Secure Real Time Database Systems

Binto George Jayant Haritsa
SERC, Indian Institute of Science, Bangalore 560012, India
{binto, haritsa}@serc.iisc.ernet.in

1 Introduction

A **Real-Time Database System (RTDBS)** is a transaction processing system that is designed to handle workloads where transactions have completion deadlines. The objective of the system is to meet these deadlines, that is, to process transactions before their deadlines expire. Research on real-time database systems has focused mainly on identifying the appropriate choice, with respect to meeting the real-time goals, for the various database system policies such as priority assignment, concurrency control, memory management, etc. [9]. However, the issue of providing *security* in real-time database systems has not yet been considered. We feel that this is a major shortcoming since real-time database applications arise primarily in safety-critical installations and military systems, where enforcing security is crucial to the success of the enterprise. In this paper we investigate the performance implications, with respect to meeting transaction deadlines, of providing security in a real-time database system.

Database Security

Most secure database systems have access control mechanisms based on the Bell–LaPadula model [8]. 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. For example, when a process accesses a data file for input/output operations, the process is the subject and the data-file is the object. Each object in the system has a *classification* level (e.g. Top Secret, Secret, Classified, 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:

- A subject is allowed read access to an object only if the former's clearance is greater than the latter's classification.
- 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.

The Bell–LaPadula conditions prevent direct unauthorized access to secure data, but cannot protect from "covert channels". A covert channel is an *indirect* means by which a high security clearance process can transfer information to a low security clearance process [7]. For example, if a low security process requests access to an exclusive resource, it will be delayed if the resource is already possessed by a high security process, otherwise it will immediately be granted the resource. The presence or absence of the delay can be used to encode information by a high security process that is conspiring to pass on information to the low security process.

One method for preventing covert channels is to ensure that low security transactions do not "see" high security transactions – this is done by providing higher priority to low security transactions whenever a resource conflict occurs between a low security transaction and a high security transaction.

Real-Time Database Security

In a secure real-time database system, there are two goals, namely, that of *simultaneously* providing security and minimizing the number of missed deadlines. Unfortunately, the mechanisms for achieving the individual goals often work at cross-purposes: In a real-time database system, high priority is usually given to transactions with earlier deadlines in order to help their timely completion. On the other hand, in secure database systems, low security transactions are given high priority in order to avoid covert channels (as described above). Now consider the situation wherein a high security process submits a transaction with a tight deadline in a secure real-time database system. 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. However, as security is a *correctness* criterion, whereas the number of missed deadlines is a *performance* criterion, the database system is forced to assign priorities based primarily on security clearance levels and only secondarily on deadlines. This means that high security transactions will have more difficulty in meeting their deadlines as compared to low security transactions.

Our Work

In our research work, we have started investigating the problem of how to minimize, as far as possible, the number of missed transaction deadlines *without* compromising security. As a first step towards achieving this goal, we have conducted a detailed discrete event simulation study to evaluate the performance impact, in terms of missed transaction deadlines, of providing complete security (i.e., no covert channels). Our simulation model captures a real-time database system with an open transaction arrival process. Transactions are assigned security levels and have corresponding restrictions on their data accesses. Each transaction also has a deadline and the deadline is "firm" – that is, transactions which miss their deadlines are considered to be worthless and are immediately discarded from the system without being executed to completion [5]. Transaction priorities for resource access are based on a combination of their security levels and their deadlines.

In a database system, conflicts can arise for both *physical* resources (CPU, disk, etc.) and *logical* resources (data). For physical resources, simple priority-based scheduling schemes such as

Preemptive-Resume and Head-Of-Line are available and are used in our study. For data access, however, conflicts have to be resolved in a manner that maintains database consistency. The data access policies used in database systems are commonly referred as **concurrency control** techniques. Concurrency control techniques preserve the database integrity by ensuring that transactions operate in a *serializable* schedule [4]. In recent years, several concurrency control mechanisms that are specially tailored for real-time database systems have been developed. In our study, we use the well-known 2PL High Priority (2PL-HP) algorithm [1], which modifies the classical two-phase locking (2PL) protocol [4] by incorporating a high priority conflict resolution scheme for ensuring that the high priority transactions are not delayed by low priority transactions. In 2PL-HP, when a transaction requests a lock on a data item that is held by one or more higher priority transactions in a conflicting lock mode, the requesting transaction waits for the item to be released. On the other hand, if the data item is held by only lower priority transactions in a conflicting lock mode, the lower priority transactions are *restarted* and the requesting transaction is granted the desired lock. Note that if priorities are assigned uniquely (as is usually the case in real-time database systems), 2PL-HP is inherently deadlock-free.

