# STAR: Secure Real-Time Transaction Processing with Timeliness Guarantees *

Kyoung-Don Kang Sang H. Son John A. Stankovic
Department of Computer Science
University of Virginia
{kk7v, son, stankovic}@cs.virginia.edu

## Abstract

*Real-time databases are needed in security-critical applications, e.g., e-commerce, agile manufacturing, and military applications. In these applications, transactions and data items can be classified into several security levels according to their clearance and sensitivity levels. It is essential for real-time databases to prevent illegal direct/indirect transfer of sensitive data, e.g., secret trade, manufacturing, or operational data, between transactions belonging to different security levels. Further, transactions should be committed within their deadlines, i.e., before the market, manufacturing, or battle field status changes. In this paper, we present a novel real-time database architecture, in which illegal direct/indirect inter-level information flows are prevented while controlling the deadline miss ratio for admitted transactions to remain below a certain threshold. In our approach, mandatory access control mechanisms are applied for security purposes. QoS management, admission control, and feedback control schemes are applied to support certain guarantees on miss ratio against potential overload and data conflicts. A detailed simulation study shows that our approach can support the specified miss ratio preventing illegal information flows even in the presence of unpredictable workloads and varying degrees of data contention, whereas baseline approaches fail.*

## 1 Introduction

Real-time databases are needed in many security-critical applications, e.g., e-commerce, online stock trading, agile manufacturing, and military applications. In these applications, transactions and data items can be classified according to their clearance and sensitivity levels. It is essential to prevent unlawful information flows between different security levels to protect important data related to trading, manufacturing, and military operations. It is also important for real-time databases to process transactions within their deadlines, i.e., before the market, manufacturing, or battle filed status changes. Current solutions focus on preventing information leakage between different security levels at the cost of providing no performance guarantees. This paper presents a solution which does not only maintain the security property but also gives real-time performance guarantees.

Most of secure database systems apply a mandatory access control mechanism based on the Bell-LaPadula model [6], which is described in terms of *subjects* and *objects*. Objects are data items and subjects are active processes, i.e., transactions in databases, which request access to objects. Data and transactions are classified into several security levels, e.g., Top Secret, Secret, Classified, and Unclassified. In the Bell-LaPadula model, two main restrictions are imposed on all data accesses. A transaction can read an object only if the transaction's security (clearance) level is equal to or higher than the object's security (sensitivity) level. A transaction can write an object only if its security level is equal to or lower than the object's security level. Informally, these access restrictions are called "read-below" and "write-above" restrictions, respectively.

The Bell-LaPadula access restrictions can prevent direct information flows between transactions belonging to different security levels. However, it is not sufficient to prevent indirect information flows, called *covert channels* [14], in which transactions can conspire for an illegal inter-level information transfer. A higher (security) level transaction can either hold or release an exclusive resource or lock on a data item. A conspiring lower level transaction requests the same resource or data item. As a result, the lower level transaction is delayed when the higher level one holds the corresponding resource or lock. The lower level transaction can interpret the presence and absence of the delay as 0 and 1, respectively.

Covert channels based on data and resources can be prevented by favoring lower level transactions in the resolution of inter-level data/resource conflicts, which occur between transactions belonging to different security levels. In this

*Supported, in part, by NSF grants EIA-9900895 and CCR-0098269.

way, lower level transactions can *not* know the presence of higher level transactions, i.e., no covert channel. This notion of *non-interference* should be satisfied to prevent covert channels [9]. However, this may increase the deadline miss ratio, especially for high security level transactions. This is because high security level transactions will be preempted, aborted, and restarted for security purposes in case of inter-level data/resource conflicts. This can be a serious problem, possibly leading to a system failure in secure *real-time* databases, in which timely transaction processing is essential.

In this paper, we present a novel real-time main memory database architecture, called **STAR** (Security and Timeliness Assurance in Real-time databases). STAR fully supports the Bell-LaPadula model and notion of non-interference to prevent direct/indirect information leakage between different security levels. At the same time, our approach provides certain guarantees on average/transient miss ratio for admitted transactions, regardless of their security levels. To our best knowledge, this is the first approach, which prevents illegal direct/indirect information flows while providing timeliness guarantees in real-time databases.

In this paper, transactions are classified into two security levels. In the remainder of this paper, we follow a convention in which low and high security classes are called Classes 0 and 1, respectively. In the CPU scheduling and concurrency control, Class 0 transactions are always treated in a preferred manner to prevent both resource and data based covert channels. We apply QoS management, admission control, and feedback control schemes to support the specified average/transient (deadline) miss ratio for both Classes 0 and 1. Based on the current performance error, i.e., the difference between the specified miss ratio and currently measured one, the feedback control loop computes the required CPU utilization adjustment. QoS management and admission control schemes enforce the required CPU utilization adjustment to support the specified average/transient miss ratio.

In a simulation study, we show that our approach can achieve a significant performance improvement, in terms of miss ratio, compared to several baseline approaches. For a large range of workloads and different degrees of data contention, our approach can support the specified average/transient miss ratio preventing illegal direct/indirect information flows, while other approaches fail.

The rest of this paper is organized as follows. In Section 2, our real-time database model is discussed. Main performance metrics and performance specification issues are also described. Section 3 discusses our real-time database architecture for security (in terms of access control) and miss ratio guarantees in detail. The performance evaluation results are presented in Section 4. Related work is discussed in Section 5. Finally, Section 6 concludes the paper and discusses the avenues for future work.

## 2 Database Model and Performance Metrics

In this section, our real-time database model is discussed and main performance metrics are described. Performance specification issues are discussed in terms of main performance metrics.

### 2.1 Real-Time Database Model

In our approach, a main memory database model, in which the CPU is considered the main system resource, is applied. Main memory databases have been increasingly used for real-time data management such as online auction, stocking trades, e-commerce, and voice/data networking. This is mainly because of their relatively high performance and decreasing main memory cost [2, 5, 24]. We consider firm deadline semantics. Tardy transactions − transactions that have missed their deadlines − are considered valueless, and therefore, are aborted upon their deadline misses.

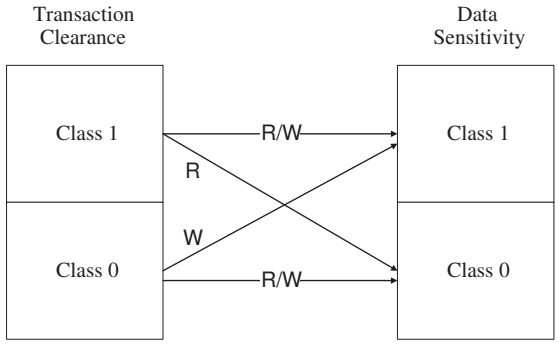


**Figure 1. Access Restrictions in the Bell-LaPadula Model**

In our approach, all transactions and data items are classified into either Class 0 or 1 (i.e., the low or high security class). Transactions can read/write data items according to the access restrictions required by the Bell-LaPadula model as shown in Figure 1. Note that transactions can read/write any data at the same security class. However, Class 1 transactions can not write Class 0 data. In contrast, Class 0 transactions can can not read Class 1 data items. In this way, *direct* information flows between Classes 0 and 1 can be prevented. We also prevent indirect information flows between the two classes.

Separate from the security classes, we also assume that each transaction has two versions, which support different QoS levels, i.e., level 0 and 1. At QoS level 1 (i.e., the

higher QoS level), a transaction requires more CPU cycles. When overloaded, some transactions' QoS can be degraded from level 1 to level 0 to reduce the CPU utilization. For the clarity of presentation, a detailed discussion about the security support and QoS management including their interactions with other components in our real-time database architecture is given in Section 3.

