Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations

Mehdi Amirijoo, Jörgen Hansson, *Member*, *IEEE*, and Sang Hyuk Son, *Senior Member*, *IEEE*

Abstract—Real-time applications such as e-commerce, flight control, chemical and nuclear control, and telecommunication are becoming increasingly sophisticated in their data needs, resulting in greater demands for real-time data services that are provided by real-time databases. Since the workload of real-time databases cannot be precisely predicted, they can become overloaded and thereby cause temporal violations, resulting in damage or even a catastrophe. Imprecise computation techniques address this problem and allow graceful degradation during overloads. In this paper, we present a framework for QoS specification and management consisting of a model for expressing QoS requirements, an architecture based on feedback control scheduling, and a set of algorithms implementing different policies and behaviors. Our approach gives a robust and controlled behavior of real-time databases, even for transient overloads and with inaccurate runtime estimates of the transactions. Further, performance experiments show that the proposed algorithms outperform a set of baseline algorithms that uses feedback control.

Index Terms—Real-time and embedded systems, real-time data services, imprecise computation, feedback control, modeling techniques.

1 Introduction

Lately, the demand for real-time data services has increased in a number of applications such as manufacturing, Web-servers, and e-commerce. Further, they are becoming increasingly sophisticated in their real-time data needs [1], [2]. The data normally 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 nonuniform access patterns, the workload of real-time databases (RTDB) cannot be precisely predicted and, hence, the RTDBs can become overloaded. As a result, uncontrolled deadline misses and freshness violations may occur during the transient overloads. To provide reliable service quality, we propose a quality of service (QoS) sensitive approach that guarantees a set of requirements on the performance of the database, even in the presence of unpredictable workloads. Further, for some applications (e.g., Web service), it is desirable that the QoS does not vary significantly from one transaction to another. Here, it is emphasized that the individual QoS needs requested by transactions are enforced and, hence, any deviations from the QoS needs should be uniformly distributed among the clients to ensure QoS fairness.

Imprecise computation techniques [3] have been introduced to allow flexibility in operation and to provide means for achieving graceful degradation during transient overloads. These techniques make it possible to trade off resource needs for the quality of a requested service. Imprecise computation has been successfully applied to applications where timeliness is emphasized, but where a certain degree of imprecision can be tolerated [4], [5], [6], [7]. In our approach, we employ the notion of imprecise computation on transactions as well as data, i.e., we allow data objects to deviate, to a certain degree, from their corresponding values in the external environment. This combined approach of imprecise computation presents a greater challenge, but gives better efficiency in managing QoS and overload management.

In this paper, we present a framework for specification and management of QoS in imprecise RTDBs. The contributions of this paper are

1. a model for expressing QoS requirements,
2. an architecture based on feedback control to satisfy a given QoS specification,
3. a new scheduling algorithm that enhances QoS fairness, and
4. a model of the controlled system that is used to synthesize feedback controllers.

To the best of our knowledge, this is the first paper on QoS management of RTDBs using imprecise computations and feedback control.

Starting with the QoS specification, the expressive power of our QoS specification model allows a database operator

- *M. Amirijoo is with the Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden. E-mail: meham@ida.liu.se.*
- *J. Hansson is with the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890. E-mail: hansson@sei.cmu.edu.*
- *S.H. Son is with the Department of Computer Science, School of Engineering and Applied Science, University of Virginia, 151 Engineer's Way, PO Box 400740, Charlottesville, VA 22904-4740. E-mail: son@virginia.edu.*

Manuscript received 2 Apr. 2004; revised 29 Apr. 2005; accepted 6 July 2005; published online 20 Jan. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0112-0404.

to specify not only the desired steady-state performance, representing the nominal system operation, but also the transient-state performance describing the worst-case system performance and system adaptability in the face of unexpected failures or load variation. Continuing with the second contribution, we notice that the main challenge with managing QoS such that the given specification is satisfied is the unpredictability of workload in terms of unknown arrival patterns and inaccurate execution time estimates. Traditional approaches for providing performance guarantees [8] rely on known worst-case conditions, e.g., worst-case execution times and worst-case arrival patterns of tasks; this knowledge is often lacking for systems operating in highly unpredictable environments. Using feedback control has been shown to be very effective for a large class of real-time systems that exhibit unpredictable workload [9], [10], [11], [12], [13]. Therefore, to provide QoS guarantees without a priori knowledge of the workload, we apply feedback control, where the performance of the RTDB is continuously monitored and compared to the desired performance as given by the QoS specification.

To tune feedback controllers that are efficient in managing the performance of real-time systems, it is necessary to have a model that accurately describes the behavior of the controlled system [14]. As the fourth contribution, we present a novel model that results in a feedback loop with a significant improvement in QoS adaptation compared to the performance achieved using a previously presented model [11]. This result aids a system operator in configuring RTDBs to be highly reactive to changes in applied load and execution time estimation errors, resulting in increased performance reliability and enhanced QoS adaptation. Finally, we present a set of experimental results where we evaluate the performance of the proposed algorithms. Our studies show that the presented algorithms ensure robust and insensitive behavior, even in the presence of transient overloads. An equally important feature of this set of algorithms is their ability to adapt to various workloads and tolerate inaccurate estimates of execution times still conforming to a given QoS specification.

This paper is organized as follows: The detailed 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 present the related work, followed by Section 7, where conclusions and future work are discussed.

2 Problem Formulation

In our model, data objects in an RTDB are updated by update transactions, representing sensor values, while user transactions represent user requests, e.g., complex read-write operations. As mentioned previously, the notion of imprecision may be applied at data object and/or user transaction level. Starting with data imprecision, we observe that, although a real-time database models an external environment that changes continuously, the values of data objects that are slightly different in age or in precision can be used as consistent read data for user transactions. This is due to the fact that data objects cannot, in general, be updated continually to perfectly track the dynamics of the real world. The time it takes to update a data object alone introduces a time delay, which means that the value of the data object cannot be the same as the corresponding real-world value at all times. Hence, for a data object stored in an RTDB and representing a real-world variable, we can allow a certain degree of deviation compared to the real-world value. We can then discard an update transaction that holds a value sufficiently close to the stored value in the RTDB. The more the values of the data objects in the database deviate from the external environment, as given by the values of the update transactions, the more imprecise the data objects are. To measure data imprecision, we introduce the notion of data error, denoted de_i, which gives an indication of how much the value of a data object d_i stored in the RTDB deviates from the corresponding real-world value given by the latest arrived update transaction. Note that the latest arrived update transaction is discarded if it holds a value that is sufficiently close to the value already stored in the database. Hence, d_i may hold the value of an earlier update transaction. We say that quality of data (QoD) increases as the data error of the data objects decreases.

Imprecision at user transaction level can be expressed in terms of certainty, accuracy, and specificity [4]. Certainty refers to the probability of a result being correct, accuracy refers to the degree of accuracy of a value returned by an algorithm (typically through a bound on the difference from the exact solution), and specificity reflects the level of detail of the result. For example, if filters are used in control loops, greater accuracy is achieved. Specificity is used to define user transaction imprecision in the context of image coding or decoding. The imprecision of the result of a user transaction increases as the resource available for the user transaction decreases. For simplicity, we refer to the imprecision of the results of the user transactions as quality of transaction (QoT). We say that QoT increases as the imprecision of the results of the user transaction decreases.

Usually, system developers know how much data imprecision an application can tolerate such that the end result is within acceptable limits. Therefore, we assume that sufficiently precise data values stored in the database are regarded as having no effect on the result of a transaction. Hence, we model QoT and QoD as orthogonal entities. System developers can then focus on finding appropriate precision requirements and avoid modeling QoT as functions of QoD. This significantly reduces the complexity of the QoS specification process.

QoT is manipulated by adjusting the admitted user transaction load and the admitted update transaction load. The CPU resource allocated for each user transaction decreases as the number of admitted user transactions and the number admitted update transactions increase, resulting in a decrease in QoT. The update transaction load is reduced by discarding update transactions according to an upper bound for the data error given by the maximum data error, denoted mde. Note, discarding update transactions reduces QoD, however, we assume that QoT is not affected by QoD as they are modeled to be orthogonal. 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 than or equal to mde (i.e., $de_i \leq mde$). If mde increases, more update transactions are discarded, degrading the QoD. This results in more resources available for user transactions and, hence, an increase in QoT. Similarly, if mde decreases, fewer update transactions are discarded, resulting in a greater QoD and, consequently, a lower QoT. The goal of our work consists of two parts. We want to derive: 1) algorithms for adjusting data error using mde such that QoD and QoT satisfy a given QoS specification and the deviation in imprecision of user transaction results is minimized, i.e., QoS fairness is maximized, and 2) a feedback loop architecture that is highly reactive and adaptive to changes to workload characteristics. The second part implies, as argued in Section 1, that we need to find accurate models of the controlled system to provide efficient QoS adaptability and performance reliability even in the presence of unpredictable workload.

3 Data and Transaction Model

We consider a main memory database model where there is one CPU as the main processing element. Main memory databases have been increasingly applied to real-time data management due to their relatively high performance, decreasing main memory cost, fast response time (since I/O overhead is decreased), and the emergence of embedded systems lacking disks [15], [16]. In our data model, data objects can be classified into two classes, temporal and nontemporal [17]. 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 length of the absolute validity interval of d_i (denoted avi_i), i.e., $currenttime > timestamp_i + avi_i$. For a data object d_i, let data error $de_i = \Phi(cv_i, v_i)$ be a nonnegative function of the current value cv_i of d_i and the value v_i of the latest arrived transaction that updated d_i or that was to update d_i but was discarded. Remember an update transaction may be discarded if its update value is close enough to the value stored in the RTDB. Our approach does not have any restrictions on the structure of Φ. For example, it may be defined as the absolute deviation between cv_i and v_i, i.e., $de_i = |cv_i - v_i|$, or the relative deviation as given by $de_i = \frac{|cv_i - v_i|}{|cv_i|}$. Update transactions arrive periodically and may only write to base data objects. User transactions arrive aperiodically and may read temporal and read/write nontemporal data. User and update transactions (T_i) are composed of one mandatory subtransaction m_i and $|O_i| \geq 0$ optional subtransactions $o_{i,j}$, where $o_{i,j}$ is the jth optional subtransaction of T_i. For the remainder of the paper, we let t_i denote a subtransaction of T_i. As updates do not use complex logical or numerical operations, we assume that each update transaction consists only of a single mandatory subtransaction, i.e., $|O_i| = 0$.

