Maintaining integrity constraints and security in real-time database systems

Q. N. Ahmed and S. V. Vrbsky
Department of Computer Science, The University of Alabama
101 Houser Hall, Box 870290, Tuscaloosa, AL 35487-0290,
U.S.A, Phone 205-348-6363, Fax 205-348-0219,
E-mail {qahmed, vrbsky@cs.ua.edu}

Abstract

Many real-time database systems are contained in environments that exhibit restricted access to information, such as government agencies, hospitals and military institutions, where mandatory access control for security is required. In addition to such security constraints, real-time database systems have real-time integrity constraints. These real-time constraints require deadlines to be met and data to be temporally consistent. Conventional multi-level secure database models are inadequate for time-critical applications and conventional real-time database models do not support security constraints. We propose a new concurrency control algorithm for secure real-time databases. We implement the algorithm and study the performance using a real-time database system simulation model. Results show that the algorithm performs fairly well in terms of security and timeliness compared to the non-secure algorithm. We argue and show that achieving more security does not necessarily mean a great deal of sacrifice in maintaining real-time constraints.

Keywords

Multi-level secure databases, real-time database, real-time constraints

1 INTRODUCTION

Real-time database systems (Ozsoyglu 1995, Ramamritham 1993, Wolfe 1997) have real-time integrity constraints in addition to the integrity constraints found in conventional databases. Specifically, real-time database systems have timing constraints and temporal consistency constraints. Some examples of real-time databases are avionics, radar tracking, managing automated factories, robot navigation, program stock trading, and military command and control. The timing constraints are typically in the form of deadlines which require a transaction to be completed by a specified time. Failure to meet such a deadline causes the results to lose their value, and in some cases a result produced too late may have a negative value. The temporal consistency constraints require data to be up-to-date as well as data that is close in time. Much of the data in a real-time database is only valid during a specified interval. Failure to meet these temporal consistency constraints compromises the integrity of the real-time database.

Many real-time database systems are contained in environments that exhibit hierarchical propagation of information. Such real-time database systems may require restricted access by users to the data. Mandatory access control is used to ensure the security of data in hierarchical environments, and is typically implemented by multilevel secure (MLS) databases (Bell 1974, Denning 1988, Jajodia 1991). However, major efforts to design secure MLS databases have not considered databases with the real-time constraints of deadlines and temporal consistency. A secure real-time system has to simultaneously satisfy two goals: ensure that the real-time constraints are satisfied and provide security. These two goals can conflict with each other and to achieve one goal is to sacrifice the other. The objective of our work is to study the factors involved in security control of real-time databases, develop suitable concurrency control algorithms, and using a real-time database simulation, study the effect on real-time integrity of maintaining security in real-time databases.

All MLS models (Ramamritham 1993, Jajodia 1991) are based on the classification of the system elements, where classifications are expressed by security levels. Data objects have security levels and users have clearance levels. A user can read a certain object only if the subject's clearance level dominates the object's security level. According to the Bell-LaPadula properties (George 1997) for MLS databases, a subject cannot read an object of a higher or incomparable security level than the subject and all writes must take place at the subject's security level or higher. However, the concurrent execution of transactions results in contention for data objects. As a result, it is possible to have an indirect flow of information from objects at higher levels to subjects at lower levels due to a covert channel (Moskowitz 1994, Qian 1994). For example, if the results from a lower security level transaction are delayed when there is a higher level security transaction, then the lower security level user can determine there are transactions at higher levels, and may even be able to receive information from the length of the delay.

Enforcing database security can compromise the real-time integrity by causing deadlines to be missed and data to become temporally inconsistent. For example, suppose there is a transaction with an earlier deadline at a high security level and a transaction with a later deadline at a low security level, and there is a data conflict between them. If the low security level transaction gets the data and blocks the high security transaction, then although security is maintained, the real-time constraints may be violated. The high security transaction has an earlier deadline, and due to its blocking may miss its deadline. If the reverse is allowed to happen to maintain the real-time constraints, then security is violated as a covert channel is introduced.

Whether to maintain real-time constraints or security is dependent upon the system. If the system requires that security be maintained regardless of the real-time constraints, then conflict must be resolved in favor of security. On the other hand, if the system requires the real-time constraints be maintained, then security must be sacrificed in favor of the real-time constraints. If the system allows a compromise between security and priority, then the goal is to maintain as much security as possible without violating the real-time constraints significantly. In this paper we present a new concurrency control algorithm based on 2-phase locking (2PL). The algorithm recognizes the constraints of real-time transactions as well as security. The algorithm can be used for systems where security can be compromised for real-time constraints and vice versa, and also for systems where security is a correctness criteria and must be maintained.

The rest of the paper is organized as follows. In section 2, we discuss related work in secure real-time concurrency control and the secure real-time factor. In section 3, we present the secure concurrency control algorithm and metrics for security maintenance, and in section 4 we describe the simulation model and the results. In section 5 we present our conclusions.

2 SECURE REAL-TIME CONCURRENCY CONTROL

The few works which address the security of real-time databases are described in (George 1997, David 1995, Son 1995). In (David 1995) a concurrency control strategy is presented which trades off security for improved timeliness if the system does not provide the desired deadline miss percentage. They use a measure called "capacity" to adjust the covert channel to get better real-time performance. In (Son 1995) a secure two-phase locking algorithm is used to allow partial violations of security for improved timeliness. Decisions are made concerning the trade off between security and meeting deadlines by comparing two measures to resolve conflicts: the security factor and the deadline miss factor. This comparison is used to determine if a lower level transaction should be aborted or proceed when it conflicts with a higher level transaction. If the security properties are more important, then any conflict is resolved in favor of the lower level transaction, otherwise if meeting deadlines is more important, then the higher priority transaction is given precedence. Our work also considers the need to tradeoff between security and timeliness but shows that security can be achieved with negligible sacrifice in maintaining real-time constraints.