### 2.2 Main Performance Metrics

We consider security an all-or-nothing correctness criterion rather than a performance metric, similar to [8]. Unlike several real-time database research projects such as [3, 21, 22], in which covert channels can be momentarily allowed to improve the miss ratio, we do not consider security a trade-off item for performance reasons. Therefore, deadline miss ratio is considered a single performance metric in our approach:

- *Average Miss Ratio*: For the admitted transactions in Class $i$ $(0 \leq i \leq 1)$, let $\#Tardy_i$ and $\#Timely_i$ represent the number of transactions that have missed their deadlines and the number of transactions that have finished within their deadlines, respectively. The miss ratio of the transactions belonging to Class $i$ measured within a certain time interval is:

$$MR_i = 100 \times \frac{\#Tardy_i}{\#Tardy_i \ + \ \#Timely_i} \ (\%)$$

  In STAR, the average miss ratio for Classes 0 and 1 should be below a certain miss ratio threshold, e.g., 5%. Note that we manage $MR_0$ and $MR_1$ separately instead of considering the aggregate miss ratio. This is to prevent an undesirable situation, in which one security class suffers a high miss ratio, whereas the aggregate miss ratio is below the specified value.

- *Transient Miss Ratio*: Long-term performance metrics such as average miss ratio are not sufficient for the performance specification of dynamic systems, since their performance can widely vary in a short time interval. To address this problem, transient performance metrics are adopted from control theory for a real-time system performance specification [16]. *Overshoot* ($M_p$) is the worst-case system performance in the transient system state, i.e., the highest miss ratio over the specified threshold in the transient state. *Settling time* ($t_s$) is the time for a transient overshoot to decay and reach the steady state performance, in which $0 \leq MR_i \leq \theta_i + 0.01 \times \theta_i$ where $MR_i$ $(0 \leq i \leq 1)$ is the miss ratio of Class i transactions, and $\theta_i$ is the miss ratio threshold required for Class i.

### 2.3 Performance Specification

In this paper, we consider the following performance specification as an example to illustrate the applicability of our approach for secure transaction processing with miss ratio guarantees: "*Perf-Spec* $= \{MR_0,\ MR_1 \leq 5\%,\ M_p \leq 20\%,\ t_s \leq 80sec,\ CPU\ Utilization \geq 80\%\}$". This specification requires us to limit the average miss ratio below 5% for Classes 0 and 1, respectively. We also set $M_p$ = 20%, therefore, an $MR_0$ or $MR_1$ overshoot should not exceed $6\% = 5\% \times (1 + 0.2)$. A potential overshoot should decay within 80sec. Given a sampling rate, the overshoot and settling time have a trade-off relation with each other. It is very hard, if at all possible, to minimize both the overshoot and settling time for a given sampling rate [20]. In this paper, we consider a somewhat long settling time, e.g., think time between trades, is tolerable as long as the corresponding overshoot is low. We also require the CPU utilization to be at least 80% to avoid a trivial case in which average/transient $MR_0$ and $MR_1$ are satisfied due to the underutilization. In summary, this entire set of performance specification is the performance guarantee supported by our approach.

## 3 An Architecture for Security and Timeliness Guarantees

In this section, we present our real-time database architecture for security and timeliness assurance as shown in Figure 2. Incoming real-time transactions with firm deadlines are scheduled in one of the two ready queues according to their security classes. Scheduling and concurrency control policies are provided for transaction processing. At each sampling instant, $MR_0$, $MR_1$, and the CPU utilization are monitored. The miss ratio and utilization controllers compute the required CPU utilization adjustment ($\Delta U$ in Figure 2) based on the current performance error such as the miss ratio overshoot or CPU underutilization. When $\Delta U < 0$ (i.e., overloaded), the QoS manager degrades the transaction QoS. The admission controller enforces the remaining utilization adjustment after potential QoS degradation, i.e., $\Delta U_{new}$, to prevent overload. In the following subsections, each component is discussed in detail.

### 3.1 Transaction Handler

The transaction handler is a fundamental component for real-time database services. It consists of a concurrency controller (CC) and a basic scheduler. The basic scheduler schedules transactions in either $Q_0$ or $Q_1$ according to their security classes, i.e., Class 0 (Class 1) transactions are scheduled in $Q_0$ ($Q_1$). The Bell-LaPadula access restrictions are applied for data accesses. CPU scheduling and

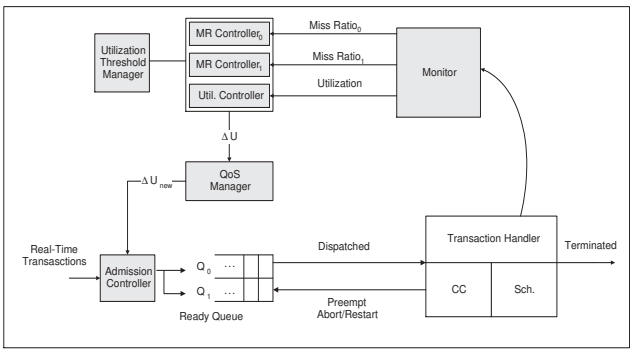


**Figure 2. STAR Architecture**

concurrency control are performed in a manner to support the notion of non-interference as follows.

To prevent covert channels, Class 0 transactions (scheduled in $Q_0$) always receive a higher priority than Class 1 transactions. A transaction in $Q_1$ can be scheduled if there is no ready transaction in $Q_0$. Also, a Class 1 transaction is preempted upon the arrival of a Class 0 transaction. In this way, Class 0 transactions can not know the presence of Class 1 transactions, therefore, the notion of non-interference is supported. In each queue, transactions are scheduled in an EDF manner. Since illegal information flows are not applicable inside each security class, the EDF scheduling policy can be applied to improve the miss ratio in each class.

For concurrency control, we use two phase locking high priority (2PL-HP) [1], in which a low priority transaction is aborted upon a conflict. 2PL-HP is well studied in real-time database research and it is free of a priority inversion. In our approach, upon an inter-level (i.e., between Classes 0 and 1) data conflict a Class 0 transaction always receive a higher priority. In this way, we support the notion of non-interference. Class 0 transactions can not know the presence of Class 1 transactions, since Class 0 transactions are never delayed by Class 1 transactions due to inter-level data conflicts. (We assume the delay for aborting a transaction is negligible. Hence, we assume a covert channel can not be established using the abort delay.) Note that intra-level data conflicts are resolved in favor of a transaction with the earliest deadline, since there is no security problem (in terms of mandatory access control) within one security class.

Note that STAR provides guarantees on average/transient miss ratio for both Classes 0 and 1. In real-time databases, it is essential to support consistent timing guarantees in terms of both long-run average and transient miss ratio, regardless of security classes. In this way, users can be assured of secure transaction processing with consistent timing guarantees even when the system is in the transient state. For several reasons, it is very challenging to support the specified average/transient miss ratio guarantees for both Classes 0 and 1, especially for Class 1. As the Class 0 workload increases, a multitude of Class 1 transactions can be preempted, aborted, and restarted due to possible inter-level data/resource conflicts. This is because the CPU scheduling and concurrency control schemes are performed in favor of Class 0 to support the notion of non-interference as discussed before. Further, precise workload models and possible data/resource conflicts are usually unknown a priori in database applications. To provide $MR_0$ and $MR_1$ guarantees despite these challenges, we apply a feedback-based approach depicted in Figure 3, and described in the following subsections.

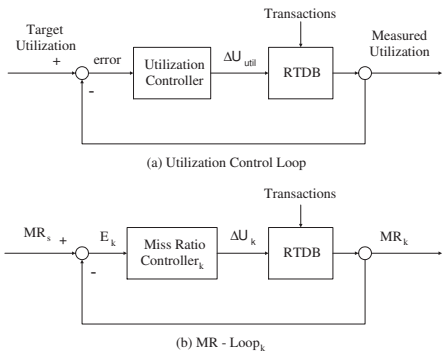


**Figure 3. Miss Ratio/Utilization Controllers**

## 3.2 Feedback Control

Feedback control is well known for its effectiveness to support a specified performance when the system model may include uncertainties [20]. The specified performance can be achieved by dynamically adapting the system behavior considering the current performance error measured in the feedback control loop. In this paper, we adapt and extend a feedback control scheduling policy for service differentiation in real-time databases [13] to provide average/transient guarantees for both $MR_0$ and $MR_1$ as follows.

