QoS Management in Replicated Real-Time Databases *

Yuan Wei Sang H. Son John A. Stankovic K.D. Kang †
Department of Computer Science
University of Virginia
Charlottesville, Virginia, 22904-4740
E-mail: {yw3f, son, stankovic, kk7v}@cs.virginia.edu

Abstract

Providing quality-of-service guarantees for data services in a distributed environment is a challenging task. The presence of multiple sites in distributed environments raises issues that are not present in centralized systems. The transaction workloads in distributed real-time databases may not be balanced and the transaction access patterns may be time-varying and skewed. Data replication is an effective method to help database systems meet the stringent temporal requirements of real-time applications. We have designed an algorithm that provides quality-of-service guarantees for data services in distributed real-time databases with full replication of temporal data. The algorithm consists of heuristic feedback-based local controllers and global load balancers (GLB) working at each site. The local controller controls the admission process of incoming transactions. The global load balancers collect the performance data from other nodes and balance the system-wide workload. The simulation results show that the new algorithm successfully balances the workloads in distributed real-time databases and provides tight transaction miss ratio guarantees under various transaction workloads.

1 Introduction

There is a growing need for real-time data services in distributed environments. For example, in ship-board control systems, data is shared in a distributed real-time database embedded in the ship [7]; in traffic control and agile manufacturing, transactions should be processed within their deadlines using fresh (temporally consistent) data that reflects the current real-world status [14]. For many of these applications, providing real-time data services in distributed environments is essential. The issues involved in providing predictable real-time data services in centralized database systems have been studied and the results are promising [11] [12]. However, we are not aware of research results for providing data services with *Quality-of-Service*(QoS) guarantees in distributed real-time database environments.

In distributed environments, it is challenging to provide data services with QoS guarantees while still meeting transaction temporal requirements needed by different real-time applications. One of the reasons is that a distributed system's performance depends on the workload distribution. Transaction workload fluctuations cause uneven distribution of the workload among the sites even if on the average, all sites receive a similar workload. A site may experience transient overloads caused by burst arrivals. Moreover, transaction access patterns may be time-varying and skewed. With skewed access patterns, many transactions may access a set of data items stored only at a specific site, overloading the site. In addition, the overloading point also changes dynamically. The QoS management algorithm must deal with these situations and guarantee the specified QoS requirements.

Data replication can help database systems meet the stringent temporal requirements of real-time applications. Data replication greatly improves the system performance when the majority of operations on data replicas are read operations. It also helps avoid the data access skew problem mentioned above because transactions can access locally available data replicas.

Load balancing is a technique to provide better QoS in distributed systems. By transferring transactions from highly overloaded sites to the less loaded sites, the overload situation is alleviated and the QoS of transactions are maintained. In this paper, we study the issues involved in providing QoS in distributed real-time databases and propose a QoS management algorithm that controls and balances the workloads in distributed real-time database systems. The algorithm consists of local feedback controllers and heuristic feedback-based global load balancers (**FB-GLB**) running at each site.

*This work was supported, in part, by NSF grants EIA-9900895 and IIS-0208758.

†K.D. Kang is now with the Department of Computer Science, State University of New York at Binghamton, Binghamton, NY 13902-6000, USA.

The local controller controls the admission process of the incoming transaction workload. The global load balancers collect the performance data from other sites and balance the workloads. A simulation study shows that strict QoS requirements are guaranteed under a wide range of workloads.

The rest of the paper is organized as follows. Section 2 describes our real-time database model. The real-time database QoS management architecture is presented in Section 3. In Section 4, the algorithm for distributed environments is described. Section 5 shows the details of the simulation settings and presents the evaluation results. Related work is discussed in Section 6 and Section 7 concludes the paper and discusses future work.

2 Real-time Database Model and Performance Metrics

Before we present our QoS management algorithm, we first introduce the distributed real-time database system model and the performance metrics considered in this paper.

2.1 Real-time Database Model

In this paper, we consider a distributed real-time database system which consists of a group of main memory real-time database systems connected by a Local Area Network(LAN). Due to the high performance of main memory accesses and the decreasing main memory cost, main memory databases have been increasingly used for data management in real-time applications [3]. We focus our study on medium scale distributed databases (in the range of 5 to 10 sites), since the load balancers need full information from every sites to make accurate decisions. Several applications that require distributed real-time data services fall in that range. For example, a ship-board control system which controls navigation and surveillance consists of 6 distributed control units and 2 general control consoles located throughout the platform and linked together via a ship-wide redundant Ethernet to share distributed real-time data and coordinate the activities [7]. We leave it as the future work to make our solution applicable to large scale distributed real-time applications with 100s of sites involved, using only a partial information from a subset of sites.

In this paper, we apply firm deadline semantics in which transactions add value to the application only if they finish within their deadlines. Hence, tardy transactions (transactions that have missed their deadlines) are aborted upon their deadline miss. Firm deadline semantics are common in several real-time database applications. A late commit of a real-time transaction may incur the loss of profit or control quality, resulting in wasted system resources. Our objective is to provide QoS guarantees for real-time data services in those applications.

2.1.1 Data Composition

In our system model, data objects are divided into two types, namely, *temporal data* and *non-temporal data*. Temporal data are the sensor data from physical world. In ship-board control applications, they could be ship maneuvering data such as position, speed and power; in stock trading, they could be real-time stock prices. Temporal data objects have *validity interval*s and are updated by periodic sensor update transactions. Non-temporal data do not change dynamically with time. Thus they do not have validity intervals and there are no periodic system updates associated with them.

In our distributed real-time database system model, a local site is called a *node*. Each node hosts a set of temporal data objects and non-temporal objects. The node is called the *primary node* for those data objects. Each node also maintains a set of replicas of temporal data objects hosted by other nodes. The fresh value of temporal data objects are periodically submitted from outside to their primary nodes and propagated to the replicas. In our replication model, temporal data objects are fully replicated and the replicas are updated as soon as the fresher data are available. Non-temporal data objects are not replicated because replicating non-temporal data objects will not improve the system performance if the majority of operations are not read.

2.1.2 Transaction Model

In our system, transactions are divided into two types, *system update transactions* and *user transactions*. System update transactions are temporal data (sensor data) update transactions and temporal data replica update transactions. User transactions are queries or updates from applications. User transactions are divided to different *service class*es, e.g., class 0, 1 and 2. The lower the service class number, the higher the priority the transaction has during the execution. Class 0 is the service class that has the best quality of service guarantee.

