Secure Buffering in Firm Real-Time Database Systems

Binto George   Jayant R. Haritsa*

Database Systems Lab
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore 560012, India
{binto,haritsa}@dsl.serc.iisc.ernet.in

Abstract

The design of **secure buffer managers** for database systems supporting real-time applications with firm deadlines is studied here. We first identify the design challenges and then present **SABRE**, a new buffer manager that aims to address these challenges. SABRE guarantees covert channel-free security, employs a fully dynamic one-copy allocation policy for efficient usage of buffer resources, and incorporates several optimizations for reducing the number of killed transactions and for decreasing the unfairness in the distribution of killed transactions across security levels. Using a detailed simulation model, the real-time performance of SABRE is evaluated against unsecure conventional and real-time buffer management policies. Our experiments show that SABRE provides security with only a modest drop in real-time performance. Finally, we present **FSABRE**, an adaptive admission control-augmented version of SABRE, which efficiently ensures close to ideal fairness across transaction security levels while remaining within the information leakage bandwidth limits specified in military standards.

1 Introduction

Many applications of real-time database systems **(RTDBS)** arise in safety-critical installations and military systems where enforcing **security** is crucial to the success of the enterprise. Security violations in an RTDBS can occur if, for example, information from the "secret" database is transferred by corrupt high security transactions to the "public" database where they are read by conspiring low security transactions. Such *direct* violations can be eliminated by implementing the classical Bell–LaPadula security model [13] which imposes restrictions on the data operations permitted to transactions, based on their security levels. The Bell–LaPadula model is not sufficient, however, to protect from "covert channels". A covert channel is an *indirect* means by which a high security transaction can transfer information to a low security transaction [12]. For example, if a low security transaction requests access to an exclusive resource, it will be delayed if the resource is already held by a high security transaction, otherwise it will be granted the resource immediately. The presence or absence of the delay can be used as a "signaling" or encoding mechanism by a high security transaction passing secret information to the low security transaction. Note that, from the system perspective, nothing "obviously illegal" has been done in this process by the conspiring transactions.

Covert channels can be prevented by ensuring that low security transactions do not "see" high security transactions – this notion is formalized in [7] as *non-interference*, that is, *low security transactions should not be able to distinguish between the presence or absence of high security transactions*. This can be implemented, for example, by providing *higher priority to low security transactions* whenever a conflict occurs between a low security transaction and a high security transaction. From a system perspective, it translates to developing database managers that support the non-interference feature.

In a recent study [6], we made a detailed investigation of the performance implications of providing covert-channel-free security in the context of real-time applications with "firm-deadlines" [9]. For such applications, completing a transaction after its deadline has expired is of no utility and may even be harmful. Therefore, transactions that miss their deadlines are "killed", that is, immediately aborted and discarded from the system without being executed to com-

*Supported in part by a research grant from the Department of Science and Technology, Government of India.

Proceedings of the 24th VLDB Conference
New York, USA, 1998

pletion. Accordingly, the performance metric is the *percentage of killed transactions*.[1]

The focus in our earlier study was on the design of high-performance secure *concurrency control managers*. We move on, in this paper, to considering the equally important and related issue of designing *buffer managers* that can both guarantee security and provide good real-time performance.

Design Challenges

Buffer managers take advantage of the temporal locality typically exhibited in database reference patterns to enhance system performance by minimizing disk activity. In doing so, however, they open up possibilities for covert channels – in fact, *many more than those associated with concurrency control*. For example, the presence or absence of a delay in acquiring a free buffer slot, or the presence or absence of a specific data page in the buffer pool, or the allocation of a particular (physical) buffer slot, could all be used as channel mediums, whereas in concurrency control, data access time is the primary medium. Apart from this "multitude of channel mediums" problem, there are several additional problems that arise while integrating security into the RTDBS framework in general, and into the buffer manager in particular:

First, a secure RTDBS has to *simultaneously* satisfy two requirements, namely, provide security and minimize the number of killed transactions. Unfortunately, the mechanisms for achieving the individual goals often work at cross-purposes. In an RTDBS, high priority is usually given to transactions with earlier deadlines in order to help their timely completion. On the other hand, in secure DBMS, low security transactions are given high priority in order to avoid covert channels (as described earlier). Now consider the situation wherein a high security process submits a transaction with a tight deadline in a secure RTDBS. In this case, it becomes difficult to assign a priority since assigning a high priority may cause a security violation whereas assigning a low priority may result in a missed deadline.

Second, a major problem arising out of the preferential treatment of low security transactions is that of "fairness" – high security transactions usually form a disproportionately large fraction of the killed transactions. Note that this is an especially problematic issue because it is the "VIPs", that is, the high security transactions, that are being discriminated against in favor of the "common-folk", that is, the low security transactions.

Unfortunately, achieving fairness without permitting covert channels appears to be *fundamentally* impossible for dynamically changing workloads. The issue then is whether it is possible to design fair systems while still guaranteeing that the covert channel information leakage bandwidth is within acceptable levels.

Third, it is straightforward to eliminate covert channels by using "static" resource allocation policies wherein each transaction security class has a pre-assigned buffer space, or by using "replication"-based policies wherein multiple copies of data pages are maintained. However, such policies may result in very poor resource utilization. The challenge therefore is to design "dynamic" and "one-copy" policies that are demonstrably secure.

Finally, unlike concurrency control, which essentially deals only with regulating data access, a buffer manager has *multiple* components – buffer allocation, buffer replacement and pin synchronization [8], all of which have to be made secure.

In summary, for all of the above reasons, making a real-time buffer manager implementation secure involves significant design complexity. We have conducted a detailed study of this issue and report on the results here. To the best of our knowledge, these results represent the *first work* in the area of secure real-time buffer management.

2 Related Work

The only prior research we are aware of on secure buffer management is the recent study by Warner et al [16] in the context of *conventional* (i.e. non-real-time) DBMS. A number of design alternatives for secure buffer allocation, replacement and synchronization were explored in this study. In particular, they *statically* divided the buffer pool among the various security levels,[2] but also presented a *dynamic* scheme called "slot stealing" wherein buffers, currently underutilized at the low security level, could be borrowed by high security transactions. Later, if needed, these borrowed slots could be reclaimed by the low security transactions. They also considered replication-based schemes wherein multiple copies of the same disk page could be present in the buffer pools of various security levels. Finally, the performance of these policies was evaluated on a simulated model of a buffer manager.

