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

Mehdi Amirijoo[◇], Jörgen Hansson[◇], Sang H. Son[‡]

[◇]Dept. of Computer Science
University of Linköping
Sweden
{meham,jorha}@ida.liu.se

[‡]Dept. of Computer Science
University of Virginia
Virginia, USA
son@cs.virginia.edu

Abstract

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

Keywords: *Quality of Service, Quality of Data, Real-time Databases, Imprecise Computation, Feedback Control Scheduling.*

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 span from low-level

*Correspondence regarding this paper can be sent to Jörgen Hansson, Phone: +46 13 282846, Fax: +46 13 284020. This work was funded, in part by CUGS, CENIIT under contract 01.07, and NSF grant IIS-0208758.

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 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. webservice) it is desirable that the quality of service does not vary significantly from one transaction to another. Here, it is emphasized that the individual QoS needs requested by clients and transactions are enforced, and hence, any deviations from the QoS needs should be uniformly distributed among the clients to ensure QoS fairness.

We employ the notion of imprecise computation [14], where it is possible to trade off resource needs for the quality of requested service. Imprecise computation has successfully been applied to applications where timeliness is emphasized (e.g. [26, 5, 7, 17, 16, 18]). We believe that our approach is important to applications that require timely execution of transactions, but where certain degree of imprecision can be tolerated. For example, in multi-media applications the system load is adapted by providing media of varying quality [3]. Anytime algorithms have shown to be useful for providing approximate results when complete resources are not available [26]. For many applications, such as avionics and engine control, strict consistency between the RTDB and the external environment may be relaxed as small changes to variables in the external environment may be considered insignificant and, hence, some updates may be skipped.

In our previous work we presented two algorithms, FCS-IC-1 and FCS-IC-2, for managing QoS using imprecise computation [4]. In this paper we extend that work by defining a general model of transaction error, and we present two new scheduling algorithms, FCS-HEF and FCS-HEDF, that enhance QoS fairness (i.e. decrease the deviation in quality of service among admitted transactions) and provide an improved transient state performance. Given a QoS specification in terms of data and transaction impreciseness, FCS-HEF and FCS-HEDF adapt the behavior of a RTDB such that the QoS specification is satisfied. 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 impreciseness. To solve this issue we apply feedback control scheduling [15] to provide robustness under these conditions. We say that a system is robust if it has good regulation or adaptation in the face of changes in system parameters (e.g. execution time estimation error and applied load), but also has good disturbance rejection (in RTDBs disturbances occur due to concurrency control, but also arrival and termination of transactions).

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. 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 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. However, it is clear that transaction quality is related to data quality since user transactions may read data that is imprecise, hence, degrading the results computed by user transactions. In the future, we will extend our model to capture more advanced relations between transaction quality and data 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. If such deviation can be tolerated, arriving updates may be discarded during transient overloads. In order to measure data quality we introduce the notion of *data error* (denoted DE_i), which gives an indication of how much the value of a data object d_i stored in the RTDB deviates from the corresponding real-world value, 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 a greater data quality.

Moreover, we introduce the notion of transaction error (denoted TE_i), inherited from the imprecise computation model [14], to measure the quality of a transaction, T_i. Here, the quality of the result given by a transaction depends on the processing time allocated to the transaction. The transaction returns more precise results (i.e. lower TE_i) as it receives more processing time.

The goal of our work is to derive algorithms for adjusting data error, such that the data quality and the

[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.

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 [20]. For temporal data we only consider base data, i.e. data that hold the view of the real-world and are updated by sensors. A base data object d_i is considered temporally inconsistent or stale if the current time is later than the timestamp of d_i followed by the absolute validity interval of d_i (denoted AVI_i), i.e. $CurrentTime > TimeStamp_i + AVI_i$.

For a 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* (M_i) and $\#O_i$ *optional subtransactions* ($O_{i,j}$, $0 \leq j \leq \#O_i$). For the remainder of the paper, we let $t_i \in \{M_i, O_{i,1}, \ldots, O_{i,\#O_i}\}$ denote a subtransaction of T_i.

We use the milestone approach [14] to transaction impreciseness. Thus, we have divided transactions into subtransactions according to milestones. A mandatory subtransaction is completed when it is completed in a traditional sense. The mandatory subtransaction is necessary for an acceptable result and must be computed to completion before the transaction deadline. The optional subtransactions may be processed if there is enough time or resources available. While it is assumed that all subtransactions (t_i) of a transaction (T_i) arrive at the same time, the first optional subtransaction (i.e. $O_{i,1}$) becomes ready for execution when the mandatory subtransaction is completed. In general, an optional subtransaction, $O_{i,j}$, becomes ready for execution when $O_{i,j-1}$ (where $2 \leq j \leq \#O_i$) completes. Hence, there is a precedence relation given by $M_i \prec O_{i,1} \prec O_{i,2} \prec \ldots \prec O_{i,\#O_i}$.

We set the deadline of all subtransactions (t_i) of a transaction to the deadline of the transaction (T_i). 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. $O_{i,\#O_i}$) completes or one of its subtransactions misses its deadline. In the latter case, all subtransactions that are not completed are terminated as well. If a transaction is terminated when its last optional subtransaction is complete, then the corresponding transaction error is zero and we say that the transaction is precisely scheduled. If all transactions are precisely scheduled, we say that the schedule is precise.

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, since updates do not use complex logical

Attribute	Description	Update Transactions	User Transactions
EET_i	estimated (average) execution time of T_i	$EET_i = EET_i[M_i]$	$EET_i = \sum_{\forall t_i} EET_i[t_i]$
AET_i	average execution time of T_i	$AET_i = AET_i[M_i]$	$AET_i = \sum_{\forall t_i} AET_i[t_i]$
AT_i	arrival time of T_i	$AT_i[t_i] = AT_i$	$AT_i[t_i] = AT_i$
TE_i	transaction error of T_i	NA	$TE_i(\#COS_i) = \left(1 - \frac{\#COS_i}{\#O_i}\right)^{n_i}$
P_i	period of T_i	$P_i[t_i] = P_i$	NA
EIT_i	estimated inter-arrival time of T_i	NA	$EIT_i[t_i] = EIT_i$
AIT_i	average inter-arrival time of T_i	NA	$AIT_i[t_i] = AIT_i$
D_i	relative deadline of T_i	$D_i[t_i] = D_i = P_i$	$D_i[t_i] = D_i = AIT_i$
EU_i	estimated utilization of T_i	$EU_i[t_i] = EET_i[t_i]/P_i$ $EU_i = EET_i/P_i$	$EU_i[t_i] = EET_i[t_i]/EIT_i$ $EU_i = EET_i/EIT_i$
AU_i	average utilization of T_i	$AU_i[t_i] = AET_i[t_i]/P_i$ $AU_i = AET_i/P_i$	$AU_i[t_i] = AET_i[t_i]/AIT_i$ $AU_i = AET_i/AIT_i$

Table 1. Transaction Model.

or numerical operations and, hence, normally have a lower execution time than user transactions.

For a user 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 [8]. By choosing n_i we can model and support multiple classes of transactions showing different error characteristics (see Figure 1). For example, it has been shown that anytime algorithms used in AI exhibit error characteristics where n_i is greater than one [26].

A summary of the attributes and characteristics of the transactions and their corresponding subtransactions is given in Table 1. 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 of T_i.

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-HEF and FCS-HEDF.

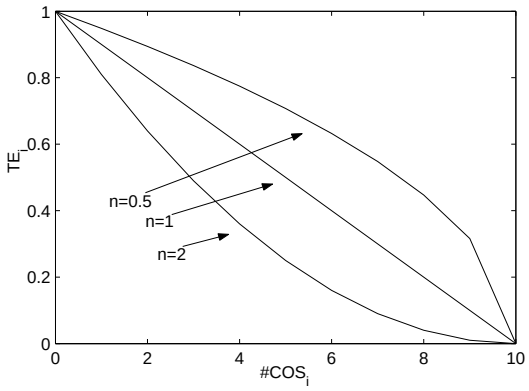

Figure 1. Contribution of $\#COS_i$ to TE_i.

4.1 Performance Metrics and QoS specification

We apply the following steady-state and transient state performance metrics. $Terminated(k)$ denotes the set of terminated transactions during period k.

- Average Transaction Error ($ATE(k)$). A DBA can specify the desired average transaction error of admitted user transactions. The average transaction error during period k,

$$ATE(k) = \frac{\sum_{i \in Terminated(k)} TE_i}{|Terminated(k)|}$$

  gives the preciseness of the results produced by user transactions.[2]

- Maximum Data Error ($MDE(k)$). This metric gives the maximum data error tolerated for the data objects (as described in section 2) during period k.