Transactions are represented as a sequence of *operations* on data objects. The operation of a system update transaction is always *write*. For user transactions, the operation on non-temporal data objects could be *read* or write while the operation on temporal data could only be read. There is a certain execution time associated with each operation and the execution time of a transaction is the sum of the execution time of all its operations.

Operations of one transaction are executed in *sequential* fashion. One operation can not be executed unless all previous operations are finished.

2.2 Major Performance Metric

In our distributed real-time database system model, the main performance metric is *per-class deadline miss ratio*. The Miss Ratio for service class i is defined as:

$$MR_i = 100 \times \frac{\#tardy_i}{\#tardy_i + \#timely_i} (\%)$$

where $\#tardy_i$ and $\#timely_i$ represent the number of class i transactions that have missed and met their deadlines, respectively. The DBA (Database Administrator) can specify a tolerable miss ratio threshold (e.g., 2%), for a specific class of real-time transactions. Since database workloads and access patterns of transactions vary dynamically, it is reasonable to assume that some deadline misses are inevitable. A few deadline misses are considered acceptable unless they exceed the specified tolerance threshold. To guarantee QoS, an admission control is applied, and hence the deadline miss ratio is accounted for admitted transactions only.

2.3 Other Performance Metrics

In addition to the deadline miss ratio, we use other performance metrics to measure the system's performance.

2.3.1 Transient Performance Metrics

Long-term performance metrics such as average miss ratio are not sufficient for the performance specification of dynamic systems in which the system performance can be time-varying. For this reason, transient performance metrics such as overshoot and settling time are adopted from control theory for a real-time system performance specification [19]:

- *Miss Ratio Overshoot* ($MROS_i$) is the maximum transient miss ratio of class i transactions.
- *Settling time* (t_s) is the time for the transient miss ratio overshoot to decay and reach the steady state where the miss ratios are below the specified average values.

2.3.2 System Resources Utilization and Throughput

In our main memory database model, the CPU time is the main system resource for consideration. Using the system throughput, we show that our algorithm does not sacrifice the transaction throughput to provide the QoS guarantees.

- *CPU Utilization:* The CPU time utilization at each individual node.
- *Throughput(TP):* The number of completed transactions per second.

2.4 QoS Specifications

The transactions at each node are divided into several service classes. Each service class has certain QoS specifications. In this paper, we consider the following QoS specification as an example to illustrate the applicability of our approach for service differentiation.

$$QoS_{spec} = \{(MR_{QoS_0} \leq 1\%, MROS_{QoS_0} \leq 2\%, t_s \leq 50 seconds), (MR_{QoS_1} \leq 10\%), (MR_{QoS_2} = best-effort)\}$$

Note that this specification requires that the average miss ratio is below 1% for Class 0. In the ship-board system, class 0 transactions relate to tracking important targets. Due to the environmental uncertainty, 100% guarantees are not possible. We also set $MROS_0 \leq 2\%$, therefore, a miss ratio overshoot of class 0 transactions should not exceed 2%, and the overshoot should decay within 50 seconds. The average miss ratio should be below 10% for Class 1. Class 1 transactions are those that track less important targets, such as "friendly" targets. The best-effort service is specified for Class 2. Class 2 transactions include environmental monitoring and certain display activities.

In our previous work [11], we presented an approach for service differentiation in a centralized real-time database system. In this paper, we extend the feedback-based miss ratio control to distributed real-time databases. It is challenging to provide average and transient miss ratio guarantees in distributed environments, while differentiating real-time data services among the service classes.

3 System Architecture

The system architecture of one node is shown in Fig. 1. The real-time database system consists of an admission controller, a scheduler, a transaction manager, and blocked transaction queues. A local system performance monitor, a local controller and a global load balancer are added to the system for QoS management purpose. In Fig. 1, the solid arrows represent the transaction flows in the system and the dotted arrows represent the performance and control information flows in the system.

The system performance statistics are collected periodically by the transaction manager. At each sampling period, the local monitor samples the system performance data from the transaction manager and sends it