While their dynamic allocation scheme is an interesting approach, it suffers from two problems: First, it is only *partially dynamic*, since low security transactions are not allowed to utilize the currently unused buffer slots of high security transactions. Therefore, resource wastage could still result. Second, it must be ensured that the number of unpinned clean buffer slots in the high security buffer pool must *at all times* be as large as the number of slots borrowed from the low security pool. This is necessary to support the immediate return of slots reclaimed by the low security transactions and thereby to prevent covert channels. The utility of the slot stealing is diminished by this constraint since it places restrictions on the high security accesses.

Our work differs from the above in that, apart from addressing *real-time* applications, we consider (a) fully dynamic and unconstrained one-copy allocation policies, (b) the buffer manager's performance role in the context of an *entire system* where all remaining components are secure, and (c) the issue of fairness across security levels.

Just as there has been little prior research on secure buffer management, there has been a similar paucity with regard to (unsecure) real-time buffer management. The

[1] Or equivalently, the percentage of missed deadlines.

[2] A scheme where the static buffer allocation is periodically reviewed and altered is also presented, but this scheme can result in covert channels.

only work that we are aware of are [10, 11] in which real-time versions of the popular LRU [4] and DBMIN [2] policies were developed and evaluated. The results of these studies were inconclusive, however, since they arrived at differing conclusions regarding the utility of adding real-time information to buffer management.

3 Security Model

Most secure database systems have access control mechanisms based on the Bell–LaPadula model [13]. This model is specified in terms of *subjects* and *objects*. An object is a data item, whereas a subject is a process that requests access to an object. Each object in the system has a *classification* level (e.g., Secret, Classified, Public, etc.) based on the security requirement. Similarly, each subject has a corresponding *clearance* level based on the degree to which it is trusted by the system.

The Bell–LaPadula model imposes two restrictions on all data accesses:

1. A subject is allowed **read** access to an object only if the former's clearance is *higher than or identical to* the latter's classification

2. A subject is allowed **write** access to an object only if the former's clearance is *identical to or lower than* the latter's classification.

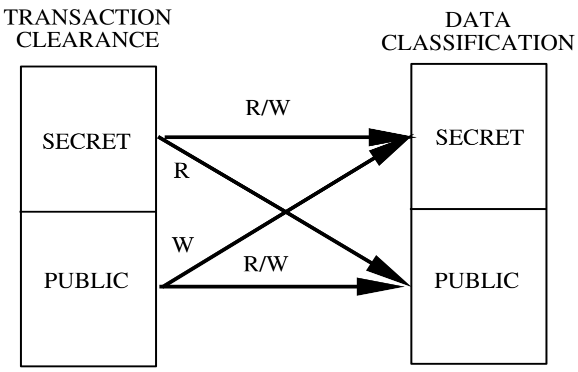

Figure 1: Bell-LaPadula access restrictions

The Bell-LaPadula conditions effectively enforce a "read below, write above" constraint on transaction data accesses (an example is shown in Figure 1), and thereby prevent *direct* unauthorized access to secure data. They are not sufficient, however, to protect from "covert channels", as described in the Introduction.

One approach to tackling covert channels is by introducing "noise" in the form of *dummy* transactions. The problem with this solution, however, is its inefficiency stemming from the considerable waste of system resources by the dummy transactions. An alternative approach is the *non-interference* formalism described in the Introduction – we exclusively use this approach here.

3.1 Orange Security

For many real-time applications, security is an "all-or-nothing" issue, that is, it is a *correctness* criterion. In such "full-secure" applications, metrics such as the number of killed transactions or the fairness across transaction clearance levels are secondary *performance* issues. However, there are also applications for whom it is acceptable to have well-defined *bounded-bandwidth* covert channels in exchange for performance improvement. For example, the US military's security standards, which are defined in the so-called "Orange Book" [3], specify that covert channels with bandwidth of less than *one bit per second* are typically acceptable – we will hereafter use the term "orange-secure" to refer to such applications.

In this study, we consider the design of buffer managers for both full-secure and orange-secure real-time applications.

4 Buffer Model

Buffer managers attempt to utilize the temporal locality typically exhibited in database reference patterns to maximize the number of buffer hits and thereby reduce disk activity. Three kinds of reference localities are usually observed: *inter-transaction locality* (i.e. "hot spots"), *intra-transaction locality* (transaction's internal reference locality), and *restart locality* (restarted transactions make the same sequence of accesses as their original incarnation).

At any given time, the slots in the buffer pool can be grouped into the following four categories: **Pinned** – Buffer slots containing valid pages that are currently being accessed (in read or write mode) by executing transactions; **Active** – Buffer slots containing valid pages that have been earlier accessed by currently executing transactions; **Dormant** – Buffer slots containing valid pages that have not been accessed by any currently executing transaction (these pages were brought in by previously completed transactions); **Empty** – Buffer slots that are empty. Further, the first three categories (Pinned, Active, and Dormant) can be further subdivided into **clean** and **dirty** groups, which contain the set of clean and dirty (i.e. modified) pages, respectively, of that category.

In addition to the traditional buffer management components [8] of *allocation*, *replacement* and *pin synchronization*, an additional component that arises specifically in the secure real-time domain is *pin preemption*. Consider the case, for example, where a low clearance transaction requests buffer space but the slots currently held by high clearance transactions which would normally be candidates for replacement cannot be forced out because they are all pinned. In this case, one option is for the low clearance transaction to wait for slots to become unpinned but this would immediately result in covert channel possibilities. Therefore, there appears to be no choice but to "break the pin", that is, permit the replacement with the proviso that the high clearance transaction holding the original pin is immediately notified that the page it had assumed to be stable had actually been replaced – the high clearance transaction can then either abort or, preferably, redo the corrupted operation at a later time.

Security considerations are not the only reason for supporting pin preemption. Consider, for example, the case where a tight-deadline transaction wishes to utilize the slot currently pinned by a slack-deadline transaction. If the urgent transaction is blocked, this would mean that high priority transactions are being blocked by low priority transactions, a phenomenon known as *priority inversion* in the real-time literature [15]. Priority inversion can cause the affected high-priority transactions to miss their deadlines and is clearly undesirable. Therefore, pin preemption is useful from *real-time* considerations also.

5 Buffer Management Policies

In this section, we present the set of buffer management policies evaluated in our study, which cover the spectrum from unsecure non-real-time policies to fully secure real-time policies.

5.1 The SABRE Policy

Our new **SABRE** (Secure Algorithm for Buffering in Real-time Environments) policy provides complete covert-channel-free security. It is *fully dynamic* since there is *no preallocation* of buffer slots to security levels, and *unconstrained* in that no restrictions are placed on high clearance transaction accesses. Further, only *one copy* of a data page is maintained in the buffer pool. These features ensure efficient use of buffer resources. Finally, it incorporates several optimizations both for reducing the overall number of killed transactions and for decreasing the unfairness in the distribution of killed transactions across clearance levels. We describe its design in more detail below.