- Overshoot (M_p) is the worst-case system performance in the transient system state (see Figure 2) and it is usually given in percentage. The overshoot is applied to ATE and MDE.

- Settling time (T_s) is the time for the transient overshoot to decay and reach the steady state performance (see Figure 2), hence, it is a measure of system adaptability, i.e., how fast the system converges towards the desired performance.

- In order to measure QoS fairness among transactions, we introduce Standard Deviation of Transaction Error, $SDTE(k)$, which gives a measure of how much the transaction error of terminated transactions deviates from the average transaction error. $SDTE(k)$ in the k^{th} sampling instant is given by,

$$SDTE(k) = \sqrt{\frac{1}{|Terminated(k)| - 1} \sum_{i \in Terminated(k)} (TE_i - ATE(k))^2}.$$

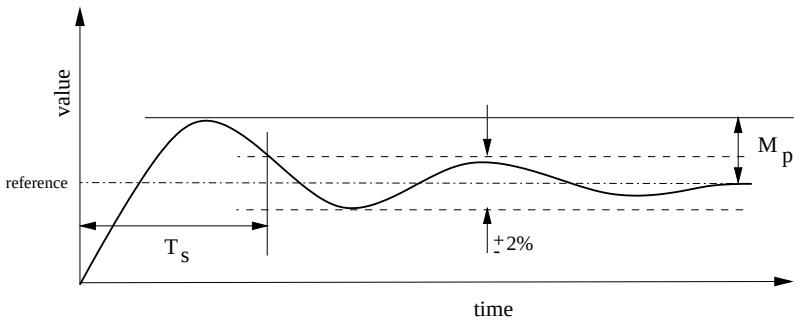

Figure 2. Definition of settling time (T_s) and overshoot (M_p)

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

In our approach, the DBA can explicitly specify the required database QoS, defining the desired behavior of the database. The QoS specification is given in terms of a set of target levels or references for ATE and MDE.[3] A QoS requirement can be specified as the following: $ATE_r = 20\%$ (i.e. reference ATE, meaning that we wish $ATE(k)$ to equal ATE_r for all $k > 0$), $MDE_r = 5\%$ (i.e. reference MDE), $T_s \leq 60s$, and $M_p \leq 30\%$. This gives the following transient performance specifications: $ATE \leq ATE_r \times (M_p + 1) = 26\%$, and $MDE \leq MDE_r \times (M_p + 1) = 6.5\%$.

4.2 Feedback Control Scheduling Architecture

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

The general outline of the feedback control scheduling architecture is given in Figure 3. Admitted transactions are placed in the ready queue. The transaction handler manages the execution of the transactions. At each sampling instant, the controlled variable ATE is monitored and fed into the average transaction error controller, which compares the performance reference, ATE_r, with ATE to get the current performance error. Based on 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 control, which enforces the remaining utilization adjustment.

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

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

[3] $SDTE$ is not included in our QoS specification.

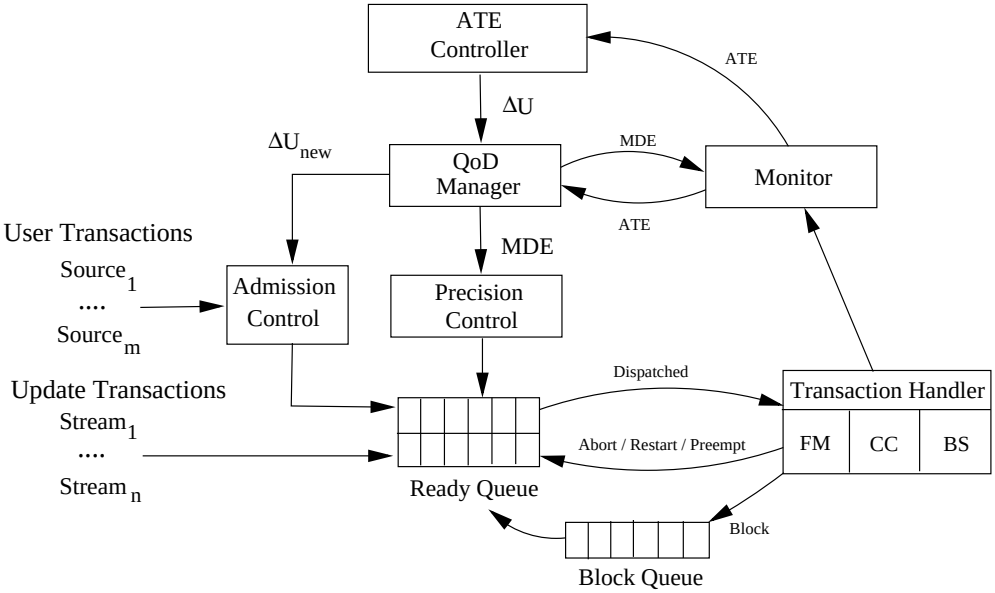

Figure 3. Feedback Control Scheduling Architecture

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 object. We employ two-phase locking with highest priority (2PL-HP) [2] for concurrency control. 2PL-HP is chosen since it is free from priority inversion and has well-known behavior. We consider three different scheduling algorithms as basic schedulers:

Earliest Deadline First (EDF): Transactions are processed in the order determined by their absolute deadlines; the next transaction to run is the one with the earliest deadline.

Highest Error First (HEF): Transactions are processed in the order determined by their transaction error; the next transaction to run is the one with the greatest transaction error.

Highest Error Density First (HEDF): Transactions are scheduled according to their 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, and where the transaction with the highest transaction error density is processed first.

For all three basic schedulers (EDF, HEF, and HEDF) the mandatory subtransactions have higher priority than the optional subtransactions and, hence, scheduled before them.

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 control 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.

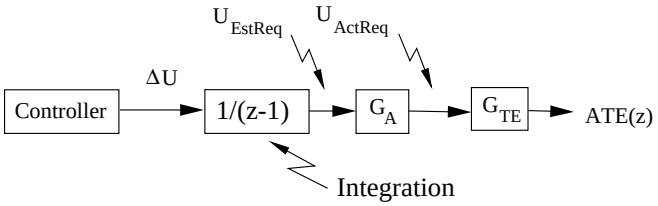

Figure 4. Model of the controlled system

4.3 System Modeling and Model Tuning

4.3.1 System Modeling

We have modeled the controlled system, i.e. RTDB, according to the analytical approach proposed in [15]. The approach has been adapted to support average transaction error.

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. Further, we let $X(k)$ and $X(z)$ denote the time domain and the z-domain respectively of the variable X.

The controlled system includes AC, QoD manager, BS, and monitor. The input to the controlled system is the change in the total estimated requested utilization (ΔU). The output of the system is the controlled variable, i.e. ATE. A block diagram of the controlled system with input and output relations is depicted in Figure 4. The goal is to derive a transfer function describing the relation between the manipulated variable, i.e. $\Delta U(z)$, and the controlled variable, i.e. $ATE(z)$.

Starting from the control input, the estimated requested utilization is the integration of the control input ΔU. Formally, the estimated requested utilization in period $k+1$,

$$U_{EstReq}(k+1) = U_{EstReq}(k) + \Delta U(k) \tag{1}$$

is the summation of the estimated requested utilization ($U_{EstReq}(k)$) and estimated utilization adjustment $\Delta U(k)$ in period k. Note, the estimated utilization adjustment, $\Delta U(k)$ refers to the desired change in utilization during period $k+1$. The z-transform of Equation (1) is given in Equation (2).

$$\frac{U_{EstReq}(z)}{\Delta U(z)} = \frac{1}{z-1} \tag{2}$$

Further, the actual requested utilization ($U_{ActReq}(k)$) may differ from $U_{EstReq}(k)$, due to incomplete knowledge about the controlled system, e.g. unknown execution times of the transactions and data conflicts. Therefore, $U_{ActReq}(k)$ is modeled according to the following,

$$U_{ActReq}(k) = G_A \times U_{EstReq}(k) \tag{3}$$

where G_A, the utilization ratio represents the extent of worst case workload variation in terms of actual total required utilization.

The next step becomes to model the average transaction error based on the actual requested utilization. The relationship between U_{ActReq} and ATE is non-linear due to saturation. We define the *precisely schedulable threshold* (denoted $U_{th}(k)$) as the utilization threshold in the k^{th} period for which the admitted transactions are precisely schedulable. The average transaction error becomes saturated[4] when it is within its saturation zone, given by $U_{ActReq}(k) \leq U_{th}(k)$. When the average transaction error is saturated ATE remains zero despite changes to ΔU and, hence, U_{ActReq}. However, when outside the saturation zones (i.e. $U_{ActReq}(k) > U_{th}(k)$), the average transaction error increases nonlinearly. One way of linearizing the relationship between $U_{ActReq}(k)$ and $ATE(k)$ is to take the derivative at the reference average transaction error (i.e. ATE_r) given by the QoS specification. Hence, we let the average transaction error factor,

$$G_{ATE} = \frac{dATE(k)}{dU_{ActReq}(k)} \tag{4}$$

denote the gradient of the average transaction error at ATE_r.