### 3.2.1 Miss Ratio Controllers

In our approach, digital PI (proportional and integral) controllers are employed to control average/transient $MR_0$ and $MR_1$.[1] As shown in Figure 3, a miss ratio control loop $MR - LOOP_k$ belonging to Class $k$ $(0 \leq k \leq 1)$ computes a miss ratio control signal $\Delta U_k$ based on the current performance error, $E_k = MR_s - MR_k$, i.e., the difference between the specified miss ratio and the currently measured

[1] A P controller alone can not eliminate a constant steady state error [20]. We verified that P controllers can not meet *Perf-Spec*, i.e., the CPU is underutilized in our experiments. In this paper, we do not consider more complex controllers, since PI controllers closely meet *Perf-Spec*.

miss ratio for Class $k$. Let $E_k(i)$ represent the miss ratio error for Class k at the $i_{th}$ sampling instant. At the $n_{th}$ sampling instant, the miss ratio control signal for Class k is:

$$\Delta U_k = KP \times E_k(n) + KI \times \sum_{i=1}^{n} E_k(i) \qquad (1)$$

When overloaded, $\Delta U_k$ can become negative to request the reduction of the CPU utilization via QoS management and/or admission control. KP and KI stand for proportional and integral controller gains to be tuned to support the specified average/transient miss ratio.

#### 3.2.2 Profiling and Controller Tuning

To support the specified average/transient miss ratio, miss ratio controllers are tuned as follows.

- $MR - LOOP_0$: To tune a miss ratio controller, the miss ratio gain, $G_{MR} = Max\{\frac{Miss\ Ratio\ Increase}{Unit\ Load\ Increase}\}$, should be derived under the worst case set-up to support a certain miss ratio guarantee [17]. To derive $G_{MR}$, the performance of the controlled system, i.e., a real-time database, should be profiled. Average $MR_0$ was measured for loads increasing from 60% to 200% by 10%. To consider the worst case, all incoming transactions are admitted, and all transactions are processed at QoS level 1. From this, we derived the $MR_0$ gain $G_{MR_0} = 1.295$ when the load increases from 120% to 130%. We set the sampling period to 5sec (for both miss ratio and utilization control). According to control theory, frequent sampling could improve the transient performance such as overshoot and settling time. However, too frequent sampling could incur the sudden QoS degradation in our approach. For this reason, we selected the longest sampling period among several tested values, which can support the required overshoot and settling time. (To do this, we measured the overshoot and settling time in Matlab by increasing the sampling period by 1sec starting from 1sec.) Using $G_{MR_0}$ and the sampling period, we applied the Root Locus design method in Matlab [20] to tune KP and KI to satisfy the average/transient $MR_0$ specified in *Perf-Spec*. We have selected the closed loop poles at $p_0, p_1 = 0.778, 0.539$. The feedback control system is stable, since the closed loop poles are inside the unit circle. The corresponding values are KP = 0.224 and KI = 0.176.

- $MR-LOOP_1$: For $MR-LOOP_1$, we also measured $MR_1$ for loads increasing from 60% to 200%. For $MR_1$, the gain $G_{MR_1}$ = 2.288 when the load increases from 80% to 90%. It is necessary to profile $MR_1$ as well, since we aim to support not only the average, but also the transient miss ratio specification for Class 1. Note that $G_{MR_1}$ is higher than $G_{MR_0}$, and the sharpest $MR_1$ increase is observed at lower loads ($80\% - 90\%$) compared to Class 0. This is mainly because the CPU scheduling and concurrency control are performed in favor of Class 0 for security reasons. To tune $MR - LOOP_1$, we applied the Root Locus method to support the average/transient miss ratio specified in *Perf-Spec*, similar to the tuning process of $MR - LOOP_0$. The closed loop poles are located inside the unit circle for the stability of $MR - LOOP_1$. The corresponding values are KP = 0.127 and KI = 0.176.

#### 3.2.3 Utilization Controller

To prevent a potential underutilization, one utilization control loop is employed. Note that it is necessary to use both miss ratio and utilization controllers to avoid control saturation problems. A utilization controller saturates at utilization 100%. In contrast, a miss ratio controller saturates when the system is underutilized (0 miss ratio as a result). The miss ratio (utilization) controller is activated when the system is overloaded (underutilized) to support the stability of the feedback control system by avoiding the controller saturation problems, similar to [16, 17].

At each sampling instant, a PI controller derives the utilization control signal $\Delta U_{util}$ based on the current utilization error, i.e., the difference between the target and current utilization as shown in Figure 3. The utilization control signal is derived in a PI controller, similar to Eq. 1. The utilization controller is also tuned using the Root Locus method, but we do not discuss the details due to space limitations. To further improve the CPU utilization, we also employ a utilization threshold manager as follows.

#### 3.2.4 Utilization Threshold Manager

For many complex real-time systems, e.g., real-time databases, the schedulable utilization bound is unknown or can be very pessimistic [17]. In real-time databases, the utilization bound is hard to derive, if at all possible. This is partly because database applications usually include unpredictable aborts/restarts due to data/resource conflicts. It is a hard problem to decide a proper utilization threshold, which can avoid both the possible underutilization and excessive deadline misses due to an overly pessimistic and optimistic utilization threshold, respectively.

For this reason, we apply an online approach in which the utilization threshold (the target utilization in Figure 3 (a)) is dynamically adjusted considering the current real-time system behavior. The utilization threshold is initialized at a relatively low set point ($U_{low}$), e.g., 80%. If $MR_0$ and $MR_1$ are currently zero, the utilization threshold is incremented by a certain step size, e.g., 2%, unless the new uti-

lization threshold exceeds 100%. The utilization threshold will be continuously increased as long as $MR_0$ and $MR_1$ are zero. The utilization threshold will be switched back to $U_{low}$ when the miss ratio controller takes control. This can be considered a somewhat conservative approach, since the utilization threshold could be decreased to the initial value upon a few deadline misses. The main objective of this back-off scheme is to prevent a potential miss ratio overshoot in real-time databases because of an overly optimistic utilization threshold. In some applications where a transient miss ratio overshoot can be considered relatively tolerable, e.g., web information systems, the utilization back-off can be performed in a stepwise manner.

Note that our approach is computationally lightweight and self-adaptive requiring no a priori knowledge about a specific workload model. Using this approach, the potentially time-varying utilization threshold can be closely approximated [13].

#### 3.2.5 Derivation of a Single Control Signal

For the sake of consistency in control, we derive a single control signal out of two miss ratio signals and one utilization control signal. We first derive a single miss ratio control signal, $\Delta U_{MR}$ from $\Delta U_0$ and $\Delta U_1$. For this derivation, we need to consider two cases: (1) both $\Delta U_0$ and $\Delta U_1$ are negative at the current sampling instant, (2) only one of the two is negative or none of them is negative. In the first case, both $MR_0$ and $MR_1$ are violated. Thus, we set $\Delta U_{MR} = \sum_{k=0}^{1} \Delta U_k$ to require the enough CPU utilization reduction to avoid a significant miss ratio increase in the consecutive sampling instants. Otherwise, we set $\Delta U_{MR} = Minimum(\Delta U_0, \Delta U_1)$ to support a smooth transition from one system state to another, similar to [17]. After deriving $\Delta U_{MR}$, we set the current control signal $\Delta U = Minimum(\Delta U_{util}, \Delta U_{MR})$ for a similar reason.

