I/O-Aware Deadline Miss Ratio Management in Real-Time Embedded Databases *

Woochul Kang, Sang H. Son, and John A. Stankovic
Department of Computer Science
University of Virginia
{wk5f,son,stankovic}@cs.virginia.edu

Mehdi Amirijoo
Department of Computer and Information Science
Linköping University
meham@ida.liu.se

Abstract

Recently, cheap and large capacity non-volatile memory such as flash memory is rapidly replacing disks in embedded systems. While the access time of flash memory is highly predictable, deadline misses may occur if data objects in flash memory are not properly managed in real-time embedded databases. Buffer cache can be used to mitigate this problem. However, since the workload of a real-time database cannot be precisely predicted, it may not be feasible to provide enough buffer space to satisfy all timing constraints. Several deadline miss ratio management schemes have been proposed, but they do not consider I/O activities. In this paper, we present an I/O-aware deadline miss ratio management scheme in real-time embedded databases whose secondary storage is flash memory. We propose an adaptive I/O deadline assignment scheme, in which I/O deadlines are derived from up-to-date system status. We also present a deadline miss ratio management architecture where a control theory-based feedback control loop prevents resource overload both in I/O and CPU. A simulation study shows that our approach can effectively cope with both I/O and CPU overload to achieve the desired deadline miss ratio.

1 Introduction

A number of emerging applications require real-time data services to handle large amounts of data in a timely fashion. Examples include traffic control, target tracking, and network management. Unlike traditional applications, they require transactions to be completed within their deadlines. Several approaches using real-time database systems (RTDBS) have been proposed to handle these time-constrained transactions [4][20][14]. Most of them are based on main-memory databases due to the inherent unpredictability of disk I/O. In main-memory databases, it is assumed that the database is small enough to fit into main memory, thus eliminating most I/O operations. However, this assumption does not hold when the database size is greater than main memory. Another problem with main-memory databases is that they are vulnerable to system failures. Recent advancement in semiconductor technology enables us to overcome the unpredictability of disks with cheap and large capacity non-volatile memories. In particular, flash memory is rapidly replacing disks not only in real-time embedded systems, but also in high-performance servers [2] . Unlike disks, the access time to flash memory is not affected by mechanical parts; thus, flash memory is several orders of magnitude faster and highly predictable. These desirable characteristics make flash memory more suitable for RTDBS. However, flash memory still has high overhead in access time compared to volatile memory such as SRAM. Table 1 shows the overhead of flash memory in comparison to SRAM.

Type	Read	Write	Erase
SRAM	10ns	10ns	N/A
NAND Flash	10μs	200μs	2ms

Table 1: Characteristics of memory devices [17].

Such gaps in access time are critical for RTDBS because high I/O overheads may incur deadline misses if data objects are not properly managed. System designers may provide buffer memory to mitigate the high overhead of flash memory accesses. However, in many cases, the workloads are unknown at design time and they can change dynamically. Therefore, it may not be feasible to provide enough buffer space to satisfy all timing constraints. In particular, resource-constrained embedded systems are extremely cost and space sensitive, and cannot afford to have large buffers.

In this paper, we propose an I/O-aware deadline miss ratio management scheme for real-time embedded database systems (RTEDBS) whose secondary storage is flash memory. The contributions of this paper are three-fold:

1. an adaptive I/O deadline assignment scheme,
2. a model of deadline miss ratio in terms of I/O and CPU workloads, and
3. a feedback control architecture to satisfy a given deadline miss ratio.

*This research work was funded in part by NSF CNS-0614886

1052-8725/07 $25.00 © 2007 IEEE
DOI 10.1109/RTSS.2007.19

IEEE computer society

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

To the best of our knowledge, this is the first paper on deadline miss ratio management of RTEDBS with flash storage using feedback control to consider both I/O and CPU workloads. Previous approaches on deadline miss ratio management in RTDBS assumed main-memory databases [14][5]. As a result they only considered CPU workloads, which is insufficient when managing the Quality of Service (QoS) of modern RTEDBS consisting of volatile memory as well as non-volatile flash memory. Several I/O-aware approaches [3][8] have been proposed. However, they assumed disks as secondary storage, and their primary research focus was deadline-driven disk scheduling to mitigate the unpredictability of disk operations.

A key issue in deadline miss ratio management with different kinds of resources, e.g. I/O and CPU, is to find out which resource is the bottleneck that causes deadline misses. In RTEDBS, a deadline miss can happen either because of an overload in I/O or CPU, or both. By properly setting deadlines for each resource request and observing its deadline misses, we can tell which resource is the bottleneck. However, deriving a deadline for each resource is not straightforward because the deadlines are set for transactions, not for individual resource requests. In this paper, we assume a 2-phase transaction model, where each transaction consists of an I/O phase and a computation phase. The I/O deadlines and subsequent CPU deadlines for respective I/O phases and computation phases are derived from transaction deadlines with up-to-date system overload status.

Having I/O deadlines and CPU deadlines enables us to measure and model the system in terms of I/O and CPU workloads and their respective deadline miss ratios. A straightforward approach would be to build separate models for I/O and CPU. However, our experiments show that CPU and I/O deadline miss ratios are coupled and affect each other, thus necessitating multiple-input/ multiple-output (MIMO) modeling of the system. In this paper, the RTEDBS is modeled as a MIMO system to capture this coupling of control inputs and system outputs.

The use of feedback controllers has shown to be effective for real-time systems with unpredictable workloads [5][14][16]. A feedback control architecture is proposed to guarantee the desired deadline miss ratio. At each sampling instant, the feedback control loop measures I/O and CPU miss ratios and computes control signals, e.g. the required I/O and CPU workload adjustment. In particular, our approach controls both I/O and CPU workloads at the same time because of close interactions between them.

The rest of the paper is organized as follows. Section 2 describes our system and data model for RTEDBS. In Section 3, our deadline miss ratio management architecture is described. In Section 4, the performance evaluation results are presented. In Section 5, we discuss the related work. Finally, Section 6 concludes the paper.

2 System and Data Model for RTEDBS

2.1 System Model

A real-time embedded system that includes a CPU, main memory, and flash memory is considered. The buffer pool is a cache between the flash memory and the CPU. It is shared between transactions to reduce the data storage access time. The maximum size of the buffer pool is configured at deployment time and does not change over its lifetime. However, the proportion of allocated buffer pool size between different classes is not fixed since a fixed allocation can incur inefficient use of the buffer resource.

