions is over 90% [2]. Consequently, since the miss percentage of optional subtransactions does not reach 90%, the miss percentage of mandatory subtransactions remains at zero.

Average Miss Percentage of Optional Subtransactions. For Baseline-1 and Baseline-2, the miss percentage of optional subtransactions (M^O) increases as the load increases, violating the reference miss percentage, M_r^O, at loads exceeding 150%. In the case of FCS-IC-1, M^O is near zero at loads 150% and 200%. Even though the miss percentage is low, it does not fully satisfy the QoS specification. This is in line with our earlier discussions regarding the behavior of FCS-IC-1. The low miss percentage is due to the utilization controller since it attempts to reduce potential overshoots by reducing the utilization, which in turn decreases the miss percentage. FCS-IC-2 on the other hand shows a better performance. The average M^O at 150% and 200% is $8.5 \pm 0.1\%$, which is fairly close to M_r^O. In our model tuning of the controlled system, we have assumed worst-case setups and set $EstErr$ to one. In this experiment we have set $EstErr$ to zero, resulting in a certain model error[5]. If $EstErr$ is set to one, we can see that that the average M^O is close to M_r^O. This is shown in Section 5.5.

Average MDE. The average MDE for Baseline-1 and Baseline-2 violates the reference MDE set to 2%. In contrast, in the case of FCS-IC-1, MDE is significantly lower than MDE_r. Since the miss percentages are kept low at all times, they are not likely to overshoot. Consequently, the control signal from the miss percentage controllers is likely to be positive, which is interpreted by the QoD manager as an QoD upgrade and, hence, MDE will not reach the level of MDE_r. This is further explained in Section 5.6, where the transient performance of the algorithms is discussed. FCS-IC-2 provides an average MDE closer to MDE_r, given by $1.78 \pm 0.024\%$ at loads 150% and 200%. However, MDE does not reach MDE_r since MDE is set according to the relative M^O (which does not reach M_r^O).

Average Utilization. For all approaches, the utilization satisfies the QoS specification as it is above the specified 80% for loads between 100-200%, reaching almost 100% at 200% applied load.

5.5 Results of Varying EstErr

The setup of the experiment is given below, followed by the presentation of the results. Figure 6 shows the average M^O and MDE.

Experimental setup. We measure M^M, M^O, MDE, and U. The experiment setup is as follows. 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.

Average Miss Percentage of Mandatory Subtransactions. As in the previous experiment (see Section 5.4), M^M is zero for all approaches and $EstErr$.

[5] By model error we mean the deviation of the model used compared with the actual system being controlled.

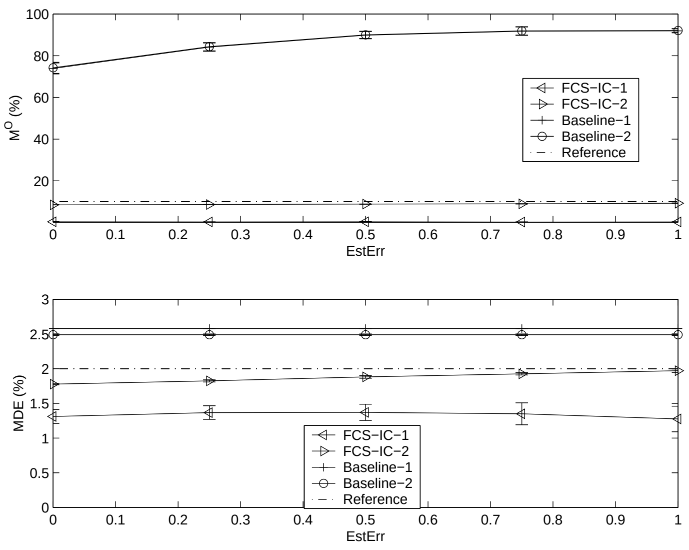

Fig. 6. Average performance for *EstErr* = 0, 0.25, 0.50, 0.75, and 1.0, *Load* = 200%

The discussion regarding average miss percentage of mandatory subtransactions given in Section 5.4 also apply here and are not further discussed.