As mentioned earlier, there are several ways of implementing imprecise computations, e.g., multiple versions, use of sieve functions, and the milestone approach [3]. The focus of this paper is not on how to apply different imprecise computation techniques in the context of RTDBs since this area has already been explored, as shown in Section 6. Previous work indicates that iterative and recursive algorithms, generating monotonically improving answers, can be efficiently used to solve problems in a wide class of applications, such as, numerical algorithms, e.g., Newton's method and FFT [18], graph algorithms [4], and also query processing [6], [7]. Iterative and recursive algorithms can easily be modeled using the milestone approach, where the k first iterations or recursions correspond to the mandatory part and the remainder are given by the optional part. For this reason, we use the milestone approach [3] to transaction impreciseness. Thus, we divide 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 should 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 of a transaction T_i arrive at the same time, the first optional subtransaction (if any) $o_{i,1}$ becomes ready for execution when the mandatory subtransaction m_i 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. We set the deadline of every subtransaction t_i to the deadline of the transaction T_i. A subtransaction is terminated if it has completed or has missed its deadline. A transaction T_i is terminated when $o_{i,|O_i|}$ completes or one of its subtransactions misses its deadline. In the latter case, all subtransactions that are not completed also miss their deadlines and are therefore terminated as well.

We introduce the notion of transaction error (denoted te_i), inherited from the imprecise computation model [3], to measure the imprecision of a user transaction result, T_i. Transaction error may be modeled as a function of completed optional subtransactions. This requires knowledge about the transactions and/or the data sets they read. Although our work does not require detailed knowledge about the transactions, in many applications, this knowledge is available to the designer and transaction error may be derived through experiments [4], analytical expressions, e.g., accuracy bounds for numerical iterative algorithms [19], or the experience of designers or engineers. The exact details of the above-mentioned methods are beyond the scope of this paper and the reader is referred to the appropriate literature. In applications where it is possible to formally model the precision of the answers given by transactions in terms of completed optional subtransactions, we model transaction error through the use of error functions [20]. For a transaction T_i, we use an error function to approximate its corresponding transaction error given by $te_i(|COS_i|) = (1 - \frac{|COS_i|}{|O_i|})^{n_i}$, where n_i is the order of the error function and $|COS_i|$ denotes the number of completed optional subtransactions. By choosing n_i, we can model and support multiple classes of transactions showing different error characteristics. For example, it has been shown that, anytime algorithms used in AI exhibit error characteristics, n_i is greater than one [4].

We assume the workload model presented by Lu et al. [11], where update transactions have a period and user transactions have a mean interarrival time. The estimated load of a task is the estimated execution time of the task divided by its relative deadline. The actual load of a task is the actual execution time of the task divided by its relative deadline.

4 Approach

Next, we describe our approach for managing the performance of an RTDB in terms of QoT and QoD. First, we start by defining performance metrics in Section 4.1. The QoS specification models are described in Section 4.2. An overview of the feedback control scheduling architecture is given in Section 4.3, followed by the description of QoS controllers in Section 4.4. In Section 4.5, we present the algorithms PC-MPU (precision control miss percentage utilization), PC-MP (precision control miss percentage), $\text{PC-ATE}_{\text{HEF}}$ (precision control average transaction error highest error first), and $\text{PC-ATE}_{\text{HEDF}}$ (precision control average transaction error highest error density first). These algorithms determine how QoD is adjusted, i.e., to what extent the precision of the data objects is modified based on the current system performance. In Section 4.6, we present two models, describing the dynamics of RTDBs, which are used to tune the QoS controllers.

4.1 Performance Metrics

In our approach, the database operator[1] can explicitly specify the required database QoS, defining the desired behavior of the database. Long-term performance metrics such as average deadline miss ratio are not sufficient to specify the desired performance of real-time systems that require stringent QoS enforcement [11]. Therefore, in this work, we adapt both long-term performance metrics, referred to as steady-state performance metrics, and transient-state performance metrics. We adapt the following notation of describing discrete variables in the time-domain: $a(k)$ refers to the value of the variable a at the time kT, where T is the sampling period and k is the sampling instant.[2]

QoT Metrics. We consider the following metrics for measuring QoT of admitted transactions:

- Deadline miss percentage of mandatory user subtransactions is given by $m^M(k) = 100 \times \frac{|deadlmiss^M(k)|}{|term^M(k)|} (\%)$ where $|deadlmiss^M(k)|$ denotes the number of mandatory subtransactions that have missed their deadline and $|term^M(k)|$ is the number of terminated mandatory subtransactions.
- Deadline miss percentage of optional user subtransactions is given by $m^O(k) = 100 \times \frac{|deadlmiss^O(k)|}{|term^O(k)|} (\%)$, where $|deadlmiss^O(k)|$ denotes the number of optional subtransactions that have missed their deadline and $|term^O(k)|$ is the number of terminated optional subtransactions. Note, $|deadlmiss^O(k)|$ and $|term^O(k)|$ include the optional subtransactions that are not completed.
- Average transaction error is defined as

$$ate(k) = 100 \times \frac{\sum_{i \in term(k)} te_i}{|term(k)|} (\%),$$

where $term(k)$ denotes the set of terminated transactions.

QoD Metric. The maximum data error $mde(k)$ gives the maximum data error tolerated for the data objects (as described in Section 2).

QoS Fairness Metric. For some applications, it is desirable to measure QoS fairness among transactions and, therefore, we introduce the standard deviation of transaction error,

$$sdte(k) = \sqrt{\frac{\sum_{i \in term(k)} (100 \times te_i - ate(k))^2}{|term(k)| - 1}},$$

which is a measure of how much the transaction error of terminated transactions deviates from the average transaction error.

System Utilization. We measure system utilization $u(k)$ to acquire a better understanding of the performance of the algorithms. Using the utilization of the system, we can show whether our algorithms provide high throughput.

Steady-State and Transient-State Performance Metrics. The desired performance of the system is given by a set of references specifying the desired level of the controlled variables, which represent the actual system performance. We consider the following transient-state performance metrics (see Fig. 1a). Overshoot M_p is the worst-case system performance in the transient system state and it is given in the percentage by which a controlled variable overshoots its reference. Settling time T_s is the time for the transient overshoot to decay and settle around the steady state performance and it is a measure of system adaptability, i.e., how fast the system converges toward the desired performance. Hence, the performance of the controllers is distinguished by how well they force a controlled variable $y(k)$ to follow or track a desired level given by a reference $y_r(k)$, despite the presence of disturbances in the controlled system. It is therefore interesting to measure the difference between $y_r(k)$ and $y(k)$ over a period of time, which is obtained using the functions $J_a = \frac{1}{N}\sum_{k=1}^{N} |y_r(k) - y(k)|$ and $J_s = \frac{1}{N}\sum_{k=1}^{N} (y_r(k) - y(k))^2$, where N is the number of samples taken. The lower J_s and J_a are, the better a controller is able to keep y near y_r and, also, the faster y converges toward y_r.

4.2 QoS Specification Models

The maximum data error provides a direct measure of the precision of the data objects and, hence, we express QoD in terms of mde. An increase in QoD refers to a decrease in

1. By a database operator, we mean an agent, human or computer, that operates the database, including setting the QoS.

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

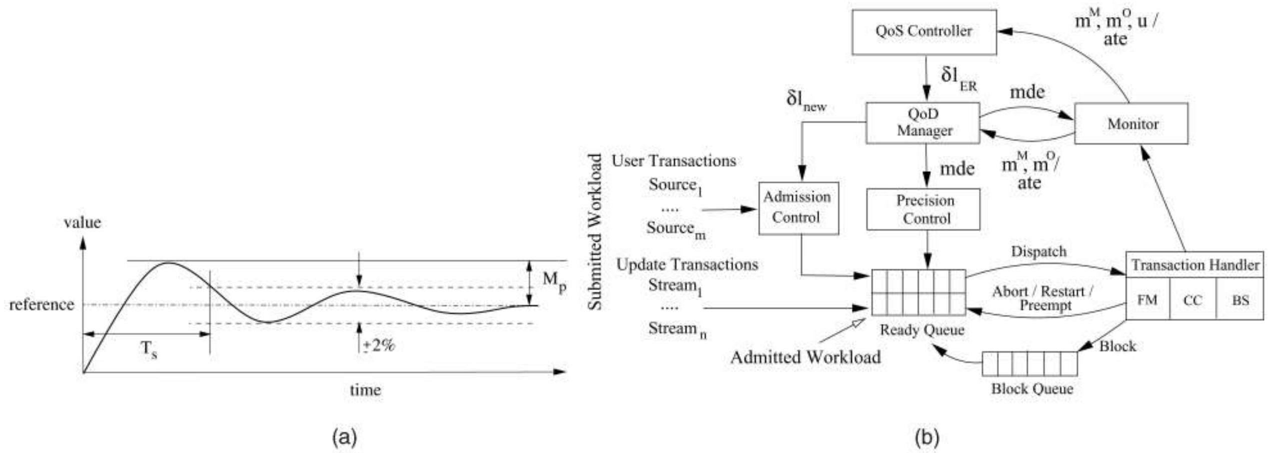

Fig. 1. Performance specification and system architecture. (a) Settling time and overshoot. (b) Feedback control scheduling architecture.

mde, i.e., an increase in data precision. In contrast, a decrease in QoD refers to an increase in mde. We consider two alternative ways of defining QoS, below referred to as QoS specification type A and type B, where they differ in the way QoT is expressed.

QoS Specification Type A. In the case when it is not possible to model transaction error using error functions, we have to express QoT by other means. We know that the more optional subtransactions we complete before the deadline, the smaller the transaction error will be. Therefore, the deadline miss ratio of optional subtransactions qualifies as an approximate measure of the true transaction error and, hence, we define QoT in terms of m^O. QoT decreases as m^O increases (similarly, QoT increases as m^O decreases). The database operator can specify steady-state and transient-state behavior for m^M, m^O, u, and mde. The specification for u is given by a lower bound u_l for u. A QoS requirement can be specified as the following: $m_r^M = 1\%$ (i.e., reference of m^M), $m_r^O = 10\%$, $mde_r = 2\%$, $u_l = 80\%$, $T_s \leq 60s$, and $M_p \leq 30\%$. This gives the following transient-state performance specifications: $m^M \leq m_r^M \times (M_p + 100\%) = 1.3\%$, $m^O \leq 13\%$, and $mde \leq 2.6\%$.

QoS Specification Type B. Having error functions to describe the transaction error, we can directly define QoT in terms of average transaction error (ate). QoT decreases as ate increases (similarly, QoT increases as ate decreases). The database operator can specify steady-state and transient-state behavior for ate and mde. A QoS requirement can be specified as the following: $ate_r = 20\%$ (i.e., reference of ate), $mde_r = 5\%$, $T_s \leq 60s$, and $M_p \leq 30\%$. This gives the following transient-state performance specifications: $ate \leq ate_r \times (M_p + 100\%) = 26\%$ and $mde \leq mde_r \times (M_p + 100) = 6.5\%$.