The I/O load between main memory and flash memory occurs only when an explicit I/O request is issued for a data object not present in the buffer pool. Implicit I/O requests such as page faults are not considered because virtual memory is typically not supported in embedded systems.

2.2 Data and Transaction Model

In our data model, data objects can be classified into two classes, temporal and non-temporal data. Temporal data objects are updated periodically by update transactions. In contrast to update transactions, user transactions may read both temporal and non-temporal data objects and modify non-temporal data objects.

Since embedded platforms are assumed for our RTEDBS, transactions are *canned transactions*, whose characteristics including data requirement and worst-case computation time is known at the design time. However, workload and data access patterns of the whole RTEDBS can be unpredictable and change dynamically because the invocation frequency of each transaction is unknown. Since data requirements are known for each transaction, data requests of each transaction can be gathered before its computation to improve the response time. To this end, we model each transaction as a two-phase operation, an I/O phase and a computation phase. In the I/O phase, data objects for the transaction are brought to the buffer pool from the flash memory. In a single transaction, the computation phase can begin only after all its required data objects are present in the buffer pool. However, the I/O and the computation phase of different transactions can overlap. For example, while transaction i is under the I/O operation, transaction j can perform its computation. A transaction can commit after the computation phase by updating a copy of the data object in the memory buffer. The time required to update the data object in the memory buffer is ignored in this transaction model because it is relatively small compared to flash memory accesses. The buffer manager will $eventually$ write the updated buffers back to flash memory when the buffer manager is running out of buffers.

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

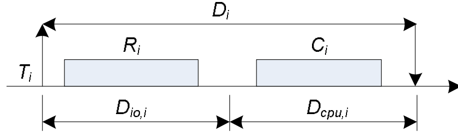


Figure 1: The timing of a typical transaction.

Figure 1 shows the timing of a typical transaction i, where T_i is a release time, D_i is a relative deadline, R_i is a I/O time, C_i is a computation time, $D_{io,i}$ is a relative I/O deadline, and $D_{cpu,i}$ is a relative CPU deadline. While the deadline of each transaction i is set by the application in consideration of the worst case I/O time and computation time, $D_{io,i}$ and $D_{cpu,i}$ are dynamically derived from the relative deadline D_i and the current deadline miss ratio. When the I/O deadline is missed, the transaction is aborted and its computation phase is not initiated. This dynamic I/O deadline assignment scheme is explained in the Section 3.2.

Transactions are classified into classes by their importance. Two service classes, a guaranteed service class and a best effort service class, are assumed in this study for simplicity. Resources including the buffer space are shared between service classes. Instead of providing each service class fixed amount of resources, the allocation of resources to each service class is adaptive in our RTEDBS. Our adaptive buffer space allocation scheme is presented in Section 3.4.

3 Approach

A system overload due to a transient surge of the workload is the main source of deadline misses in soft real-time systems. When the number of deadline misses increases because of the scarcity of a specific resource, the feedback control loop for the resource reduces the workload by adjusting the system parameters. In this section, we present a QoS management architecture which controls both CPU and I/O workloads with a feedback control.

3.1 Performance Metrics

In our approach, the main performance metric is the deadline miss ratio of real-time transactions. A deadline miss can be caused by either an I/O deadline miss or a CPU deadline miss.

CPU deadline miss: A transaction misses its deadline and all data objects needed by the transaction are present in the buffer pool by the I/O deadline.

I/O deadline miss: Some data objects needed by the transaction are not present in the buffer pool at the I/O deadline. In this case, the computation phase is not initiated.

When a deadline miss happens, it can be attributed to either a CPU or an I/O deadline miss, but not both. Accordingly, CPU and I/O deadline miss ratios are obtained as follows,

$$m_{\{cpu|io\}} = \frac{\text{\# of \{CPU |I/O\} deadline misses in admitted transactions}}{\text{\# of admitted transactions}} \quad (1)$$

$$\text{where, total miss ratio} = m_{cpu} + m_{io} \leq 1. \quad (2)$$

By observing the deadline miss ratios for CPU and I/O, we can tell which resource is the current bottleneck in the RTEDBS.

The desired levels of miss ratio, m_{ref}, for the guaranteed service class is expressed in the QoS specification. The desired level of deadline miss ratio for CPU, $m_{ref,cpu}$, and I/O, $m_{ref,io}$, are set separately such that their sum is equal to m_{ref}. The reference deadline miss ratios for I/O and CPU are weighted according to the system overload status as follows,

$$m_{ref,\{cpu|io\}}(t) = \rho_{\{cpu|io\}}(t) \times m_{ref} \quad (3)$$

where,

$$\rho_{\{cpu|io\}}(t) = \frac{\text{Total number of \{CPU | I/O\} deadline misses}}{\text{Total number of deadline misses}} \quad (4)$$

Note that $\rho_{cpu}(t)$ and $\rho_{io}(t)$ are defined only when the RTEDBS has deadline misses, and the sum of $\rho_{cpu}(t)$ and $\rho_{io}(t)$ equals to one.

3.2 Adaptive I/O Deadline Assignment

Since a transaction deadline is set for the transaction, not for I/O or computation phases, deriving an I/O deadline is not straightforward. In this paper, instead of setting I/O deadlines statically for each transaction, we define I/O deadlines recursively as a time-varying function of the deadline miss ratio of the past sampling period as follows,

$$\text{I/O deadline for transaction } i = T_i + D_i - C_i \times \left(1 + m_{cpu}(t) \times \frac{D_i - C_i}{C_i}\right), \quad (5)$$

where $m_{cpu}(t)$ is the CPU deadline miss ratio during the past sampling period. The initial value for $m_{cpu}(0)$ is set to zero. The I/O deadline reflects the up-to-date resource status and the amount of slack time that is required for the computation phase to finish within the transaction deadline; in our two-phase transaction model, the deadline for the computation phase is equal to the transaction deadline.

An I/O deadline can be as long as $T_i + D_i - C_i$ when the CPU is not overloaded, thus, m_{cpu} is close to zero; this implies that the computation phase requires less slack time to meet the transaction deadline. In contrast, an I/O deadline can be as short as T_i when the CPU resource is the bottleneck and most I/O requests can be handled in the data buffer without incurring physical I/O operations and I/O deadline misses in the end. In this case, the computation phase requires more slack time. Overall, I/O deadlines are set in

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

inverse proportion to the CPU deadline miss ratio of the latest sampling period. However, in practice, the two extreme cases rarely happen since the controller keep the deadline miss ratio ($m_{cpu} + m_{io}$) under m_{ref}, e.g. 0.01.