#### 3.2.6 Integrator Antiwindup

All feedback controllers in Figure 3 are digital PI controllers. An integral controller can improve the performance of the feedback control system when used with a proportional controller. However, it is necessary to avoid for the integrator to erroneously accumulate control signals, since this can lead to a substantial overshoot later [20]. For this reason, we apply the integrator antiwindup technique [20] as follows.

- **Case 1** ($\Delta U_{MR} > \Delta U_{util}$): Turn on the integrator for the utilization controller, but turn off all integrators of $MR - LOOP_k$ $(0 \leq k \leq 1)$, since the current $\Delta U = \Delta U_{util}$. This prevents an erroneous accumulation of miss ratio control signals.
- **Case 2** ($\Delta U_{MR} \leq \Delta U_{util}$): In this case, turn off the integrator for the utilization controller, since the current $\Delta U = \Delta U_{MR}$. This prevents an erroneous accumulation of the utilization control signal. For miss ratio controllers, if $\Delta U_0 < 0$ and $\Delta U_1 < 0$, turn on the integrators for both $MR - LOOP_0$ and $MR - LOOP_1$. This is because the current $\Delta U = \sum_{k=0}^{1} \Delta U_k$. Otherwise, only turn on the integrator of $MR - LOOP_k$ whose miss ratio control signal is smaller than the other. This prevents an erroneous accumulation of the miss ratio control signal by one of the two miss ratio controllers whose signal is greater than the other.

### 3.3 QoS Manager and Admission Controller

When the system is overloaded, i.e., $\Delta U < 0$, the CPU utilization can be reduced by degrading the QoS level from 1 to 0 for some transactions in the system. More specifically, QoS degradations are performed by applying an imprecise computation technique, called milestone approach [15].

The admission controller is another system component that can enforce the control signal together with the QoS Manager, if necessary. As shown in Figure 2, the QoS Manager informs the admission controller of the new control signal, called $\Delta U_{new}$, adjusted after possible QoS degradations. Admission control is necessary, since under severely overloaded conditions the QoS Manager itself might not be able to enforce the current $\Delta U$ entirely. In contrast, more transactions should be admitted when underutilized. (We assume that all incoming transactions are already authenticated, therefore, our admission controller only considers workload adjustments instead of dealing with security problems such as denial of service attacks.)

In our approach, a newly incoming transaction can be admitted, if its estimated CPU utilization requirement is currently available. The current utilization is examined by aggregating the utilization estimates of the previously admitted transactions.

## 4 Performance Evaluation

In this section, we analyze the performance of our approach. The main objective of the performance evaluation is to observe whether or not STAR can support *Perf-Spec*, while supporting the Bell-LaPadula access restrictions and notion of non-interference. For this, we have developed a real-time database simulator. Baseline approaches are introduced for performance comparisons. We describe workload variables and performed sets of experiments, and present the performance evaluation results.

### 4.1 Simulation Model

Our simulation model is described as follows.

- **Execution Time and Deadline** A source, $Source_i$, generates a group of real-time transactions whose inter-arrival time is exponentially distributed. $Source_i$ is associated with an estimated execution time ($EET_i$) and an average execution time ($AET_i$). We set $EET_i = Uniform(5ms, 20ms)$. By generating multiple sources, we can derive transaction groups with different average execution time and average number of data accesses in a statistical manner. By increasing the number of sources, we can also increase the workload applied to the simulated real-time database. This is because more transactions will be generated in a certain time interval. We set $AET_i = (1 + EstErr) \times EET_i$, in which $EstErr$ is used to introduce the execution time estimation errors. Note that STAR and all baseline approaches are only aware of the estimated execution time. Upon the generation of a transaction, $Source_i$ generates an actual execution time by applying the normal distribution $Normal(AET_i, \sqrt{AET_i})$ to introduce the execution time variance in the corresponding transaction group. We set *deadline = arrival time + average execution time* $\times$ *slack factor*. A slack factor is uniformly distributed in a range (10, 20).

- **QoS**: Transactions at QoS level 0 are assumed to require a half of the original execution time, i.e., $actual\ execution\ time_0 = 0.5 \times actual\ execution\ time_1$. In our performance evaluation, we directly relate QoS levels 0 and 1 to numeric values 0 and 1, respectively. By taking the average of QoS levels for all processed transactions, we can observe the fraction of transactions serviced at their full QoS levels. For example, when the measured average QoS = 0.6, 60% of the transactions are processed at QoS level 1 (and 40% of transactions are processed at QoS level 0).

- **Number of Data Accesses**: The number of data accesses for $Source_i$ is derived in proportion to the length of $EET_i$, i.e., $N_{DATA_i} =$ *data access factor* $\times$ $EET_i = (5, 20)$. Therefore, longer transactions access more data in general. Upon the generation of a transaction, $Source_i$ associates the actual number of data accesses with the transaction by applying $Normal(N_{DATA_i}, \sqrt{N_{DATA_i}})$ to introduce the variance in the transaction group.

- **Security and Data**: Each source is associated with a security class, i.e., Class 0 or 1, and transactions generated from a source are accordingly scheduled in $Q_0$ or $Q_1$ as shown in Figure 2. All data items in the database are also classified into Classes 0 and 1. When accessing data, each transaction follows the access restrictions imposed by the Bell-LaPadula model, i.e., read-below and write-above. We also vary the database size from 1,000 to 100,000 data items to vary the degree of data contention. A more detailed discussion is given in Section 4.3.

### 4.2 Baselines

We have developed three baseline approaches to compare with STAR:

- **Open**: In this approach, all incoming transactions are admitted and serviced with the full QoS regardless of the current miss ratio. The closed-loop scheduling is not employed. Hence, all the shaded components in Figure 2 are turned off. Note that most database systems take this open-loop and non-adaptive approach.

- **Open-AC**: This is similar to *Open* except that admission control is applied in this approach. Hence, potential overloads can be partially managed by admission control.

- **Covert-AC**: This approach is similar to *Open-AC*, however, the notion of non-interference is not supported for performance reasons. This approach only considers transaction deadlines to resolve both intra- and inter-level data/resource conflicts. A mandatory access control mechanism based on Bell-LaPadula model is applied to prevent illegal *direct* information transfer between Classes 0 and 1. However, the indirect information transfer using covert channels are not prevented at all. By considering this baseline approach, we can measure the performance penalty, if any, required to fully support the notion of non-interference in our approach. Note that *Covert-AC* could show a better performance, especially in terms of $MR_1$, compared to existing approaches such as [3, 21, 22], in which inter-level data/resource conflicts are temporarily resolved in terms of deadlines instead of security levels, to trade off security to improve the miss ratio, if necessary.

### 4.3 Workload Variables

In this section, we describe the workload variables used in the performance evaluation.

- **AppLoad** (Applied Load): In general, computational systems may show different, possibly worse, performance for increasing loads, especially when overloaded. We use a variable, called $AppLoad$, to apply

**Table 1. Load Relieved by QoS degradation**

| Avg. QoS | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|
| Rel. Load | 37.5% | 30% | 22.5% | 15% | 7.5% | 0% |
| Rel. Load | 50% | 40% | 30% | 20% | 10% | 0% |

different workloads in the simulation. Note that this variable indicates the load assuming that all incoming transactions are admitted and all transactions are serviced at their full QoS levels. The actual load can be reduced in a tested approach by applying admission control and degrading the QoS, if necessary. The workload relieved from QoS degradation (Rel. Load) = $0.5 \times (1 - \textit{Average QoS}) \times \textit{AppLoad}$, since the transaction execution time at QoS level 0 is a half of that at level 1 as discussed before. Table 1 presents the loads relieved from QoS degradation when $AppLoad = 150\%$ and 200%, respectively. Remaining potential overload, if any, should be managed by admission control to prevent a miss ratio overshoot.

- **EstErr** (Execution Time Estimation Error): In our experiments, $EstErr$ is used to introduce errors in execution time estimates as described in Section 4.1. A high execution time estimation error could generally induce a difficulty in real-time scheduling.