4.3 QoS Management Architecture

The architecture of our QoS management scheme is shown in Fig. 1b. Admitted transactions are placed in the ready queue. The transaction handler manages the execution of the transactions. We choose m^M, m^O, and u as controlled variables when the QoS is specified according to QoS specification type A, while ate is the controlled variable when QoS specification type B is used. At each sampling instant, the controlled variable(s) (i.e., m^M, m^O, and u, or ate) is monitored and fed into the QoS controller, which compares the performance reference (i.e., m_r^M and m_r^O or ate_r) with the controlled variable to get the current performance error. Based on the result, the controller computes a change, denoted δl_{ER}, to the total estimated requested load. We refer to δl_{ER} as the manipulated variable. Based on δl_{ER}, the QoD manager changes the total estimated requested load by adapting the QoD (i.e., adjusting mde). The precision controller discards an update transaction writing to a data object d_i having an error less than 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 timestamp of d_i is updated. The portion of δl_{ER} not accommodated by the QoD manager, denoted δl_{new}, is returned to the admission controller (AC), which enforces the remaining load adjustment. The transaction handler provides a platform for managing transactions. It consists of a freshness manager (FM), a unit managing the concurrency control (CC), and a basic scheduler (BS). The FM checks the freshness before accessing a data object, using the timestamp and the absolute validity interval of the data. We employ two-phase locking with highest priority (2PL-HP) [21] for concurrency control. 2PL-HP is chosen since it is free from priority inversion and has well-known behavior. We use three different scheduling algorithms as basic schedulers:

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 (for an elaborate discussion on EDF, see, e.g., [8]).

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 $ted_i = \frac{te_i}{at_i + rd_i - currenttime}$, where at_i and rd_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.

Note that HEF and HEDF cannot be used in the case when error functions for transactions are not available as they are error-cognizant and require knowledge of te_i. For all three basic schedulers (EDF, HEF, and HEDF), the mandatory subtransactions have higher priority than the optional subtransactions and, hence, are scheduled before them.

4.4 QoS Controllers

Depending on the algorithms used, we apply different feedback control loops to control QoT in the presence of unpredictable workload and inaccurate execution time estimates. PC-MPU employs one utilization controller and two miss percentage controllers, i.e., one controller to adjust the utilization u according to a reference u_r and two controllers to adjust m^M and m^O according to the references m_r^M and m_r^O, respectively. Transactions in RTDBs often make unpredictable aborts or restarts due to data and resource conflicts. Further, the execution time of the transactions depends on their data needs, which may vary over time. This makes the deadline miss percentages prone to overshoot. To avoid overshoots greater than M_p, the load of the system is constantly changed according to a linear increase/exponential decrease scheme. Initially, the utilization reference u_r is set to u_l. As long as the miss percentages are below their references, u_r is increased by a certain step. As soon as one of the miss percentages is above its reference, u_r is reduced exponentially according to $u_r(k+1) = \frac{u_r(k)+u_l}{2}(\%)$, where $u_r(k+1)$ is the new utilization reference. This way, we are certain that the system is not underutilized, while, at the same time, great deadline miss ratio overshoots are avoided. This approach is self-adapting and does not require any knowledge about the underlying runtime estimates. PC-MP uses two deadline miss percentage control loops, one for each of the controlled variables m^M and m^O. The algorithms $\text{PC-ATE}_{\text{HEF}}$ and $\text{PC-ATE}_{\text{HEDF}}$ use a single average transaction error control loop to control ate (i.e., the controlled variable). Here, ate is monitored and fed into the controller, which computes δl_{ER} according to ate and ate_r.

Using several controllers raises the question of integration of the signals from each controller. In case PC-MP, where miss percentage controllers are used, we need to integrate the signals from the mandatory and the optional subtransaction miss percentage controllers. Further, in PC-MPU, where a combination of miss percentage and utilization controllers is used, an integrated signal from both the miss percentage and the utilization controllers is computed and returned. Let δl_M denote the control signal computed by the m^M controller, δl_O denote the control signal computed by the m^O controller, and δl_u denote the control signal computed by the u controller. The integrated control signal from both miss ratio controllers δl_{MP} is computed as follows: If both miss percentage control signals are negative (i.e., $\delta l_M < 0 \wedge \delta l_O < 0$), we set $\delta l_{MP} = \delta l_M + \delta l_O$ to the sum of both control signals. This is necessary since both miss percentages are above their references and both signals must be considered to compensate for miss percentage overshoots. If the above does not hold, we set $\delta l_{MP} = \min(\delta l_M, \delta l_O)$ to the minimum of the control signals. If one of the control signals is negative (due to an overshoot), we return the negative one to reduce the miss percentage of the corresponding subtransaction type. If both are positive, the *min* operator provides a smooth transition between low and high miss percentages among mandatory and optional subtransactions [11]. In the case when only miss percentage controllers are used, we set δl_{ER} to δl_{MP}. However, when a utilization controller is used as well, we derive δl_{ER} by taking the minimum of δl_{MP} and δl_u. This is necessary for similar reasons as those mentioned above.

4.5 QoD Management Algorithms

We recall that setting $mde(k+1)$ greater than $mde(k)$ results in more discarded update transactions and, hence, a decrease in update transaction load. Similarly, setting $mde(k+1)$ less than $mde(k)$ results in fewer discarded update transactions and, hence, an increase in update transaction load. To compute $mde(k+1)$ given a certain $\delta l_{ER}(k)$, we use a function $f(\delta l_{ER}(k))$ that returns, based on $\delta l_{ER}(k)$, the corresponding $mde(k+1)$. The function f holds the following property: If $\delta l_{ER}(k)$ 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 $\delta l_{ER}(k)$ is greater than zero, then $mde(k+1)$ is set such that $mde(k+1)$ is less than $mde(k)$, i.e., QoD is improved. We will discuss the function f in more detail later in this section.

The algorithms PC-MPU and PC-MP control QoT by monitoring m^M, m^O, and u and adjusting mde such that a given QoS specification according to QoS specification type A is satisfied. Here, we use EDF as a basic scheduler. The algorithms $\text{PC-ATE}_{\text{HEF}}$ and $\text{PC-ATE}_{\text{HEDF}}$ are error-cognizant and control QoT by monitoring ate and adjusting mde such that a QoS specification in terms of QoS specification type B is satisfied. Furthermore, $\text{PC-ATE}_{\text{HEF}}$ and $\text{PC-ATE}_{\text{HEDF}}$ are designed to enhance QoS fairness among transactions (i.e., decrease the deviation in te_i among admitted transactions). We use the same feedback control policy for $\text{PC-ATE}_{\text{HEF}}$ and $\text{PC-ATE}_{\text{HEDF}}$, but use different basic schedulers, i.e., $\text{PC-ATE}_{\text{HEF}}$ schedules the transactions using HEF and $\text{PC-ATE}_{\text{HEDF}}$ schedules the transactions using HEDF. The details of the algorithms are given below.

PC-MPU. The system monitors the deadline miss percentages and the CPU utilization. At each sampling instant, the CPU utilization adjustment, $\delta l_{ER}(k)$, is derived. If $\delta l_{ER}(k)$ is greater than zero, upgrade QoD as much as $\delta l_{ER}(k)$ allows. However, when $\delta l_{ER}(k)$ is less than zero, degrade QoD, i.e., increase mde according to δl_{ER}, but not greater than the highest allowed mde (i.e., $mde_r \times (M_p + 100)$). Degrading the data further would violate the upper limit of mde, given by the QoS specification. When $\delta l_{ER}(k)$ is less than zero and mde equals $mde_r \times (M_p + 100)$, no QoD adjustment can be issued and, hence, the system has to wait until some of the currently running transactions terminate. An outline of PC-MPU is given in Fig. 2a.

PC-MP. In PC-MPU, 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 PC-MP, the utilization controller is removed to keep the miss percentages at the specified references. One of the characteristics of the miss percentage controller is

Monitor $m^M(k)$, $m^O(k)$, and $u(k)$
Compute $\delta l_{ER}(k)$
if $\delta l_{ER}(k) > 0 \wedge mde(k) > 0$ **then**
  Upgrade QoD, $mde(k+1) := f(\delta l_{ER}(k))$
  Subtract utilization lost from $\delta l_{ER}(k)$
else if $\delta l_{ER}(k) < 0 \quad \wedge$
    $mde(k) < mde_r \times (M_p + 100)$ **then**
  Downgrade QoD, $mde(k+1) := f(\delta l_{ER}(k))$
  Add utilization gained to $\delta l_{ER}(k)$
end if
Set δl_{new} to the new $\delta l_{ER}(k)$

(a)

Monitor $m^M(k)$ and $m^O(k)$
Compute $\delta l_{ER}(k)$
if $\delta l_{ER}(k) \geq 0$ **then**
  $mde(k+1) :=$

$$\min(\frac{m^O_{MA}(k)}{m^O_r} mde_r, mde_r \times (M_p + 100))$$

  if $mde(k) < mde(k+1)$ **then**
    Add utilization gained to $\delta l_{ER}(k)$
  else
    Subtract utilization lost from $\delta l_{ER}(k)$
  end if
else if $\delta l_{ER}(k) < 0 \quad \wedge$
    $mde(k) < mde_r \times (M_p + 100)$ **then**
  $mde(k+1) := f(\delta l_{ER}(k))$
  Add utilization gained to $\delta l_{ER}(k)$
end if
Set δl_{new} to the new $\delta l_{ER}(k)$

(b)

Fig. 2. QoD Management algorithms. (a) PC-MPU. (b) PC-MP.

that, as long as m^O is below its reference (i.e., $m^O \leq m^O_r$), the controller output δl_{ER} is positive. Due to the characteristics of f (i.e., $\delta l_{ER}(k) > 0 \Rightarrow mde(k+1) < mde(k)$), a positive δl_{ER} is interpreted as a QoD improvement. Consequently, even if m^O is just below its reference, QoD remains high.