Setting I/O deadlines can serve two purposes: time-cognizant I/O scheduling and I/O workload control. In this paper, we use I/O deadlines only for I/O workload control purposes; the controller controls the I/O workload based on m_{io}. In terms of I/O scheduling, a *First-come/First-service* scheduling policy is used for its simplicity. We reserve the study on the impact of the time-cognizant I/O scheduling as our future work.

3.3 Deadline Miss Ratio Management Architecture

Figure 2 shows the deadline miss ratio management architecture for RTEDBS. Transactions issued by sensors (update transactions) and users (user transactions) are placed on the ready queue. The transaction queue can have several service classes. The figure shows two classes: a guaranteed service queue and a best effort service queue. The transactions in the best effort service queue are dispatched only if the ready queue for the guaranteed service class is empty. The dispatched transactions are managed by the transaction handler, which consists of a buffer manager (BM), a freshness manager (FM), a concurrency control (CC), and a scheduler (S). Transactions are observed by the monitor and the statistics of monitored transactions, including deadline miss ratios, (m_{io} and m_{cpu}), and utilizations (u_{io} and u_{cpu}), are reported to the QoS controllers on every sampling period.

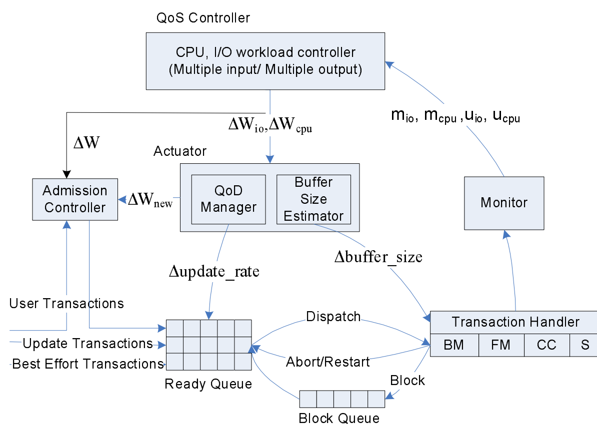


Figure 2: Feedback control architecture

Controllers calculate the workload adaptations required to meet the reference miss ratios, $m_{ref,io}$ and $m_{ref,cpu}$, by comparing them to current miss ratios. Once the workload changes are determined, they are enforced by two actuators independently. For CPU workload adaptation, the update rates of temporal data are adjusted by the *Quality of Data* (QoD) manager. For I/O workload adaptation, the buffer size of the guaranteed service is adjusted by the *buffer size estimator*. If the workload needs to be adjusted more than the actuators can handle, the admission controller adapts the workload by allowing or denying user transactions.

3.4 I/O and CPU Workload Adjustment

In I/O-aware deadline miss ratio management, the workload should be separately defined for I/O and CPU. The CPU workload, W_{cpu}, can be measured as the amount of requested computation to do at a given time. However, we need to be more specific about I/O workload because the I/O workload of a transaction depends on the buffer cache; In the presence of buffer cache, I/O requests from a transaction incur I/O operations only if data objects are not found in the buffer. In our study, I/O workload, W_{io} for the past sampling period is estimated as follows,

$$W_{io} = \frac{\text{\# of buffer access} \times (1 - HIT) \times \text{buffer page size}}{\text{bandwidth} \times \text{sampling period}}, \quad (6)$$

where, HIT is the buffer hit ratio during the past sampling period. Note that we consider explicit I/O requests and subsequent buffer misses as the only source of I/O workload. Once the target I/O workload, $W_{io,target}$, is provided by the controller (in Section 3.5), the target buffer hit ratio can be obtained from Equation (6). The buffer size to achieve that target buffer hit ratio can be estimated with an estimation function [6][9]. In this paper, we use the technique proposed by Brown et al. [6], in which the relation between the buffer size and the buffer hit ratio is linearly approximated with the last two hit ratio observations.

It may be argued that adaptive buffer adjustment is unnecessary because providing larger buffer space will solve I/O overload problem. However, memory is a scarce resource in most embedded systems and we cannot afford to have large buffer memory. Therefore, we need to increase the total utilization of the buffer memory by adaptively allocating buffer space to each service classes. For instance, by reducing the buffer space of a service class when its I/O is under-utilized, the RTEDBS can provide more buffer space to the other service classes that requires it.

In contrast to the I/O workload, the CPU workload can be adjusted by changing the precision of transactions [5] or the freshness of temporal data [14]. In this paper, we change the freshness of temporal data by changing update intervals of temporal data. For details, readers are referred to [14].

If further workload adaptation is not possible by changing update intervals and buffer size, admission control is applied. By allowing or denying more user transactions, workloads are adjusted. However, we should be careful in applying admission control because it changes both I/O and CPU workloads together. Aggressive admission control can make systems unstable. For instance, if I/O allows

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

50% additional user transaction while CPU needs only 10% additional user transactions, then allowing 50% additional user transaction can cause an overshoot in CPU miss ratios. To prevent an excessive overshoot, we take a conservative approach by taking the average of two. Another interesting issue in applying admission control occurs when each resource wants to adjust workload in different directions; e.g., CPU wants to increase its workload by allowing 20% additional user transactions while I/O wants to decrease its workload by 20%. In this case, the admission rate is determined by the resource that has a higher deadline miss ratio.

3.5 Control Loop Design

In this section, we model RTEDBS in terms of I/O and CPU deadline miss ratios and build a controller to manage deadline misses.

3.5.1 System Modeling

The first step in the design of a feedback control loop is the modeling of the controlled system [11]; the RTEBDS in our study.

Unlike the previous work [5][14], which has single-input and single-output (SISO), the RTEDBS in this paper has multiple inputs (W_{cpu} and W_{io}) and multiple outputs (m_{cpu} and m_{io}). We may choose to use two separate SISO models for each pair of control input and system output; (W_{cpu}, m_{cpu}) and (W_{io}, m_{io}), respectively. However, if an input of the system is highly affected by another input, then a Multiple Inputs/Multiple Outputs (MIMO) model should be considered [10][11]. Having two SISO models does not capture the interaction between different control inputs and system outputs; for example, the interaction between the I/O workload and the CPU deadline miss ratio cannot be modeled with two SISO models. To understand the interaction between control inputs and system outputs in the RTEDBS, we performed a series of experiments by applying a discrete sine wave input while the other input was fixed. We did this for both CPU workload and I/O workload. The workloads were adjusted by controlling the buffer size and update rates of temporal data for I/O and CPU respectively. Figure 3(a) shows the result of applying the sine wave CPU workload with 40% amplitude while the I/O workload was fixed at 120%. While the CPU workload changes between 80% and 120%, the I/O workload stays at around 110% without a significant deviation. This result shows that the I/O workload is not affected by CPU workload change. On the contrary, an I/O workload change has a significant impact on the CPU workload as shown in Figure 3(b). In Figure 3(b), a discrete sine wave input was applied to the I/O while the CPU workload was fixed at 130%. Even though we fixed the CPU workload, it was affected by changes to the I/O workload, and, consequently, the CPU deadline miss ratio was also affected by this. Interestingly enough, the results show that the CPU workload is inversely proportional to the I/O workload. This is because transactions are aborted when their I/O deadlines are missed. The higher the I/O deadline miss ratio is, the more transactions are aborted by I/O deadline misses. Therefore, CPU workload decreases, thus, decreasing the CPU deadline miss ratio. This experiment shows that a MIMO model is more appropriate for RTEDBS than SISO models.