From Equations (1)-(4), we can derive the following. Under the condition that $U_{ActReq} > U_{th}$, there exists a transfer function,

$$P = \frac{G_A G_{ATE}}{z - 1}$$

from the control input ΔU to average transaction error ATE.

4.3.2 Model Tuning

We tune the model by profiling the system under nominal system operation.[5] In our work, we have used a simulator to evaluate our algorithms. Detailed information about system related issues, i.e., simulator settings, is given in section 5.

We have turned off AC during system profiling. This allows us to derive the relationship between actual load and the average transaction error without the intervention of an AC. Further, in this profiling we assume that mandatory subtransactions have higher priority than optional subtransactions. For the purpose of performance evaluation, we consider two different transactions sets, $TSet1$ and $TSet2$, with different transaction error characteristics (see section 5). Also, the performance of the proposed algorithms are compared with regards to different QoS specifications, $QoSSpec1$ and $QoSSpec2$. We have profiled the system based on the transaction sets and QoS specifications used in the experiments.

The model parameter G_{ATE} is tuned by measuring ATE under loads of 50-200%, by steps of 10%. Graphs illustrating average transaction error for $TSet1$ and $TSet2$ are given in Figure 5 and Figure 6, respectively.[6] We

[4] A controlled variable becomes saturated when it remains unchanged independently of the manipulated variable.

[5] We do not derive system model parameters by considering the worst case set-up as in [15]. By modeling the system under nominal set-up, we believe that we can obtain a more accurate model of the system at run-time.

[6] $TSet1$ and $TSet2$ denote two sets of transactions with different error characteristics.

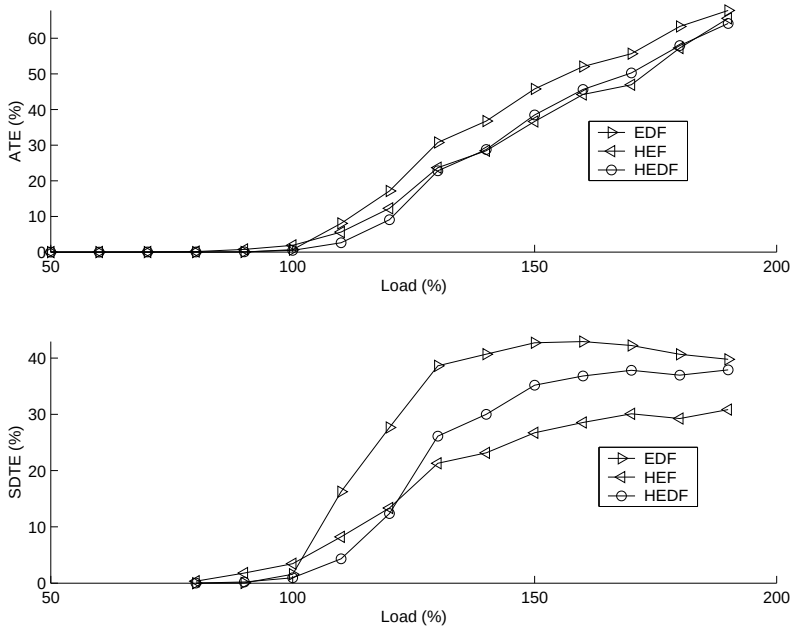

Figure 5. System profiling and model tuning ($TSet1$)

have derived 95% confidence intervals but have not included these in the graphs for clarity of presentation. For both graphs, ATE and $SDTE$ have been observed to be within a 95% confidence interval of $\pm 5\%$.

4.3.3 Model Tuning for QoS Specification $QoSSpec1$

In $QoSSpec1$, the reference transaction error (ATE_r) is set to 20%. From the Figures 5 and 6, we can see that the gradient at 20% average transaction error is approximately 1.3 for all scheduling algorithms and transaction sets ($TSet1$ and $TSet2$). We have set G_{ATE} to 1.3 when tuning the controllers used in experiments where $QoSSpec1$ is assumed.[7]

4.3.4 Model Tuning for QoS Specification $QoSSpec2$

In $QoSSpec2$, the reference transaction error (ATE_r) was set to 10%. From the Figures 5 and 6, we can see that the gradient at 10% average transaction error is approximately 0.9 for all scheduling algorithms and transaction sets, $TSet1$ and $TSet2$. We have set G_{ATE} to 0.9 when tuning the controllers used in experiments where $QoSSpec2$ is assumed.

[7] From the figures 5 and 6, we can see that the gradient of ATE is not exactly 1.3 for all scheduling policies and transaction sets. We can however safely set G_{ATE} to 1.3 when tuning the controllers. This is possible since PI controllers are not sensitive to model errors (deviation between a model and its corresponding real-world system) and, hence, an approximate G_{ATE} suits our purposes.

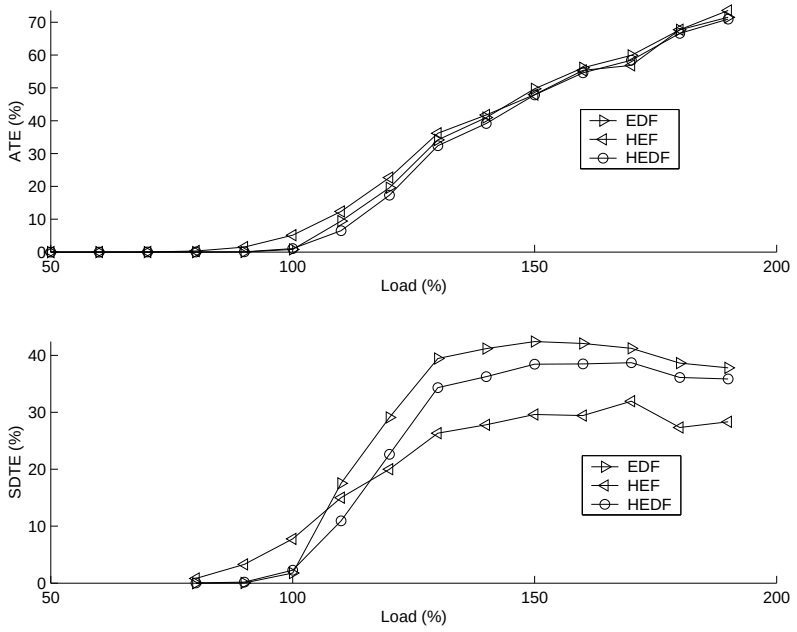

Figure 6. System profiling and model tuning ($TSet2$)

4.3.5 Performance Analysis of Scheduling Algorithms

Below we discuss some of the characteristics that can be observed when scheduling transactions in $TSet1$ and $TSet2$ under different scheduling policies.

Scheduling $TSet1$. Considering the average transaction error, we can notice from Figure 5 that during overloads (i.e. applied load greater than 100%), the scheduling algorithms HEF and HEDF perform better than EDF. As given by Table 5, we can see that about 50% of the transactions adopt error functions where the order of the error functions are greater than one, i.e. $n_i > 1$ (see Figure 1). Hence, for a great portion of the transactions TE decay significantly at the completion of the initial optional subtransactions and less as more subtransactions are completed. During overloads where transactions can not be precisely scheduled, it is more feasible to distribute the resources to transactions such that only the first optional subtransactions are completed, as completing the remaining subtransactions does not decrease TE as much. This is done by HEF since resources are allocated to transactions with the highest TE and given that a a great portion of the transactions have error functions with orders greater than one, the average transaction error is minimized. EDF on the other hand, allocates resources to transactions with no regard to the actual lost in TE when completing a subtransaction.

Turning to the standard deviation of the transaction error, we can see that in Figure 5, the standard deviation increases as the transaction error increases.[8] When the system is underutilized (load less than 100%), $SDTE$ is higher for HEF than the other algorithms. This is due to the high ATE in the underutilized condition. During overloads, $SDTE$ is significantly lower under HEF scheduling compared to EDF scheduling. Under HEF scheduling

[8] We have observed this property also when studying the transient behavior of ATE under EDF, HEF, and HEDF scheduling.

$SDTE$ is less as resources are allocated with regards to TE (resource is given to the transaction with the highest TE), while under EDF scheduling resources are distributed with regards to deadlines. Hence, under EDF scheduling, two transactions with deadlines near each other may receive different amount of CPU power and, hence, they may terminate with a significant deviation in transaction error, while in the same case, under HEF scheduling the resources are divided such that both transactions terminate with transaction errors close to each other.