It is desirable to let m^O, which corresponds to QoT, increase and decrease together with QoD given by mde. For this reason, mde is set considering both δl_{ER} and m^O. When δl_{ER} is less than zero (i.e., m^O overshoots), mde is set according to f. However, when δl_{ER} is greater than or equal to zero, mde is set according to the moving average of m^O, computed by $m^O_{MA}(k) = \alpha m^O(k) + (1-\alpha) m^O_{MA}(k-1)$, where α $(0 \leq \alpha \leq 1)$ is the forgetting factor [22]. The moving average is used to reduce large deviations from one sampling period to another. Setting α close to 1 results in a fast adaptation, but also captures any high-frequency changes of m^O, whereas setting α close to 0 results in a slow but smooth adaptation. 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 is increased, but to a maximum value of $mde_r \times (M_p + 100)$ since a further increase violates the given QoS specification. The outline of PC-MP is given in Fig. 2b.

PC-ATE$_{\text{HEF}}$ and **PC-ATE$_{\text{HEDF}}$**. These algorithms are two variants of PC-MP, but where QoT is measured in terms of ate, instead of m^O. Hence, we replace the miss percentage control loops for a single average transaction error control loop. Here, mde is adjusted based on the control signal δl_{ER} and the moving average of ate, denoted $ate_{MA}(k)$. We do not provide full algorithm descriptions for PC-ATE$_{\text{HEF}}$ and PC-ATE$_{\text{HEDF}}$, but refer instead to Fig. 2b, where m^O_{MA} is replaced with ate_{MA}.

The precision of the data is controlled by the QoD manager by setting mde depending on the system behavior. When f is used to compute $mde(k+1)$ based on $\delta l_{ER}(k)$, the following scheme is used: Discarding an update results in a decrease in CPU load, which we refer to as the gained load. Let

$$gl(k) = \frac{1}{T} \sum_{T_i \in discarded(k)} eet_i$$

be the sum of the estimated execution time of the discarded update transactions divided by the sampling period T, where $discarded(k)$ is the set of discarded update transactions and eet_i is the estimated execution time of the update transaction T_i. In our approach, we profile the system and measure gl for varying mde and linearize the relationship between them. During each sampling period, $gl(k)$ is monitored and $\mu(k) = \frac{mde(k)}{gl(k)}$ and its moving average $\mu_{MA}(k)$ are computed. Consequently, the relation between mde and gl is updated to capture the current state of the system. Having the relationship between gl and mde, we introduce the help function,

$$h(\delta l_{ER}(k)) = \min(\mu_{MA}(k) \times (gl(k) - \delta l_{ER}(k)), \\ mde_r \times (M_p + 100)).$$

Since mde is not allowed to overshoot more than $mde_r \times (M_p + 100)$, we use the min operator to enforce this requirement. Further, since mde by definition cannot be less than zero, we apply the max operator on h and obtain $mde(k+1) = f(\delta l_{ER}(k)) = \max(h(\delta l_{ER}(k)), 0)$.

4.6 System Modeling

To tune feedback controllers that are efficient in managing the desired performance and that react rapidly to changes in workload, it is necessary to have a model that accurately describes the behavior of the controlled system [14]. The particular form of the models we construct, i.e., linear models, enables us to use a set of powerful analytical methods that are available in control theory, e.g., root locus [14]. For analysis purposes, we apply the principles of $\mathcal{Z}$-transform theory [14]. Using $\mathcal{Z}$-transforms enables us to reduce the complexity of large dynamic systems into the simpler representation by $\mathcal{Z}$-transforms. Further, $\mathcal{Z}$-transforms are used in many controller tuning procedures, e.g., root locus [14]. We adopt the following notation, where $A(z)$ denotes the $\mathcal{Z}$-transform of the variable $a(k)$. The goal is to derive a transfer function describing the relation between the manipulated variable, i.e., $\Delta L_{ER}(z)$, and the controlled variables, i.e., $M^M(z)$, $M^O(z)$, $ATE(z)$, and $U(z)$. In this section, we first present the STA model, previously presented [11], where some dynamic relations are approximated by static relations (hence, the name STA referring to statics). Then, we propose a new model, called DYN, which generalizes the STA model [11] by capturing additional system dynamics (the name DYN refers to dynamics).

4.6.1 STA

The estimated requested workload of admitted user transactions l_{ER} in the next sampling period is changed through the manipulated variable δl_{ER}, given by

$$l_{ER}(k+1) = l_{ER}(k) + \delta l_{ER}(k). \tag{1}$$

Hence, l_{ER} is the integration of the control input δl_{ER}. Now, the estimated admitted workload of user transactions l_E may differ from l_{ER} since external load applied on the database may not be sufficient to satisfy l_{ER} or the admitted workload is decreased due to deadline misses and, consequently, early termination of transactions. Here, however, we approximate the estimated load of admitted transactions by l_{ER}, i.e., $l_E = l_{ER}$ (in DYN, we take a different approach).

The actual workload, denoted $l_A(k)$, may differ from $l_E(k)$ due to incomplete knowledge about the controlled system, e.g., unknown execution times of the transactions and data conflicts. Therefore, we get $l_A(k) = g_A(k) l_E(k)$, where the workload ratio $g_A(k)$ represents the workload variation in terms of actual total requested workload. For example, $g_A(k)$ equal to two means that the actual workload is twice the estimated workload. It is obvious that $g_A(k)$ cannot be deterministically modeled.[3] However, by profiling the controlled system, we can compute $g_A(k)$ for each sampling and form the average of $g_A(k)$, denoted g_A, which is then used in our model to describe the relation between l_E and l_A in the average case, i.e.,

$$l_A(k) = g_A l_E(k). \tag{2}$$

The relationship between the actual workload l_A and the utilization u is nonlinear due to saturation, as given by the following:

$$u(k) = \begin{cases} l_A(k), & l_A(k) \le 100\% \\ 100\%, & l_A(k) > 100\%. \end{cases} \tag{3}$$

When $l_A(k)$ is less than or equal to 100 percent, i.e., the CPU is underutilized, $u(k)$ is outside its saturation zone and equals $l_A(k)$. However, when $u(k)$ is within its saturation zone, i.e., $l_A(k)$ is greater than 100 percent, then $u(k)$ remains at 100 percent, despite changes to l_A.

From classical scheduling theory, it is possible to determine whether a set of tasks can be scheduled such that no deadline misses occur [8]. Turning to the case of the miss percentages, we define the schedulable threshold for mandatory subtransactions, denoted $l_{th}^M(k)$, and optional subtransactions, denoted $l_{th}^O(k)$, as the load threshold in the kth sampling period for which no deadline miss can be observed for the respective type of subtransaction. When the miss percentages are saturated, i.e., they are inside the saturation zones given by $l_A(k) \le l_{th}^M(k)$ and $l_A(k) \le l_{th}^O(k)$, no deadline misses are observed. At this condition, adjustments of δl_{ER} and, consequently, $l_A(k)$ will not affect the miss percentages until the actual load becomes greater than the threshold and miss percentages start increasing. However, when outside the saturation zones, i.e., $l_A(k) > l_{th}^M(k)$ or $l_A(k) > l_{th}^O(k)$, the miss percentage for either subtransaction increases nonlinearly. Note, since mandatory subtransactions have a higher priority than optional subtransactions, the schedulable threshold for mandatory subtransactions is greater than the threshold for optional subtransactions, i.e., $l_{th}^M(k) > l_{th}^O(k)$.

Since feedback control relies on linear systems, we linearize the relationship between l_A and m^M by deriving the ratio between l_A and m^M in the vicinity of the miss ratio reference m_r^M. Similarly, we form the linear relationship between l_A and m^O by deriving the ratio between l_A and m^O in the vicinity of the miss ratio reference m_r^O, giving the equations,

$$g_m^M = \frac{m^M}{l_A}, \qquad m^M = m_r^M, \tag{4}$$

$$g_m^O = \frac{m^O}{l_A}, \qquad m^O = m_r^O. \tag{5}$$

We model the average transaction error similarly to the miss percentages. The relationship between l_A and ate is nonlinear due to saturation. We define the precisely schedulable threshold $l_{th}^{ATE}(k)$ as the load threshold in the kth period for which all admitted subtransactions meet their deadlines. The average transaction error becomes saturated when it is within its saturation zone, given by $l_A(k) \le l_{th}^{ATE}(k)$. When the average transaction error is saturated, ate remains zero despite changes to δl_{ER} and, hence, l_A. However, when outside the saturation zones, i.e., $l_A(k) > l_{th}^{ATE}(k)$, the average transaction error increases nonlinearly. We linearize the relationship between l_A and ate by taking the ratio between l_A and ate in the vicinity of the reference ate_r, i.e.,

$$g_{ate} = \frac{ate}{l_A}, \qquad ate = ate_r. \tag{6}$$

From (1)-(6), we can derive a transfer function for each of the controlled variables when they are outside their saturation zones. Hence, under the condition $l_A \le 100\%$, there exists a transfer function $G_{STA,U}(z) = \frac{g_A}{z-1}$ from the control input $\Delta L_{ER}(z)$ to CPU utilization $U(z)$. Similarly, under the conditions $l_{th}^M(k) < l_A$, $l_{th}^O(k) < l_A$, and $l_{th}^{ate}(k) < l_A$, the transfer functions

$$G_{STA,m}^M(z) = \frac{g_A g_m^M}{z-1},\ G_{STA,m}^O(z) = \frac{g_A g_m^O}{z-1},\ G_{STA,ate}(z) = \frac{g_A g_{ate}}{z-1}$$

relate the control input $\Delta L_{ER}(z)$ to the controlled variables $M^M(z)$, $M^O(z)$, and $ATE(z)$, respectively.

4.6.2 DYN

In DYN, we extend the model STA to include additional system dynamics. In STA, we assumed that there are static relations between l_{ER}, u, m^M, m^O, and ate. We here show that, in fact, there are dynamic relations between these variables and STA fails to capture them. Starting from the model input $\delta l_{ER}(k)$, we compute $l_{ER}(k)$ according to (1).

3. If $g_A(k)$ can be deterministically modeled, then we can compute exact execution times given estimated execution times.

Given a certain $l_{ER}(k)$, the estimated workload of admitted user transactions,

$$l_E(k) = g_L \min(l_{ER}(k), \bar{l_E}(k)), \tag{7}$$

is the product of the requested load factor g_L and the minimum of the estimated requested load and the maximum estimated load $\bar{l_E}(k)$ that can be made available. The upper limit of the admitted load, given by $\bar{l_E}(k)$, makes the relationship between $l_{ER}(k)$ and $l_E(k)$ nonlinear due to saturation. $l_E(k)$ becomes saturated when it is within its saturation zone, given by $l_{ER} > \bar{l_E}(k)$. However, when outside the saturation zone, $l_E(k)$ increases linearly with $l_{ER}(k)$. Furthermore, even though outside the saturation zone, the estimated admitted workload $l_E(k)$ may be lower than the estimated requested workload $l_{ER}(k)$ due to early termination of the tasks caused by deadline misses. To capture this difference, we derive the ratio between $l_{ER}(k)$ and $l_E(k)$ in the vicinity of $l_E(k)$ corresponding to m_r^O or ate_r and we obtain requested load factor g_L.