Another issue in modeling a computing system is its non-linearity and time-variant characteristics. Complex systems such as RTEDBS can show a non-linear response to inputs. For example, the CPU deadline miss ratio develops quite differently when the CPU is saturated than when it is not saturated. However, the system can be approximated quite closely with linear time invariant models such as the *ARX* model by choosing an operating region where the system's response is approximately linear [11]. Even when the system's response is highly non-linear, the system can be modeled with linear models by dividing the operating region into several sub-operating regions, where each region is approximately linear; in this case, adaptive control techniques such as gain scheduling [11] can be used for control. In case of RTEDBS, the operating region of the controller is set so that the deadline miss ratio is greater than 0 and less than 50%. As we will show later, the accuracy of the linear model in the operating region is acceptable; hence, the linear model is sufficient for our purpose. The form of the linear time invariant model that we use for RTEDBS is shown in (7), with parameters $\mathbf{A}$ and $\mathbf{B}$.

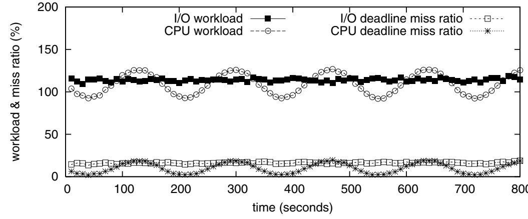
$$\begin{pmatrix} m_{io}(k+1) \\ m_{cpu}(k+1) \end{pmatrix} = \mathbf{A} \cdot \begin{pmatrix} m_{io}(k) \\ m_{cpu}(k) \end{pmatrix} + \mathbf{B} \cdot \begin{pmatrix} W_{io}(k) \\ W_{cpu}(k) \end{pmatrix} \quad (7)$$

Because the RTEDBS is modeled as a MIMO system, $\mathbf{A}$ and $\mathbf{B}$ are 2x2 matrices. A RTEDBS simulator which will be introduced in Section 4 was used for *system identification* [15]. In the system identification, relatively prime sine wave workloads for CPU and I/O were applied simultaneously to the simulator to get the parameters. In our study, the RTEDBS model has $\mathbf{A} = \begin{pmatrix} 0.5403 & 0.1740 \\ 0.2405 & 0.4500 \end{pmatrix}$, and $\mathbf{B} = \begin{pmatrix} 0.2400 & -0.1208 \\ -0.0100 & 0.1774 \end{pmatrix}$ as its parameters. All eigenvalues of $\mathbf{A}$ are inside the unit circle; hence, the system is stable [11].

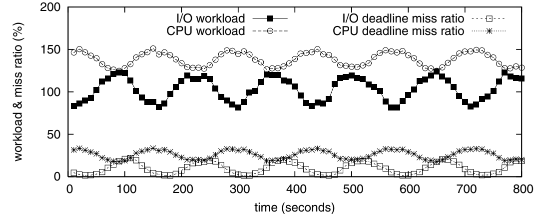
In terms of the system order, note that we model the RTEDBS as a first-order system; the current outputs are determined by their inputs and the outputs of the last sample. The accuracy of the model is satisfactory and, hence, the chosen model order is sufficient for our purposes.

The model can be validated by comparing the experimental result to what the model predicts. Figure 4 plots the experimental response of the RTEDBS and the prediction of the model. We can see that the model provides

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.



(a) sine wave CPU workload



(b) sine wave I/O workload

Figure 3: SISO inputs to RTEDBS.

highly accurate predictions. The accuracy metric $R^2 = 1 - \frac{variance\text{(experimental value - predicted value)}}{variance\text{(experimental value)}}$ is 0.93 and 0.78 for the I/O and CPU dead line miss ratio respectively. Usually, $R^2 \geq 0.8$ is considered acceptable [11]. With regard to R^2 and multi-step validation in Figure 4, the suggested first-order linear model can be considered acceptable.

3.5.2 Controller Design

For RTEDBS, we choose to use a proportional integral (PI) control function given by,

$$U(k) = K_p \cdot E(k) + K_I \cdot \sum_{j=1}^{k-1} E(j). \tag{8}$$

At each sampling instant k, the controller computes the control input $U(k) = [W_{IO}(k)\ \ W_{CPU}(k)]^T$ by monitoring the control error $E(k) = [m_{ref,IO}(k) - m_{IO}(k)\ \ m_{ref,CPU}(k) - m_{CPU}(k)]^T$. K_p and K_I are controller gains.

One important design consideration in computing systems such as RTEDBS which have a stochastic nature is to control the trade-off between short settling times and over-reacting to random fluctuations. If a controller is too aggressive, then the controller over-reacts to this random fluctuation. To this end, we choose to use the linear quadratic regulator (LQR) technique to determine control gains, which is accepted as a general technique for MIMO systems [11]. In LQR, control gains are set to minimize the quadratic cost function,

$$J = \sum_{k=0}^{\infty} [E(k)\ V(k)] \cdot Q \cdot \begin{pmatrix} E(k) \\ V(k) \end{pmatrix} + U(k)^T \cdot R \cdot U(k). \tag{9}$$

The cost function includes the control errors $E(k)$, accumulated errors $V(k)$, and weighting matrices Q and R. LQR allows us to better negotiate the trade-offs between speed of response and over-reaction to random fluctuation by selecting appropriate Q and R matrices. Q quantifies the cost of control errors and R quantifies the cost of control effort. Since controlling the I/O workload by changing the buffer size incurs higher cost than controlling the CPU workload by changing the update interval, we impose a higher weight on I/O. We choose R=diag(1/3, 1/8) where 1/3 is the cost of I/O control and 1/8 is the cost of CPU control. Then, we choose Q=diag(0.1, 0.1, 0.001, 0.001) to weight the control errors more heavily than the integrated control errors. For more details on the LQR technique, readers are referred to [11].