Scheduling $TSet2$. Studying Table 5, we can observe that about 50% of the transactions have error functions with n less than one. Hence, a great portion of the transaction follow a error scheme where the transaction error is decreased significantly when completing the final subtransactions (as opposite to $TSet1$). From Figure 6, we can see that EDF performs better than HEF for loads 80-140%. This is due to the optimality of EDF, but also the fact that EDF does not allocate resources evenly among the transactions. 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 infinetly 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 when the resources are not evenly distributed.

For $SDTE$ the same argument as in the case of scheduling of $TSet1$ holds and, hence, this is not further discussed.

4.4 Controller Design and Tuning

We employ a policy similar to FC-M [15], to control user transaction quality in the presence of unpredictable workload and inaccurate execution time estimates. FC-M uses a deadline miss percentage control loop to control the system deadline miss percentage with regards to a reference, while in our approach we exchange deadline miss percentage to average transaction error.

4.4.1 Transaction Error Controller

The DBA specifies in the QoS specification the desired database behavior through a set of references, e.g. ATE_r. We employ a PI controller to compute a control signal, ΔU, based on the current performance error, E_{ATE}, which is the difference between the reference and the measured performance (i.e. $E_{ATE} = ATE_r - ATE$). If ATE overshoots its reference, then ΔU becomes negative as a request to lower the estimated requested utilization.

4.4.2 Integrator Antiwindup

For real-time systems, it may happen that a control variable is within its saturation point (e.g. when $U_{ActReq} \leq U_{th}$). When this happens the feedback control loop is broken and the system runs as an open loop because the controlled variable will remain unchanged independently of the controller output. This may have an undesired effect, as the performance error will continue to be integrated, growing to a large value. We say that the integrator "winds up". It is then required that the performance error has the opposite sign for a long time before a controlled variable reaches its respective reference. The same effect happens when a controlled variable is far away from its reference. For example, consider the case when the system is underutilized and ATE is below ATE_r. This results in the average transaction error controller accumulating the performance error. If at a later stage ATE overshoots its reference, then it may take some time for the transaction error controller to respond to the high average transaction error. For this reason we use an *integrator antiwindup* mechanism [22, 23, 9]. We implement the integrator antiwindup by using a *conditional integration* [22] technique, where the integration is switched off when ΔU becomes greater than a threshold ϵ. We employ the following scheme for conditional integration:

- If ΔU is less than ϵ, the integrator (see Equation (5)) of the average transaction error controller is turned on.
- If ΔU is greater than ϵ, the integrator of the average transaction error controller is turned off. This is to prevent the integrator accumulating the performance error.

By employing the above described integrator antiwindup scheme, the controller can react more efficiently to changes in workload and prevent significant overshoots.

4.4.3 Controller Implementation

A digital version of the PI controller is given below. Equations (5) and (6) are equivalent, but equation (6) is more efficient at run-time.

$$\Delta U(k) = K_P(E_{ATE}(k) + K_I \sum_{j=0}^{k} E_{ATE}(j)) \tag{5}$$

$$\Delta U(k) = \Delta U(k-1) + K_P((K_I+1)E_{ATE}(k) - E_{ATE}(k-1)) \tag{6}$$

The z-transform of equation (6) is given by:

$$C(z) = \frac{g(z-r)}{z-1} \qquad g = (K_P(K_I+1)), r = \frac{1}{K_I+1}$$

The parameters that need to be tuned for the transaction error controller are K_P and K_I. The tuning, based on system model and QoS requirements, is given in section 4.4.4.

Controller	K_P	K_I	M_p	T_s	Roots
ATE (assuming $QoSSpec1$)	0.26	0.3	$\approx 25\%$	≈ 50.0 s	0.54
ATE (assuming $QoSSpec2$)	0.40	0.3	$\approx 25\%$	≈ 50.0 s	0.54

Table 2. Controller parameters and time domain performance for QoS specifications $QoSSpec1$ and $QoSSpec2$.

4.4.4 Controller Tuning

Having a tuned model of the system and a QoS specification, the next step is to tune the controllers based on the closed loop of the system. We use $C(z)$ to denote the average transaction error controller.

Given the model of the controlled system, i.e. P, the transfer function of the feedback control loop,

$$H(z) = \frac{C(z)P(z)}{1 + C(z)P(z)} \tag{7}$$

can be established. The z-transform of $ATE(k)$,

$$ATE(z) = \frac{ATE_r z}{z-1} H(z)$$

can be derived by using equation (7).

We have applied the root locus method in Matlab [1] to tune the controller parameters such that the performance specifications as given by Table 6 are satisfied. Assuming the system workload having a worst-case utilization ratio of two (i.e. $G_A = 2$), we set K_P and K_I according to Table 2. We have set the sampling period to 5 seconds.

Below we evaluate the performance of the controllers with regard to overshoot, settling time, stability, and steady-state performance error.

Time Domain Performance: Given a unit step in Matlab, the transient performance of the tuned feedback controllers are established. As we can see from Table 2, the overshoot and the settling time for ATE meet the requirements given in the QoS specifications ($QoSSpec1$ and $QoSSpec2$).

Stability: The closed loop systems are stable since all the poles of H are within the unit circle.

Steady-state Error: Since the closed loop systems are stable, the steady-state error of ATE,

$$\begin{aligned} E_{ss} &= ATE_r - \lim_{z \to 1}(z-1)\frac{ATE_r}{z-1}\frac{1}{1+C(z)P(z)} = \\ &= ATE_r - \lim_{z \to 1}(z-1)\frac{ATE_r}{z-1}\frac{z-1}{1+g(z-r)+G_A G_{ATE}} = 0 \end{aligned}$$

can be computed by applying the final value theorem to the closed loop transfer function.

4.5 Algorithms for Data and Transaction Error Management

We propose two algorithms, FCS-HEF and FCS-HEDF, that adopt the data and transaction management policy used in FCS-IC-2 [4], where the transaction quality is adjusted based on the deadline miss percentage of optional subtransactions. In this work, we have extended FCS-IC-2 such that it manages the transaction quality based on the average transaction error, which captures the transaction quality to a greater extent compared to the miss percentage of optional subtransactions. Moreover, FCS-HEF and FCS-HEDF are transaction error cognizant and are designed to enhance QoS fairness among transactions (i.e. decrease the deviation in QoT among admitted transactions), whereas FCS-IC-2 uses EDF to schedule the transactions and since EDF is only time cognizant, it may not be able to enforce QoS fairness. We use the same basic algorithm for FCS-HEF and FCS-HEDF, but use different basic schedulers, e.g. FCS-HEF schedules the transactions with HEF. We also introduce FCS-EDF, which is similar to FCS-HEF and FCS-HEDF, but where the transactions are scheduled under EDF. We compare the performance of FCS-HEF and FCS-HEDF with FCS-EDF, since the behavior of EDF is well-known. The following scheme for managing the data and transaction quality is common for all three algorithms.

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 4.6.

One of the characteristics of the average transaction error controller is that as long as the average transaction error is below its reference (i.e. $ATE \leq ATE_r$), the controller output ΔU will be positive.[9] Due to the characteristics of f (i.e. $\Delta U(k) > 0 \Rightarrow MDE(k+1) < MDE(k)$), a positive ΔU is interpreted as a QoD upgrade. Consequently, even if the average transaction error is just below its reference, QoD remains high. We would rather that the average transaction error, which corresponds to QoT, increases and decreases together with QoD. For this reason, MDE is set not only by considering ΔU, but also according to the current ATE. When ΔU is less than zero (i.e. ATE overshoot), MDE is set according to f. However, when ΔU is greater or equal to zero, MDE is set according to the moving average of ATE, computed by $ATE_{MA}(k) = \alpha ATE(k) + (1-\alpha)ATE_{MA}(k-1)$,

[9] 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.

where α ($0 \leq \alpha \leq 1$) is the forgetting factor [24]. Setting α close to 1 results in a fast adaptation, but will also capture the high-frequency changes of ATE, whereas setting α close to 0, results in a slow but smooth adaptation. When ATE_{MA} is relatively low compared to ATE_r, MDE is set to a low value relative to MDE_r. As ATE_{MA} increases, MDE is increased but to a maximum value of $MDE_r \times M_p$, since a further increase will violate the given QoS specification. The outline of FCS-HEF, FCS-HEDF, and FCS-EDF is given in Figure 7.