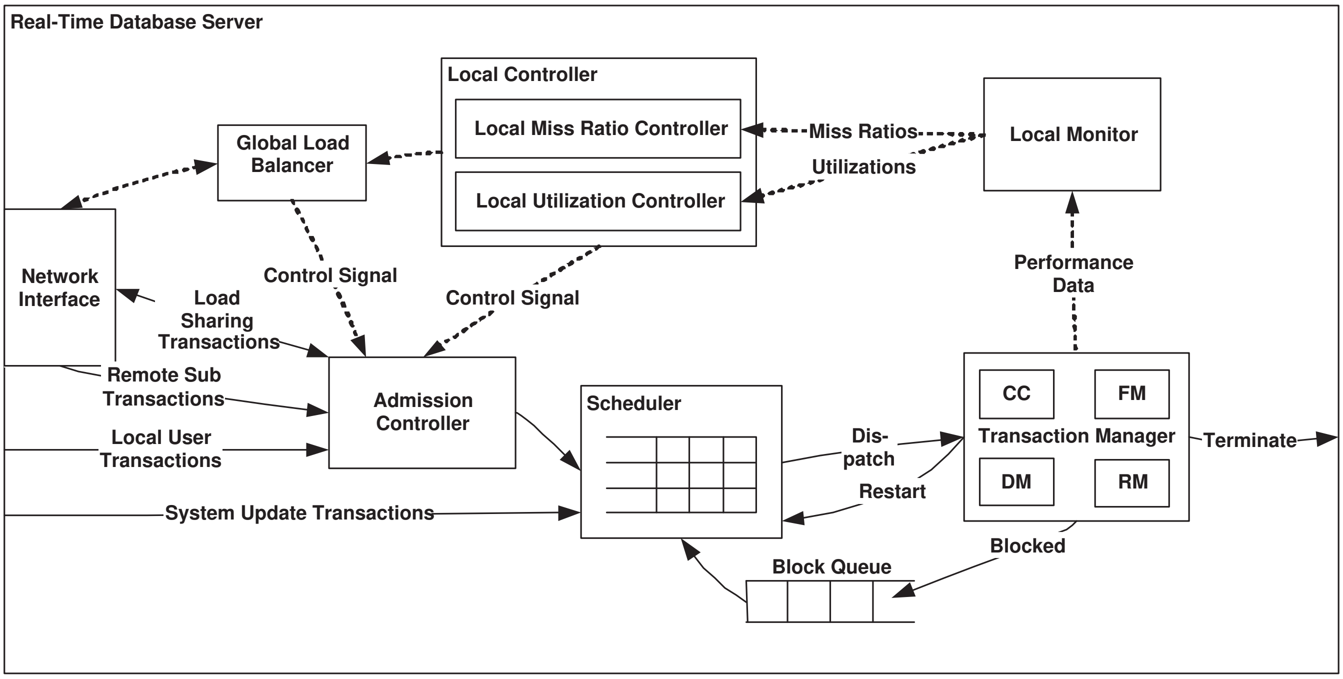

Figure 1. Real-time Database Architecture for QoS Guarantee

to the local controller. The local miss ratio controller and utilization controller generate local control signals based on the sampled local miss ratio and utilization data. The details of the controller are discussed in the next section.

The admission controller is used to avoid overloading the system. It is based on *estimated CPU utilization* and the *target utilization* set point. At each sampling period, the target utilization parameter is set by the local controller. The estimated execution time of an admitted transaction is credited to the estimated CPU utilization. Transactions are rejected if the estimated CPU utilization is higher than the target utilization set by the controller. To provide better services to transactions of higher service classes, priority-aware admission control is used, i.e., all arrived class 0 transactions are admitted to the system. The underlying assumption is that the system should be designed with sufficient capacity to handle a significant number of incoming transactions of class 0.

The transaction manager consists of a concurrency controller (CC), a freshness manager (FM), a data manager (DM) and a replica manager (RM). For concurrency control, we use 2PL-HP (Two Phase Locking - High Priority) [2]. 2PL-HP is selected since it is free of a priority inversion and is shown to work well in real-time databases.

During the transaction execution, if it needs temporal data hosted by other nodes, the transaction manager uses the local copy. If it needs to access non-temporal data hosted by other nodes, the transaction manager sends sub-transaction initiation request to the primary node of the data. The remote node then sets up a sub-transaction, which executes on behalf of the original transaction. The two-phase commit protocol is used to ensure the serializability of concurrent transactions.

The FM checks the freshness before accessing a data item using the corresponding *absolute validity interval (avi)*. It blocks a user transaction if the target data item is stale. The blocked transaction(s) is resumed and transferred from the block queue to scheduler as soon as the corresponding data object is updated. FM also checks the freshness of accessed data just before a transaction commits. If the accessed data item is stale, the transaction is restarted. In this way, the data objects accessed by committed transactions are always 100% fresh at commit time.

The user transactions are scheduled in one of multi-level queues according to their service classes. A fixed priority is applied among the multi-level queues. A transaction in a low priority queue is scheduled to run only when there are no ready transactions at the higher priority queues. A low priority transaction is preempted upon arrival of a high priority transaction. Within each queue, transactions are scheduled using Earliest Deadline First (EDF). The system update transactions are executed together with user transactions. Since they update the data objects needed by user transactions, system update transactions are given higher priority than user transactions.

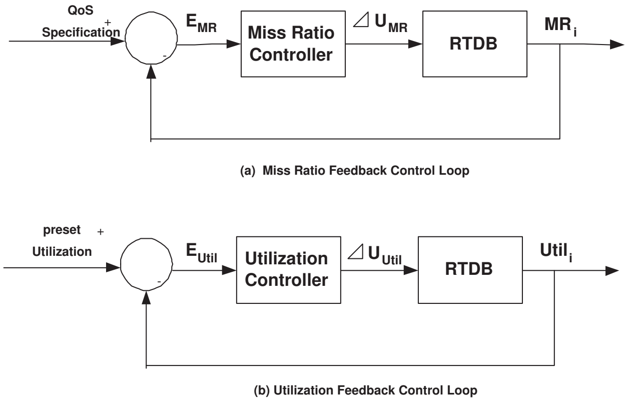

Figure 2. Utilization and Miss Ratio Controllers in Centralized Systems

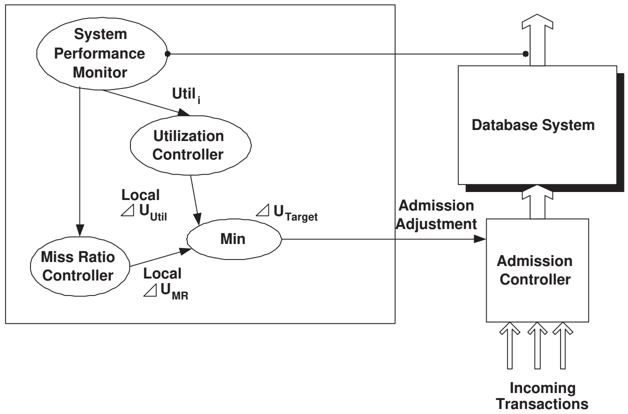

Figure 3. Local Control Algorithm Architecture

4 Algorithm for QoS Guarantees

In this section, the QoS management algorithm in distributed real-time databases is presented. We first introduce a feedback-based control algorithm for centralized systems. Then we present the details of our decentralized load balancing algorithm and the integration of the two algorithms.

4.1 Algorithm in Centralized Systems

Feedback control has been proven to be very effective in supporting a required performance specification when the system model includes uncertainties. Basically, the target performance can be achieved by dynamically adapting the system behavior based on the performance deviation measured in the feedback loop. Feedback control has recently been applied to various computational systems to provide performance guarantees [18] [22] [24].

4.1.1 Centralized Control Loops

At each node, there are a local miss ratio controller and a local utilization controller. As shown in Fig. 2 (a), the local miss ratio controller takes the miss ratios from the latest sampling period, compares them with the QoS specification and computes the local miss ratio control signal ΔU_{MR}, which is used to adjust the target utilization at the next sampling period. The equation used in this paper to derive ΔU_{MR} is as follows.

$$\Delta U_{MR} = \sum_{i=1}^{n} P_{MR} \times (MR_i - MR_{QoS_i}) + \sum_{i=1}^{n} I_{MR} \times (LTMR_i - MR_{QoS_i})$$

MR_i is the miss ratio of class i transactions of last period and $LTMR_i$ is the long term average miss ratio of class i transactions; MR_{QoS_i} is the specified miss ratio requirement by the QoS specification; n is the specified QoS level; P_{MR} and I_{MR} are two controller parameters.