Finally, in terms of the sampling interval, we sample every 10 seconds. In RTEDBS, the buffer management affects the choice of the sampling interval in particular because the buffer hit ratio changes slowly after adjusting the buffer size. If the sampling interval is too short, controlling the buffer size may not have an effect until the next sampling period, thus wasting the control effort.

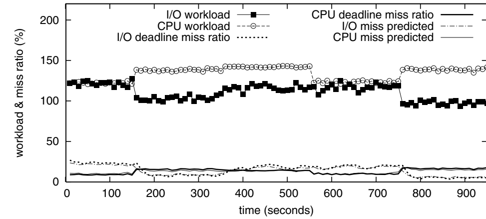


Figure 4: Model validation.

4 Experiment

The main objective of the experiment is to test the effectiveness of controlling I/O and CPU workloads together instead of considering only one in the presence of unpredictable workloads. A scheme which is not I/O-aware is compared to our scheme. For experiments, we developed a simulator that models the proposed RTEDBS. Various workloads were applied to the simulator to test its performance.

4.1 Simulation Settings

The simulated workload consists of sensor data update transactions and user transactions. User transaction workloads are synthesized to test a wide range of workloads and patterns. Update transaction workloads follow similar settings as in [14]. I/O components including NAND flash memory are modeled to follow the performance characteristics typically found in commercial products [2].

4.1.1 Data and Update Transactions

3,000 temporal data objects and another 3,000 non-temporal data objects reside in the database. Among them, the update

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

Parameter	Value
# of temporal data objects	3000
Update interval (P_i)	$Uniform(100ms, 50sec)$
EET_i	$Uniform(2ms, 4ms)$
Actual exec. time	$Normal(EET_i, \sqrt{EET_i})$
Relative deadline	$2 \times P_i$
# data object access/update	1
Update CPU load	$\approx 50\%$
Update I/O load	$\ll 1\%$

Table 2: Update transaction settings.

Parameter	Value
# of non-temporal data objects	3000
$EECT_i$	$Uniform(3ms, 5ms)$
Actual exec. time	$Normal(EET_i, \sqrt{EET_i})$
$EEIT_i$	$NUM_{data} \times ReadAccessTime/page$
Relative deadline	$(EECT_i + EEIT_i) \times$ slack factor
Slack factor	$Uniform(5, 10)$
NUM_{data}(I/O intensive)	$Normal(150, 30)$
NUM_{data}(balanced)	$Normal(100, 20)$
NUM_{data}(CPU intensive)	$Normal(50, 10)$

Table 3: User transaction settings.

Parameter	Value
Read access time / page	$30\mu s$
Write access time	$300\mu s$
Erase time	N/A
Page size	512 bytes
Bank interleaving	N/A
Flash memory max. bandwidth	16MB/sec
BUS bandwidth	∞
Buffer size	30 - 3000 data objects
Buffer replacement algorithm	LRU

Table 4: Flash memory and buffer settings.

stream updates only temporal data. The update period, p_i, follows a uniform distribution $Uniform(100ms, 50sec)$. The expected execution time (EET) of an update transaction is uniformly distributed in the range (3ms, 6ms). The actual execution time is given by normal distribution $Normal(EET_i, \sqrt{EET_i})$. The relative deadline of an update is set to $2 \times p_i$. An update transaction incurs I/O operations if its target data object is not found in the buffer. However, I/O deadlines for update transactions are not set because update transactions are not I/O-intensive. The default settings shown in Table 2 generate about 50% CPU load and less than 1% I/O load when the buffer can hold 50% of total temporal data objects.

4.1.2 User Transaction

A user transaction accesses both temporal and non-temporal data and updates only non-temporal data. The arrival rate of user transactions to the database follows the Poisson distribution. In terms of the number of data accesses per transaction (NUM_{data}), three settings are tested; I/O intensive settings where NUM_{data} is given by $Normal(150, 30)$, balanced settings whose NUM_{data} follows $Normal(100, 20)$, and CPU intensive settings where NUM_{data} is given by $Normal(50, 10)$. The execution of user transactions consists of an I/O phase and a computing phase. The expected execution time ($EECT_i$) of the computation phase is given by the $Uniform(3ms, 5ms)$. The actual execution time follows Normal ($EECT_i$,$\sqrt{EECT_i}$). The expected execution time of the I/O phase ($EEIT_i$) is given by the multiplication of NUM_{data} and the read access time to flash memory per page. The actual execution time of the I/O phase is determined by the number of data objects found in the buffer. The deadline of a user transaction is $(EECT_i + EEIT_i) \times$ slack factor. The slack factor is uniformly distributed ranging from 5 to 10. In I/O intensive settings, user transactions result in more I/O workload than CPU workload; when 100 user transactions arrive per second in addition to default update transactions, about 55% and 30% additional I/O and CPU loads are incurred respectively. In CPU intensive settings, 18% I/O load and 30% CPU load are incurred. In balanced settings, 35% I/O load and 30% CPU load are incurred. In experiments, the arrival rates are adjusted to increase or decrease the I/O and CPU load.

4.1.3 Flash Memory and Buffer

When a transaction accesses data objects, the buffer pool in the main memory is first searched, and if not found, the data objects in the persistent storage are brought to the buffer in main memory. The *Least Recently Used (LRU)* buffer replacement scheme is used for buffer management. The maximum size of the buffer pool is set to hold 3000 data objects (1/2 of total data objects). The buffer pool is shared between the guaranteed service and the best-effort service. The ratio of buffer pool sizes between two services is adjusted dynamically. A NAND flash memory is assumed for persistent data storage. Read operations occur in the unit of a page. The size of a page is set to 512 bytes. Read time per page is set to 30μs. Write time is set to $300\mu s$ and the write operations are done by a translation layer [12] without block-erase; therefore, the erase time is not considered. We assume the size of flash memory is big enough, so garbage collection occurs infrequently [7]. The flash memory is assumed to have only one bank and read/write requests are serialized to the bank. Because we assume no interleaving between banks, the maximum bandwidth of the flash memory accesses is determined only by the access latencies. The obtained maximum bandwidth is about 16MB/sec. The bandwidth of the interconnection bus between the main memory and the flash memory is assumed to be much greater than the flash memory bandwidth, thus avoiding interferences from other bus operations; this assumption is reasonable when we consider the two most common bus technologies, USB [1] and PCI [19], that have 40 MB/sec and 133.3 MB/sec maximum bandwidth respectively.