Now, under the condition $l_{ER} < \bar{l_E}$, it can be observed that, given a certain increase in estimated requested workload l_{ER}, it takes some time before the estimated load l_E and, hence, the utilization u reach l_{ER}. The time it takes for l_E to reach l_{ER} is determined by the amount of workload submitted to the system $\bar{l_E}$ or, more specifically, the arrival rate of the transactions submitted to the system. The greater $\bar{l_E}$ is, the faster we can fill up the workload, reaching l_{ER} earlier. Similarly, a decrease in l_{ER} does not result in an immediate decrease in l_E since currently running transactions have to terminate. Hence, an increase/decrease in l_{ER} does not result in an immediate increase/decrease in l_E and, consequently, we have a dynamic relation between l_{ER} and l_E. The relation between l_E and l_A is linearized according to (2), i.e., we describe the actual workload in terms g_A.

Furthermore, we argue that the relation between l_A and m^M, m^O, ate is nonstatic. We give the rationale only for the case of l_A and a miss ratio m, as the dynamics of the relationship between l_A and ate is described by the same line of argument. Given a certain l_A, the time it takes for m to reach $g_m l_A$, shown in (4), depends on the actual basic scheduler used. Under EDF scheduling, newly admitted transactions are placed further down in the ready queue since they are less likely to have earlier deadlines than transactions admitted earlier. This means that an actual change to m is not noticed until the newly admitted transactions are executing, which may take a while until the older ones have terminated. Consider the example given in Fig. 3, where the execution time of the transactions is two time units. We want to increase the number of deadline misses during interval $[kT, (k+1)T]$ and, hence, at sampling k, we increase the load by raising the admission rate of transactions, which are scheduled to be executed later than the already admitted transactions, i.e., in interval $[(k+1)T, (k+2)T]$. As shown in Fig. 3, the number of deadline misses does not increase in the interval $[kT, (k+1)T]$, instead it increases in the interval $[(k+1)T, (k+2)T]$, which causes a delay between the issuing of load increase and the observed increase in number of deadline misses. Under HEF scheduling, newly

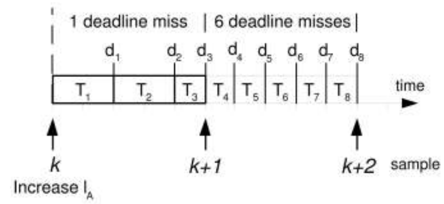

Fig. 3. A change to l_A is not noticed immediately when EDF is applied.

arrived transactions are more likely to have higher priority than old transactions (since they have greater transaction error) and, hence, they are placed at the front of the ready queue. The set of newly admitted transactions are therefore executed instantaneously and, hence, a change to m is noticed earlier than compared to EDF scheduling. Hence, under HEF scheduling, the controlled variable is more responsive to changes in the manipulated variable as m converges faster toward $g_m l_A$.

We have now established that, given a certain change in l_{ER}, it takes some time before the controlled variables u, m^M, m^O, and ate reach their final values u_S, m_S^M, m_S^O, and ate_S in the steady-state. The speed by which a controlled variable reaches its final values is determined by the time constant of the system. Let T denote the sampling period, and T_u, T_m^M, T_m^O, and T_{ate} denote the time constants of u, m^M, m^O, and ate, respectively. To give a concise statement of the modeling, we only examine the case for m^M as the dynamics of u, m^O, and ate are modeled similarly. The difference equation,

$$m^M(k+1) = \frac{T(m_S^M(k) - m^M(k))}{T_m^M} + m^M(k), \tag{8}$$

relates m^M and its final value m_S^M such that it takes a number of samples for m^M to reach m_S^M. Initially, when the difference between m_S^M and m^M is large, m^M converges rapidly toward m_S^M. However, the speed of convergence decreases as the difference between m_S^M and m^M decreases. It shows that the speed of convergence is determined by T_m^M, i.e., an increase in T_m^M results in a slower convergence. The $\mathcal{Z}$-transform of (8) is given by

$$G_{D,M}^M(z) = \frac{T}{T_m^M} \frac{1}{z - 1 + \frac{T}{T_m^M}}. \tag{9}$$

A step function with amplitude m_S^M applied on (9) gives the time domain solution,

$$m^M(k) = m_S^M \left[1 - \left(1 - \frac{T}{T_m^M} \right)^k \right].$$

Here, it is clearly shown that the greater T_m^M is, the more time it takes for m^M to reach the final value m_S^M, i.e., the slower m^M reacts to changes in m_S^M. Let us now examine the step response of the controlled system from l_{ER} to m^M and measure the time, δ_m^M, it takes for $m^M(k)$ to reach $(1 - e^{-1})m_S^M$, i.e., approximately 63 percent of the final value of $m^M(k)$. At time δ_m^M, we have that

$$m^M(k) = m^M\left(\frac{\delta_m^M}{T}\right) = (1-e^{-1})m_S^M$$
$$= m_S^M\left[1-\left(1-\frac{T}{T_m^M}\right)^{\frac{\delta_m^M}{T}}\right]$$

and solving for T_m^M gives

$$T_m^M = \frac{T}{1-e^{-\frac{T}{\delta_m^M}}}. \tag{10}$$

Hence, we compute T_m^M according to (10), where, given a step on l_{ER}, δ_m^M is the time it takes for m^M to reach 63 percent of the final value. We now give the following results, based on (7)-(9).

Under the conditions $l_{ER} \leq \bar{l_E}$ and $l_A \leq 100\%$, there exists a transfer function,

$$G_{DYN,u}(z) = \frac{g_L g_A T T_u^{-1}}{(z-1)(z-1+TT_u^{-1})},$$

from the control input $\Delta L_{ER}(z)$ to $U(z)$, where l_{ER} and u are dynamically related. Under the conditions $l_{ER} \leq \bar{l_E}$, $l_{th}^M(k) < l_A$, $l_{th}^O(k) < l_A$, and $l_{th}^{ATE}(k) < l_A$, the transfer functions

$$G_{DYN,m}^M(z) = \frac{g_L g_A g_m^M T (T_m^M)^{-1}}{(z-1)(z-1+T(T_m^M)^{-1})}$$
$$G_{DYN,m}^O(z) = \frac{g_L g_A g_m^O T (T_m^O)^{-1}}{(z-1)(z-1+T(T_m^O)^{-1})}$$
$$G_{DYN,ate}(z) = \frac{g_L g_A g_{ate} T T_{ate}^{-1}}{(z-1)(z-1+TT_{ate}^{-1})}$$

relate the control input $\Delta L_{ER}(z)$ to the controlled variables $M^M(z)$, $M^O(z)$, and $ATE(z)$, respectively.

As presented above, we have extended the STA model and the new model, DYN, captures additional dynamics of the controlled system, by giving more accurate time-domain relations between the control input δl_{ER} and the outputs u, m^M, m^O, and ate. We have also described how to compute the models parameters g_L, g_A, g_m^M, g_m^O, g_{ate}, T_u, T_m^M, T_m^O, and T_{ate}.

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 Performance Evaluation Goals

Considering the goal of our work, stated in Section 2, the two objectives of the performance evaluation are 1) to determine if the presented algorithms can provide QoS guarantees according to a QoS specification and 2) to determine the suitability of the proposed model DYN for describing the performance of RTDBs. Considering our first objective, we have studied and evaluated the behavior of the algorithms under various conditions, where a set of parameters have been varied. They are: 1) Load ($load$) as 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. 2) Execution time estimation error ($esterr$) as, 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. The second objective is investigated by comparing the controller tuned using DYN with the controller tuned using STA with regard to performance reliability and performance adaptation.

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. We have used a workload based on the analysis of the NYSE stock trades and commercial databases [23]. Update transactions occupy approximately 50 percent of the workload. In our experiments, one simulation run lasts for 10 minutes of simulated time. For all the performance data, we present the average of 10 simulation runs. We have derived 95 percent confidence intervals based on the samples obtained from each run and using the t-distribution [24]. The workload model of the update and user transactions is described as follows: We use the following notation, where the attribute x_i refers to the transaction T_i and $x_i[t_i]$ is associated with the subtransaction t_i of T_i.

Data and Update Transactions. The DB holds 1,000 temporal data objects (d_i), where each data object is updated by a stream ($Stream_i, 1 \leq i \leq 1{,}000$).[4] 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 actual execution time of an update is given by the normal distribution $N : (eet_i, \sqrt{eet_i})$. The average update value (av_i) of each $Stream_i$ is given by $U : (0, 100)$. The actual value (v_i) of an update is set according to $N : (av_i, av_i \times varfactor)$, where $varfactor$ is uniformly distributed in (0, 1). The deadline is set to $arrivaltime_i + p_i$. We define data error as the relative deviation between cv_i and v_i as given by $de_i = 100 \times \frac{|cv_i - v_i|}{|cv_i|}(\%)$.

User Transactions. Each $Source_i$ generates a transaction T_i, consisting of one mandatory subtransaction, m_i, and $|O_i|$ ($1 \leq |O_i| \leq 10$) optional subtransaction(s), $o_{i,j}$ ($1 \leq j \leq |O_i|$). $|O_i|$ is uniformly distributed between 1 and 10. The estimated (average) execution time ($eet_i[t_i]$) of the mandatory and the optional subtransactions is given by $U : (5ms, 15ms)$. The execution time estimation error factor $esterr$ is used to introduce execution time estimation error in the average execution time given by $aet_i[t_i] = (esterr + 1) \times eet_i[t_i]$. 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 $arrivaltime_i + eet_i \times slackfactor$. The slack factor is uniformly distributed according to $U : (20, 40)$. Transactions are evenly distributed in four classes representing error function orders of 0.5, 1, 2, and 5 (e.g., 25 percent of the transactions have an error order of 1).

We use the QoS specifications

4. In current automotive engine control units (ECUs), a typical RTDB consists of approximately 500 data objects with temporal constraints. Future ECUs with more functionality will increase the data volume further.