5.1.1 Priority Assignment

As mentioned in the Introduction, assigning priorities in a secure RTDBS is rendered difficult due to having to satisfy multiple functionality requirements. Given the paramount importance of security, the database system is forced to assign transaction priorities based primarily on clearance levels and only secondarily on deadlines. In particular, priorities are assigned as a vector P = (LEVEL, INTRA), where LEVEL is the transaction clearance level and INTRA is the value assigned by the priority mechanism used *within* the level. Clearance levels are numbered from zero upwards, with zero corresponding to the lowest security level. Further, priority comparisons are made in *lexicographic order* with lower priority values implying higher priority.

With the above scheme, transactions at a lower clearance have higher priority than all transactions at a higher clearance, a necessary condition for non-interference. For the intra-level priority mechanism, any priority assignment that results in good real-time performance can be used. For example, the classical Earliest Deadline assignment [14] wherein transactions with earlier deadlines have higher priority than transactions with later deadlines. In this case, the priority vector would be P = (LEVEL, DEADLINE) – this priority assignment is used in SABRE.

5.1.2 Security Features

Merely assigning priorities in the manner described above is not *sufficient*, however, to make the buffer manager secure in terms of the non-interference requirement. We describe below the main additional features incorporated in SABRE to ensure covert-channel-free security (the complete set of features is available in [5]):

1. The contents of a buffer slot are not "visible" to a transaction if the existence of the page in the slot is a consequence of actions performed by higher clearance transactions. Such a slot will appear as a *(pseudo)-empty* slot to the requesting transaction.

   More specifically, for a transaction of a specific clearance level, all the buffer pages of all higher clearance levels are not visible whereas the Pinned and Active buffer pages of its own level and all lower clearance levels are visible (the visibility of Dormant pages is discussed below in Feature 3). This means that, for example, a low clearance transaction is not given immediate access to a page that has been earlier brought into the buffer pool by a high clearance transaction. Instead, it is forced to wait for the same period that it would have taken to bring the page from disk had the page not been in the buffer pool (the determination of this period is discussed in Section 5.1.5).

2. High clearance transactions can replace the Dormant slots of low clearance transactions.

3. Dormant pages, of *any* level, are visible only to transactions of the *highest* clearance level – for no other transactions are they visible. That is, a low clearance transaction is not given immediate access to a Dormant page even at its own or lower clearance levels. Instead, it is forced to wait for the same period that it would have taken to bring the page from disk had the page not been in the buffer pool.

4. Priority-based pin preemption is supported at all clearance levels.[3]

5. When selecting from among the set of (really) Empty slots, the slot is *randomly chosen* and not in a predefined order.

Feature 1 above is an obvious requirement to ensure the absence of signaling between high clearance and low clearance transactions. Feature 2, which allows high clearance transactions to "steal" the Dormant slots of low clearance transactions, is included for the following reason: If high

[3] Supporting pin preemption at the *lowest* clearance level is not essential for security but is retained for performance reasons (see Section 5.1.3).

clearance transactions could not replace *any* pages of low clearance transactions, then the buffer pool would very quickly fill up with the Active and Dormant pages of the low clearance transactions and after this the high clearance transactions would not be able to proceed further. That is, *starvation* of high clearance transactions would occur.

Note, however, that there is a price to pay for ensuring that covert channels do not result in spite of slot-stealing. This is expressed in Feature 3 wherein low clearance transactions are made to wait for access to Dormant pages at even their *own and lower* clearance levels, thereby partially losing the benefits that could be gained from inter-transaction locality. Feature 4 ensures that high clearance accesses can be unrestricted without causing covert channels since borrowed slots can be returned immediately even if they are currently pinned. Finally, Feature 5 ensures that low clearance transactions cannot "guess" that they are being given a pseudo-empty slot by virtue of the fact that the slot they receive is not the first in the list of slots that they perceive to be empty.

5.1.3 Real-Time Features

SABRE incorporates the following features to enhance its *real-time* performance:

1. Transactions of a particular clearance level cannot replace the Pinned or Active pages of higher priority transactions belonging to the same level.
2. Within each clearance level, priority-based pin preemption is supported.
3. An optimized "comb" slot selection algorithm to decide the slot in which to host a new data page.

Feature 1 helps to utilize the intra-transaction and inter-transaction locality of high priority transactions, while Feature 2 ensures the absence of priority inversion. The optimized slot selection algorithm mentioned in Feature 3 is described below in Section 5.1.4.

5.1.4 Search and Slot Selection

When a transaction requests a data page, the *visible* portion of the buffer pool (corresponding to the transaction's clearance level) is searched for the page and if the search is successful, SABRE returns the address of the slot in which the page is present. If the page is already pinned in a conflicting mode by a higher priority transaction, then the transaction has to wait for it to be unpinned before accessing the contents. Otherwise, it can access the page immediately after gaining a pin on the page.

A search could be unsuccessful for one of two reasons: (a) Because the page is really not in the buffer pool, or (b) Because it is in the non-visible portion of the buffer pool. In the first case, if really empty buffer slots are available, SABRE brings the page into one of these slots and returns the address of the slot to the requesting transaction. Otherwise, an existing buffer page is chosen for replacement according to the selection algorithm described below. The victim page, after being flushed to disk if dirty, is replaced by the requested page and the associated slot address is returned to the transaction. In the second case, the requested page is made to *appear* to have been brought from disk into the buffer slot where it currently exists by "unveiling" the slot only after waiting for the equivalent disk access time.

The slot selection algorithm used in SABRE is shown in Figure 2a. In this algorithm, starting from really Empty slots, the selection works its way up the security hierarchy looking for Dormant slots and then in a "comb-like" fashion works its way downwards from the top clearance level looking for Active and Pinned slots until it reaches the requester's clearance level. This route is shown pictorially in Figure 2b.

SABRE's comb slot selection algorithm incorporates the security and real-time features described earlier. In addition, it includes the following optimizations designed to improve the real-time performance:

1. The search for replacement buffers is ordered based on slot category – the Dormant slots are considered first and only if that is unsuccessful, are the Active and Pinned slots considered. This ordering is chosen to maximize the utilization of *intra-transaction* locality.
2. Given a set of candidate replacement slots (as determined by the slot selection algorithm), these candidate slots are grouped into clean and dirty subsets – only if the clean group is empty is the dirty group considered. Within each group the classical **LRU** (Least Recently Used) policy [4] is used to choose the replacement slot. This "LRU (clean|dirty)" ordering is based on the observation that (a) it is faster to replace a clean page than a dirty page since no disk writes are incurred, and (b) the cost of writing out a disk page is amortized over a larger number of updates to the page.