Monitor $ATE(k)$

Compute $\Delta U(k)$

if ($\Delta U(k) \geq 0$) **then**

  Adjust $MDE(k)$ according to

$$MDE(k+1) = \min(\frac{ATE_{MA}(k)}{ATE_r} MDE_r, MDE_r \times M_p)$$

  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 $MDE(k) < MDE_r \times M_p$) **then**

  Downgrade QoD according to $MDE(k+1) = f(\Delta U(k), MDE(k))$

  Inform AC about the portion of $\Delta U(k)$ not accommodated by QoD downgrade

else

  {i.e. $\Delta U(k) < 0$ and $MDE(k) = MDE_r \times M_p$}

  Reject any incoming transactions

end if

Figure 7. Algorithm outline of FCS-HEF, FCS-HEDF, and FCS-EDF

4.6 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,

$$MDE = g^{-1}(PGU) = \mu \times PGU \tag{8}$$

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

$$PGU(k+1) = \begin{cases} GU(k) - \Delta U(k), & \Delta U(k) < GU(k), \\ 0, & \Delta U(k) \geq GU(k). \end{cases} \tag{9}$$

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 (8) and (9), 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.
- 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 are dependent on the actual application. For this reason, we consider the impact of transactions showing different transaction error characteristics. We evaluate the performance of our algorithms with respect to different sets of transactions, representing different applications.

5.2 Simulation Setup

In our simulator, the 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. 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 intervals. The workload model of the update and user transactions are given in Tables 3 and 4, and described as follows.

5.2.1 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

Parameter	Value
$\#DataObjects$	1000
P_i	$U : (100ms, 50s)$
EET_i	$U : (1ms, 8ms)$
AV_i	$U : (0, 100)$
AET_i	$N : (EET_i, \sqrt{EET_i})$
V_i	$N : (AV_i, AV_i \times VarFactor)$
$VarFactor$	$U : (0, 1)$

Table 3. Workload settings for data and update transactions

$Stream_i$ 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 (V_i) according to $N : (AV_i, AV_i \times VarFactor)$, where $VarFactor$ is uniformly distributed in (0,1). The deadline is set to $AT_i + D_i$.

5.2.2 User Transactions

Each $Source_i$ generates a transaction T_i, consisting of one mandatory subtransaction, M_i, and $\#O_i$ ($1 \leq \#O_i \leq 10$) optional subtransaction(s), $O_{i,j}$ ($1 \leq j \leq \#O_i$). $\#O_i$ is uniformly distributed between 1 and 10.

The estimated (average) execution time of the mandatory and the optional ($EET_i[t_i]$) subtransactions is given by $U : (5ms, 15ms)$. The estimation error $EstErr$ is used to introduce execution time estimation error in the average execution time given by $AET_i[t_i] = (1 + EstErr) \times EET_i[t_i]$. Further, upon generation of a transaction, $Source_i$ associates an actual execution time to each subtransaction t_i, which is given by $N : (AET_i[t_i], \sqrt{AET_i[t_i]})$. The deadline is set to $AT_i + EET_i \times SlackFactor$. The slack factor is uniformly distributed according to $U : (20, 40)$.

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.

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 this set, for 50% of the transactions TE decrease significantly when completing the initial optional subtransactions, while TE does not decrease as much upon the completion of the final subtransactions. 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. Here, for 50% of the transactions TE decrease significantly when completing the final subtransactions. The characteristics

Parameter	Value
$\#O_i$	$1 \leq \#O_i \leq 10$,
	$P(\#O_i = 1) = \ldots = P(\#O_i = 10) = \frac{1}{10}$
$EET_i[t_i]$	$U : (5ms, 15ms)$
$AET_i[t_i]$	$AET_i[t_i] = (1 + EstErr) \times EET_i[t_i]$
Act Exec. Time	$N : (AET_i[t_i], \sqrt{AET_i[t_i]})$
D_i	$D_i = AT_i + EET_i \times SlackFactor$
$SlackFactor$	$U : (20, 40)$
Act. Data Accesses	$N : (\#DA_i[t_i], \sqrt{\#DA_i[t_i]})$

Table 4. Workload settings for user transactions

Transaction Set	Distribution
$TSet1$	$P(n_i = 0.5) = P(n_i = 1) = P(n_i = 2) = P(n_i = 5) = 0.25$
$TSet2$	$P(n_i = 0.5) = 0.5$, $P(n_i = 1) = 0.3$, $P(n_i = 2) = 0.15$, $P(n_i = 5) = 0.05$

Table 5. User transaction sets.

of the transaction sets, $TSet1$ and $TSet2$ are summarized in Table 5.

5.3 QoS Specifications

We consider two QoS specifications referred to as *QoSSpec1* and *QoSSpec2* and are listed in Table 6. Note, ATE_r and MDE_r are steady-state specifications, whereas transient specifications for respective metric can be obtained by considering the overshoot M_p (e.g. $ATE \leq ATE_r \times M_p$) and settling time.

5.4 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

QoS Metric	QoSSpec1	QoSSpec2
ATE_r	20%	10 %
MDE_r	5%	10 %
M_p	30%	30 %
T_s	60 s	60 s

Table 6. QoS specifications used for performance evaluation

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, to study the impact of the workload on the system. We also compare the behavior of FCS-HEF and FCS-HEDF with FCS-EDF since EDF is optimal (in minimizing deadline misses) and has well-known behavior. Here, we can establish the efficiency of FCS-HEF and FCS-HEDF 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. If ATE is greater than its reference, the utilization has to be lowered, which is achieved by discarding more update transactions, i.e. increasing MDE. Consequently, the preciseness of the data is adjusted based on the relative average transaction error. MDE is set according to $MDE(k+1) = \min(\frac{ATE(k)}{ATE_r} MDE_r, MDE_r \times M_p)$. 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. In order to prevent a potential overshoot, we increase MDE as soon as ATE is greater than zero. In Baseline-1, a significant change in MDE may introduce oscillations in the average transaction error. This is avoided in Baseline-2 by increasing and decreasing MDE stepwise. If $ATE(k)$ is greater than zero, increase $MDE(k)$ stepwise until $MDE_r \times M_p$ is reached (i.e. $MDE(k+1) = \min(MDE(k) + MDE_{step}, MDE_r \times M_p)$). If $ATE(k)$ is equal to zero, decrease $MDE(k)$ 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.

FCS-EDF. FCS-EDF is similar to FCS-HEF and FCS-HEDF in that it is based on the algorithm given in Figure 7, but where EDF is used as a basic scheduler whereas FCS-HEF and FCS-HEDF use HEF and HEDF, respectively.

5.5 Performance Metrics

We analyze the following metrics in our simulations:

- Average Transaction Error, $ATE(k)$.
- Maximum Data Error, $MDE(k)$.
- Standard Deviation of Transaction Error, $SDTE(k)$.
- CPU Utilization (below referred to as utilization), U.

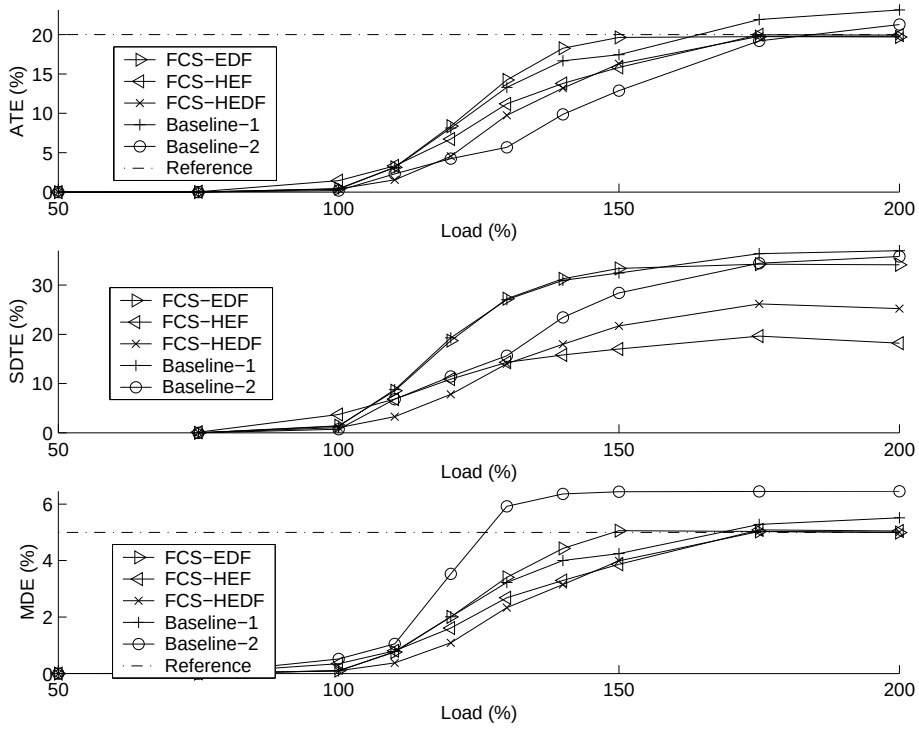

Figure 8. Average Performance: $EstErr = 0$, $QoSSpec1$, and $TSet1$.