- **HCR** (High Priority Class Ratio): As the Class 0 workload increases, it could be more challenging to support the specified average/transient miss ratio, especially for $MR_1$. This is because Class 0 transactions receive a higher priority in the CPU scheduling and concurrency control to prevent covert channels in all tested approaches except *Covert-AC*. To adjust the Class 0 workload, we define a workload variable:

$$HCR = 100 \times \frac{\textit{\#Class 0 Transactions}}{\sum_{i=0}^{1} \textit{\#Class i Transactions}} (\%).$$

- **DB_SIZE** (Database Size): Database performance can vary as the degree of the data contention changes [1, 10]. To model this, we applied our approach for different $DB_SIZE$ (database sizes), i.e., 1,000, 10,000 and 100,000 data items. In general, a small database size may incur a higher degree of data contention for given a set of transactions [1]. In $Experiment\ Sets$ 1 − 3 (discussed in the next subsection), we only present the case of 1,000 data items, which may have the highest data contention due to the smallest database size among the tested ones. In $Experiment\ Set$ 4, we show that our approach can support *Perf-Spec* for a different database size, i.e., 100,000. (We have also measured the performance of our approach for the size of 10,000 and found similar performance result. To avoid repetition, we only present the performance results for the two ends of the spectrum, i.e., 1,000 and 100,000.)

### 4.4 Experiments

Using the workload variables, the most representative sets of experiments performed in our simulation study are described as follows.

- **Experiment Set 1**: For performance evaluation, we applied $AppLoad = 70\%, 100\%, 150\%$, and 200% and we set $HCR = 20\%$. We set $EstErr = 0$ in this set of experiments. This is the best case set-up in our experiments.

- **Experiment Set 2**: To show the applicability of our approach against inaccurate execution time estimates, we have evaluated the performance for $EstErr$ = 0, 0.25, 0.5, 0.75, and 1. We set $AppLoad = 200\%$ and $HCR = 20\%$.

- **Experiment Set 3**: We evaluate the performance for $HCR$ = 20%, 40%, 60%, 80%, and 100%. We set $AppLoad = 200\%$ and $EstErr = 1$ for this set of experiments. This is the worst case set-up in our experiments due to the increasing $HCR$ in addition to the highest $AppLoad$ and $EstErr$ among the tested values.

- **Experiment Set 4**: In this set of experiments, we show that our approach can also support the specified average/transient $MR_0$ and $MR_1$ for the other database sizes tested. Other workload variables except $DB_SIZE$ are equal to $Experiment\ Set$ 3. In this way, we can show the general applicability of our approach for different sizes of databases under the worst case set-up among the tested settings.

In our experiments, one simulation run lasts for 10 minutes of simulated time. For all performance data, we have taken the average of 10 simulation runs and derived the 90% confidence intervals. Confidence intervals are plotted as vertical bars in the graphs showing the performance evaluation results. (For some performance data, the vertical bars may not be noticeable due to the small confidence intervals.)

### 4.5 Experiment Set 1: Effects of Increasing Loads

In this section, we compare the performance of *Open, Open-AC, Covert-AC,* and $STAR$ for increasing $AppLoad$. We compare the average performance of all tested approaches and present the transient miss ratio of STAR.

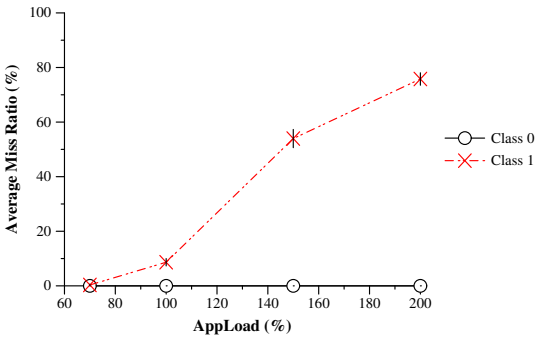


**Figure 4. Average Miss Ratio for Open**

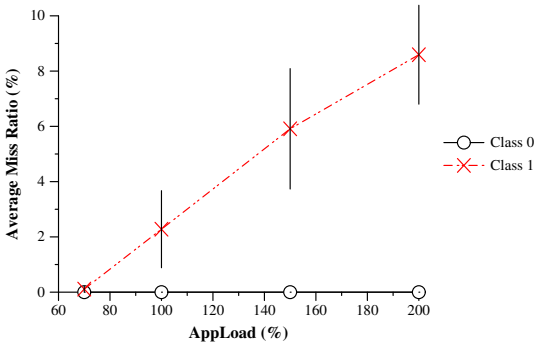


**Figure 5. Average Miss Ratio for Open-AC**

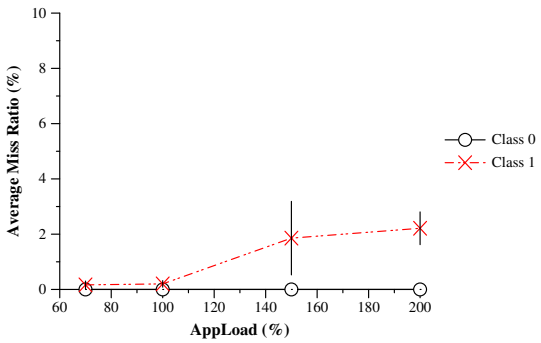


**Figure 7. Average Miss Ratio for STAR**

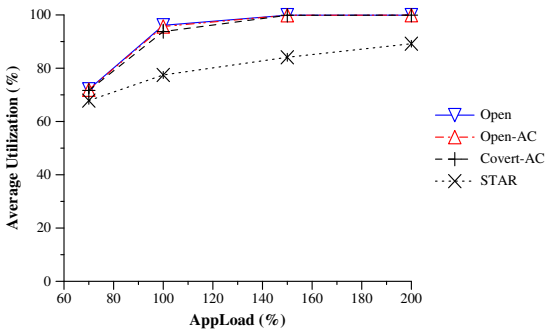


**Figure 8. Average Utilization**

#### 4.5.1 Average Performance

As shown in Figures 4 – 7, all tested approaches (i.e., all baseline approaches and STAR) show near zero $MR_0$ (miss ratio for Class 0) for increasing $AppLoad$. However, tested approaches showed different $MR_1$ as follows.

As shown in Figure 4, for $Open$ $MR_1$ is approximately 80% when $AppLoad = 200\%$. As shown in Figure 5, by applying admission control *Open-AC* significantly improved $MR_1$ compared to $Open$: $MR_1$ is $8.59 \pm 1.79\%$ when $AppLoad = 200\%$. However, this exceeds the average miss ratio 5% specified in *Perf-Spec*. As shown in Figure 6, *Cover-AC* shows near zero $MR_0$ and $MR_1$ by allowing potential covert channels. As shown in Figure 7, STAR shows $MR_1$ of $2.21 \pm 0.59\%$ when $AppLoad = 200\%$, which meets *Perf-Spec* but this is slightly worse than *Covert-AC*. This is because *Covert-AC* only considers transaction deadlines for scheduling and concurrency control. As a result, sensitive information in real-time databases may not be protected. Also, in this set of experiments we set $EstErr = 0$. However, precise execution time estimates are generally not available. In fact, *Covert-AC* showed a relatively poor performance when only approximate execution time estimates are available. It did not satisfy even the