5.1.5 Fairness Features

As mentioned in the Introduction, due to the covert-channel-free requirements, high clearance transactions in secure RTDBS are discriminated against in that they usually form a disproportionately large fraction of the killed transactions. To partially address this issue, the following optimizations are incorporated in SABRE:

(a) Proxy Disk Service: The delays prescribed in Security Features 1 and 3 (Section 5.1.2) can be easily implemented by actually retrieving the desired page from disk into the buffer slot where it already currently exists. Obviously, this approach is wasteful. A more useful strategy would be to use the disk time intended for servicing this request to instead serve the pending requests, if any, of higher clearance transactions for the same disk. That is, the high clearance transaction clandestinely acts as a "proxy" for the low clearance transaction.

a: Slot Selection Algorithm

```
if (Really Empty Slots exist) then
     slot = randomly chosen Empty Slot
else if (Dormant Slots exist) then
     /* Slot Stealing from Lower Security Levels OR
        Slot Confiscating from Higher Security Levels */
     find the lowest security level at which Dormant Slots exist
     slot = LRU (clean|dirty) among these Dormant Slots
else if (Slots of Higher Security Levels exist) then
     /* Slot Confiscating from Higher Security Levels */
     find the highest level above requester's level for which slots exist
     if (Active Slots exist) then
          find the lowest priority transaction with Active Slots
          slot = LRU (clean|dirty) among these slots
     else /* All Slots are Pinned */
          abort lowest priority transaction and release its slots
          slot = LRU (clean|dirty) among these slots
else if (Slots of Requester's Level exist) then
     if (Lower Priority Transactions with Active Slots exist) then
          find the Active Slots of lowest priority transaction
          slot = LRU (clean|dirty) among these slots
     else if (Lower Priority Transactions with Pinned Slots exist) then
          abort the lowest priority transaction and release its slots
          slot = LRU (clean|dirty) among these slots
     else /* All Slots belong to Higher Priority transactions */
          insert request in the wait queue which is
          maintained in (LEVEL, DEADLINE) priority order
else /* All Slots belong to Lower Security Level transactions */
     insert request in the wait queue which is
     maintained in (LEVEL, DEADLINE) priority order
```

b: COMB Slot Selection Route

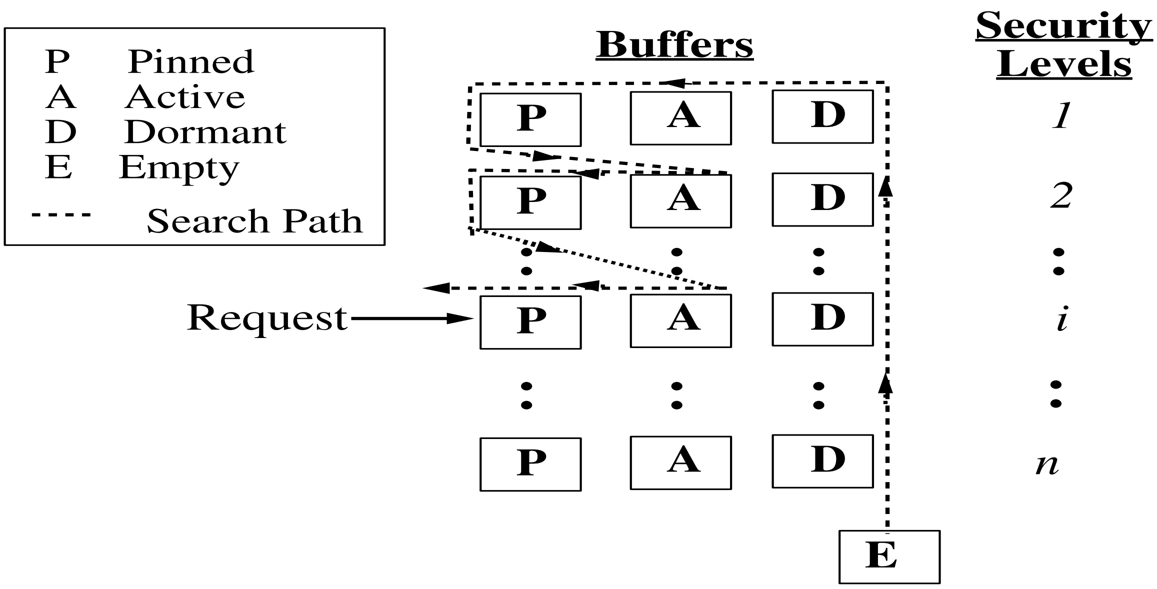

Figure 2: The SABRE Slot Selection Policy

```
Initial condition: Admit[Public] = 1.0, KillPercent[Public] = 0.0,
                                        KillPercent[Secret] = 0.0
LOOP:  if KillPercent[Secret] > 5.0 then
           FairFactor = KillPercent[Public] / KillPercent[Secret]
           if [FairFactor < 0.95] then
                Admit[Public] = Admit[Public] * 0.95
           else if [FairFactor > 1.05] then
                Admit[Public] = Admit[Public] * 1.05
       else
           Admit[Public] = 1.0
       sleep T seconds
       measure KillPercent[Public] and KillPercent[Secret]
       goto statement LOOP
```

Figure 3: The FSABRE Admission Control Algorithm

(b) Comb Route: In the Comb algorithm, the search for a replacement slot among the Dormant slots begins with the *lowest* clearance level. This approach maximizes the possibility of slot-stealing from lower clearance levels, resulting in increased fairness.

5.2 The FSABRE Policy

As discussed above, the SABRE policy has features intended to reduce the unfairness associated with providing security. However, providing complete fairness in a *workload adaptive* manner without incurring covert channels appears to be *fundamentally impossible* since the dynamic fairness-inducing mechanism can *itself become the medium of information compromise*. Therefore, for full-secure applications, unfairness can only be mitigated to an extent, but not eliminated. It would be attractive, however, if for applications that do not insist on full security, complete fairness could be provided with *bounded-bandwidth* covert channels. We present below, **FSABRE**, an adaptive admission-control-augmented version of SABRE, that attempts to provide such fairness while guaranteeing "orange-security" (i.e. leakage bandwidth of less than one bit per second).