In order to prevent under-utilization, a utilization feedback loop is added. This is used to avoid a trivial solution, in which all the miss ratio requirements are satisfied due to under-utilization. At each sampling period, the local utilization controller compares the utilization of the last period with the preset utilization threshold and generates the local utilization control signal ΔU_{Util} as follows.

$$\Delta U_{Util} = P_{Util} \times (Util - Util_{preset}) + I_{Util} \times (LTUtil - Util_{preset})$$

$Util$ is the CPU utilization of the last sampling period and $LTUtil$ is the long term average CPU utilization of the system; $Util_{preset}$ is the preset target CPU utilization; P_{Util} and I_{Util} are the controller parameters.

The controller parameters determine the behavior of the controllers. The process of tuning their values is called *controller tuning*. The controller analysis and tuning are not the focus of this paper. Details of the analysis and tuning used in our controller design are provided in [19], [12].

The local control architecture is shown in the Fig. 3. At each sampling period, the system utilization and transaction miss ratios are input into the utilization controller and miss ratio controller. The smaller output of the two controllers is used to adjust the *target utilization* of the admission controller.

4.2 Global Load Balancer

To balance the workload between the nodes and thus provide distributed QoS management, decentralized global load balancers (GLB) are used. GLBs sit at each node in the system, collaborating with each other for load balancing. As discussed before, in this paper we consider GLBs utilizing full information from every node, which is reasonable for medium size distributed real-time database applications such as ship-board control systems. At each sampling period, nodes in the system exchange their system performance data. The GLB at each node compares the performance of different nodes. If a node is overloaded while some other nodes in the system are not, the GLB at the overloaded node will send some workload to other nodes that are not overloaded in the next period.

4.2.1 System Performance Indexes

To measure the system performance of one node, we integrate the miss ratios of different service classes into the performance indexes, *Miss Ratio Index* (MRI) and *Long Term Miss Ratio Index* (LTMRI). MRI is a measure of system performance deviation from the specifications. It is defined as follows.

$$MRI = \sum_{i=0}^{n} W_{MR_i} \times (MR_i - MR_{QoS_i})$$

In the definition, MR_i is the miss ratio of class i transactions and MR_{QoS_i} is the specified miss ratio guarantee for class i transactions. W_{MR_i} is the predefined *Miss Ratio Weight* for transaction service class i. Larger miss ratio weights are associated with transaction service classes with higher priority because transaction deadline misses of transactions with higher priority are more serious than the deadline misses of lower class transactions.

The long term miss ratio index $LTMRI$ is the long term average of the miss ratio index. It is calculated using the following equation:

$$LTMRI[k] = \alpha \times LTMRI[k-1] + (1-\alpha) \times MRI[k]$$

where $0 \leq \alpha \leq 1$ and $LTMRI[n]$ is the long term miss ratio index of period n.

4.2.2 Load Transferring Factor

The load sharing process is guided by the *load transferring factor* (LTF). The LTF at each node is an array of real numbers which denote the amount of workload the local node transfers to other nodes during the next sampling period. The LTF_{ij} is defined as follows.

- LTF_{ij}: The workload that local node i could transfer to node j the next sampling period.

The LTF is measured by required CPU time of transactions that one node can transfer to other nodes. For example, if LTF_{ij} is 0.2 and the sampling period is 2 seconds, node i can transfer to node j a set of transactions that require $0.2 \times 2 = 0.4$ second of CPU time for execution. In case different nodes have different processing capacity, a standardized CPU time unit may be used.

4.2.3 Decentralized Global Load Balancing Algorithm

When one node collects the performance data from the other nodes, a feedback-based load balancing algorithm is carried out to calculate the LTF. The algorithm is divided into two steps, *Workload Imbalance Test* and *LTF Adjustment.*

- **Workload Imbalance Test:** The first step is to test whether there exists load imbalance between nodes. To do that, we calculate the mean deviation of MRIs from different nodes. The mean deviation is defined as follows.

  $$Mean\ Deviation = \frac{1}{n} \times \sum_{i=1}^{n} (ABS(MRI_i - MEAN(MRI)))$$

  where MRI_i is the miss ratio index of node i; $ABS(MRI_i)$ returns the the absolute value of MRI_i and $MEAN(MRI)$ returns the mean of MRIs; n is the number of nodes in the system.

  The mean deviation of MRI is a measure for workload distribution in the system. A high value of the mean deviation means that the workloads are not balanced while a low value of the mean deviation means the system workloads are well balanced. Depending on the value of the mean deviation, the algorithm makes different LTF adjustments. A system parameter, *Mean Deviation Threshold*, is used to test whether there exists load imbalance in the system. When the measured mean deviation is larger than this threshold, the system workload is not considered to be balanced. Otherwise, the system workload is considered to be balanced.

- LTF **Adjustment:** The LTF adjustment is divided into two cases depending on whether the system workload is balanced.

  - Workload Not Balanced: When there is load imbalance in the system, i.e., the mean deviation of MRIs is larger than the threshold, it is necessary to share the load between nodes. The load balancing algorithm at the overloaded nodes will shift some workload to the less loaded nodes. A node i is considered to be overloaded compared to other nodes if and only if the difference between its MRI and MRI mean is larger than the preset mean deviation threshold, i.e., $(MRI_i - MEAN(MRI)) > MeadDeviationThreshold$. A node j is considered less overloaded if its MRI is less than the MRI mean, i.e., $(MRI_j - MEAN(MRI)) < 0$.

For an overloaded node i, the algorithm generates the control signal ΔLTF_{ij} for load transferring factor LTF_{ij}, the transaction workload that is transferred from node i to the less loaded node j. The load transferring factor incremental value ΔLTF_{ij} is calculated using the following equation.

$$\Delta LTF_{ij} = P_{LTF} \times (MRI_i - MRI_j) + I_{LTF} \times (LTMRI_i - LTMRI_j)$$

where MRI_n and $LTMRI_n$ are the miss ratio index and long term miss ratio index of node n; P_{LTF} and I_{LTF} are tunable system parameters that determine the weights of MRI and $LTMRI$ on the control signal.