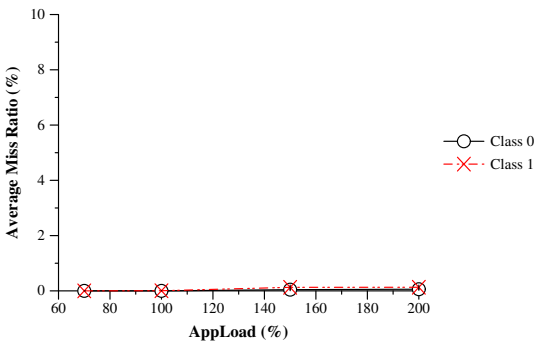


**Figure 6. Average Miss Ratio for Covert-AC**

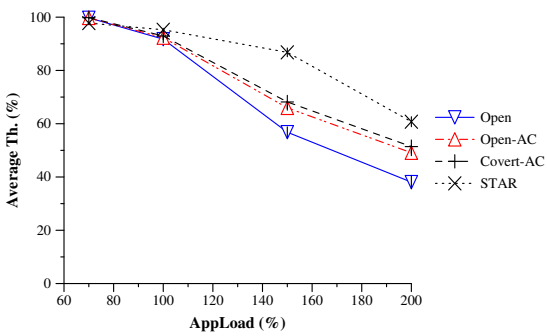


**Figure 9. Average Throughput**

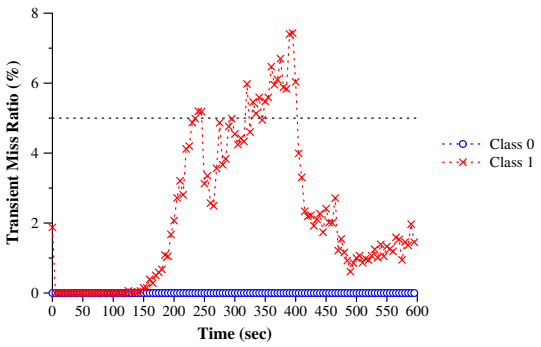


**Figure 10. Transient $MR_0$ and $MR_1$ for STAR**

average miss ratio specification, which can be considered relatively easy to meet than the specified transient miss ratio. (A detailed discussion is given in Section 4.6.)

As shown in Figure 8, for *Open, Open-AC*, and *Covert-AC* the utilization quickly reaches 100% as $AppLoad$ increases leading to potential overload and the resulting miss ratio increase. In comparison, for $STAR$ the utilization ranges between $70\% - 90\%$ for increasing $AppLoad$ as shown in Figure 8. This is because in our approach the CPU utilization is controlled in the feedback control loop to avoid both overload and underutilization. Under overload, the CPU utilization is reduced by admission control and QoS degradation, if necessary.

One may argue that STAR can provide a good miss ratio, since admission control drops many transactions. However, we show that admission control can actually improve the *real-time database throughput* $= \frac{\#\texttt{Timely}}{\#\texttt{Submitted}}$ where $\#\texttt{Timely}$ and $\#\texttt{Submitted}$ represent the number of transactions which commit within their deadlines and that submitted to system (before admission control).

As shown in Figure 9, $Open$ shows the lowest throughput among the tested approaches as $AppLoad$ increases. Since $Open$ simply admits all incoming transactions, the simulated real-time database is overloaded incurring many deadline misses. As a result, the throughput of $Open$ is below 40% when $AppLoad = 200\%$. By applying admission control (and allowing potential covert channels in *Covert-AC*), *Open-AC* and *Covert-AC* improve the throughput by approximately 10% compared to $Open$ when $AppLoad = 200\%$. From these results, observe that admission control is a sensible approach to prevent overload (and improve the throughput) in real-time databases.

Compared to *Open*, *Open-AC*, and *Covert-AC*, STAR shows a relatively high throughput especially given a high $AppLoad$ as shown in Figure 9. When $AppLoad = 200\%$, the throughput of STAR is $66.67 \pm 3.67\%$ achieving more than 25% throughput improvement compared to $Open$ at the expense of the degraded QoS. For STAR, the average QoS decreases from 100% (when $AppLoad = 70\%$) down to $63.17 \pm 7.03\%$ when $AppLoad = 200\%$, i.e., approximately 55% of transactions are processed at QoS level 1. (In Experiment Sets $2 - 4$, STAR showed the average QoS between $50\% - 60\%$, while achieving the throughput between $60\% - 70\%$, similar to the results presented in this section. We do not include the results due to space limitations.)

In all baseline approaches, the average QoS is 100%, since they do not adapt the QoS regardless of the current system status such as miss ratio. In real-time systems, QoS degradation using some techniques such as the mile stone approach [15] is generally considered more tolerable than deadline misses. For example, it might be better to degrade the resolution of radar images rather than having lots of deadline misses in target tracking. Also, it was empirically observed that in web-based e-commerce systems a significant fraction of potential clients leave the corresponding web sites if responses for queries take longer than a certain time, e.g., 5sec [7]. In this case, the QoS of the web page can be degraded by only returning the textual information to improve the response time. Large objects such as images can be transmitted later when the network and server loads are low.

### 4.5.2 Transient Miss Ratio of STAR

In this section, we present the transient miss ratio observed for our approach. We do not present the transient miss ratio for baseline approaches, since they can not even support the specified average miss ratio under many simulation settings as discussed before. In Figure 10, transient $MR_0$ and $MR_1$ are presented for our approach when $AppLoad = 200\%$, i.e., the highest load tested. A dotted horizontal line in the figure represents the specified miss ratio threshold, i.e., 5%. $MR_0$ is near zero through the experiment. The $MR_1$ overshoot of 7.43% is observed at 395sec slightly exceeding the specified overshoot $M_p = 6\%$. Since the feedback control system is *reactive* to the performance error, some overshoot might be inevitable [20]. Especially, it is hard to meet the transient $MR_1$ specification in our approach due to the scheduling and concurrency control favoring Class 0 to support the notion of non-interference. For this set of experiments, the degree of data conflicts can also be very high due to the smallest database size among the tested ones. In Section 4.8, we show that the transient miss ratio is improved for larger database sizes. A further investigation is reserved for future work.

In terms of settling time, the specified settling time of 80sec is satisfied; a few miss ratio overshoots exceeding $M_p(= 6\%)$ first observed at 360sec decays until 400sec. Therefore, we may consider that our approach has closely satisfied the transient miss ratio specification.

### 4.6 Experiment Set 2: Effects of Increasing Execution Time Estimation Error

In this section, we compare the performance of *Open-AC, Covert-AC*, and *STAR* for increasing $EstErr$. $Open$ is dropped due to its poor performance as discussed in Section 4.5. We set $AppLoad = 200\%$.

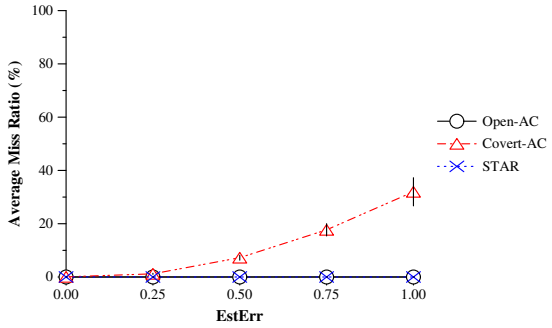


**Figure 11. Average $MR_0$**

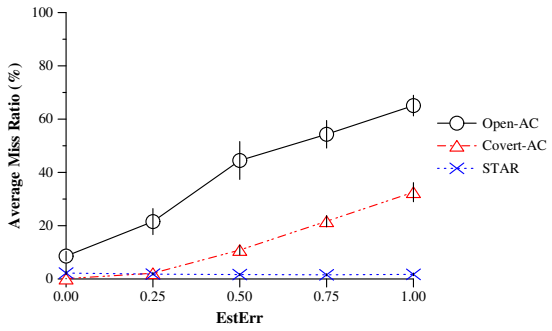


**Figure 12. Average $MR_1$**

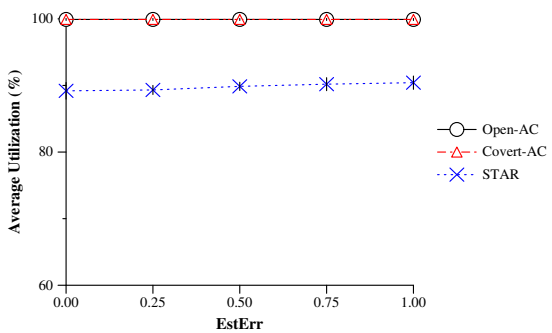


**Figure 13. Average Utilization**

#### 4.6.1 Average Performance

*Covert-AC*, which showed the best performance in $Experiment\ Set\ 1$, shows over 30% of $MR_0$ and $MR_1$ as