For ease of exposition, we will assume that there are only two clearance levels: *Secret* and *Public*. For this environment, the FSABRE admission control algorithm is shown in Figure 3. The basic idea in the algorithm is that based on the imbalance in the transaction kill percentages of the Secret and Public classes, the admission of Public transactions into the system is *periodically* controlled every T seconds (the setting of T is discussed below). The $FairFactor$ variable, which is the ratio of the kill percentages for the Public and Secret classes, captures the degree of unfairness during the last observation period. Ideally the FairFactor should be 1.0 and so, if there is a significant imbalance ($FairFactor < 0.95$ or $FairFactor > 1.05$), what is done is to decrease or increase the admit probability of the Public transactions accordingly. The increase or decrease margin has been set to a nominal 5%. The hope is that this mechanism will eventually result in the multiprogramming level of the Public transactions reaching a value that ensures that the Secret transactions are not unduly harmed. Finally, to ensure that the admission control becomes operative only when the Secret class is experiencing sustained missed deadlines, a 5% threshold Secret kill percentage is included.

5.2.1 Guaranteeing Orange Security

In the FSABRE policy, a corrupt Secret user can signal information to collaborating Public users by modulating the outcomes (increase, decrease, or constant) of the Admit[Public] computation. Corresponding to each computation, $log_2 3 = 1.6$ bits of information can be transmitted[4] and since the computation is made once every T seconds, the total channel bandwidth is $1.6/T$ bits per second. By setting this equal to 1 bit per second, the condition for being orange-secure, we get $T_{min} = 1.6$ seconds to be the minimum recomputation period. In our experimental evaluations of FSABRE's performance, T is set to this value.

5.3 Unsecure Buffer Policies

Since, to our knowledge, SABRE (and FSABRE) represent the *first* secure real-time buffer management policies, we have compared their performance with that of representative *unsecure* buffer management policies – in particular, **CONV** and **RT**, policies for conventional DBMS and for RTDBS, respectively.

For the CONV and RT policies, the *entire* buffer pool is always visible since they are not security-cognizant. In CONV, for a successful search, the requesting transaction can access the page immediately unless it has already been pinned in a conflicting mode – in this case the transaction has to wait for the page to be unpinned before accessing its contents. RT also follows a similar policy for successful searches except that in the case of conflicting pins, if the pin holder is of lower priority than the requesting transaction, the pin is preempted by aborting the lower priority holder.

For unsuccessful searches, CONV and RT follow the slot selection algorithms shown in Figures 4 and 5, respectively. CONV essentially implements the classical LRU approach, including also the (clean|dirty) and category-based search optimizations of SABRE, while RT implements a prioritized version of the same approach.

[4] Based on information-theoretic considerations.

```
if (Empty Slots exist) then
     slot = any Empty Slot
else if (Dormant Slots exist) then
     slot = LRU (clean|dirty) among these Dormant Slots
else if (Active Slots exist) then
     slot = LRU (clean|dirty) among these Active Slots
else  /* all slots are pinned */
     insert request in the wait queue which is
     maintained in FCFS order
```

Figure 4: The CONV Slot Selection Algorithm

```
if (Empty Slots exist) then
     slot = any empty slot
else if (Dormant Slots exist) then
     slot = LRU (clean|dirty) among the Dormant Slots
else if (Lower Priority Transactions with Active Slots exist)
     find the active slots of the lowest priority transaction
     slot = LRU (clean|dirty) among these slots
else if (Lower Priority Transactions exist)
     abort the lowest priority transaction and release its slots
     slot = LRU (clean|dirty) among these slots
else  /* all slots are with higher priority transactions */
     insert request in the wait queue which is
     maintained in DEADLINE priority order
```

Figure 5: The RT Slot Selection Algorithm

5.4 Baseline Policies

To isolate and quantify the effects of buffer management on the system performance, we also evaluate two artificial baseline policies, **ALLHIT** and **ALLMISS**, in the simulations. ALLHIT models an ideal system where every page access results in a buffer pool "hit" , while ALLMISS models the other extreme – a system where every page access results in a buffer pool "miss".

6 Simulation Model

We used a detailed simulation model, similar to that of our previous study [6], of a firm-deadline RTDBS to evaluate the real-time performance of the buffer management policies. Due to space constraints, we highlight only the main features of the model here – the complete details are available in [5]. A summary of the key model parameters is given in Table 1.

In our model, the system consists of a shared-memory multiprocessor DBMS operating on disk-resident data. The database is modeled as a collection of $DBSize$ pages that are uniformly distributed across all the disks. The database is equally partitioned into $ClassLevels$ classification levels. Transactions are generated in a Poisson stream with rate $ArrivalRate$ and each transaction has an associated clearance level and a firm completion deadline. A transaction is equally likely to belong to any of the $ClearLevels$ clearance levels. For simplicity, we assume in this study that the categories (e.g., Secret, Public) for data classification and transaction clearance are identical.

Transaction deadlines are assigned using the formula $D_T = A_T + SlackFactor * E_T$, where D_T, A_T and E_T are the deadline, arrival time and execution time, respectively, of transaction T.[5] The $SlackFactor$ parameter controls the tightness/slackness of deadlines.

A transaction consists of a sequence of page read and page write accesses, generated in accordance with the Bell–LaPadula restrictions. The number of pages accessed by a transaction varies uniformly between 0.5 and 1.5 times the value of $TransSize$. The $WriteProb$ parameter determines the probability that a transaction operation is a write. If a transaction has not completed by its deadline, it is immediately killed (aborted and discarded from the system).

The physical resources consist of $NumCPU$ processors, $NumDisk$ disks, and $NumBuf$ buffers. There is a single common queue for the CPUs and the service discipline is Pre-emptive Resume, with preemptions based on transaction priorities. Each of the disks has its own queue and is scheduled according to a Head-Of-Line (HOL) policy, with the request queue ordered by transaction priority. The $PageCPU$ and $PageDisk$ parameters capture the CPU and disk processing times per data page, respectively. A single level buffer comprising of $NumBuf$ identical page-sized buffer slots is modeled and these slots as well as the associated wait queue are managed according to the specific buffer policy in use. A page that is brought into the buffer pool is pinned for a uniformly distributed duration in the range $[MinPin, MaxPin]$.

[5] The execution time is the time taken to complete the transaction when executed *alone* in the system.