Related Work

Although there has been considerable work in recent years on real-time database systems, especially with regard to transaction concurrency control, comparatively little attention has been paid to security issues in real-time database systems. The only work that we are aware in this area is by David and Son [2, 3], in which they propose a concurrency control protocol that attempts to *balance* the dual requirements of security and timeliness. Their scheme is a "combination protocol" in that transactions dynamically choose, with equal probability, between a deadline-cognizant 2PL-HP concurrency controller and a secure version of 2PL (developed by them) that does not take deadlines into account. Due to this combination, full security is not provided by the database system. In contrast, in our work, we have assumed that full security is a fundamental requirement and that it is not permissible to improve the timeliness performance at the cost of security.

2 Simulation Model

As mentioned in the Introduction, we have conducted a performance study of the impact of providing security on the number of missed transaction deadlines. To this end, we developed a detailed simulation model of a real-time database system (similar to that described in [5]), which has six components: a *Database* that models the data and its layout; a *Source* that generates the transaction workload; a *Transaction Manager* that models the execution of transactions; a *Resource Manager* that models the hardware resources; a *Concurrency Control (CC) Manager* that controls access to shared data; and a *Sink* that gathers statistics on transactions exiting the system.

In our model, the system consists of a shared-memory multiprocessor DBMS operating on disk-resident data (for simplicity, we assume that all data is accessed from disk and buffer pool

Parameter	*Meaning*
ArrivalRate	Transaction arrival rate
ClearLevels	Number of Clearance Levels
SlackFactor	Slack Factor in Deadline formula
TransSize	Average transaction size (in pages)
WriteProb	Write probability

Table 1: Workload Parameters

considerations are therefore ignored). The database itself is modeled as a collection of pages that are uniformly distributed across all of the disks. Transactions are generated in a Poisson stream and each transaction has an associated security clearance level and a completion deadline. A transaction consists of a sequence of page read and page write accesses. A read access involves a concurrency control request to get access permission, followed by a disk I/O to read the page, followed by a period of CPU usage for processing the page. Write requests are handled similarly except for their disk I/O – their disk activity is deferred until the transaction has committed. Here we assume that the RTDBS has sufficient buffer space to allow the retention of updates until commit time, and we also assume the use of a log-based recovery scheme where only log pages are forced to disk prior to commit. A transaction that is restarted due to a data conflict follows the same data access pattern as its original incarnation. If a transaction has not completed by its deadline, it is immediately aborted and discarded. The following two subsections describe the workload generation process and the hardware resource configuration.

Workload Model

The workload model characterizes transactions in terms of their security clearance levels, their deadlines and their data accesses. Table 1 summarizes the key parameters of the workload model.

The *ArrivalRate* parameter specifies the mean rate of transaction arrivals. A transaction is equally likely to belong to any of the $ClearLevels$ security clearance levels. Deadlines are assigned using the formula $D_T = A_T + SF * R_T$, where D_T, A_T and R_T are the deadline, arrival time and resource time, respectively, of transaction T, while SF is a slack factor. The resource time is the total service time at the resources that the transaction requires for its data processing. The *SlackFactor* parameter is a constant that provides control over the tightness/slackness of transaction deadlines.

The number of pages accessed by a transaction varies uniformly between half and one-and-a-half times the value of $TransSize$. Due to security reasons, each transaction can only access data from a specific part of the database, and page requests are generated by randomly sampling (without replacement) from the database over this range, denoted $RangeOfAccess$. The $RangeOfAccess$ is determined by both the security clearance level of the transaction and the operation (read or write) which the transaction wants to perform on the data page. The database is equally partitioned

Parameter	*Meaning*
DatabaseSize	Number of pages in the database
NumCPUs	Number of processors
NumDisks	Number of disks
PageCPU	CPU time for processing a data page
PageDisk	Disk service time for a page
ClassLevels	Number of classification levels in the system

Table 2: System Parameters

into $ClassLevels$ security classification levels (for example, if the database has 1000 pages and the number of classifications is 5, pages 1 through 200 belong to level 1, 201 through 400 belong to level 2, and so on). Transaction access to data pages is given strictly based on Bell–LaPadula specifications. That is, a transaction cannot read pages classified above its clearance level. Also, a transaction is not allowed to write to pages that have a security classification below its own clearance level. The $WriteProb$ parameter determines the probability that a transaction operation is a write.