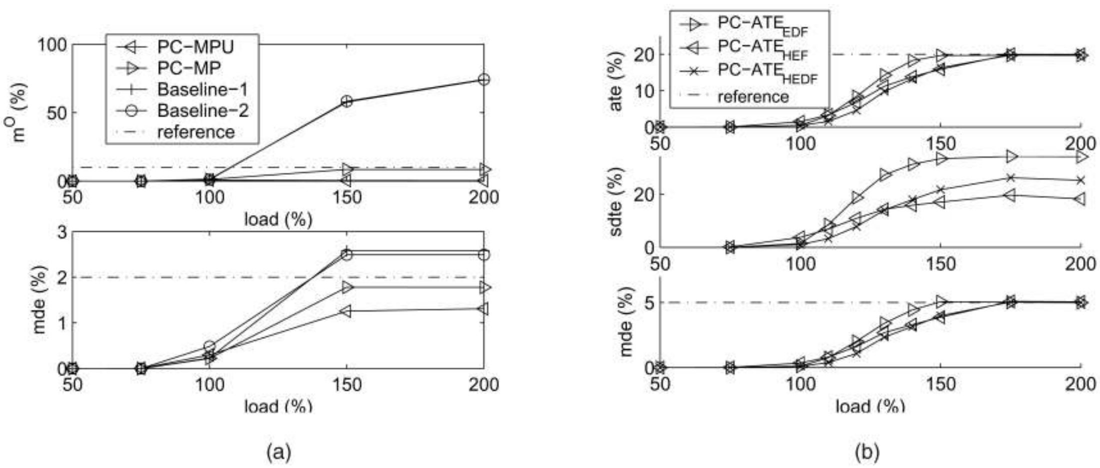

Fig. 4. Experiment 1: Average performance when varying load. (a) PC-MPU and PC-MP. (b) PC-ATE.

$$QoSSpecA = \{m_r^M = 1\%, m_r^O = 10\%, mde_r = 2\%,\\ T_s \leq 60s, M_p \leq 30\%\}$$

for PC-MPU and PC-MP, and

$$QoSSpecB = \{ate_r = 20\%, mde_r = 5\%, T_s \leq 60s, M_p \leq 30\%\}$$

for $\mathrm{PC\text{-}ATE_{EDF}}$, $\mathrm{PC\text{-}ATE_{HEF}}$, and $\mathrm{PC\text{-}ATE_{HEDF}}$.

5.3 Baselines

To the best of our knowledge, there has been no earlier work on techniques for managing data imprecision and transaction imprecision aimed at satisfying specific QoS or QoD requirements. For this reason, we have developed two baseline algorithms, Baseline-1 and Baseline-2, to study the impact of the workload on the system. In our performance evaluation, we also include $\mathrm{PC\text{-}ATE_{EDF}}$, which is working as a reference algorithm in addition to the baselines. We choose EDF since it is optimal in minimizing deadline misses and has well-known behavior. The algorithm outline of Baseline-1 and Baseline-2 is given below. Depending on the given QoS specification type, let v be either m^O or ate.

Baseline-1. If v (i.e., m^O or ate) is greater than its reference, the utilization is lowered by discarding more update transactions, i.e., increasing mde. Consequently, the preciseness of the data is adjusted based on v. mde is set according to $mde(k+1) = \min(\frac{v(k)}{v_r} mde_r, mde_r \times (M_p + 100))$. A simple AC is applied, where a transaction (T_i) is admitted if the estimated utilization of admitted subtransactions and eet_i is less or equal to 80 percent.

Baseline-2. To prevent a potential overshoot, we increase mde as soon as v is greater than zero. In Baseline-1, a significant change in mde may introduce oscillations in v. This is avoided in Baseline-2 by increasing and decreasing mde stepwise. If $v(k)$ is greater than zero, increase $mde(k)$ stepwise until $mde_r \times (M_p + 100)$ is reached (i.e., $mde(k+1) = \min(mde(k) + mde_{step}, mde_r \times (M_p + 100))$). If $v(k)$ 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.

5.4 Experiment 1: Results of Varying Load

We apply loads from 50 percent to 200 percent. The execution time estimation error factor is set to zero (i.e., $esterr = 0$). Controllers tuned using STA are used. Fig. 4a shows the performance of PC-MPU and PC-MP and Fig. 4b shows the performance of $\mathrm{PC\text{-}ATE_{HEF}}$, $\mathrm{PC\text{-}ATE_{HEDF}}$, and $\mathrm{PC\text{-}ATE_{EDF}}$. Dash-dotted lines indicate references. For clarity of presentation, we do not include the baselines in Fig. 4b.

m^M has been observed to be zero[5] for all four algorithms and, therefore, this has not been included in Fig. 4a. The specified miss percentage reference (m_r^M) has been set to 1 percent and this is not reached. This is due to the higher priority of mandatory subtransactions compared to optional subtransactions. According to our investigations, the miss percentage of mandatory subtransactions starts increasing when the miss percentage of optional subtransactions is over 90 percent. Consequently, since the miss percentage of optional subtransactions does not reach 90 percent, the miss percentage of mandatory subtransactions remains at zero.

Turning to m^O, the 95 percent confidence intervals for all algorithms are less than $[m^O - 3.4\%, m^O + 3.4\%]$. 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 percent. In the case of PC-MPU, m^O is near zero at loads 150 percent and 200 percent. 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 PC-MPU. The low miss percentage is due to the utilization controller attempting to reduce potential overshoots by reducing the utilization, which, in turn, decreases the miss percentage. PC-MP, on the other hand, shows a better performance. The average m^O at 150 percent and 200 percent is $8.5 \pm 0.1\%$, which is fairly close to m_r^O.

The 95 percent confidence intervals of ate are less than $[ate - 2.1\%, ate + 2.1\%]$. For Baseline-1 and Baseline-2, the ate increases as the load increases, violating the reference, ate_r, at loads exceeding 175 percent. In the case of $\mathrm{PC\text{-}ATE_{EDF}}$, ate reaches the reference at 150 percent of

5. We have not observed any deadline misses.

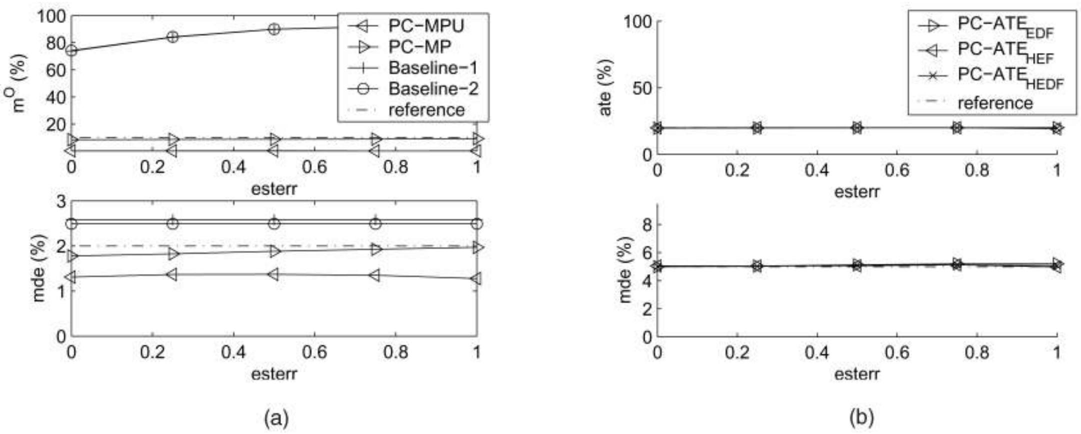

Fig. 5. Experiment 2: Average performance when varying execution time estimation error. (a) PC-MPU and PC-MP. (b) PC-ATE.

applied load. For PC-ATE$_{\mathrm{HEF}}$ and PC-ATE$_{\mathrm{HEDF}}$, ate reaches the reference at 175 percent. All PC-ATE algorithms provide a robust performance since ate is kept at the specified reference during overloads.

Considering $sdte$, the 95 percent confidence intervals for all algorithms are less than $[sdte - 1.92\%, sdte + 1.92\%]$. For all algorithms, $sdte$ increases as load and ate increase. At 200 percent load, the corresponding $sdte$ for PC-ATE$_{\mathrm{EDF}}$, PC-ATE$_{\mathrm{HEDF}}$, and PC-ATE$_{\mathrm{HEF}}$ is 34.1 percent, 25.2 percent, and 18.2 percent, respectively. Consequently, deviation of transaction error is minimized when HEF scheduling is used. It is worth mentioning that, under HEF scheduling, $sdte$ is less as resources are allocated with regard to transaction errors, while, under EDF scheduling, resources are distributed with regard to deadlines. Under EDF scheduling, two transactions with deadlines near each other may receive different amounts of resources and, hence, they may terminate with a significant deviation in transaction error, while, in the same case, under HEF scheduling, the resources are divided such that both transactions terminate with transaction errors close to each other.

The 95 percent confidence intervals of mde are less than $[mde - 0.13\%, mde + 0.13\%]$. Starting with Fig. 4a, the average mde for Baseline-1 and Baseline-2 violates the reference mde set to 2 percent. In contrast, in the case of PC-MPU, 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 a 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. PC-MP provides an average mde closer to mde_r, given by $1.78 \pm 0.024\%$ at loads 150 percent and 200 percent. However, mde does not reach mde_r since mde is set according to m^O (which does not reach m^O_r). Turning to Fig. 4b, the 95 percent confidence intervals of mde are less than $[mde - 0.5\%, mde + 0.5\%]$. The average mde for Baseline-1 and Baseline-2 violates the reference mde set to 5 percent at applied loads of 175 percent and 130 percent, respectively. In contrast, in the case of PC-ATE$_{\mathrm{EDF}}$, PC-ATE$_{\mathrm{HEF}}$, and PC-ATE$_{\mathrm{HEDF}}$, mde is at the reference during overloads.

The experiments have shown that PC-MPU produces a miss percentage significantly lower than the specified miss percentages and, hence, it does not fully satisfy the given QoS specification. PC-MP, on the other hand, produces miss percentages close to the given references and, consequently, the given QoS specification is satisfied with regard to steady-state performance. We have also seen that PC-ATE$_{\mathrm{HEF}}$, PC-ATE$_{\mathrm{HEDF}}$, and PC-ATE$_{\mathrm{EDF}}$ are robust against varying applied loads. Moreover, PC-ATE$_{\mathrm{HEF}}$ outperforms the other algorithms with regard to QoS fairness of admitted transactions.

5.5 Experiment 2: Results of Varying esterr

We apply 200 percent load and vary the execution time estimation error according to $esterr$ = 0.00, 0.25, 0.50, 0.75, and 1.00. Controllers tuned using STA are used. Fig. 5a and Fig. 5b show the performance of PC-MPU, PC-MP, PC-ATE$_{\mathrm{HEF}}$, PC-ATE$_{\mathrm{HEDF}}$, and PC-ATE$_{\mathrm{EDF}}$. Dash-dotted lines indicate references.