Table 1: Simulation Model Parameters

Parameter	Meaning	Value
$DBSize$	No. of pages in the database	1000
$ClassLevels$	No. of Classification Levels	2
$ArrivalRate$	Transaction arrival rate	—
$ClearLevels$	No. of Clearance Levels	2
$SlackFactor$	Slack Factor in Deadline	4.0
$TransSize$	Average transaction size	16pages
$WriteProb$	Page write probability	0.5
$NumCPU$	No. of processors	10
$NumDisk$	No. of disks	20
$NumBuf$	No. of buffers	50
$PageCPU$	Page processing time at CPU	10ms
$PageDisk$	Page processing time at Disk	20ms
$MinPin$	Minimum buffer pin time	0ms
$MaxPin$	Maximum buffer pin time	100ms
$NumGPS$	No. of GPS (Global Page Sets)	100
$SizeGPS$	Size of each GPS	200
$GRefCnt$	References from current GPS	500
$InterLoc$	Inter-transaction locality Factor	0.02
$IntraLoc$	Intra-transaction locality Factor	0.8
$LocalProb$	Local pageset probability	0.8

6.1 Access Locality

The generation of *inter-transaction* and *intra-transaction* locality is handled in a manner similar to that used in [16]. In this scheme, two types of pagesets are created, *global pagesets* and *local pagesets*. The inter-transaction locality is associated with the global pagesets while the intra-transaction locality is associated with the local pagesets.

Global pagesets are generated by sampling (with replacement) from the database using identical (truncated) normal distributions with variance $1/InterLoc^2$ – only the page location of the center-point of the distribution is a function of the individual global pageset. The $NumGPS$ parameter specifies the number of global pagesets generated and the size of each pageset is given by the $SizeGPS$ parameter. At the outset, one of the global pagesets is designated as the *current* global pageset and subsequent page references are generated from this pageset. After $GRefCnt$ number of references have been generated from this pageset, another pageset is uniformly randomly chosen from the remaining pagesets to become the current pageset.

The pageset for a transaction T is created in the following manner: A local pageset is first created by uniformly randomly choosing pages from the current global pageset and the number of pages chosen is given by the formula $LocalPageSetSize_T = (1 - IntraLoc) * TransSize_T$. After this, the transaction's pageset is created by successively choosing pages randomly from within the local pageset or from within the current global pageset. The probability that a given page access is chosen from the local pageset is specified by the $LocalProb$ parameter.

Finally, *restart locality* is modeled by ensuring that a restarted transaction has the same clearance level and the same sequence of data accesses as its original incarnation.

6.2 Priority Assignment and Concurrency Control

As mentioned earlier, we wish to evaluate the performance of the buffer managers in an environment where *the rest of the RTDBS is completely secure*. Accordingly, the transaction priority assignment used at all the other RTDBS components is P = (LEVEL, DEADLINE). Among the buffer managers, the priority assignment in RT is P = DEADLINE and in SABRE it is P = (LEVEL,DEADLINE), whereas CONV is priority-indifferent.

The secure version [6] of the 2PL–HighPriority real-time protocol [1] is used for concurrency control and, for simplicity, only page-level locking is modeled.

6.3 Performance Metrics

The primary performance metric of our experiments is *KillPercent*, which is the percentage of input transactions that the system is *unable* to complete before their deadlines. We compute this percentage also on a *per-clearance-level* basis. An additional performance metric is *ClassFairness* which captures how evenly the killed transactions are spread across the various clearance levels. This is computed, for each class i, as the ratio $\frac{100-KillPercent_i}{100-KillPercent}$. With this formulation, a protocol is ideally fair if the fairness value is 1.0 for all classes.

7 Experiments and Results

Using the above simulation model, we evaluated the performance of the various buffer managers for a variety of security-classified transaction workloads and system configurations. Due to space constraints, we present results for only two representative experiments here – the others are available in [5]. The settings of the workload and system parameters for these experiments are listed in Table 1. These settings were chosen to ensure significant locality in the reference patterns and significant buffer contention, thus helping to bring out the performance differences between the various buffer management policies.

7.1 Experiment 1: Cost of Complete Security

In this experiment, we evaluate the cost of providing covert-channel-free security in the real-time environment for a system with two security levels: *Secret* and *Public*. For this configuration, Figure 6a shows the overall KillPercent behavior as a function of the (per-second) transaction arrival rate for the SABRE, CONV, RT, ALLHIT and ALLMISS buffer policies. We observe here that, as expected, the ALLHIT and ALLMISS baseline policies exhibit the best and worst performance, respectively. What is more interesting is the *vast gap* between the performance of ALLHIT and that of ALLMISS indicating the extent to which intelligent buffer management can play a role in improving system performance and highlighting the need for well-designed buffer policies.

Among the practical protocols, we observe that the real-time performance of SABRE and RT is similar, with RT