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

4.2 Baseline

To our best knowledge, the issues of simultaneous control of I/O and CPU workloads for deadline miss ratio management have hardly been studied in real-time databases. Therefore, we compare our scheme (**I/O-CPU**) with the following baseline scheme which was introduced in [14].

CPU-ONLY: This scheme is not I/O-aware; the I/O deadline is not set for transactions, and I/O and CPU deadlines are not distinguished. When deadline miss ratio deviates from the desired miss ratio, only the CPU workload is adjusted by changing the update rates of the temporal data and applying admission control; the I/O workload is not dynamically controlled. This scheme was originally designed for main-memory real-time database that has no or negligible I/O workload. For comparison to our approach, the RTEDBS was modeled by a first-order SISO model; the CPU workload is the control input and the deadline miss ratio is the system output. A PI controller is used.

4.3 Results

Each simulation is run at least 5 times, and their average is taken. 90% confidence intervals are drawn for each data point. For deadline miss ratios, confidence intervals are not shown because they are no more than 0.5%. For experiments, the reference miss ratio is set to 3%. Two metrics are used to compare our approach to the baseline: the average miss ratio and throughput. The average miss ratio indicates if the miss ratio requirement is satisfied, and the throughput indicates if underutilization is occurred to achieve the miss ratio requirement. The throughput is defined as the percentage of timely transactions over the total number of submitted transactions.

4.3.1 Experiment 1: Varying Loads

Computational systems usually show different behavior for different workloads, especially when overloaded. In this experiment, workloads are varied by applying an increasing number of user transactions. Overload can result from either I/O or CPU, or both. We apply three different sets of workloads by applying three different sets of user transactions.

Balanced I/O and CPU: NUM_{data}, the number of accesses data per transaction, follows $Normal(100, 20)$. The user transactions incur almost the same amount of I/O and CPU workload. In this setting, the I/O workload varies from 50% to 190% by applying more user transactions. The CPU workload varies accordingly.

CPU intensive: NUM_{data} follows $Normal(50, 10)$. Each user transaction incurs about 1.5 times more CPU load than I/O load. In this setting, the CPU workload varies from 50% to 190%. The I/O workload varies accordingly.

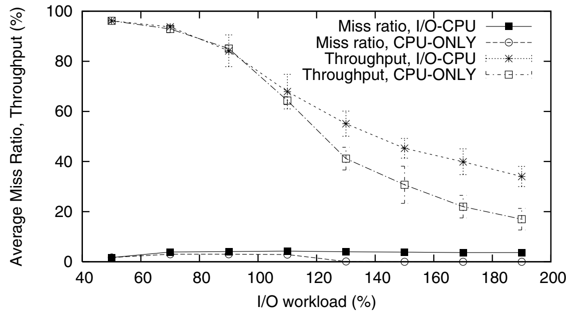
I/O intensive: NUM_{data} follows $Normal(150, 30)$. Each user transaction incurs about 100% more I/O load than CPU load. In this setting, the I/O workload varies from 50% to 190%. The CPU workload varies accordingly.

Note that the $workload$ in the three settings indicates the amount of workload applied to the simulated RTEDBS when all transactions are admitted and no workload control is applied. Furthermore, because the measured I/O workload changes with the size of buffer space and its subsequent buffer hit ratio, the same set of transactions can incur different I/O workloads in different RTEDBS settings. In our experiments, sets of transactions are prepared to incur $x\%$ workload in a RTEDBS setting, whose buffer size is 1000 data objects. The same sets of transactions are applied to each experiment to generate $x\%$ workload. The actual measured workload in each experiment may be different due to different settings, e.g, different buffer sizes, admission controls and controllers

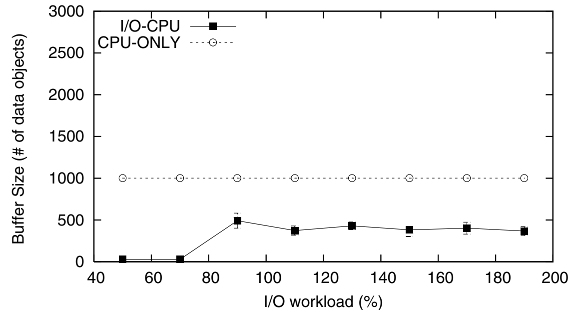
The result is shown in Figure 5-7. The results show that both I/O-CPU and CPU-ONLY effectively achieve the desired miss ratio, that is 3%, in all three different workloads. However, the throughputs of the two schemes to achieve the desired miss ratio are quite different. In all three workloads, I/O-CPU shows much higher throughputs. For instance, when 190% I/O workload (or CPU workload in CPU-intensive) is applied, I/O-CPU achieves 17%-28% higher throughput than CPU-ONLY. The throughput gap between I/O-CPU and CPU-ONLY increases as the workload increases. I/O-CPU shows significantly better throughput than CPU-ONLY when the workload is I/O intensive as in Figure 7. This is because I/O-CPU can effectively use more buffer space as shown in Figure 7-(b). For CPU-ONLY, the buffer size is fixed at 1000 data objects. As the I/O workload increases, it incurs more buffer cache misses. For I/O-CPU, the I/O workload is effectively reduced by utilizing unused buffer space of the best-effort service class, thus increasing the buffer hit ratio.

Interestingly enough, I/O-CPU consumes less buffer than CPU-ONLY both in balanced workloads and CPU-intensive workloads even though it achieves higher throughput than CPU-ONLY as in Figure 5 and 6. For instance, in Figure 6, I/O-CPU achieves 20% more throughput when a 190% CPU workload is applied even though it consumes almost zero buffers. At first, this seems counter-intuitive because providing more buffer cache should reduce the I/O load, thus reducing deadline misses due to I/O overload. This result can be attributed to the close interaction between I/O

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

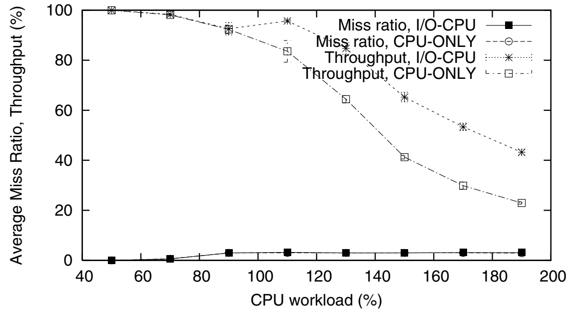


(a) Miss ratio and Throughput

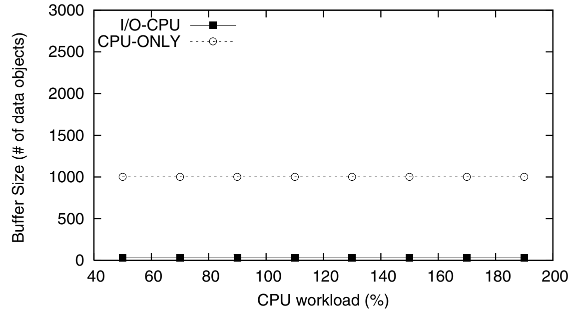


(b) Buffer size

Figure 5: Average performance when varying balanced I/O and CPU workload.