As in Experiment 1, m^M is zero for all approaches and $esterr$. Turning to m^O, the 95 percent confidence intervals for all algorithms are less than $[m^O - 2.7\%, m^O + 2.7\%]$. As expected, Baseline-1 and Baseline-2 do not satisfy the QoS specification, whereas PC-MPU and PC-MP are insensitive against varying $esterr$, as m^O and mde do not change considerably when varying $esterr$. Studying ate, we note that Baseline-1 and Baseline-2 do not satisfy the QoS specification as ate reaches 52 percent when $esterr$ equals 1. The 95 percent confidence intervals for PC-ATE algorithms are less than $[ate - 0.3\%, ate + 0.3\%]$. PC-ATE$_{\mathrm{EDF}}$, PC-ATE$_{\mathrm{HEF}}$, and PC-ATE$_{\mathrm{HEDF}}$ are insensitive against varying $esterr$ as ate and mde do not change with varying $esterr$. From the above, we can conclude that PC-MPU, PC-MP, PC-ATE$_{\mathrm{HEF}}$, and PC-ATE$_{\mathrm{HEDF}}$ are insensitive to changes to execution time estimation and, hence, they can easily adapt when accurate runtime estimates are not known.

5.6 Experiment 3: Transient Performance

Studying the average performance is often not enough when dealing with dynamic systems and, therefore, we study the transient performance of PC-MPU, PC-MP, PC-ATE$_{\mathrm{HEF}}$, PC-ATE$_{\mathrm{HEDF}}$, and PC-ATE$_{\mathrm{EDF}}$. We set $load$ to 200 percent and $esterr$ to 1.

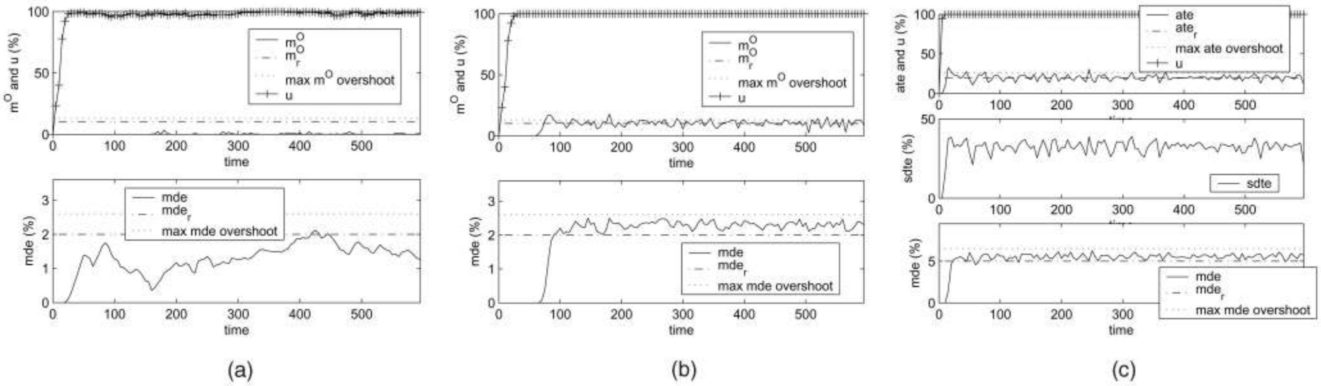

Fig. 6. Experiment 3: Transient performance. (a) PC-MPU. (b) PC-MP. (c) $\text{PC-ATE}_{\text{EDF}}$.

TABLE 1
J_s and J_a with 95 Percent Confidence Intervals

Model	STA		DYN	
Performance Metric	J_s	J_a	J_s	J_a
$\text{PC-ATE}_{\text{EDF}}$	139.62 ± 27.91	9.58 ± 1.04	79.11 ± 5.51	7.12 ± 0.22
$\text{PC-ATE}_{\text{HEF}}$	77.78 ± 7.68	7.17 ± 0.38	71.31 ± 6.24	6.82 ± 0.36
$\text{PC-ATE}_{\text{HEDF}}$	105.76 ± 17.14	8.46 ± 0.61	84.67 ± 4.15	7.58 ± 0.20

5.6.1 Results of Controllers Tuned Using STA

Fig. 6a, Fig. 6b, and Fig. 6c show the transient behavior of PC-MPU, PC-MP, and $\text{PC-ATE}_{\text{EDF}}$ with controllers tuned using STA. We refer to Table 1 for a summary of the performance of $\text{PC-ATE}_{\text{HEF}}$, $\text{PC-ATE}_{\text{HEDF}}$, and $\text{PC-ATE}_{\text{EDF}}$ with controller tuned using STA and DYN. The dash-dotted line indicates the reference, while the dotted line indicates maximum overshoot. For all algorithms, m^O and ate overshoots decay faster than 60s, which is less than the settling time requirement given in the QoS specification. Starting with PC-MPU, 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 δl_{ER} 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 PC-MPU, u_r is increased only 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 δl_{ER} and, hence, u_r being increased too late. Consequently, the negative δl_{ER} leads to an increase in mde. PC-MP 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. This is due to disturbances in load due to data conflicts, resulting in restarts or aborts of transactions, and inaccurate execution time estimations. The highest overshoot for $\text{PC-ATE}_{\text{EDF}}$ has been noted to 42.38 percent at time 15. For $\text{PC-ATE}_{\text{HEF}}$, the highest overshoot was noted to 32.31 percent at time 15 and, finally, the highest overshoot for $\text{PC-ATE}_{\text{HEDF}}$ was observed to be 37.11 percent at time 15. 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 runtime estimates contribute to disturbances in an RTDB, complicating the control of ate (note that we have set $esterr$ to one).

From the discussions in Section 4.6.2, we understood that, 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 [14]. Given this, we can conclude that, under EDF scheduling, we should observe more oscillations in ate than compared with HEF scheduling, which is consistent with the data presented in Table 1. We recall from Section 4.1 that a decrease in J_s and J_a implies an improvement in control of performance and QoS. As we can see from Table 1, $\text{PC-ATE}_{\text{HEF}}$ produces fewer ate oscillations around ate_r than $\text{PC-ATE}_{\text{EDF}}$. Further, $\text{PC-ATE}_{\text{HEF}}$ is less prone to overshoot.

5.6.2 Results of Controllers Tuned Using DYN

For simplicity, we refer to controllers tuned using the model STA as STA controllers and controllers tuned using the model DYN as DYN controllers. The control signal δl_{ER} of the STA controller varies between -13 percent to 30 percent, whereas the control signal of the DYN controller varies between -1 percent to 7 percent. In other words, the STA controller controls the RTDB more aggressively, resulting in greater deviations between ate and ate_r. Since the STA controller computes δl_{ER} according to a statical relation between l_{ER} and ate, the controller does not consider the dynamics of the controlled system. Hence, it does not consider that, given a certain δl_{ER}, it may take a few samples until the corresponding ate is reached. Therefore, the STA controller persists with changing the load until the desired ate_r is reached, at which point, ate overshoots due to aggressive control. This is handled more efficiently with the DYN controller as it is tuned according to a dynamic model and, hence, the controller is more gentle when controlling the system.

Table 1 gives a summary of the performance of the controllers with respect to J_s and J_a, which show how closely the controlled variable ate follows its reference ate_r. The performance of PC-ATE$_{\mathrm{EDF}}$ and PC-ATE$_{\mathrm{HEDF}}$ is significantly improved when using DYN for tuning controllers. However, the performance improvement for PC-ATE$_{\mathrm{HEF}}$ is too small to be considered significant. From the results and discussions in this section, we learned that the scheduling policy of EDF induces a certain delay between a change in the load and ate, whereas the delay is less for the HEF scheduling policy. The delay caused by EDF is compensated for by the DYN controller as opposed to the STA controller, which does not compensate for delays, and, hence, we achieve almost the same performance as PC-ATE$_{\mathrm{HEF}}$, where a DYN controller is used. The experiments show that feedback controllers for PC-ATE$_{\mathrm{EDF}}$ and PC-ATE$_{\mathrm{HEDF}}$ tuned using DYN outperform controllers tuned using STA.

5.7 Summary of Results and Discussion

Our experiments show that PC-MPU, PC-MP, PC-ATE$_{\mathrm{HEF}}$, and PC-ATE$_{\mathrm{HEDF}}$ are robust against inaccurate execution time estimations as m^O, ate, and mde remain unaffected for varying execution time estimation errors. PC-MPU keeps m^O less than its reference and is able to efficiently suppress deadline miss percentage overshoots. PC-MPU should, therefore, be applied to RTDBs where deadline miss percentage overshoots cannot be tolerated. PC-MP provides an m^O near its reference, but generates overshoots greater than the maximum allowed overshoot. The experiments show that PC-MP is particularly useful when m^O must be near its reference, but where overshoots are accepted. It was observed that PC-ATE$_{\mathrm{HEF}}$ provides a lower $sdte$ compared to other algorithms, lowering the deviation of transaction error among terminated transactions. This property is useful in applications where QoS fairness among transactions is emphasized. Further, we saw that the control performance is greatly enhanced when using DYN as compared to STA [11]. This aids a system operator in configuring RTDBs that are highly reactive to changes in applied load and execution time estimation errors, providing increased performance reliability and enhanced QoS adaptation.

6 Related Work

Liu et al. [3] and Hansson et al. [25] presented algorithms for minimizing the total error and total weighted error of a set of tasks. Their approaches require the knowledge of accurate processing times of the tasks, which is often not available in RTDBs. Bestavros and Nagy have presented approaches for managing the performance of RTDBs, where the execution time of the transactions is unknown [26]. Each transaction contributes with a profit when completing successfully. An admission controller is used to maximize the profit of the system. The work by Liu et al. Hansson et al., and Bestavros and Nagy focuses on maximizing or minimizing a performance metric (e.g., profit). These previous approaches cannot be applied to our problem since, in our case, we want to control a set of performance metrics such that they converge toward their references as given by a QoS specification.

Lu et al. have presented a feedback control scheduling framework where they propose algorithms for managing the miss percentage and/or utilization [11]. In comparison to the proposed algorithms in this paper, they do not address the problem of maximizing QoS fairness among admitted tasks. Further, their model statically relates estimated requested load, deadline miss ratio, and utilization. In this paper, we have extended their model to capture the dynamic relationships between these variables. Parekh et al. use feedback control scheduling to control the length of a queue of remote procedure calls (RPCs) arriving at a server [10]. In contrast to their work we have chosen deadline miss percentage, utilization, and average transaction error as the controlled variables.