For transaction priority assignment, the following rule is used: if the security level of a transaction T_i is lesser than that of transaction T_j, T_i is given priority over T_j. If T_i and T_j belong to the same security class, their relative priority is according to Earliest Deadline – the transaction with the earlier deadline is given higher priority. This method of priority assignment ensures that there are no covert channels since the low security transaction is delayed only by transactions of its own class or those of lower security classes.

System Model

The physical resources of the database system consist of multiple CPUs and multiple disks. There is a single common queue for the CPUs and the service discipline is Pre-emptive Resume, with preemption being 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 being ordered by transaction priority.

Table 2 shows the resource parameters of the system. The $DatabaseSize$ parameter gives the number of pages in the database. The $NumCPUs$ and $NumDisks$ parameters specify the hardware resource composition, while the $PageCPU$ and $PageDisk$ parameters capture CPU and disk processing times per data page. We assume that preemption costs at the CPU are negligible compared to page processing times. The $ClassLevels$ parameter specifies the number of classification levels in the system.

Workload Parameter	*Value*	*System Parameter*	*Value*
TransSize	16 pages	DatabaseSize	1000 pages
WriteProb	0.50	NumCpus	10
ClearLevels	2	NumDisks	20
SlackFactor	4.0	PageCPU	10 ms
		PageDisk	20ms
		ClassLevels	2