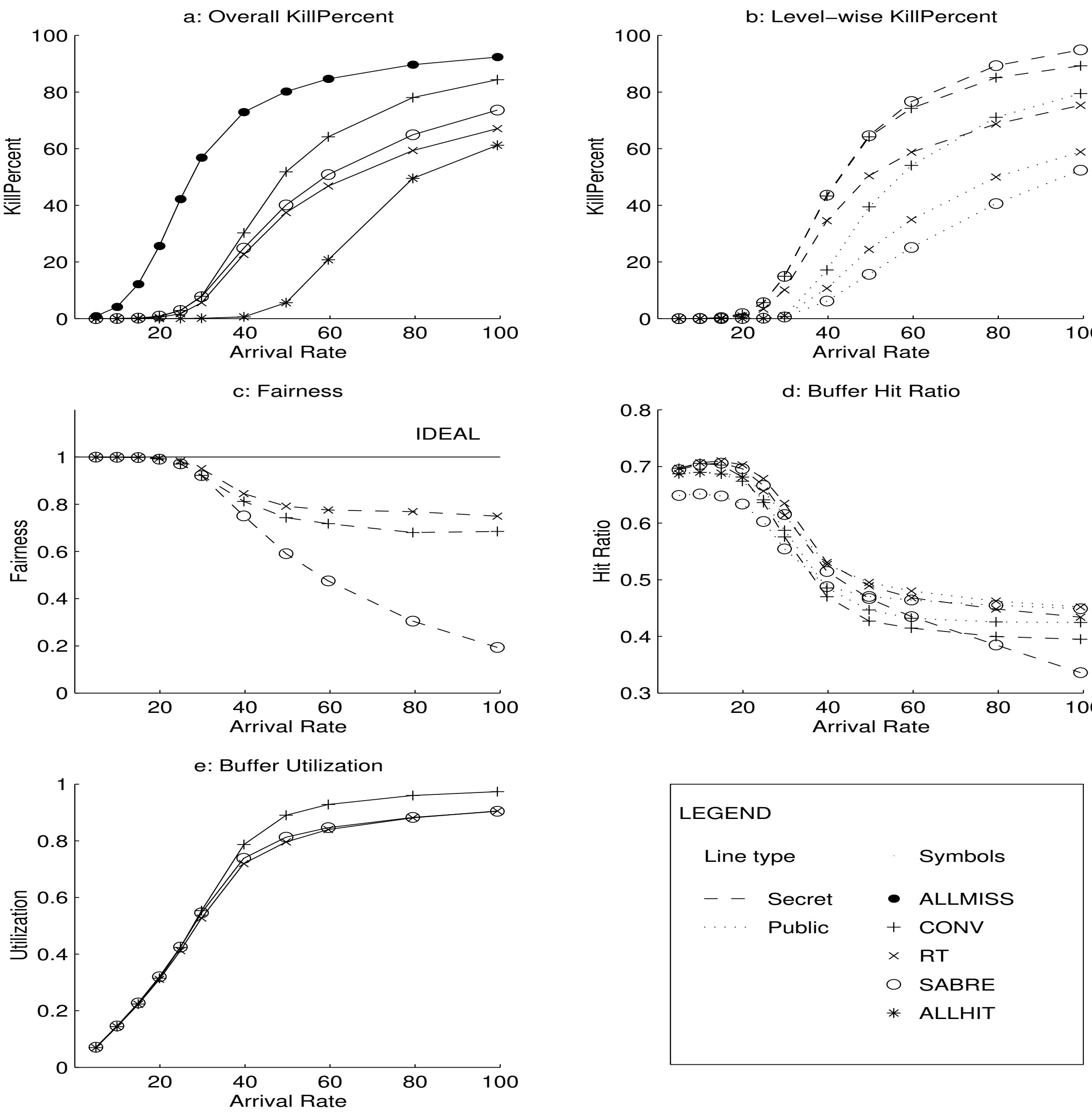

Figure 6: Cost of Complete Security

holding the edge, whereas CONV is significantly worse, especially under heavy loads. RT shows better performance than CONV because it gives preferential treatment to urgent transactions in the following ways: (1) They derive more *intra-transaction* and *restart* locality since their Active pages cannot be replaced by low priority transactions; and (2) They get "first pick" of the available slots due to the prioritized queueing for buffer slots.

The reason that SABRE performs slightly worse than RT is that since priorities are decided primarily based on clearance levels, it is still possible for a slack-deadline Public transaction to restart a tight-deadline Secret transaction, resulting in the Secret transaction missing its deadline. This effect is highlighted in Figure 6b, which shows the *level-wise* kill percentages – with SABRE, Secret transactions (dashed lines) form a much larger proportion of the killed transactions. This is further quantified in Figure 6c, which captures the ClassFairness metric, and clearly demonstrates that SABRE is extremely unfair to Secret transactions, especially under heavy loads.

It may appear surprising in Figures 6b and 6c that RT and CONV, although not security-cognizant, are still somewhat unfair to Secret transactions. This behavior is not due to the buffer policies themselves, but is a *side-effect* of the rest of the system being secure and therefore discriminating against Secret transactions, resulting in more Public page requests being received at the buffer pool.

In Figure 6d, we show the *buffer hit ratio*, that is the average probability of finding a requested page in the visible portion of the buffer pool, for the various protocols on a level-wise basis. The first point to note is that in all the protocols there is a *crossover* between the Secret and Public hit ratios – at low loads Secret is better whereas at high loads Public is better. This is explained as follows: In CONV and in RT, the low load hit ratios should have been similar since they are not security cognizant, but again there is a side-effect arising out of the rest of the system being secure – more Secret transactions are restarted for *concurrency control* reasons than Public transactions and therefore there is more *restart locality* derived for Secret transactions. Under heavy loads, however, this effect is overtaken by the Public transactions hogging most of the buffer slots and thereby making the Secret transactions hot spot mostly absent from the buffer pool.

In SABRE, the Secret transactions do better at low loads, not only because of the restart locality mentioned above, but also due to two additional factors: (1) the whole buffer pool is visible to Secret transactions; and (2) the "slot stealing" feature. Under heavy loads, however, *even more* of the buffer slots are occupied by Public transactions as compared to CONV and RT since Secret transactions cannot replace the Active pages of Public transactions, and therefore the Secret hit ratio is markedly worse. Another interesting observation with respect to SABRE is that although its design prevented the use of much of the Dormant page locality (as described in Section 5), yet its hit ratio is not materially worse than that of CONV or RT. This is because, except under very light loads, most of the pages in the buffer pool will be Active since they represent the hot spot and therefore *Active page locality predominates*.

Finally, Figure 6e shows the *buffer utilization*, that is, the overall average number of Pinned buffer slots, and it is clear here that CONV utilizes the buffers more than RT and SABRE. However, this does not result in better real-time performance because CONV is deadline-indifferent whereas RT and SABRE selectively give the Pinned slots to tight-deadline transactions.

In summary, the results of this experiment highlight the benefits of using deadline-cognizant buffer management policies. Further, it shows that SABRE provides security with only a modest drop in real-time performance. The non-interference requirement, however, causes SABRE to be rather unfair to high clearance transactions, especially under heavy loads.

7.2 Experiment 2: Class Fairness

In this experiment, we evaluate whether an orange-secure version of the FSABRE protocol could address SABRE's fairness shortcoming. The results of this experiment are shown in Figures 7a–c, which capture the overall KillPercent, the level-wise KillPercent, and the Class Fairness, respectively. In these figures we see that FSABRE achieves *close to ideal fairness*. Further, at low loads, FSABRE causes only *a small increase of the overall kill percentage* with respect to SABRE, whereas at heavy loads, FSABRE *actually does slightly better*. At low loads, due to the inherent lag involved in the feedback process, FSABRE tends to be over-conservative, unnecessarily preventing Public transactions from entering, and thereby increases the Public kill percentage. In contrast, under heavy loads, being conservative does less harm than being too liberal (since the system is essentially in a "thrashing" region), and therefore FSABRE's performance shows some improvement over that of SABRE.

In summary, the results of this experiment show that FSABRE achieves close to ideal fairness without really affecting the overall real-time performance. That is, the FSABRE policy *evenly redistributes the "pain" but does not really increase its magnitude*.

7.3 Other Experiments

Our other experiments, described in [5], explored the sensitivity of the above results to various workload and system parameters including the number of security levels, the buffer pool size, the locality factors, etc. The relative performance behaviors of the policies in these other experiments remained qualitatively similar to those seen here, but it should be noted that for applications with large number of security levels, the priority assignment in SABRE becomes much more level-based than deadline-based, resulting in an increased degradation of its real-time performance, as should be expected.

8 Conclusions

In this paper, we have quantitatively investigated, for the first time, the performance implications of the choice of buffer manager in both full-secure and orange-secure firm-deadline RTDBS. This is a followup to our earlier work on secure real-time concurrency control [6].