Kang et al. use feedback control scheduling to manage the deadline miss ratio of transactions and freshness requirements of data objects [23]. In contrast to the work by Kang et al., we have, in this paper, described a set of algorithms for managing QoS based on feedback control scheduling and imprecise computation, where QoS is defined in terms of transaction and data preciseness. Further, we have introduced QoS fairness, a set of novel QoD management algorithms including two new scheduling algorithms (HEF and HEDF), and a dynamic model giving a more accurate description of the controlled system. Kuo and Ho have introduced the notion of similarity [27], where a similarity relation gives whether two transactions produce similar results. However, the work by Kuo and Ho does not address unknown workload characteristics.

Davidson and Watters proposed a method for generating monotonically improving answers in RTDBs and distributed RTDBs [28]. A query processor, APPROXIMATE [7], produces an approximate answer if there is not enough time available. The accuracy of the improved answer increases monotonically as the computation time increases. The relational database system proposed in [29], can produce approximate answers to queries within certain deadlines. Lee et al. studied the performance of real-time transaction processing in broadcast environments [30]. In contrast to the approaches above, we have introduced precision at the transaction level and the data object level and manage QoS using feedback control.

7 Conclusions and Future Work

In this paper, we have argued for the need for increased adaptability of applications that provide real-time data services, while operating in highly unpredictable environments. Typically, transactions cannot be subject to exact schedulability analysis given the lack of a priori knowledge of the workload, making transient overloads inevitable.

Furthermore, these systems are becoming larger and more complex and, at the same time, they are being used in applications where performance guarantees are needed. To address these issues, we have proposed a QoS-sensitive approach based on imprecise computation [3] applied on transactions and data objects.

The expressive power of our QoS specification model allows a database operator to specify not only the desired steady-state performance, representing the nominal system operation, but also the transient-state performance describing the worst-case system performance and system adaptability in the face of unexpected failures or load variation. To provide QoS guarantees without a priori knowledge of the workload, we apply feedback control, where the performance of the RTDB is continuously monitored and modified according to the given QoS specification. The algorithms PC-MPU and PC-MP address QoS specifications given in terms of deadline miss percentage of optional subtransactions, while PC-ATE$_{HEF}$ and PC-ATE$_{HEDF}$ address specifications based on the notion of transaction error. Our performance evaluation shows that, given a QoS specification, the four algorithms PC-MPU, PC-MP, PC-ATE$_{HEF}$, and PC-ATE$_{HEDF}$ give a robust and controlled behavior of RTDBs in terms of transaction and data precision, even for transient overloads and with inaccurate runtime estimates of the transactions. The proposed algorithms outperform the baseline algorithms and PC-ATE$_{EDF}$, where transactions are scheduled with EDF and feedback control.

We will extend our work to manage QoS of derived data and service differentiation. In this work, we have considered the milestone approach to imprecise computation. We plan to apply other types of imprecise computation techniques.

Acknowledgments

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 US National Science Foundation grants IIS-0208578 and CCR-0329609. This work was performed when Jörgen Hansson's primary affiliation was Linköping University.

References

[1] D. Wu, Y.T. Hou, W.Z.Y.-Q. Zhang, and J.M. Peha, "Streaming Video over the Internet: Approaches and Directions," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 11, no. 3, pp. 282-300, Mar. 2001.

[2] S.-Y. Choi and A.B. Whinston, "The Future of e-Commerce: Integrate and Customize," *Computer,* vol. 32, no. 1, pp. 133-134, Jan. 1999.

[3] J.W.S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise Computations," *Proc. IEEE,* vol. 82, Jan. 1994.

[4] S. Zilberstein and S.J. Russell, "Optimal Composition of Real-Time Systems," *Artificial Intelligence,* vol. 82, nos. 1-2, pp. 181-213, 1996.

[5] X. Chen and A.M. K. Cheng, "An Imprecise Algorithm for Real-Time Compressed Image and Video Transmission," *Proc. Int'l Conf. Computer Comm. and Networks (ICCCN),* 1997.

[6] M. Yannakakis, "Perspectives on Database Theory," *Proc. Ann. Symp. Foundations of Computer Science,* 1995.

[7] S.V. Vrbsky and J.W.S. Liu, "APPROXIMATE—A Query Processor that Produces Monotonically Improving Approximate Answers," *IEEE Trans. Knowledge and Data Eng.,* vol. 5, no. 6, pp. 1056-1068 Dec. 1993.

[8] G.C. Buttazzo, *Hard Real-Time Computing Systems.* Kluwer Academic, 1997.

[9] J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury, *Feedback Control of Computing Systems.* Wiley-IEEE Press, 2004.

[10] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using Control Theory to Achieve Service Level Objectives in Performance Management," *Real-Time Systems,* vol. 23, nos. 1/2, July/Sept. 2002.

[11] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms," *Real-Time Systems,* vol. 23, nos. 1/2, July/Sept. 2002.

[12] M. Amirijoo, J. Hansson, and S.H. Son, "Algorithms for Managing QoS for Real-Time Data Services Using Imprecise Computation," *Proc. Conf. Real-Time and Embedded Computing Systems and Applications (RTCSA),* 2003.

[13] M. Amirijoo, J. Hansson, and S.H. Son, "Error-Driven QoS Management in Imprecise Real-Time Databases," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* 2003.

[14] G.F. Franklin, J.D. Powell, and M. Workman, *Digital Control of Dynamic Systems,* third ed. Addison-Wesley, 1998.

[15] *Advances in Real-Time Systems,* S.H. Son, ed., pp. 463-486. Prentice Hall, 1995.

[16] T. Gustafsson and J. Hansson, "Data Management in Real-Time Systems: A Case of On-Demand Updates in Vehicle Control Systems," *Proc. Real-Time Applications Symp. (RTAS),* 2004.

[17] K. Ramamritham, "Real-Time Databases," *Int'l J. Distributed and Parallel Databases,* no. 1, 1993.

[18] L.V. Fausett, *Numerical Methods: Algorithms and Applications.* Prentice Hall, 2003.

[19] N.J. Higham, *Accuracy and Stability of Numerical Algorithms.* SIAM, 1996.

[20] J. Chung and J.W. S. Liu, "Algorithms for Scheduling Periodic Jobs to Minimize Average Error," *Proc. Real-Time Systems Symp. (RTSS),* 1988.

[21] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Trans. Database System,* vol. 17, pp. 513-560, 1992.

[22] K.J. Åström and B. Wittenmark, *Adaptive Control,* second ed. Addison-Wesley, 1995.

[23] K.-D. Kang, S.H. Son, and J.A. Stankovic, "Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 10, pp. 1200-1216, Oct. 2004.

[24] M.H. DeGroot and M.J. Schervish, *Probability and Statistics,* third ed. Addison-Wesley, 2002.

[25] J. Hansson, M. Thuresson, and S.H. Son, "Imprecise Task Scheduling and Overload Managment Using OR-ULD," *Proc. Conf. Real-Time Computing Systems and Applications (RTCSA),* 2000.

[26] A. Bestavros and S. Nagy, "Value-Cognizant Admission Control for RTDB Systems," *Proc. Real-Time Systems Symp. (RTSS),* pp. 230-239, 1996.

[27] T.-W. Kuo and S.-J. Ho, "Similarity-Based Load Adjustment for Static Real-Time Transaction Systems," *IEEE Trans. Computers,* vol. 49, pp. 112-126, 2000.

[28] S. Davidson and A. Watters, "Partial Computation in Real-Time Database Systems," *Proc. Workshop Real-Time Software and Operating Systems,* 1988.

[29] W. Hou, G. Ozsoyoglu, and B.K. Taneja, "Processing Aggregate Relational Queries with Hard Time Constraints," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1989.

[30] V. Lee, K. Lam, S.H. Son, and E. Chan, "On Transaction Processing with Partial Validation and Timestamps Ordering in Mobile Broadcast Environments," *IEEE Trans. Computers,* vol. 51, no. 10, pp. 1196-1211, Oct. 2002.

Mehdi Amirijoo is a PhD student in the Department of Computer and Information Science at Linköping University, Sweden. He received the MSc degree in computer science and engineering from Linköping University in 2002. His interests include real-time data services, scheduling, real-time databases, QoS management, automatic control, imprecise computation techniques, and sensor networks. He received the 2003 best MSc thesis award issued by SNART (the Swedish National Real-Time Association). He served as a member of the local organization committee for the International Conference on Real-Time and Embedded Computing Systems and Applications in 2004.

Jörgen Hansson received the BSc and MSc degreed from the University of Skövde, Sweden, in 1992 and 1993, respectively. He received the PhD degree in 1999 from Linköping University, Sweden, with which he is also affiliated as an associate professor. He is a senior member of the technical staff at the Software Engineering Institute at Carnegie Mellon University. His current research interests include real-time systems and real-time database systems and he has written 40 papers and edited two books in these areas. His research has focused on techniques and algorithms for ensuring robustness and timeliness in real-time applications that are prone to transient overloads, mechanisms and architectures for handling increasing amounts of data in real-time systems, and algorithms to ensure data quality in real-time systems. His current research interests include resource management, techniques and methodologies for data repositories functioning in real-time and embedded computing systems, adaptive overload management, and component-based software architectures for embedded and real-time systems. He is a member of the IEEE.

Sang Hyuk Son received the BSc degree in electronics engineering from Seoul National University, the MSc degree from the Korea Advanced Institute of Science and Technology (KAIST), and the PhD degree in computer science from the University of Maryland, College Park, in 1986. He is a professor in the Department of Computer Science at the University of Virginia. He has been a visiting professor at KAIST, City University of Hong Kong, Ecole Centrale de Lille in France, and Linköping University in Sweden. His current research interests include real-time computing, data services, QoS management, wireless sensor networks, and information security. Dr. Son served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* for 1998-2001 and currently serves as an associate editor for the *Real-Time Systems Journal* and *Journal of Business Performance Management*. He has served as a guest editor for the *ACM SIGMOD Record*, *IEEE Transactions on Software Engineering*, *Control Engineering Practice*, *Journal of Integrated Computer-Aided Engineering*, and *NETNOMICS* on special issues covering real-time systems, real-time databases, and e-commerce. He has been on the executive board of the IEEE Technical Committee on Real-Time Systems since 2003. He has published more than 200 technical papers and served as the program chair or general chair of several real-time and database conferences, including the IEEE Real-Time Systems Symposium, IEEE Conference on Parallel and Distributed Systems, International Workshop on Real-Time Database Systems, and IEEE Conference on Electronic Commerce. He received the Outstanding Contribution Award from the IEEE Conference on Real-Time and Embedded Computing Systems and Applications in 2004. He is a senior member of the IEEE and a member of the IEEE Computer Society.