Average Miss Percentage of Optional Subtransactions. As expected, Baseline-1 and Baseline-2 do not satisfy the QoS specification. In fact, M^O increases as $EstErr$ increases, reaching a value close to 90% for both algorithms. As we can see, FCS-IC-1 and FCS-IC-2 are insensitive against varying $EstErr$. Note that when analyzing FCS-IC-2, we can see that M^O grows towards M^O_r as $EstErr$ increases in value. M^O for $EstErr$ set to zero and $EstErr$ set to one is $8.47 \pm 0.036\%$ and $9.23 \pm 0.17\%$, respectively. This is the result of the discussions given in Section 5.4. As $EstErr$ increases, the model error decreases and, hence, the controlled system becomes closer to the actual model. This gives a more accurate picture of the system and the controllers are therefore able to control the system in a more correct way.

Average MDE. Baseline-1 and Baseline-2 violate the specified MDE reference. For FCS-IC-1 average MDE does not change considerably for different $EstErr$. In the case of FCS-IC-2, average MDE grows towards MDE_r, with increasing $EstErr$. The adjustment of MDE depends on the relative M^O and, hence, the average MDE grows as the average M^O grows, reaching a value of $1.97 \pm 0.03\%$.

5.6 Transient Performance

Studying the average performance is often not enough when dealing with dynamic systems. Therefore we study the transient performance of FCS-IC-1 and

FCS-IC-2 when *Load* is set to 200% and $EstErr$ set to one. Figures 7 and 8 show the transient behavior of FCS-IC-1 and FCS-IC-2. The dash-dotted line indicates maximum overshoot.

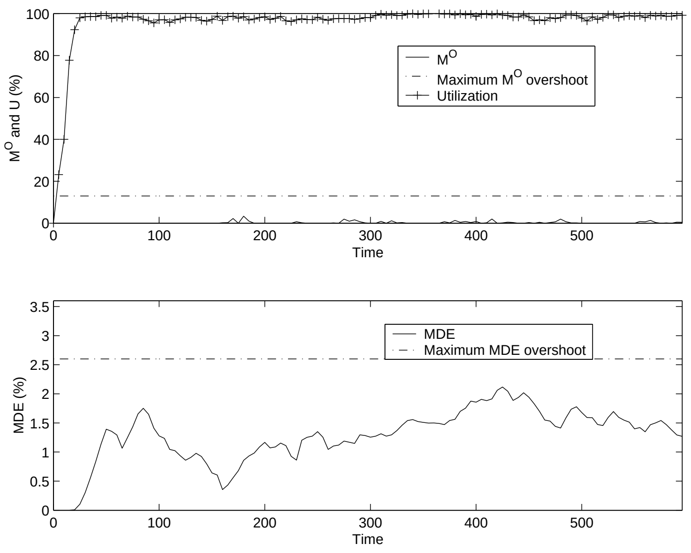

Fig. 7. Transient performance for FCS-IC-1. $EstErr = 1.0$, $Load = 200\%$

Starting with FCS-IC-1, we can note that M^O is kept low at all times. This is expected since the average M^O was shown to be low. The reader may have noticed that MDE is greater than zero in the interval 20-150 where M^O is zero. Since MDE is greater than zero, it is clear that ΔU may become negative during that period. This is due to the behavior of the utilization controller. Initially, the utilization is below the reference (U_r). As the utilization increases and no miss percentage overshoots are observed, U_r increases linearly until a miss percentage is observed (one of the miss percentage controllers takes over) in which case U_r is reduced exponentially. In FCS-IC-1, U_r is only increased if the utilization controller has taken over. Our investigations show that the utilization controller takes over once the utilization overshoots U_r, resulting in a negative ΔU and, hence, U_r being increased too late. Consequently, the negative ΔU leads to an increase in MDE.