To avoid that two nodes both have positive LTFs with each other and transfer transactions back and forth, special care is needed to make sure that transactions are only transferred in one direction between two nodes. When a node needs to update its LTFs for another node, it sends a message to the corresponding node for the purpose of LTF adjustment. The LTF adjustment process is described as follows. Assume that node i needs to adjust its LTF for node j.

* At node i: Send ΔLTF_{ij} to node j. If ΔLTF_{ij} is larger than or equal to LTF_{ji} at node j, add $\Delta LTF_{ij} - LTF_{ji}$ to LTF_{ij}; otherwise, do nothing. (Node i has the LTF_{ji} of node j because it is broadcast with the local performance data by node j)
* At node j: After receiving ΔLTF_{ij} from node i, if ΔLTF_{ij} is larger than LTF_{ji}, set LTF_{ji} to 0; otherwise subtract ΔLTF_{ij} from LTF_{ji}.

– Workload Balanced: When the mean deviation of MRI is less than the specified threshold, the system workload is considered to be balanced and the GLB reduces the load transferring factors. The LTFs are reduced in the following way.

$$LTF_{ij} = LTF_{ij} \times \gamma$$

where $0 < \gamma < 1$. γ is called the *LTF Regression Factor* which regulates the *load transferring factors regression* process. The intention is to avoid transferring workload between nodes whenever possible. It also helps remove the workload transferring circles in the system. If a LTF becomes sufficiently small (less than 0.0005), it is reset to 0.

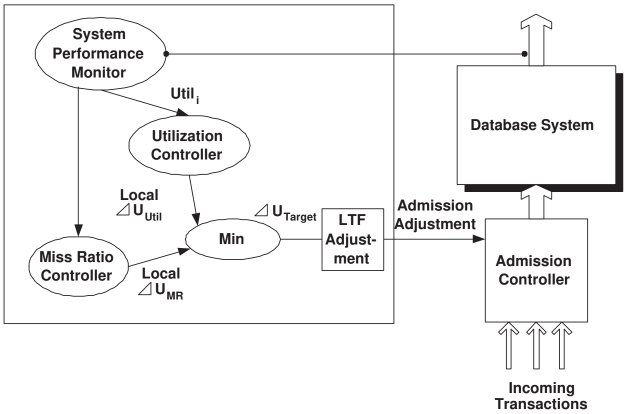

Figure 4. Integration of Local Controller and GLB

If a node fails to collect the performance data of some other nodes during a certain period, the performance data of the pervious period is used. This strategy works because message losses are very rare in wired networks. As shown in the simulation study in the next section, using history data does not cause serious problems for the correct functioning of the algorithm.

There is a possibility that a cycle of load transferring can be formed between nodes, although the probability is very low. When that happens, the load transferring factor regression process discussed above gradually removes the load transferring cycles.

4.3 Integration of Local Controller and Load Balancer

To provide QoS in distributed environments, we need both local workload control and global load balancing functionality. We integrate the local controller with the global load balancer by modifying the feedback loop in Fig. 3. The modified feedback loop is shown in Fig. 4.

As shown in the figure, an extra phase, *LTF Adjustment*, is added to the local controller. In this phase, the local controller reduces the LTFs if the specified QoS is not violated and there is extra CPU time. Note that when the LTF at one node is larger than zero, the node transfers some transactions to other nodes. When extra CPU time is available, the local controller first reduces the LTFs, thus reducing the workload it transfers to other nodes. When there are no local LTFs that are larger than 0, a controller signal is used to increase the *target utilization* parameter at the admission controller, which increases the admitted transaction workload in the next period.

The system parameters used in global load balancer and their values used in this paper are summarized in

Parameter	Value
Miss Ratio Weight 0 (W_{MR_0})	4
Miss Ratio Weight 1 (W_{MR_1})	2
Miss Ratio Weight 2 (W_{MR_2})	0.1
MRI Mead Deviation Threshold	0.1
P_{LTF}	0.02
I_{LTF}	0.02
LTF Regression Factor	0.9

Table 1. System Parameter Settings

Parameter	Value
Node #	8
Network Delay	(0.05 - 1.2) ms/ pkt
Temp Data #	200/Node
Temp Data Size	Uniform(1 - 128)bytes
Temp Data AVI	Uniform(0.5 - 5) seconds
Non-Temp Data #	10,000/Node
Non-Temp Data Size	Uniform(1 - 1024) bytes

Table 2. System Parameter Settings

Parameter	Value
Operation Time	0.2 - 2 ms
Temp Data OP #	(1 - 8) /Tran
Non-temp Data OP #	(2 - 4) /Tran
Transaction SF	10
Temp Data Skew	20%
Non-Temp Data Skew	20%
Class 0 Ratio	33%
Class 1 Ratio	33%
Class 2 Ratio	33%
Remote Data Ratio	20%
Exe Time Est Error	Normal(20%, 10%)
Arrival Rate	80 Trans/sec

Table 3. User Transaction Workload Parameter Settings

Table 1.

5 Performance Evaluation

The main objective of our performance evaluation is to test whether the proposed algorithm can provide the specified miss ratio guarantees even in the presence of unpredictable workloads. We conduct a simulation study which varies the transaction workload and study the system performance. This section presents the results of the simulation study.

5.1 Simulation Settings

For the simulations, we have chosen values that are, in general, representative of some on board ship control such as found in [7]. Precise details of these systems are not available, but we use values estimated from the details that are available. We have also chosen other values of typical of today's capabilities, e.g., network delays.

The general system parameter settings are given in Table 2. There are 8 nodes in the distributed system, each one of them hosts 200 temporal data objects and 10000 non-temporal data objects. The sizes of temporal data objects are uniformly distributed between 1 and 128 bytes and their validity intervals are uniformly distributed between 0.5 and 5 seconds. The sizes of non-temporal data objects are uniformly distributed between 1 and 1024 bytes. The network delays are modelled by calculating end-to-end transmission delay for each packet. Depending on the packet's size (64 - 1500 bytes for Ethernet), the end-to-end delay ranges from 50 microseconds to 1.2 milliseconds. If the data size exceeds one packet size, the data is put into separate packets and the transmission delay is the sum of delays for those packets. The overhead of transferring transactions is accounted by modelling the overhead of transmitting packets that contain the transactions. In the simulation, the overhead of running controllers is not modelled. The reason is that the controller computation procedure is not invoked very often (in our case, once every two seconds) and the involved computation does not consume too much CPU time.