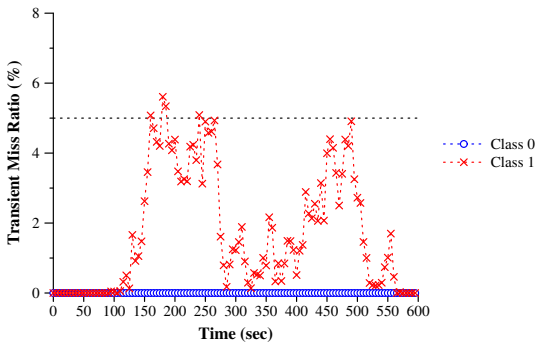


**Figure 14. Transient Miss Ratio for STAR**

shown in Figures 11 and 12. This is mainly because there can be non-trivial errors in admission control for increasing $EstErr$; too many transactions are admitted to the system and eventually miss their deadlines. As shown in Figures 11 and 12, for *Covert-AC* deadline misses are equally divided into Classes 0 and 1 in an approximate sense. This is because *Covert-AC* only considers transaction deadlines for scheduling and concurrency control. For *Open-AC* and $STAR$, $MR_1$ is relatively high compared to $MR_0$, since they support the notion of non-interference via scheduling and concurrency in favor of Class 0 as discussed before.

In Figure 11, *Open-AC* shows near zero $MR_0$. However, $MR_1$ exceeds 60% when $EstErr = 1$ as shown in Figure 12. Even though *Open-AC* might be considered secure in terms of database access control, such a high miss ratio is not acceptable for real-time database applications. In contrast, STAR shows near zero $MR_0$, and $MR_1$ is only $1.72\% \pm 0.49\%$ when $EstErr = 1$ as shown in Figures 11 and 12. This is because in our approach the system performance, in terms of miss ratio and the CPU utilization, is periodically monitored, and admission control and QoS degradation are applied, if necessary. For *STAR*, the average QoS ranged between $50\% - 60\%$ for this set of experiments.

As shown in Figure 13, for *Open-AC* and *Covert-AC* the CPU is saturated leading to significant violations of the specified miss ratio. $STAR$ managed the utilization around 90% without violating the specified miss ratio.

#### 4.6.2 Transient Miss Ratio of STAR

In Figure 14, we present transient $MR_0$ and $MR_1$ observed for STAR when $EstErr = 1$, i.e., the highest $EstErr$ tested. Similar to Figure 10, $MR_0$ is near zero through the experiment. The highest $MR_1$ is 5.6% at 160sec and it decays in 10sec (two sampling periods). Therefore, the overshoot and settling time specified in *Perf-Spec* are satisfied. One can observe that $MR_1$ overshoot is slightly lower than that presented in Figure 10. This is mainly due to the relatively low QoS observed in this set

of experiments (approximately 50% when $EstErr = 1$) compared to that measured in $Experiment\ Set\ 1$ (approximately 63% when $AppLoad = 200\%$). In this set of experiments, the feedback controller becomes more reactive for increasing $EstErr$ requiring a relatively large CPU utilization reduction under overload.

## 4.7 Experiment Set 3: Effects of Increasing the Class 0 Workload

In this section, we measure the performance of our approach for increasing $HCR$, i.e., the increasing Class 0 workload compared to Class 1. We have also measured the performance for *Open-AC* and *Covert-AC*, however, we do not include the results due to space limitations. In these approaches, the specified miss ratio is significantly violated showing the miss ratio even worse than the case where $EstErr = 1$ presented in Section 4.6.

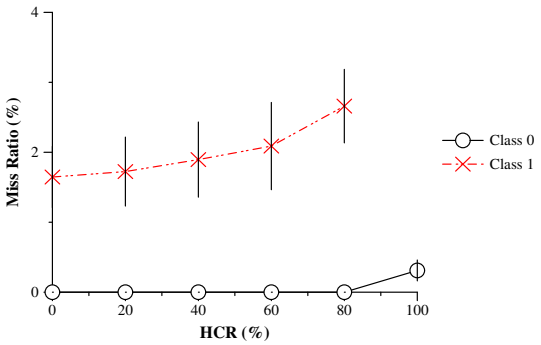


**Figure 15. Average Miss Ratio for STAR**

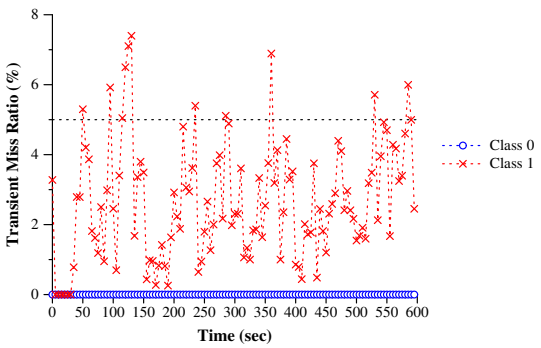


**Figure 16. Transient Miss Ratio for STAR**

### 4.7.1 Average Performance

As shown in Figure 15, $MR_0$ is below 1% when $HCR = 100\%$. $MR_1$ is also below 3% when $HCR = 80\%$, the worst case set-up for $MR_1$ tested. ($MR_1$ is not plotted for $HCR = 100\%$, since this workload does not include any Class 1 transaction.) The average utilization ranged between $87\% - 93\%$. The average QoS ranged between $50\% - 65\%$, similar to $Experiment\ Set\ 2$.

### 4.7.2 Transient Miss Ratio of STAR

As shown in Figure 16, when $HCR = 80\%$ the miss ratio overshoot of 7.39% is observed at 130sec. This slightly exceed the allowed overshoot $M_p = 6\%$, however, overshoots exceeding $M_p$ in Figure 16 decayed within at most three sampling instants, i.e., between 130sec $-$ 145sec. Note that we measured the transient miss ratio for other $HCR$ values, but only present the result for the worst case set-up, i.e., $HCR = 80\%$, due to space limitations. We verified that STAR shows the better average/transient miss ratio for other values of $HCR$.

## 4.8 Experiment Set 4: Effects of Data Contention

We also applied our approach for different sizes of simulated real-time databases to show the general applicability against varying degrees of data contention. In Figures 17 and 18, we show the performance of our approach for a modeled database of the largest size, i.e., 100,000 data items. Other workload variables including the increasing $HCR$ are same with the $Experiment\ Set\ 3$. For the baseline approaches, we have also performed the same set of experiments, but we do not include their performance results due to space limitations. In the baseline approaches, the specified miss ratio $MR_0$ and/or $MR_1$ are violated similar to the previous experiments.

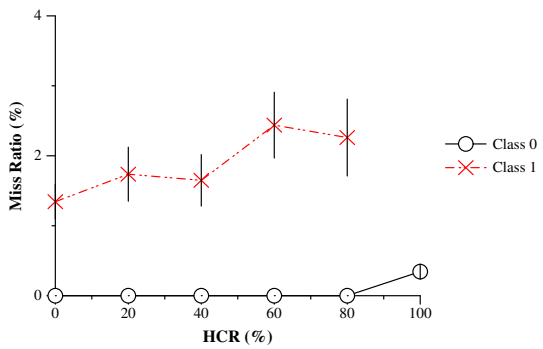


**Figure 17. Average Miss Ratio for STAR**

### 4.8.1 Average Performance

As shown in Figure 17, $MR_0$ is near zero except when $HCR = 100\%$. Also, $MR_1$ is below 3% for all $HCR$ values tested. The average utilization ranged between $88\% - 92\%$ and the average QoS ranged between $50\% - 60\%$. From this, we can conclude that our approach can support the average performance specified in *Perf-Spec*.