Making real-time buffer managers secure is complicated due to the multiplicity of covert-channel mediums and buffer components, and due to the inherent difficulties of simultaneously achieving the goals of full security, minimal KillPercent and complete ClassFairness. Our new SABRE policy addresses these challenges by (1) making buffer pool visibility a function of the clearance level, (2) not preallocating buffer slots to security levels, (3) sacrificing Dormant page locality to permit unrestricted slot stealing by higher clearance transactions, (4) supporting pin-preemption, (5) implementing a novel Comb slot selection policy, and (6) incorporating optimizations such as Proxy Disk Service.

Using a detailed simulation model of a firm-deadline RTDBS, the real-time performance of SABRE was evaluated against the CONV and RT unsecure conventional and real-time buffer managers. Our experiments showed that (a) it is essential to include deadline-cognizance in the buffer manager for good real-time performance, thereby supporting the conclusions of [11], (b) SABRE efficiently provides security in that its real-time performance is not significantly worse than that of RT, especially for applications that have only a small number of security levels, and (c) SABRE's sacrifice of Dormant page locality has little impact since Active page locality is predominant.

SABRE's main drawback of bias against higher clearance transactions was addressed by the FSABRE policy,

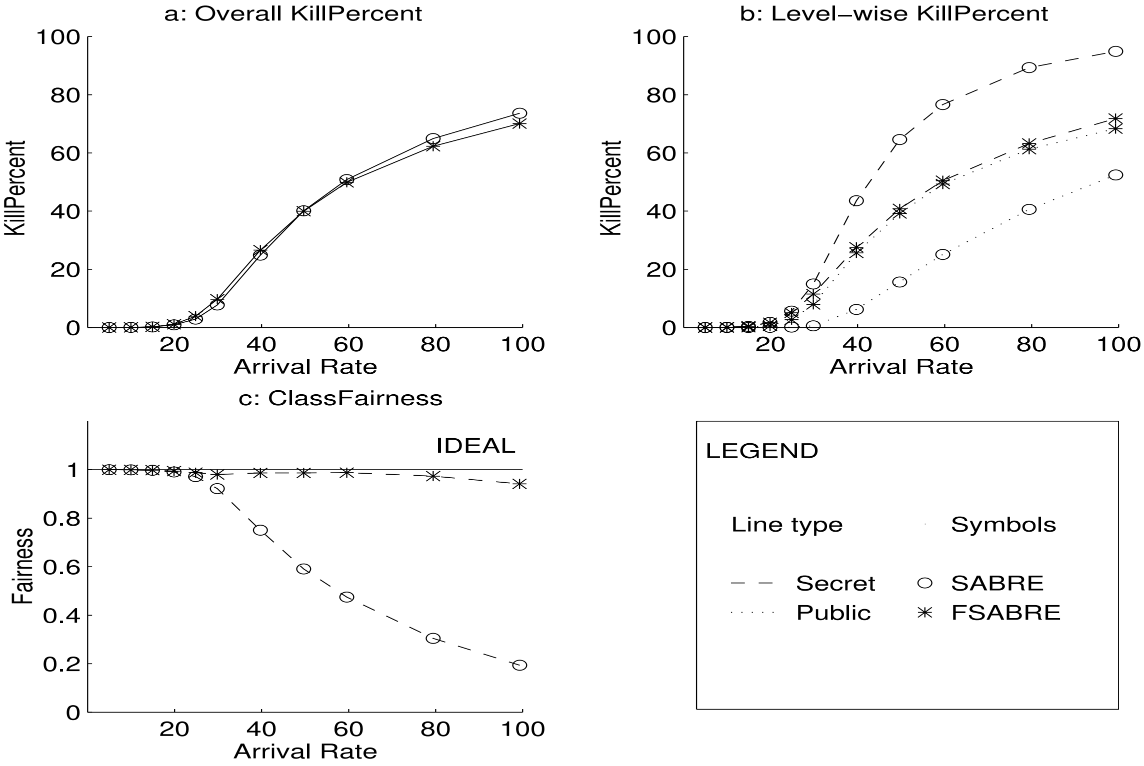

Figure 7: Class Fairness

which implemented a simple feedback-based transaction admission control mechanism. The bandwidth of the covert channel introduced by this mechanism was bounded by appropriately setting the feedback period. An orange-secure version (bandwidth less than 1 bit per second) of FSABRE was found to provide close to ideal fairness at little cost to the overall real-time performance.

References

[1] R. Abbott and H. Garcia-Molina, "Scheduling Real-time Transactions: A Performance Evaluation", *ACM Trans. on Database Systems*, September 1992.

[2] H. Chou and D. DeWitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems", *Proc. of 11th VLDB Conf.*, August 1985.

[3] Department of Defense Computer Security Center, "Department of Defense Trusted Computer Systems Evaluation Criteria", *DOD 5200.28-STD*, December 1985.

[4] W. Effelsberg and T. Haerder, "Principles of Database Buffer Management", *ACM Trans. on Database Systems*, December 1984.

[5] B. George, "Secure Real-Time Transaction Processing", *Forthcoming Ph.D. Thesis*, SERC, Indian Institute of Science, 1998.

[6] B. George and J. Haritsa, "Secure Processing in Real-Time Database Systems", *Proc. of ACM SIGMOD Conf.*, May 1997.

[7] J. Goguen and J. Meseguer, "Security Policy and Security Models", *Proc. of IEEE Symp. on Security and Privacy*, 1982.

[8] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan-Kaufmann, 1993.

[9] J. Haritsa, M. Carey and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems", *Real-Time Systems Journal*, 4(3), 1992.

[10] J. Huang and J. Stankovic, "Buffer Management in Real-Time Databases", *Tech. Rep. COINS 90-65*, Univ. of Massachusetts, July 1990.

[11] W. Kim and J. Srivastava, "Buffer Management and Disk Scheduling for Real-Time Relational Database Systems", *Proc. of 3rd COMAD Conf.*, December 1991.

[12] W. Lampson, "A Note on the Confinement Problem", *Comm. of ACM*, October 1973.

[13] L. LaPadula and D. Bell, "Secure computer systems: Unified Exposition and Multics Interpretation", *The Mitre Corp.*, March 1976.

[14] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, January 1973.

[15] L. Sha, R. Rajkumar and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", *Tech. Rep. CMU-CS-87-181*, Carnegie Mellon University, 1987.

[16] A. Warner, Q. Li, T. Keefe and S. Pal, "The Impact of Multilevel Security on Database Buffer Management", *Proc. of European Symp. on Research in Computer Security*, 1996.