5.6 Experiment 1: Results of Varying Load

The setup of the experiment is given below, followed by the presentation of the results. Figure 8 shows ATE, $SDTE$, and MDE.

Experimental setup. We measure ATE, $SDTE$, MDE, U, and number of tardy mandatory subtransactions. The experiment setup is as follows. We apply loads from 50% to 200%. The execution time estimation error is set to zero (i.e. $EstErr = 0$). $QoSSpec1$ and $TSet1$ are assumed.

Tardy Mandatory Subtransactions. We have not observed any deadline misses for any of the four algorithms.

Average Transaction Error. The confidence intervals for all algorithms are within $\pm 2.1\%$. For Baseline-1 and Baseline-2, the average transaction error (ATE) increases as the load increases, violating the reference, ATE_r, at loads exceeding 175%. Baseline-2 produces low ATE for loads 100-175%. This is due to the high MDE, and since many update transactions are discarded more resources can be allocated to user transactions. In the case of FCS-EDF, ATE reaches the reference at 150% applied load. For FCS-HEF and FCS-HEDF, ATE reaches the reference at 175%. All FCS algorithms provide a robust performance since ATE is kept at the specified reference during overloads. It is worth mentioning that ATE for FCS-EDF is higher than the other algorithms. This is in line with our discussions in section 4.3.5, where we concluded that during overloads where transactions cannot be precisely scheduled, it is more feasible to distribute the resources to transactions such that only the first optional

subtransactions are completed, as completing the last subtransactions does not decrease the transaction error as much. This is done under FCS-HEF scheduling since resources are allocated to transactions with the highest error and given that approximately 50% of the transactions have error functions with n_i greater than one, the average transaction error is minimized. FCS-EDF on the other hand, allocates resources to transactions with no regard to the actual decrease in error when completing a subtransaction.

Standard Deviation of Transaction Error. The confidence intervals for all algorithms are within $\pm 1.92\%$. $SDTE$ is higher for FCS-HEF than the other algorithms when the system is underutilized (load less than 100%). This is due to the relatively high ATE for loads less than 100%. For all algorithms, $SDTE$ increases as load and ATE increase. At 200% load, the corresponding $SDTE$ for FCS-EDF, FCS-HEF, and FCS-HEDF is 34.1%, 25.2% and 18.2%, respectively. Consequently, deviation of transaction error is minimized when FCS-HEF is used, which is consistent with our results in section 4.3.5. We would like to point out that there is a lower bound for $SDTE$ as transaction error functions are discrete and, hence, transaction errors decrease in steps. This becomes more evident the fewer subtransactions a transaction has as the steps become greater.

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

Average Utilization. For all approaches, the utilization is above 95% for loads between 100-200%.

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

5.7 Experiment 2: Results of Varying EstErr

The setup of the experiment is given below, followed by the presentation of the results. Figure 9 shows ATE, $SDTE$, and MDE.

Experimental setup. We measure ATE, $SDTE$, MDE, and number of tardy mandatory subtransactions. 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. $QoSSpec1$ and $TSet1$ are assumed.

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

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

Standard Deviation of Transaction Error. The confidence intervals for all algorithms are within $\pm 1.5\%$.

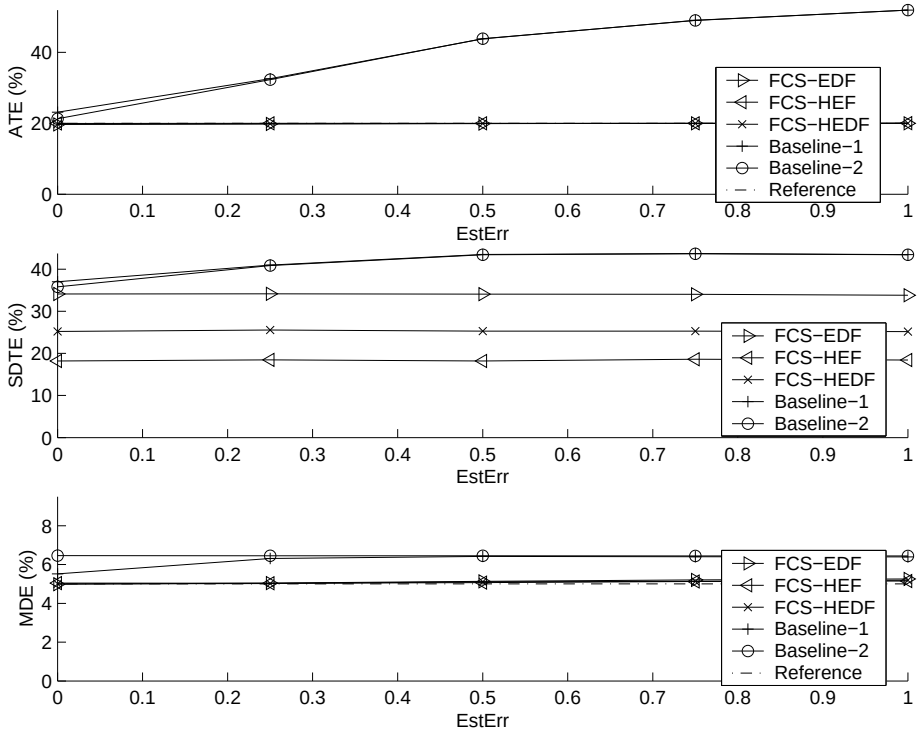

Figure 9. Average Performance: $Load = 200\%$**,** $QoSSpec1$**, and** $TSet1$**.**

For Baseline-1 and Baseline-2 $SDTE$ increases as ATE and $EstErr$ increase, reaching 43.4% at $EstErr$ equals 0.5. In the case of FCS-EDF, FCS-HEF, and FCS-HEDF, $SDTE$ does not change since the corresponding ATE is invariant.

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

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

5.8 Experiment 3: Transient Performance

Studying the average performance is often not enough when dealing with dynamic systems. Therefore we study the transient performance of the proposed algorithms. Figures 10, 11, and 12 show the transient behavior of FCS-EDF, FCS-HEF, and FCS-HEDF. The dash-dotted line indicates maximum overshoot.

Experimental setup. We measure ATE, $SDTE$, and MDE. The experiment setup is as follows. $Load$ is set to 200% and $EstErr$ set to one. $QoSSpec1$ and $TSet1$ are assumed.

Results. The highest overshoot for FCS-EDF has been noted to 35.9% at time 40. For FCS-HEF, the highest overshoot was noted to 30.7% at time 40 (the second highest overshoot was noted to 26.1% at time 555) and finally,

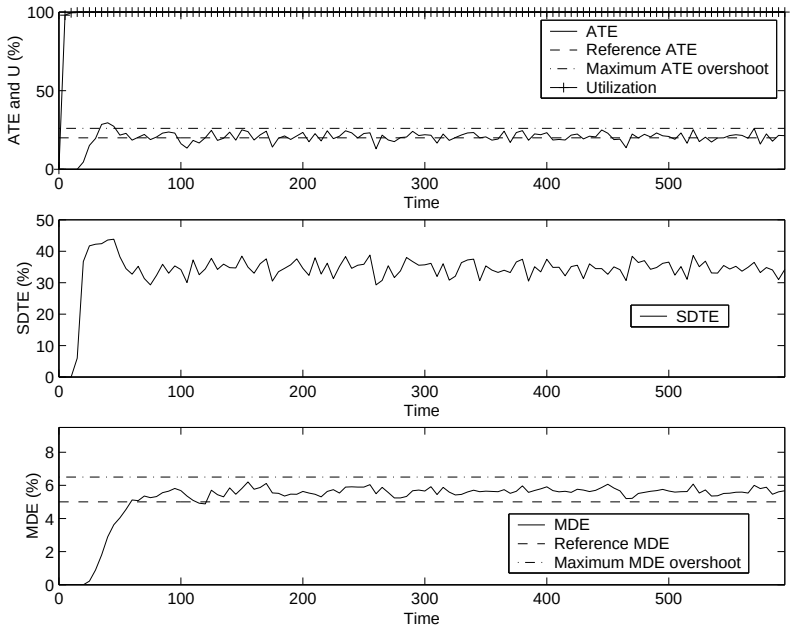

Figure 10. Transient Performance for FCS-EDF. $EstErr = 1.0$**,** $Load = 200\%$**,** $QoSSpec1$**, and** $TSet1$