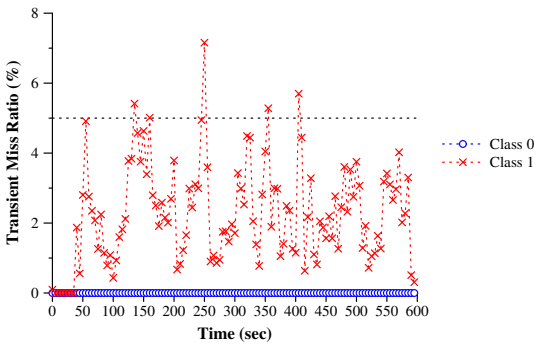


**Figure 18. Transient Miss Ratio for STAR**

#### 4.8.2 Transient Miss Ratio of STAR

Transient $MR_0$ and $MR_1$ for $HCR = 80\%$, the worst case HCR for $MR_1$ among the tested values, are presented in Figure 18. As shown in Figure 18, $MR_0$ is near zero. $MR_1$ violates the specified miss ratio less frequently compared to Figures 10 and 16. Also, the only overshoot of 7.15% exceeding the specified $M_p(= 6\%)$ at 250sec decays within one sampling period. This is because the degree of data contention could be relatively low in a larger database [1]. In this paper, we mainly presented the performance results for the smallest database size among the tested ones, since it can model the highest degree of data contention. We also verified that our approach can support the specified performance for different database sizes under general workload settings applied for our experiments presented in this paper.

## 5 Related Work

Recently, transaction processing in (non-real-time) multi-level secure databases has been actively studied [4, 12]. As a result, some commercial products such as Sybase Secure SQL Server [23], Trusted Oracle [18], and Informix-OnLine/Secure [11] can partially prevent covert channels. However, secure transaction processing in real-time databases, which are needed in security-critical environments, has attracted relatively little research efforts despite its importance.

Several research work such as [3, 21, 22] considered the security of real-time database a performance criterion, which can be traded off under overloaded conditions. In their approach, illegal direct information flows between different security levels are prevented by applying the access restrictions based on the Bell-LaPadula model. However, they did not completely prevent the illegal information transfer using covert channels. In their approaches, it was assumed that temporary trade-off of security is acceptable to improve the deadline miss ratio. However, this type of approaches can face a significant problem by allowing a security hole; sensitive information can be illegally disclosed to transactions without appropriate levels of security clearance. This possible leakage of important trading, manufacturing, and operational data might adversely affect the success of e-commerce, agile manufacturing, and military operations.

In [8], George et al. addressed this problem by preventing the illegal information transfer, both direct and indirect ones. In their approach, the Bell-LaPadula access restrictions and notion of non-interference are always enforced regardless of the current deadline miss ratio. They presented a novel concurrency control policy called dual approach. Among different security levels, data conflicts are resolved in favor of lower security level transactions to support the notion of non-interference. At each security level, data conflicts are resolved in favor of transaction deadlines to improve the miss ratio, since there is no possibility of an illegal inter-level information flow within one security level.

Our approach is similar to [8] in the sense that we do not allow illegal intra- and inter-level information flows. However, unlike their approach we provide guarantees on average/transient miss ratio. By preventing covert channels, we exceed the US military's security standards defined in the *Orange Book* [19]. In that specification, covert channels of bandwidth of less than one bit per second, which could be hard to measure, are typically considered acceptable. Furthermore, we provide guarantees on average/transient miss ratio considering timing constraints prevalent in real-time database applications. To our best knowledge, no previous work has considered providing miss ratio guarantees while preventing the illegal direct/indirect inter-level information transfer in real-time databases.

## 6 Conclusions and Future Work

In this paper, we presented a novel real-time database architecture for secure transaction processing with certain miss ratio guarantees. Considering the importance of security and transaction timeliness in many real-time database applications, e.g., e-commerce, online stock trading, agile manufacturing, and military operations, the contribution of our work could be significant. Our contribution might increase as the demand for secure real-time transaction processing increases. In a simulation study, we showed that our approach can significantly improve the miss ratio compared to several baselines, while supporting the mandatory access control mechanism based on Bell-LaPadula model and the notion of non-interference to prevent the illegal direct/indirect information transfer between different security classes. According to the performance evaluation results, our approach was able to support the specified average/transient miss ratio. In contrast, the baseline ap-

proaches failed to support the specified miss ratio. In the future, we are interested in supporting multiple security levels and considering other security models, which are more flexible than the multilevel security model.

## References


[1] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. *ACM Transactions on Database System*, 17:513–560, 1992.

[2] B. Adelberg, H. Garcia-Molina, and B. Kao. Applying Update Streams in a Soft Real-Time Database System. In *ACM SIGMOD*, 1995.

[3] Q. Ahmed and S. Vrbsky. Maintaining Security in Firm Real-Time Database Systems. In *14th Annual Computer Security Applications Conference*, 1998.

[4] Atluri et al. Multilevel Secure Transaction Processing: Status and Prospects. In *IFIP Workshop on Database Security*, 1997.

[5] Baulier et al. DataBlitz Storage Manager: Main Memory Database Performance for Critical Applications . In *ACM SIGMOD - Industrial Session: Database Storage Management*, 2000.

[6] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical report, The Multics Corp., 1976.

[7] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating User-Perceived Quality into Web Server Design. In *9th International World Wide Web Conference*, 2000.

[8] B. George and J. R. Haritsa. Secure Concurrency Control in Firm Real-Time Database Systems. *Distributed and Parallel Databases*, 5:275–320, 1997.

[9] J. Goguen and J. Meseguer. Security Policy and Security Models. In *IEEE Symposium on Security and Privacy*, 1982.

[10] M. Hsu and B. Zhang. Performance Evaluation of Cautious Waiting. *ACM Transactions on Database Systems*, 17(3):477–512, 1992.

[11] Informix Software, Inc. *Informix-OnLine/Secure Administrator's Guide*, 1993.

[12] S. Jajodia. Database Security and Privacy. *ACM Computing Surveys*, 28(1), March 1996.

[13] K. D. Kang, S. H. Son, and J. A. Stankovic. Service Differentiation in Real-Time Main Memory Databases. In *the 5th IEEE International Symposium on Object-oriented Real-time Distributed Computing*, April 2002.

[14] B. W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, 1973.

[15] K. J. Lin, S. Natarajan, and J. W. S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Real-Time System Symposium*, December 1987.

[16] C. Lu, J. Stankovic, T. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance Specifications and Metrics for Adaptive Real-Time Systems. In *Real-Time Systems Symposium*, Orlando, Florida, November 2000.

[17] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms. *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23(1/2), May 2002.

[18] Oracle, Corp. *Trusted Oracle Administrator's Guide*, 1992.

[19] DoD Trusted Computer System Evaluation Criteria. Department of Defense Standard, DoD 5200.28-STD, 1985.

[20] C. L. Phillips and H. T. Nagle. *Digital Control System Analysis and Design (3rd edition)*. Prentice Hall, 1995.

[21] S. H. Son, R. Mukkamala, and R. David. Integrating Security and Real-Time Requirements using Covert Channel Capacity. *IEEE Transactions on Knowledge and Data Engineering*, 12(6), Dec 2000.

[22] S. H. Son, R. Zimmerman, and J. Hansson. An Adaptable Security Manager for Real-Time Transactions. In *Euromicro Conference on Real-Time Systems*, pages 63–70, Stockholm, Sweden, June 2000.

[23] Sybase, Inc. *Sybase Secure SQL Server Security Administrator's Guide*, 1993.

[24] TimesTen Performance Software. *TimesTen White Paper*. Available in the World Wide Web, http://www.timesten.com/library/index.html, 2001.