The settings for user transaction workload is given in Table 3. A user transaction consists of operations on temporal data objects and non-temporal data objects. The operation time for one operation is selected between 200 microseconds to 2000 microseconds. The slack factor for transactions is set to 10. To increase the data contention, we introduce *Temporal Data Access Skew* and *Non-temporal Data Access Skew*. The 20% access skews mean that 80 percent of all transaction operations access 20 percent of the data objects. The *Remote Data Ratio* is the ratio of the number of remote data operations (operations that access data hosted by other nodes) to that of all data operations. The remote data ratio is set to 20%, which means 20 percent of all operations are remote data operations. In most real systems, it is almost impossible to know the exact execution time of transactions. To model errors in estimating the transaction execution time, the *Execution Time Estimation Error* is introduced. It is the estimation error of the execution time of user transactions. In our simulation, it conforms to a normal distribution with mean 20% and standard deviation 10%. At each node, the user transaction arrives according to a Poisson distribution and the average

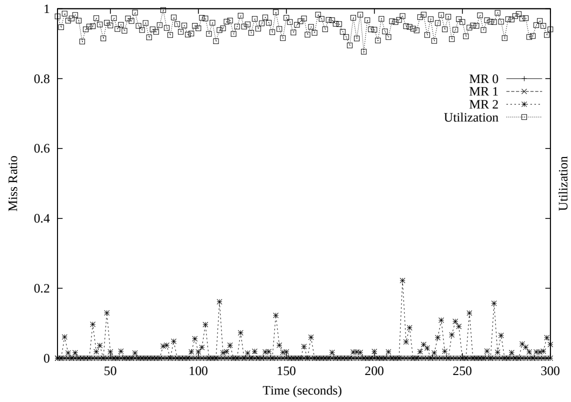

Figure 5. Average Miss Ratio of FB-GLB

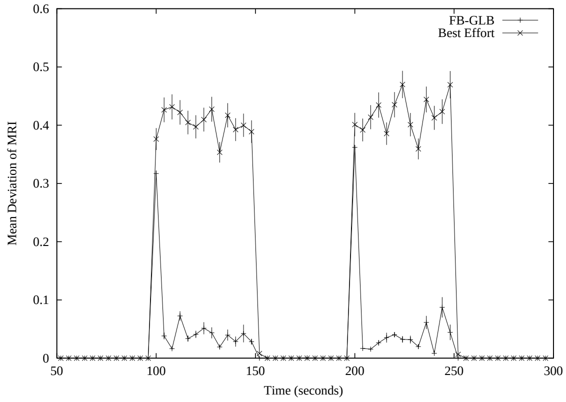

Figure 6. Load Balancing with FB-GLB

arrival rate is 80 transactions per second. User transaction are divided into three service classes and each is one third of all transactions.

5.2 Baseline Protocols

To evaluate our algorithm (FB-GLB), we compare the performance of our algorithm with two baseline algorithms.

- **Best Effort:** The system operates in best-effort manner. All arrived user transactions are admitted and no controls are taken for limiting transaction workload or balancing load among nodes.

- **Local Control Only:** Nodes employ only local feedback-based controllers, which control the system admission process based only on local system performance data.

5.3 Simulation Results

The simulation results are presented in this section. Each simulation is run 10 times and 90% confidence intervals are drawn for each data point. Confidence intervals in some graphes are not shown to improve the readability.

5.3.1 System Performance During Normal Operation

In this set of simulations, the system using FB-GLB is running under stable system conditions. The arrival rate and access patterns of user transactions do not change during the simulation. From Fig. 5 we can see that the FB-GLB algorithm keeps the miss ratios of transactions in classes 0 and 1 around zero while maintaining the CPU utilization around 95% throughout the entire simulation.

5.3.2 Load Balancing with FB-GLB

The first set of experiments evaluates the load balancing function of FB-GLB. In the experiments, we introduce two workload bursts at one node. The bursts begin at the 100th second and the 200th second and each one lasts for 50 seconds. We use the mean deviation of the MRIs of nodes to show the performance of the algorithm. As discussed in the previous section, the mean deviation of miss ratio indexes measures the performance differences between nodes. The lower the mean deviation value, the more balanced the system workloads are.

As shown in Fig. 6, the system running the best effort algorithm remains unbalanced throughout the workload burst periods; with FB-GLB, the system workload become balanced (mean deviation of MRI becomes less than 0.1) within 5 seconds. Note that the mean deviation of miss ratio indexes stays at zero most of the time because MRI is positive only when the QoS specification is violated. During the normal operation, the QoS requirements are not violated, resulting in MRIs and their mean deviation all equal to zero.

5.3.3 Handling Transaction Workload Burst

In many soft real-time systems, the transaction workload may vary significantly with time. The QoS management system should deal with this kind of workload variation and still guarantee the specified QoS requirements. The successful operation in such a situation is crucial for QoS management systems for distributed environments. To test whether our system provides good transaction services in such situations, we conduct a set of simulations where two out of eight nodes are severely overloaded. At the 100th and 200th seconds of the simulation, the workload at two nodes suddenly increased to 300% of the normal workload. The two workloads each last for 50 seconds.