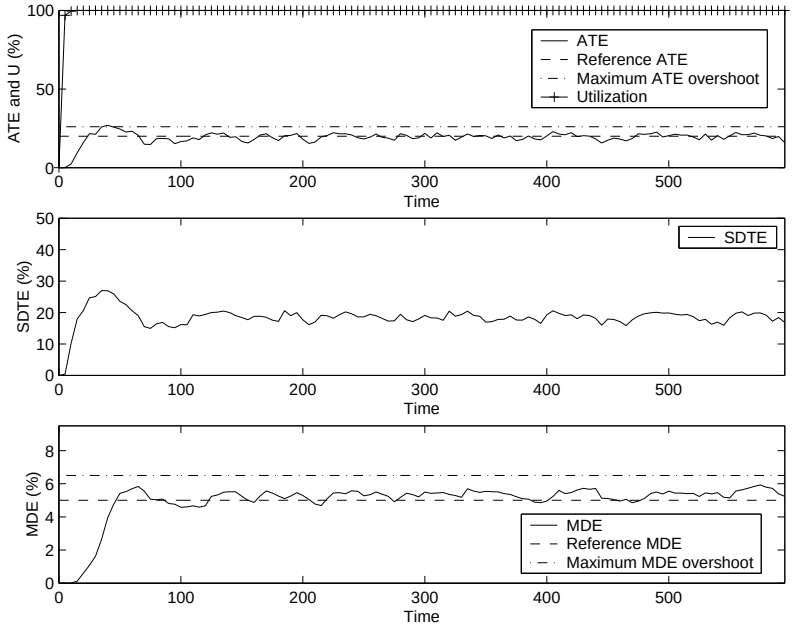

Figure 11. Transient Performance for FCS-HEF. $EstErr = 1.0$**,** $Load = 200\%$**,** $QoSSpec1$**, and** $TSet1$

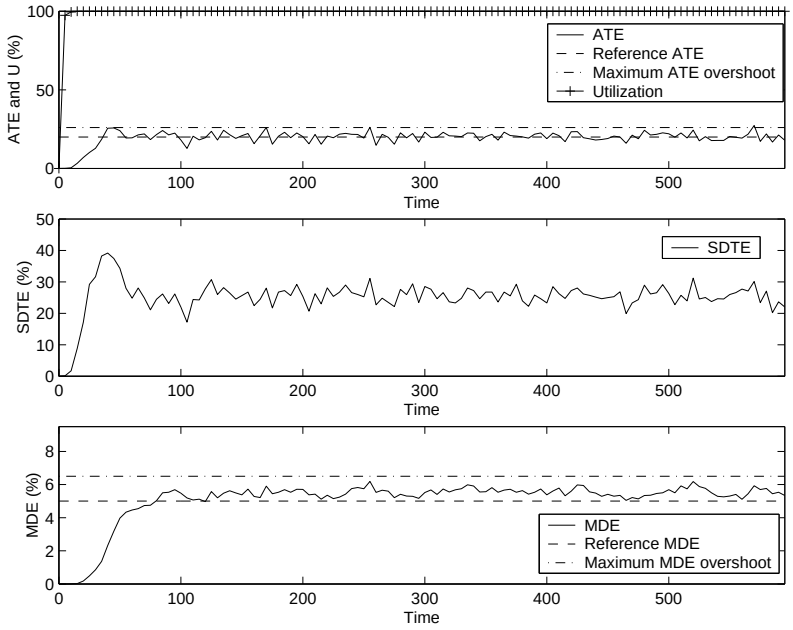

Figure 12. Transient Performance for FCS-HEDF. $EstErr = 1.0$, $Load = 200\%$, $QoSSpec1$, **and** $TSet1$

the highest overshoot for FCS-HEDF was observed to be 32.5% at time 45. As we can see, the algorithms do not satisfy the overshoot requirements given in the QoS specification (i.e. $ATE \leq 26\%$). It is worth mentioning that data conflicts, aborts or restarts of transactions and inaccurate run-time estimates contribute to disturbances in a RTDB, complicating the control of ATE (note that we have set $EstErr$ to one). Consequently, overshoots cannot completely be avoided. For all algorithms the overshoots decay faster than 60s, which are less than the settling time requirement given in the QoS specification.

As we can see from Figures 10-12, FCS-HEF produce less oscillations in ATE and consequently $SDTE$ and MDE. Further, FCS-HEF is less prone to overshoot. Consider the case where the utilization in sampling period $k+1$ differs from the utilization in period k (the utilization is modified due to a change in the number of admitted transactions). In sampling period $k+1$ the following holds. Under HEF scheduling a change in ATE is noticed at an earlier stage than under EDF scheduling. Under EDF scheduling, newly admitted transactions are less likely to have earlier deadlines than the "old" transactions (those admitted in previous periods) and, hence, they are placed further down in the ready queue. This means that an actual change to the controlled variable is not noticed until the newly admitted transactions are executing. However, this delay is removed under HEF scheduling, since newly arrived transactions are more likely to have higher priority than the old transactions (since they have greater TE) and, consequently, a transient overload or underload is observed earlier. Hence, under HEF scheduling the controlled variable is more responsive to changes in the manipulated variable. Now, from feedback control theory we know that delays in systems (low responsiveness of controlled variables) promote oscillations and may even introduce instability [23, 9]. Given this, we can conclude that under EDF scheduling we should observe more

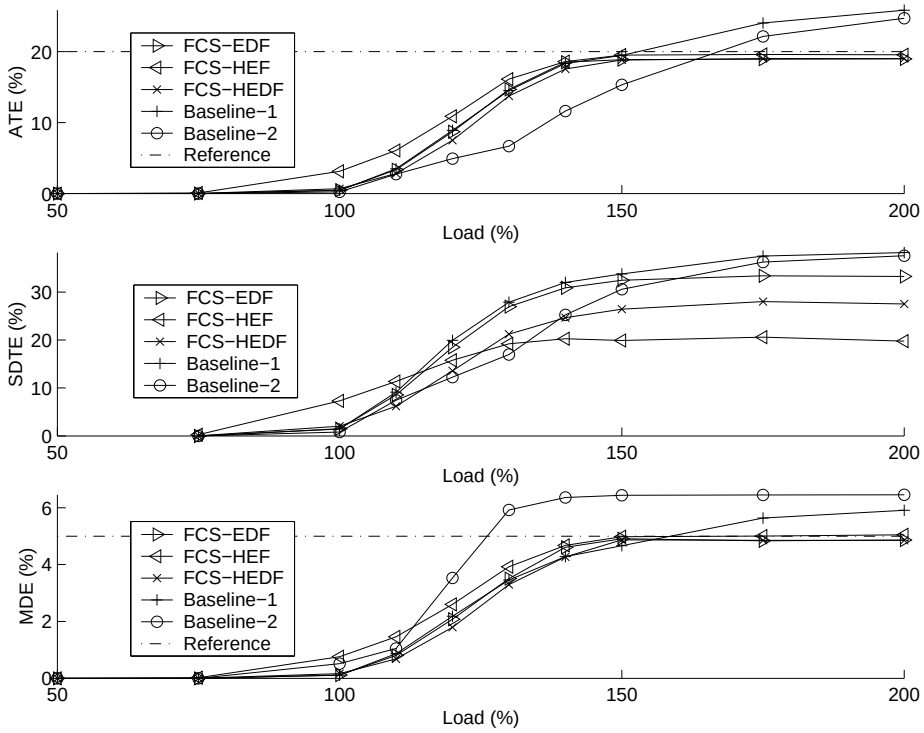

Figure 13. Average performance: $EstErr = 0$, $QoSSpec1$, and $TSet2$.

oscillations in ATE than compared with HEF scheduling.

5.9 Experiment 4: Effects of Varying Order of Transaction Error Functions

In section 5.6 we evaluated the algorithms using the transaction set $TSet1$. Below we compare the results obtained in section 5.6 using a different set of transactions. This is to evaluate the performance of the algorithms with regards to different sets of transactions having different transaction error characteristics. Below, we apply similar load pattern as in the first experiment. Figure 13 shows ATE, $SDTE$, and MDE.

Experimental setup. We assume that a uniform access pattern is used and where we measure ATE, $SDTE$, MDE, and number of tardy mandatory subtransactions. The experiment setup is as follows. We apply loads from 50% to 200%. The execution time estimation error is set to zero (i.e. $EstErr = 0$). $QoSSpec1$ and $TSet2$ are assumed.