Table 3: Parameter Settings

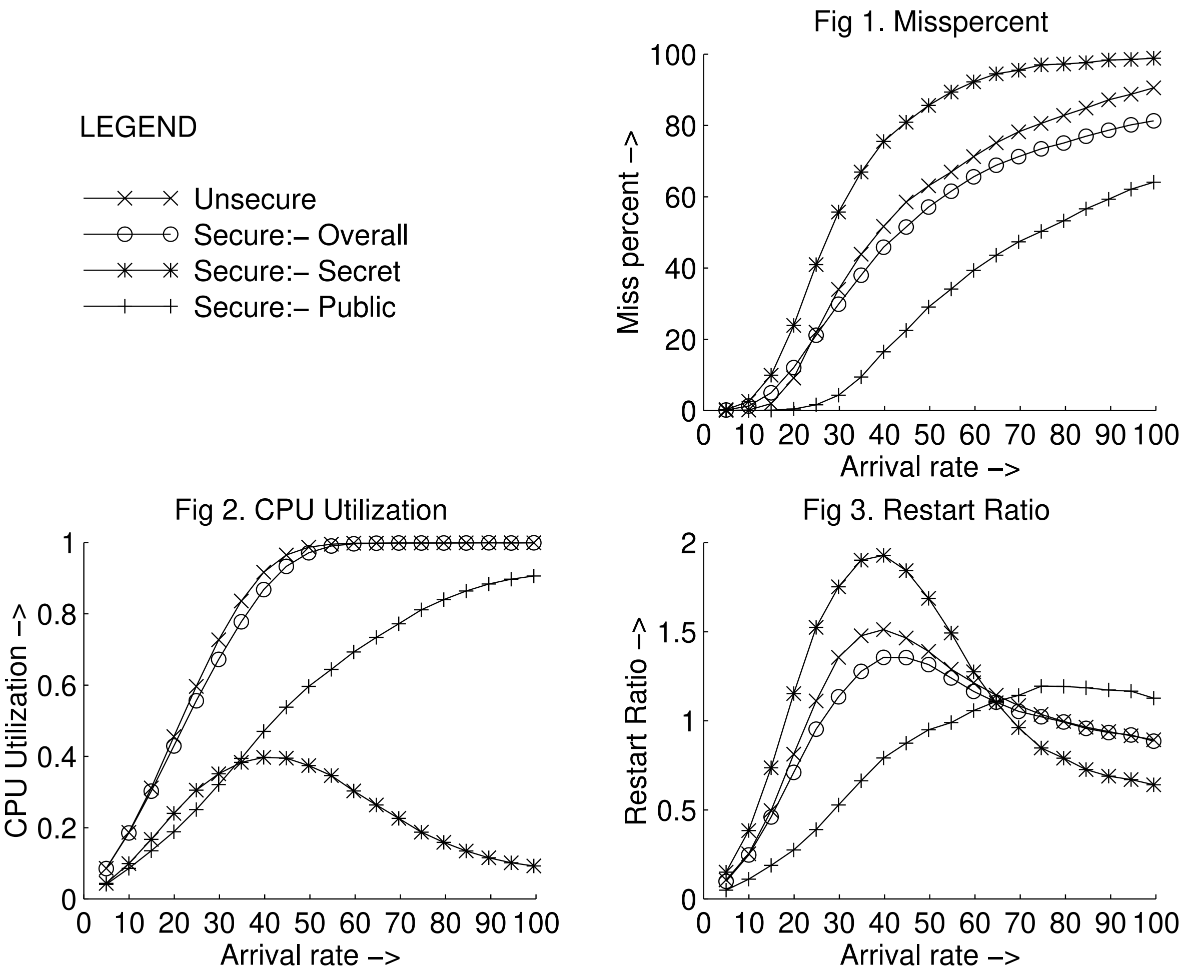
3 Experiments and Results

In this section, we present the performance results from our simulation experiments. The simulator used to obtain the results was written in Simscript II.5. In our experiments, the performance metric is $MissPercent$, which is the percentage of transactions that do not complete before their deadlines. We compute this percentage also on a per-security-class basis. To serve as a basis for comparison, the performance achievable in the *absence* of security requirements is also shown in the performance graphs.

Experiment 1: Finite Resources

In our first experiment, the workload and system parameter settings were set such that there was both significant data contention and resource contention in the system, thus helping to bring out the performance differences. The parameter settings used for this experiment are shown in Table 3. There are two classification levels: *Secret* and *Public*. Correspondingly, there are two transaction clearance levels. For this experiment, Figure 1 shows the MissPercent behavior as a function of the overall transaction arrival rate. In this figure, we first see that the overall miss percent of the secure system is slightly higher than that of the unsecure system at light loads. However, under heavy loads, the situation reverses and we observe that the secure system has a smaller miss percent than the unsecure system. In Figure 1, we also observe that the miss percent of the Secret transaction class is significantly higher than that of the Public transaction class, highlighting the price that has to be paid (in terms of timeliness of the high security class) for ensuring that there are no covert channels.

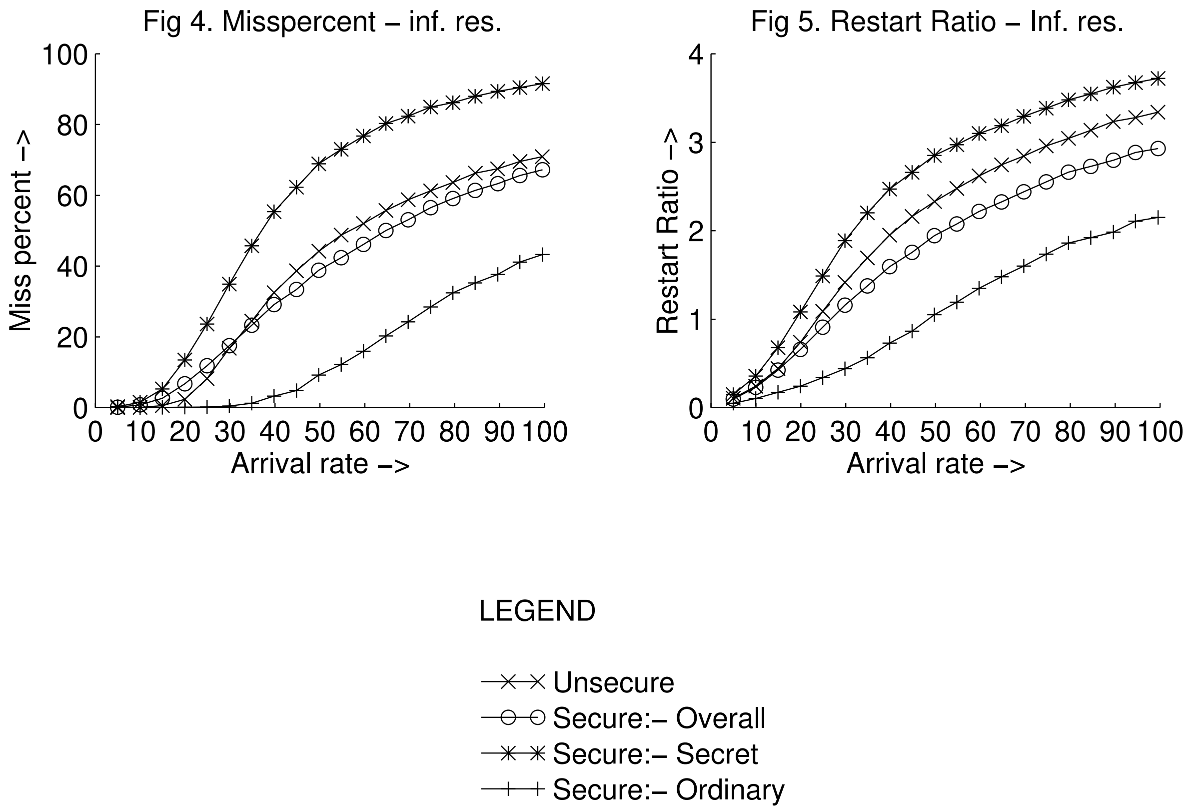
For this experiment, Figures 2 and 3 show the utilization behavior of the processors and the "restart ratio" (the average number of times a transaction is restarted) behavior. They clearly show how the Secret transaction class suffers with respect to the Public transaction class. At high transaction arrival rates, the Secret class is allowed virtually no access to the resources. Therefore, the number of restarts of this class, which was initially increasing, goes down since secret transactions are given little opportunity to access the data items.