As shown in Fig. 7, for the best-effort algorithm, QoS

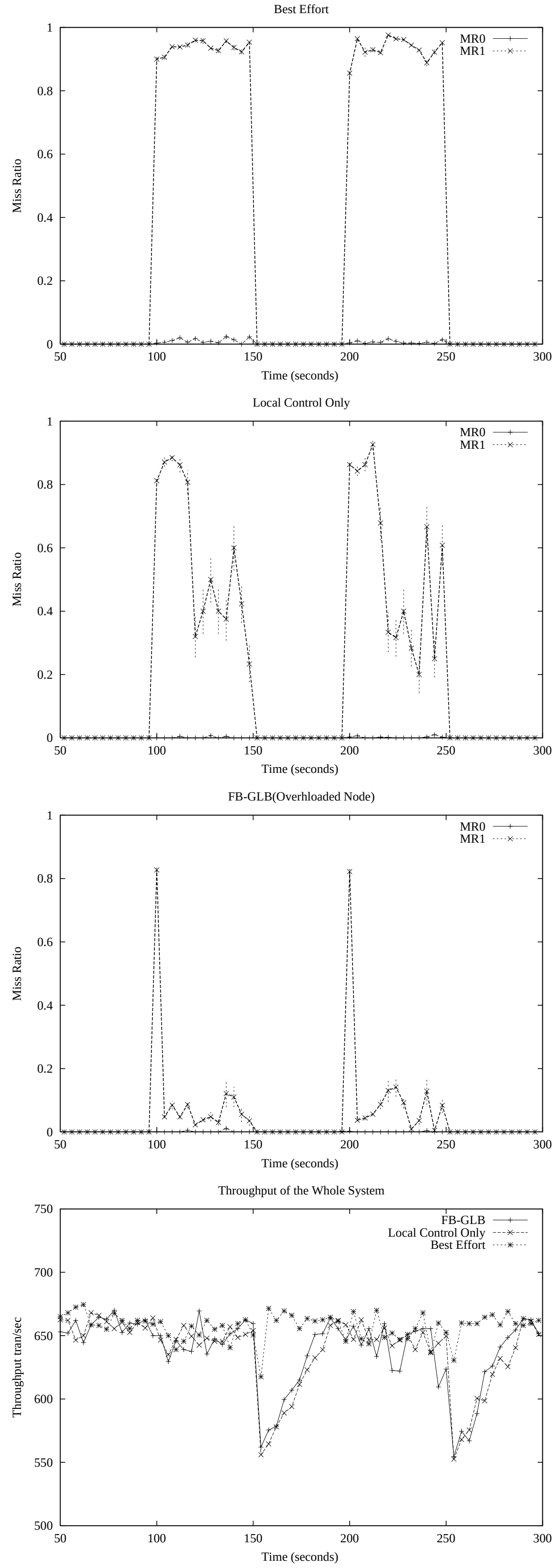

Figure 7. System Performance (Two Nodes are Overloaded)

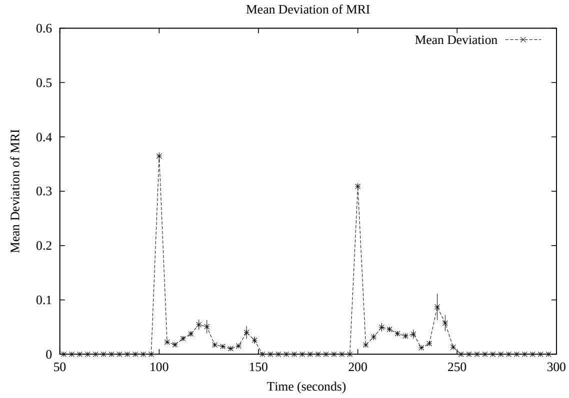

Figure 8. Load Balancing with Message Loss

requirements are violated and the miss ratio of class 1 transactions remains over 90%. For the system running only the local control algorithm, the miss ratio of class 1 transactions exceeds the QoS requirement during the workload burst period and the control algorithm does not seem to be able to keep MR_1 around the specified 10%. For the system running FB-GLB, the system adapts to the workload burst very quickly and MR_1 returns to specified 10% within 5 seconds.

As shown by the figure, the throughput of the system is not seriously affected by our QoS management algorithm. The system that runs FB-GLB has almost the same throughput as the system that runs only local controllers. The two algorithms shows lower throughput when the arrival bursts end at the 150th and 250th seconds because the two algorithms both reduce transaction admission rate for class 1 and class 2 transactions during the transaction workload bursts. Their throughput gradually catches up with the throughput of the best effort algorithm after the burst.

5.3.4 Performance with Message Loss

In our decentralized load balancing algorithm, each node needs to exchange its performance data with other nodes periodically. It is possible that one node may not be able to collect the performance data of all the other nodes. When that happens, the node uses the performance data from the pervious period. This set of simulations evaluates the performance of that strategy. In the simulations, we created transaction bursts at one node and measured the performance of algorithms with a 10% message loss rate. The results are shown in Fig. 8 and Fig. 9.

As shown in Fig. 8, the mean deviation of MRIs is kept below 0.1 immediately after each transaction arrival burst, which means the load balancing functionality of the algorithm is not affected by the 10% message loss. As shown in Fig. 9, the miss ratios of class 0 and class

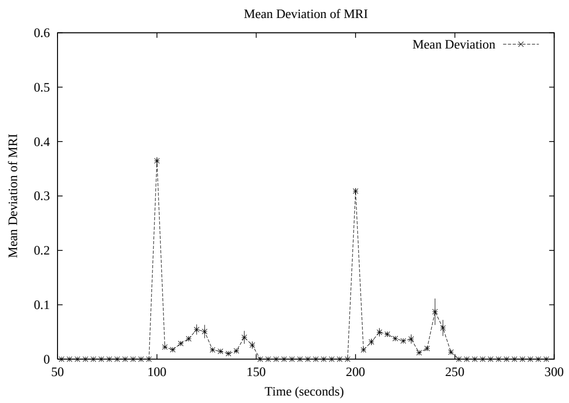

Figure 9. Miss Ratio at Overloaded Node (with Message Loss)

1 transactions at the overloaded node are kept below the specified level and the QoS is maintained.

6 Related Work

Since the major publication by Abbott and Garcia-Molina [1] in 1988, real-time databases received a lot of attention. A breath of research topics in real-time databases have been studied, including concurrency control [10] [26], scheduling algorithms [9] [13], security [21] [17] and recovery [20], to name a few.

Distributed real-time database research has drawn attention in recent years [15] [23]. In [8], Ginis et. al. discussed the design of open system techniques to integrate a real-time database into a distributed computing environment. Concurrency control mechanisms for distributed real-time databases are studied in [15]. Lee et. al. [16] built a model for wireless distributed real-time database systems and performed simulation experiments to identify the effect of wireless bandwidth on the performance of distributed real-time database systems. In [25], a state-conscious concurrency control protocol called MIRROR is proposed for replicated real-time databases. However, to the best of our knowledge, no research results have been published for providing data services with QoS guarantees in distributed real-time database systems.