Results. No deadline misses have been observed for mandatory subtransactions. The confidence intervals of ATE and $SDTE$ for all algorithms are within $\pm 1.7\%$, while for MDE the same figure is $\pm 0.4\%$. The baselines do not satisfy the given QoS specification, as ATE is greater than the reference for loads above 175%. Further, for Baseline-2, MDE violates the reference at load 130%. However, 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. It can be observed that ATE for FCS-HEF is higher than the other algorithms. Studying Table 5, we can observe that about 50% of the transactions have error functions with n less than one. Hence, a great portion of the transactions follow an error scheme where the transaction error is decreased significantly when executing and completing the final subtransactions (as opposite to $TSet1$). In section 4.3.5 we concluded that EDF produces a lower ATE compared to HEF in the case when $TSet2$ is assumed. This is due to the optimality of EDF, but also the fact that EDF does not allocate resources evenly among the transactions. This result is consistent with our observations in this experiment.

Further, we can see that FCS-HEF produces higher $SDTE$ than the other algorithms for loads less than 110%. This is due to the high ATE during the same load interval. However, $SDTE$ is lower for the other algorithms for loads above 130%. $SDTE$ for FCS-EDF, FCS-HEF, and FCS-HEDF at 200% applied load is 33.2%, 27.5% and 19.8%.

As in Experiment 1, we conclude that FCS-HEF, FCS-HEDF, and FCS-EDF are robust against varying applied load. Furthermore, FCS-HEF increases QoS fairness independent of the underlying transaction set characteristics.

5.10 Experiment 5: Varying QoS Specification

We also compare the performance of our approaches considering other QoS specifications. The idea is to see how the algorithms can adapt to different QoS specifications. Below, we apply similar load pattern as in the first experiment. Figure 14 shows ATE, $SDTE$, and MDE.

Experimental setup. We measure ATE, $SDTE$, MDE, and number of tardy mandatory subtransactions. The experiment setup is as follows. We apply loads from 50% to 200%. The execution time estimation error is set to zero (i.e. $EstErr = 0$). $QoSSpec2$ and $TSet1$ are assumed.

Results. We do not present the results of the baselines, since they have showed poor results in earlier experiments. No deadline misses have been observed for mandatory subtransactions. The confidence intervals of ATE and MDE for all algorithms are within $\pm 0.8\%$, while for $SDTE$ the same figure is $\pm 1.4\%$. In $QoSSpec2$ we set ATE_r and MDE_r to 10%. As we can see, ATE and MDE grow towards the references as the applied load increases. ATE for FCS-EDF, FCS-HEF, and FCS-HEDF are 9.4%, 9.7%, and 9.6%, respectively.

We have shown that the proposed algorithms can support different QoS specifications.

5.11 Summary of Results and Discussions

Our experiments show that the algorithms FCS-HEF and FCS-HEDF are robust against load variations and inaccurate execution time estimations as ATE and MDE have been consistent with their references for varying load and varying execution time estimation error. Also, FCS-HEF and FCS-HEDF can adapt to other types of transaction sets showing different transaction error characteristics and manage various QoS specifications. The

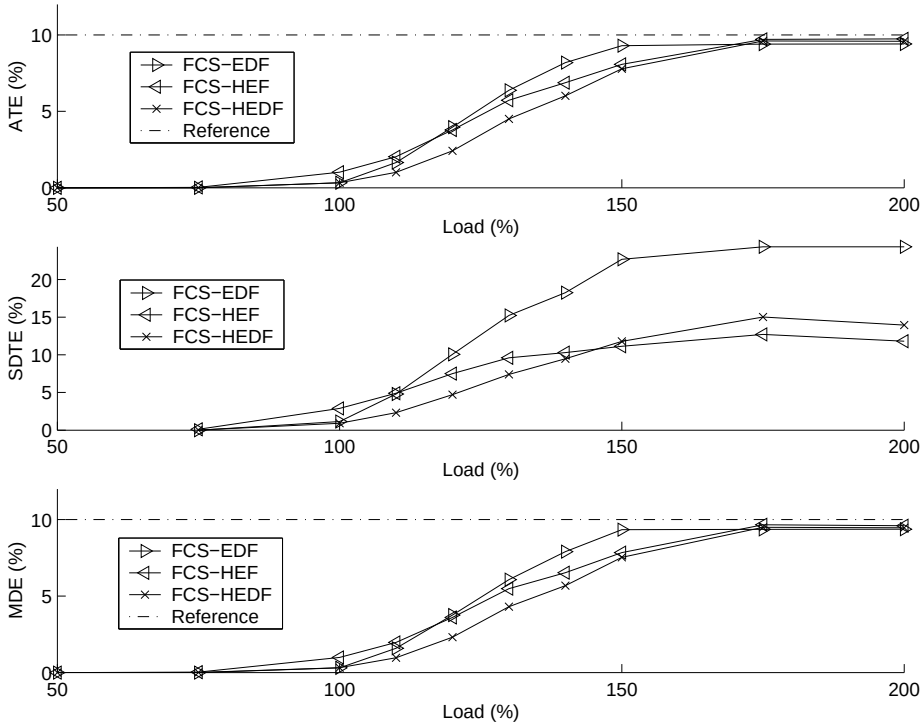

Figure 14. Average performance: $EstErr$ = 0, $TSet1$

proposed algorithms outperform the baseline algorithms and FCS-EDF and can manage the given QoS specifications well.

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

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

6 Related Work

Liu et al. proposed an imprecise computation model [14]. 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. presented two algorithms for minimizing the maximum error for a schedule that minimizes the total

error [21]. Hansson et al. proposed an algorithm, OR-ULD, for minimizing total error and total weighted error [10]. However, 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.

Feedback control scheduling has been receiving special attention in the past few years [15, 19, 6]. Lu et al. have presented a feedback control scheduling framework where they propose three algorithms for managing the miss percentage and/or utilization [15]. In comparison to the proposed approaches here, they do not address the problem of maximizing QoS fairness among admitted tasks. In the work by Parekh et al., the length of a queue of remote procedure calls (RPCs) arriving at a server is controlled [19]. Changing the periodicity of a set of tasks in response to load variations has been suggested in [6]. In contrast to FCS-HEF and FCS-HEDF, aperiodic tasks are not considered in their model.

Labrinidis et al. introduced the notion of QoD [13]. In their work, 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. In the work by Kang et al., a feedback control scheduling architecture is used to control the miss percentage and utilization by dynamically balancing update policies (immediate or on-demand) of a set of data [12]. In our previous work, we presented two algorithms for managing QoS based on feedback control scheduling and imprecise computation [4], where QoS was defined in terms of miss percentage of optional subtransactions.

The correctness of answers to databases queries can be traded off to enhance timeliness. Query processors, APPROXIMATE [25] and CASE-DB [11] are examples of such databases where approximate answers to queries can be produced within certain deadlines. However, in both approaches, impreciseness has been applied to only transactions and, hence, data impreciseness has not been addressed. Further, they have not addressed the notion of QoS. In our work, we have introduced impreciseness at data object level and considered QoS in terms of transactions and data impreciseness.

7 Conclusions and Future Work

In this paper we have argued for the need of increased adaptability of applications providing real-time data services. To address this problem we have proposed a QoS-sensitive approach based on feedback control scheduling and imprecise computation applied on transactions and data objects. Imprecise computation techniques have shown to be useful in many areas where timely processing of tasks or services is emphasized, and 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 have developed two algorithms FCS-HEF and FCS-HEDF that based on feedback control scheduling 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 and can manage the given QoS specifications well.

For our future work, we will model the relationship between data error and transaction error, expressing transaction error in terms of completed optional subtransactions and the data error of the data objects accessed by a transaction. We will also establish techniques for managing data and user transaction error in a distributed environment and we will develop policies for handling derived data. Different approaches to modeling the controlled system will also be considered.

References

[1] *Control Systems Toolbox User's Guide*. The Mathworks Inc., 1996.

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

[3] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

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

[5] L. Budin, D. Jakobovic, and M. Golub. Genetic algorithms in real-time imprecise computing. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, pages 84–89, 1999.

[6] 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.

[7] 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.

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

[9] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Addison-Wesley, third edition, 1994.

[10] 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.

[11] 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.

[12] K. Kang, S. H. Son, and J. A. Stankovic. Service differentiation in real-time main memory databases. 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing, April 2002.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] D. J. Musliner, E. H. Durfee, and K. G. Shin. Any-dimension algorithms. In *Proceedings of the Workshop on Real-Time Operating Systems and Software*, pages 78–81, 1992.

[19] 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.

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

[21] 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.

[22] K. J. Åström and T. Hägglund. *PID Controllers: Theory, Design and Tuning*. Instrument Society of America, second edition, 1995.

[23] K. J. Åström and T. Hägglund. *Computer Controlled Systems*. Prentice Hall, third edition, 1997.

[24] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addion-Wesley, second edition, 1995.

[25] 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.

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