FCS-IC-2 shows a more satisfying result as both M^O and MDE increase and decrease together. Both M^O and MDE are kept around M_r^O and MDE_r, respectively. Although the average M^O is close to M_r^O, we can see that M^O often overshoots its reference. The highest M^O has been noted to 25.7%. This is higher than the specified maximum miss percentage of 13% (i.e. $M^O \leq 13\%$). One cause to such overshoot is the various disturbances like data conflicts, resulting

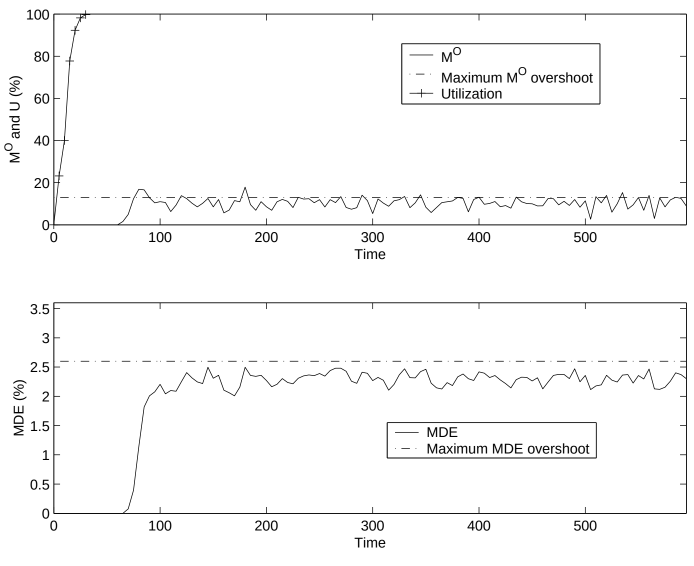

Fig. 8. Transient performance for FCS-IC-2. $EstErr = 1.0$, $Load = 200\%$

in restarts or aborts of transactions. Further, we have set $EstErr$ to one, which yields a higher overshoot than in the case when $EstErr$ is set to zero (i.e. no execution time estimation error). The results of setting $EstErr$ to zero is shown is Figure 9. Here we can see that the variance of miss percentage is much smaller than in the case when $EstErr$ is set to one.

5.7 Summary of Results and Discussions

It has been shown that FCS-IC-1 and FCS-IC-2 are insensitive against load variations and inaccurate execution time estimations. FCS-IC-1 can manage to provide near zero miss percentage for optional subtransactions. We have also seen that FCS-IC-1 can efficiently suppress miss percentage overshoots. However, the performance of FCS-IC-1 does not fully comply with the given QoS specification. Miss percentages and MDE are kept significantly lower than the references, violating the given QoS specifications. This is due to the exponential decrease in utilization every time M^O overshoots its reference.

In FCS-IC-2, M^O and MDE are consistent with their specified references. In addition, we have seen that the data and user transaction quality increase and decrease together. FCS-IC-2, however, produces overshoots higher than the maximum allowed overshoot, as given by the QoS specification.

We conclude that 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

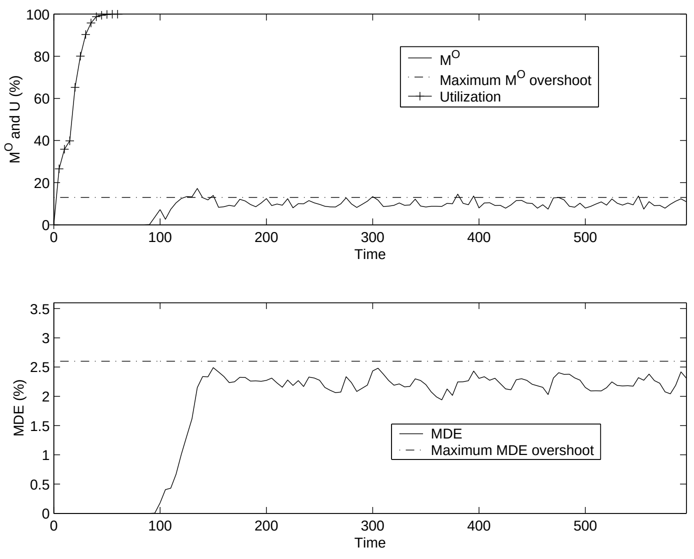

Fig. 9. Transient performance for FCS-IC-2. $EstErr = 0.0$, $Load = 200\%$

particularly useful when consistency between the controlled variables and their references are emphasized, but some overshoots higher than the maximum allowed can be accepted.