Feedback control has been applied to QoS management and real-time scheduling due to its robustness against unpredictable operating environments [24] [4]. In [19], Lu et. al. proposed a feedback control real-time scheduling framework called FC-UM for adaptive real-time systems. Stankovic et. al. [22] presented an effective distributed real-time scheduling approach based on feedback control. Kang et. al. [12] proposed an architecture to provide QoS guarantees for centralized main memory databases. In this paper, we proposed a heuristic feedback-based dynamic load sharing algorithm and integrated it with the local control algorithm to provide a QoS management in distributed real-time database systems.

Load balancing has been a research topic for general distributed systems for many years [6] [5]. In those systems, the system performance is often measured by system throughput, but QoS real-time guarantees are not considered as the most important performance metric. Further, they have not dealt with the issues of transaction deadlines and data freshness.

7 Conclusion and Future Work

The demand for real-time data services in mid-size distributed applications such as ship control is increasing. The complexity and non-determinism of these applications produce a need for QoS guarantees rather than 100% guarantees. Our solution, using feedback control, meets steady state miss ratio and transient settling time requirements. The solution is shown to be appropriate for an important class of mid-size distributed real-time systems as represented by today's ship-board control systems.

We plan to extend this work in several ways. One direction is to extend the algorithm so that it scales to large distributed systems. In large distributed systems, each node will not collect performance data of all nodes periodically. Instead, the load balancing algorithm will balance transaction workloads only among nearby nodes. Partial data replication and efficient replica management algorithms will also be added because full replication is inefficient or impossible in large distributed systems. Derived data management is another interesting extension because derived data is of particular interest for some real-time database applications such as e-commerce and online stock trading systems.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions. *SIGMOD Record*, 17(1):71 – 81, 1988.

[2] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Transactions on Database Systems*, 17(3):513–560, 1992.

[3] J. Baulier, P. Bohannon, S. Gogate, C. Gupta, S. Haldar, S. Joshi, A. Khivesera, H. Korth, P. McIlroy, J. Miller, P. P. S. Narayan, M. Nemeth, R. Rastogi, S. Seshardi, A. Silberschatz, S. Sudarshan, M. Wilder, and C. Wei. DataBlitz storage manager: Main memory database performance for critical applications. *ACM SIGMOD Record*, 28(2):519–520, 1999.

[4] N. Christin, J. Liebeherr, and T. Abdelzaher. A quantitative assured forwarding service. In *IEEE INFOCOM*, July 2002.

[5] S. Dandamudi. Sensitivity evaluation of dynamic load sharing in distributed systems. *IEEE Concurrency*, 6(3):62 – 72, 1998.

[6] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transaction on Software Engineering*, 12(5):662 – 675, 1986.

[7] P. Ericsson. An operational ship control system in a virtual environment. In *Undersea Defense Technology Europe Conference*, 2003.

[8] R. Ginis and V. Wolfe. Issues in designing open distributed real-time databases. In *the 4th International Workshop on Parallel and Distributed Real-Time Systems*, pages 106 –109, Apr 1996.

[9] J. Haritsa, M. Livny, and M. Carey. Earliest deadline scheduling for real-time database systems. In *12th Real-Time Systems Symposium*, Dec 1991.

[10] J. Huang, J. Stankovic, K. Ramamritham, and D. Towsley. On using priority inheritance in real-time databases. In *12th Real-Time Systems Symposium*, Dec 1991.

[11] K. Kang, S. Son, and J. Stankovic. Service differentiation in real-time main memory databases. In *Proc. 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC 02)*, Crystal City, VA, USA, May 2002.

[12] K. Kang, S. Son, J. Stankovic, and T. Abdelzaher. A qos-sensitive approach for miss ratio and freshness guarantees in real-time databases. In *Proc. 14th Euromicro Conference on Real-Time Systems*, Vienna, Austria, 2002.

[13] S. Kim, S. Son, and J. Stankovic. Performance evaluation on a real-time database. In *Real-Time and Embedded Technology and Applications Symposium*, pages 253 –265, 2002.

[14] K. Lam and T. Kuo. *Real-Time Database Systems: Architecture and Techniques*. Kluwer, 2001.

[15] K.-W. Lam, V. Lee, K.-Y. Lam, and S. Hung. Distributed real-time optimistic concurrency control protocol. In *4th International Workshop on Parallel and Distributed Real-Time Systems*, pages 122 –125, 1996.

[16] V. Lee, K. Lam, and W. Tsang. Transaction processing in wireless distributed real-time databases. In *10th Euromicro Workshop on Real-Time Systems*, pages 214 – 220, Jun 1998.

[17] V. Lee, J. Stankovic, and S. Son. Intrusion detection in real-time database systems via time signatures. In *6th IEEE Real-Time Technology and Applications Symposium*, pages 124 – 133, 2000.

[18] C. Lu, T. Abdelzaber, J. Stankovic, and S. Son. A feedback control approach for guaranteeing relative delays in web servers. In *Proc. 7th IEEE Real-Time Technology and Applications Symposium (RTAS 01)*, pages 51–62, Taipei, Taiwan, May 2001. IEEE.

[19] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Journal of Real-Time Systems, Special Issue on Control-theoretical Approaches to Real-Time Computing*, pages 85–126, 2002.

[20] L. Shu, J. Stankovic, and S. Son. Achieving bounded and predictable recovery using real-time logging. In *Real-Time and Embedded Technology and Applications Symposium*, pages 286 –297, 2002.

[21] S. Son. Supporting timeliness and security in real-time database systems. In *9th Euromicro Workshop on Real-Time Systems*, 1997.

[22] J. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu. Feedback control scheduling in distributed real-time systems. In *Proc. 22nd IEEE Real-Time Systems Symposium (RTSS 01)*, pages 59–72, London, UK, 2001. IEEE.

[23] J. Stankovic and S. Son. Architecture and object model for distributed object-oriented real-time databases. In *Proc. 1st International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 98)*, Kyoto, Japan, 1998.

[24] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. 3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*, pages 145–158, Berkeley, CA, Feb. 22–25 1999.

[25] M. Xiong, K. Ramamritham, J. Haritsa, and J. Stankovic. MIRROR A state-conscious concurrency control protocol for replicated real-time databases. In *Proc. 5th IEEE Real-Time Technology and Applications Symposium (RTAS 99)*, pages 100–110, 1999.

[26] P. Yu, K. Wu, K. Lin, and S. Son. On real-time databases: concurrency control and scheduling. *Proceedings of the IEEE*, 82(1):140 –157, 1994.