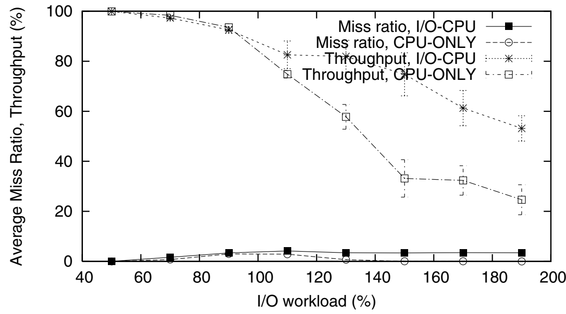


(a) Miss ratio and Throughput

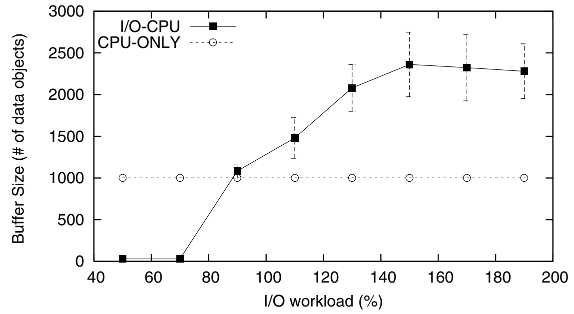


(b) Buffer size

Figure 6: Average performance when varying CPU intensive workload.

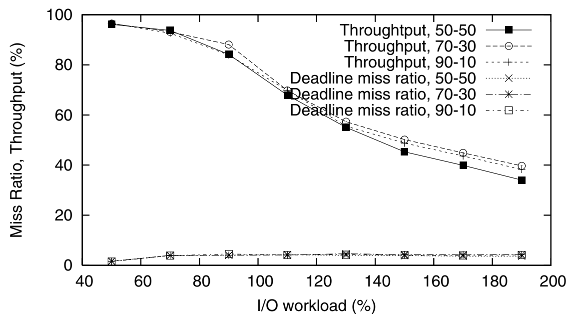


(a) Miss ratio and Throughput

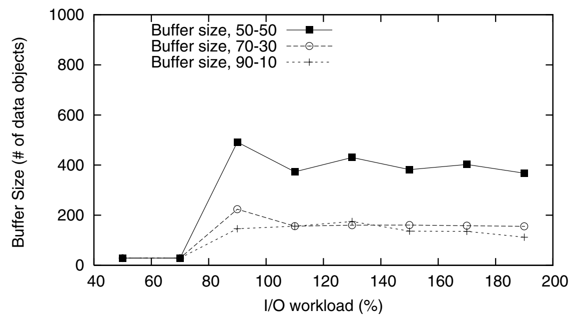


(b) Buffer size

Figure 7: Average performance when varying I/O intensive workload.



(a) Miss ratio and Throughput



(b) Buffer size

Figure 8: $x - y$ data access patterns with varying workload.

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

workloads and CPU workloads. When I/O is not the bottleneck causing transaction deadline misses, having larger buffer cache does not improve the overall miss ratio. In fact, a larger buffer cache can deteriorate the overall miss ratio. As shown in Figure 3-(b), I/O workload does not only control the I/O miss ratio but it also controls CPU miss ratio; increasing I/O workload decreases CPU workload by aborting transactions, and in contrast, decreasing I/O workload increases CPU workload, thus incurring more CPU deadline misses if CPU is already overloaded. In I/O-CPU, the transactions which are less likely to meet the deadline are selectively aborted due to small buffer cache. In other words, when CPU is overloaded while I/O is not, CPU workload can be effectively controlled by adjusting buffer cache size. Furthermore, I/O workload control via buffer cache adjustment is fine-grained; thus, it prevents overshooting. In contrast, CPU-ONLY achieves desired miss ratio only through admission control when adjusting update intervals of temporal data is not available. With admission control, the amount of workload adjustment is coarse-grained. Therefore, overshoot and subsequent underutilization can occur. Actually, CPU-ONLY suffers from underutilization in Figure 5-(a); the miss ratio is far lower than the desired value.

These results demonstrate that controlling I/O workload via buffer management not only fosters effective use of buffer cache resources but also enables fine-grained CPU workload control. Overall, our approach guarantees the desired QoS with much higher throughput; this implies resources are more effectively used in our approach.

4.3.2 Experiment 2: Varying Data Access Patterns

I/O workload is highly affected by data access patterns. By default, we assumed a uniform access pattern. However, the data access patterns can be different from a uniform access pattern. Moreover, the data access patterns can change at run-time. Therefore, the deadline miss ratio management scheme should be robust enough to cope with different data access patterns. In this section, the effect of data contention is tested using a $x - y$ access scheme as described in [14]. In the $x - y$ access scheme, $x\%$ of data accesses are directed to $y\%$ of the data in the database. For instance, with 90-10 access pattern, 90% of data accesses are directed to 10% of data in the database, thus, incurring data contention on 10% of entire data. 50-50 access pattern is essentially equal to uniform access pattern.

We test the robustness of our approach by applying three different $x - y$ access patterns; 90-10, 70-30, and 50-50 data access patterns. Because of space limitation, we only show the result when balanced I/O and CPU workload is applied in Figure 8. As shown in Figure 8-(a), our deadline miss ratio scheme achieves the desired miss ratio in all three different access patterns. Furthermore, the throughput of the three access patterns are not significantly different; 90-10 achieves no more than 5% higher throughput than 50-50. However, the buffer size to achieve the same performance with different access patterns are quite different as shown in Figure 8-(b). As the degree of data contention increases, the smaller size of buffer is enough to achieve the same degree of miss ratio and throughput. This suggests that miss ratio management should have a flexible buffer management scheme to dynamically adjust to different data access patterns. Our results demonstrate that the proposed miss ratio management scheme is robust enough to cope with different data access patterns.