6 Related Work

In the past few years, feedback control scheduling has been receiving special attention [10, 13, 3]. Lu et al. have presented a feedback control scheduling framework, where they propose three algorithms for managing the miss percentage and/or utilization [10]. 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 [3]. If the estimated load is found to be greater than a threshold, task periods are enlarged to find the desired load. In contrast to FCS-IC-1 and FCS-IC-2, aperiodic tasks are not considered in their model.

Labrinidis et al. introduced the notion of QoD [8]. Here, web pages are cached at the server and the back-end database continuously updates them. Their proposed update scheduling policy can significantly improve data freshness compared to FIFO scheduling. Kang et al., presented a feedback control scheduling architecture used to control the transaction miss percentage and utilization of a real-time database by dynamically balancing update policies (immediate or on-demand) of a set of data [7].

Liu et al. proposed an imprecise computation model [9]. They presented a set of imprecise scheduling problems associated with imprecise computing and also gave an algorithm for minimizing the total error of a set of tasks. Shih et al. presenting two algorithms for minimizing the maximum error for a schedule that minimizes the total error [15]. Hansson et al. proposed an algorithm, OR-ULD, for minimizing total error and total weighted error [5]. The approaches presented by Liu, Shih, and Hansson 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.

The correctness of answers to databases queries can be traded off to enhance timeliness. Query processors, APPROXIMATE [17] and CASE-DB [6] 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

The need for real-time data services has increased during the last years. As the run-time environment of such applications tends to be dynamic, it is imperative to handle transient overloads efficiently. It has been shown that feedback control scheduling is quite robust to errors in run-time estimates (e.g. changes in workload and estimated execution time). Further, 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 where a database administrator can specify a set of requirements on the database performance and service quality. We present two algorithms, FCS-IC-1 and FCS-IC-2, for managing steady state and transient state performance in terms of data and transaction impreciseness. FCS-IC-1 and FCS-IC-2 give a robust and controlled behavior of RTDBs, in terms of transaction and data quality, even during transient overloads and when we have inaccurate run-time estimates of the transactions.

For our future work, we are establishing techniques for managing data and user transaction impreciseness in a distributed environment and we develop policies for handling derived data. Different approaches to modeling the controlled system will be considered.

Acknowledgment

The authors wish to thank Kyoung-Don Kang at the University of Virginia, Charlottesville, for providing and helping us with the simulator used to perform the experiments.

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. Algorithms for managing QoS for real-time data services using imprecise computation, 2002. Master's Thesis Report LiTH-IDA-Ex-02/90, www.ida.liu.se/~rtslab/master/past.
3. 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.
4. X. Chen and A. M. K. Cheng. An imprecise algorithm for real-time compressed image and video transmission. In *Proceedings of the Sixth International Conference on Computer Communications and Networks*, pages 390–397, 1997.
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*, 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, and J. A. Stankovic. Service differentiation in real-time main memory databases. In *Proceedings of 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing*, April 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. P. Malinski, S. Sandri, and C. Reitas. An imprecision-based image classifier. In *The 10th IEEE International Conference on Fuzzy Systems*, pages 825–828, 2001.
12. V. Millan-Lopez, W. Feng, and J. W. S. Liu. Using the imprecise-computation technique for congestion control on a real-time traffic switching element. In *International Conference on Parallel and Distributed Systems*, pages 202–208, 1994.
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. K. J. Åström and B. Wittenmark. *Adaptive Control*. Addion-Wesley, second edition, 1995.
17. 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.
18. S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.