Experiment 2: Infinite Resources

The goal of our next experiment was to isolate the impact of data contention on the performance of the system. For this experiment, therefore, the physical resources were made "infinite" (that is, there is no queueing for resources), keeping all other parameter values the same as those of Experiment 1.

The MissPercent performance results for this system configuration are shown in Figure 4. The corresponding restart ratio behavior is shown in Figure 5. Here again we observe that the overall miss percentage of the secure system is fairly close to that of the unsecure system throughout the loading range. In addition, the performance of the Secret class is again significantly worse than that of the Public class. Note that the average number of restarts (Figure 5) for high security transactions is always more than that for low security transactions, even at high arrival rates, due to data contention being the sole performance determining factor.



4 Conclusions

In this paper, we have made a preliminary study of the performance cost introduced by maintaining covert-channel-free security in a real-time database system. Using a detailed simulation model of an RTDBS, we studied the performance of 2-level (Secret and Public) secure system against that of an equivalent unsecure system. Our experiments showed that the overall miss percent of the two systems is comparable through most of the loading range (unsecure is slightly better at low loads whereas secure is slightly better at high loads). Within the secure system, however, the performance of the Secret class is significantly worse than that of the Public class. In our future work, we plan to investigate policies by which the performance of the Secret class can be improved without compromising security.

References

[1] Abbott, R., and Garcia-Molina, H., *Scheduling Real-time Transactions: A Performance Evaluation*, Proc. of 14th Intl. Conf. on Very Large Data Bases , August 1988.

[2] David, R., Son, S., and Mukkamala, R., *Supporting Security Requirements in Multilevel Real-Time Databases*, Proc. of IEEE Symp. on Security and Privacy, May 1995.

[3] David, R., and Son, S. H., *A Secure Two Phase Locking Protocol*, Proc. of the 12th Symp. on Reliable Distributed Systems, Princeton, NJ, Oct 1993.

[4] Eswaran, K., et al, *The Notions of Consistency and Predicate Locks in a Database Systems*, Communications of ACM, November 1976.

[5] Haritsa, J., Carey, M. J., and Livny, M. *Earliest Deadline Scheduling for Real-Time Database Systems*, Proc. of 1991 IEEE Real-Time Systems Symp., December 1991.

[6] Keefe, T. F., Tsai W. T. *Multiversion Concurrency Control for Multilevel Secure Database Systems*, Proc. of the 11th IEEE Symp. on Security and Privacy, Oakland, California, pp 369-383, April 1990

[7] Lampson, W. *A Note on the Confinement Problem*, Communications of the ACM, Vol. 16, No. 10, pp 613-615, October 1973.

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

[9] *Special Issue on Real-Time Database Systems*. Journal of Real-Time Systems, September 1992.

[10] Jajodia, S. and Alturi, V. *Alternative Correctness criteria for Concurrent execution of Transactions in Multilevel secure databases*, Proc. of the IEEE Symp. on Security and Privacy, Oakland, California, pp 369-383, April 1990.

[11] Jajodia, S., Bertina E. and Atluri, V. *Achieving Stricter Correctness Requirements in Multilevel Secure Databases*, Proc. of the 14th IEEE Symp. on Research in Security and Privacy, Oakland, CA May 1992.