4.3.3 Experiment 3: Transient Performance

Average performance is not enough to show the performance of dynamic systems like RTEDBS. Transient performance including settling times and overshoots should be small enough to satisfy the requirements of applications. Figure 9 shows the result when the workload surges suddenly as a step function. Originally the system is set to have 30% I/O and 70% CPU loads. At 200 seconds, user transactions surge to increase the I/O workload by 190% as a step function. CPU load increases accordingly.

We can see that the CPU workload increases instantly to 330% after I/O workload is reduced by the controller at 230 seconds. This is because reducing I/O workload by increasing the buffer cache size allows more transactions to be issued for the CPU phase without being aborted at the I/O phase, thus, causing a sudden increase in the CPU workload. The miss ratio settles down within 70 seconds. A 70 second settling time may not be satisfactory for some real-time systems whose workloads are highly bursty. However, it satisfies the requirements of a wide range of real-time applications.

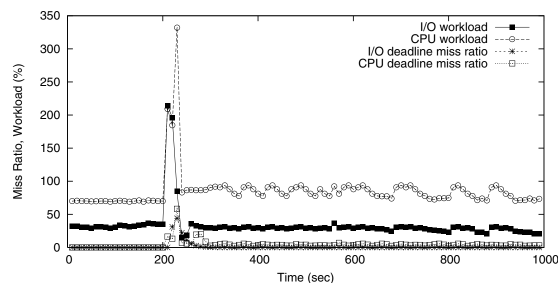


Figure 9: Sudden surge of I/O and CPU workload.

5 Related Work

In the past two decades, the research in RTDBS has received a great deal of attention [18][13]. Most of the studies assume main memory databases [4] due the inherent unpredictability of disk I/O. However, several I/O-aware approaches have been proposed, especially in terms

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.

of deadline-driven disk scheduling [3][8]. Unlike these approaches, our approach assumes flash memory as a non-volatile storage due to predictable performance.

Feedback control has been applied to QoS management in real-time systems due to its robustness against unpredictable operating environments. Lu et al. [16] proposed a feedback control real-time scheduling framework where they presented algorithms for managing miss ratio and utilization. Kang et al. [14] proposed feedback control-based QoS management architectures for main memory RTDBS to support the desired QoS. All these studies consider only CPU resource as the source of deadline misses and I/O is not considered. Consequently, their approaches are based on SISO models. Unlike these approaches, we consider both I/O and CPU resources for deadline miss ratio management. Furthermore, due to the close interaction between I/O and CPU load, we use a MIMO technique.

6 Conclusions and Future Work

Despite the abundance of flash memory as a non-volatile secondary storage in modern real-time embedded systems, the problem of managing data in flash memory for real-time applications has not been well addressed. To address this problem, in this paper we presented an I/O-aware deadline miss ratio management scheme in RTEDBS whose secondary storage is a flash memory.

We showed that I/O and CPU workloads are closely related and a MIMO technique is required to capture the interaction between them. Furthermore, a MIMO feedback control loop was designed to control I/O and CPU workload simultaneously. Our approach gives robust and controlled behavior in terms of guaranteeing the desired miss ratio and achieving high throughput in diverse workloads, access patterns, and even in the presence of transient overloads. The proposed algorithm outperforms the baseline algorithm where only CPU overload is considered. As one of the first studies on I/O-aware deadline miss ratio management, the significance of our work will increase as flash memory increasingly replaces disks in real-time embedded systems.

We will extend this work in several ways. One direction is to investigate the impact of applying different I/O scheduling algorithms such as EDF. Secondly, adptive control approaches can be considered instead of manual tuning to reduce the complexity of manual controller design and to compensate for the non-linear dynamics of the system.

References

[1] http://www.everythingusb.com/usb2/faq.htm.

[2] http://www.samsung.com/products/semiconductor/flash/.

[3] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions with disk resident data. *VLDB '89: Proceedings of the 15th international conference on Very large data bases*, pages 385–395, 1989.

[4] B. Adelberg. *STRIP: A Soft Real-Time Main Memory Database for Open Systems*. PhD thesis, Stanford University, 1997.

[5] M. Amirijoo, J. Hansson, and S. H. Son. Specification and management of QoS in real-time databases supporting imprecise computations. *IEEE Transactions on Computers*, 55(3):304–319, March 2006.

[6] K. P. Brown, M. J. Carey, and M. Livny. Goal-oriented buffer management revisited. *SIGMOD Rec.*, 25(2):353–364, 1996.

[7] L.-P. Chang, T.-W. Kuo, and S.-W. Lo. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Trans. on Embedded Computing Sys.*, 3(4):837–863, 2004.

[8] S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley. Performance evaluation of two new disk scheduling algorithms for real-time systems. *The Journal of Real-Time Systems*, 3(3):307–336, 1991.

[9] J.-Y. Chung, D. Ferguson, G. Wang, C. Nikolaou, and J. Teng. Goal-oriented dynamic buffer pool management for database systems. Technical report, IBM RC19807, October,1995.

[10] Y. Diao, N. Gandhi, and J. Hellerstein. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server. In *Network Operations and Management*, April, 2002.

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

[12] INTEL. Understanding the flash translation layer (FTL) specification. application note ap-684, December 1998.

[13] T.-W. K. E. Kam-yiu Lam. *Real-Time Database Systems: Architecture and Techniques*. Kluwer Academic Publishers, 2001.

[14] K.-D. Kang, S. H. Son, and J. A. Stankovic. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Transacctions on Knowledge and Data Engineering*, 16(10):1200–1216, October 2004.

[15] L. Ljung. *Systems Identification:Theory for the User 2nd edition*. Prentice Hall PTR, 1999.

[16] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.*, 23(1-2):85–126, 2002.

[17] C. Park, J. Seo, D. Seo, S. Kim, and B. Kim. Cost-efficient memory architecture design of nand flash memory embedded systems. *21st International Conference on Computer Design*, 2003.

[18] K. Ramamritham. Real-time databases. *Distrib. Parallel Databases*, 1(2):199–226, 1993.

[19] T. Shanley and D. Anderson. *PCI System Architecture(4th Edition)*. Addison-Wesley Professional, 1999.

[20] J. A. Stankovic, S. H. Son, and J. Liebeherr. BeeHive: Global multimedia database support for dependable, real-time applications. *Lecture Notes in Computer Science*, 1553:51–72, 1998.

Authorized licensed use limited to: Linkoping Universitetsbibliotek. Downloaded on June 17, 2009 at 07:52 from IEEE Xplore. Restrictions apply.