In (George 1997), security of firm real-time databases is addressed. In this study, security is viewed as a correctness criterion, and the number of missed deadlines as a performance issue. They do not trade off between missed deadlines and security, but instead propose to minimize the number of missed deadlines without compromising security through the choice of concurrency control strategy. They examine the performance of such strategies as a two-phase locking priority scheme, a prioritized optimistic concurrency control algorithm and a new approach, called the dual approach. The dual approach utilizes different concurrency control strategies depending on whether transactions are at the same security level or at different security levels. In contrast, our algorithm is capable of handling transactions both at the same and different security levels.

2.1 Secure real-time factor

The concurrency control algorithm for a secure real-time database system must use security levels of transactions as well as their deadlines to resolve data conflicts. Not only that, the difference in security levels of transactions also needs to be considered when a security conflict is to be resolved. If one transaction is at the highest security level and the another one is at the lowest and a covert channel is introduced between them, then the severity of the covert channel will be higher than the one where transactions are at adjacent security levels. None of the previous works discussed in the previous section recognizes the difference in security levels as a measure to resolve a security conflict. In secure real-time database systems, where satisfying real-time constraints is one of the goals to be achieved, one cannot afford to sacrifice it for some covert channel not severe enough to be concerned about. Thus, we believe that difference in security level is an important issue for determining whether a covert channel has to be closed by possibly sacrificing real-time constraints. To be able to determine the severity or consequences of security violations, we introduce the following "covert channel property" for secure real time database systems.

Covert Channel Property: The greater the difference of security levels between two transactions at data conflict, the greater the severity or the consequence if a covert channel is introduced.

The covert channel property indicates that consequence is proportional to the difference in security or access levels. In other words, the greater the difference in access levels, the more important it is to maintain security and close the covert channel. If two conflicting transactions are at two extreme security levels, the consequence of opening a covert channel is the maximum. If the two conflicting transactions are at two adjacent security levels the consequence is the minimum. Of course, if the two transactions are at the same security level, there is no covert channel, and hence no consequence. We introduce a metric to measure the consequence of introducing a covert channel, to be known as the *Covert Channel Factor* (CCF). The CCF is obtained by normalizing the difference of access levels between the two conflicting transactions:

$$\text{CCF} = \frac{\text{Difference in access levels}}{\text{Maximum difference possible}}$$

$$= \frac{\text{Difference in access levels}}{\#\text{ of access levels} - 1}$$

The maximum value of CCF is 1 when the two transactions are at two terminal access levels. The minimum value of CCF is 1/(# of access levels −1) when the two transactions are at two consecutive security levels. Of course, if two transactions are at the same security level, CCF is obviously zero meaning no covert channel.

2.2 Security tolerance

The CCF gives a measure of security violation. The greater the CCF, the greater the difference in access levels in a security conflict and hence, according to the covert channel property, the greater the severity of security violation. Depending upon the system requirements, security violation may or may not be tolerated. The security tolerance is defined as the maximum security violation a system permits. Since a security violation is measured in terms of CCF, so is the security tolerance. A security tolerance of 0 means the system does not allow any covert channel and a security tolerance of 1 means the system allows all possible covert channels. In other words, the security tolerance is the value of a CCF that corresponds to the upper limit of security violation in a system. For example, assume a system only permits the covert channels between two consecutive access levels. In this case, any covert channel having the difference in access levels greater than 1 is not allowed. The security tolerance in this system will be the value of the CCF corresponding to the covert channel with one access level difference, i.e,

$$\text{Security tolerance} = \frac{\text{Difference in access levels}}{\text{Maximum difference possible}}$$

$$= \frac{1}{\#\text{ of access levels} - 1}$$

The smaller the value of *tolerance*, the more important is the security and vice versa. In the next section we will see how we can use security tolerance to represent the importance of security. In a security conflict, if the CCF is greater than the tolerance, then a conflict is resolved in favor of security, otherwise the conflict is resolved in favor of priority based on the real-time constraints.

3 SECURE CONCURRENCY CONTROL ALGORITHMS

As mentioned earlier, secure real-time database systems have to satisfy the security constraints in addition to the real-time constraints. Security can be thought of as a correctness criteria where security must be enforced. It can also be thought of as a compromising criteria with real-time constraints where security can be sacrificed to maintain more real-time constraints. The algorithm we present here supports both types of security. Before describing our algorithm, we give a brief introduction to the existing non-secure algorithms upon which our algorithm is based. This algorithm is the 2PLHP (Abbot 1992) and our algorithm is to be known as the Secure 2PLHP algorithm. They are described in the following sub-sections.

3.1 2PLHP

The 2PL High Priority (2PLHP) algorithm is a modification of the strict two-phase locking algorithm (2PL) (Abbot 1992) and incorporates the priority of transactions. The priority of a transaction is based on its real-time constraints. The earlier its deadline, the higher its priority. When a transaction requests a lock on a data item that is held by one or more higher priority transactions, the requesting transaction waits for the data item to be released. If the data item is locked by only lower priority transactions, the lower priority transactions are aborted and restarted with the same deadline, and the lock is granted to the requesting transaction. If priority is unique, 2PLHP is deadlock free.

3.2 Secure 2PLHP

The 2PLHP algorithm does not recognize security. To incorporate security we examined all the scenarios involving deadline and access-levels between lock-holding and lock-requesting transactions. For secure real-time database systems five types of conflict can occur. We now describe the strategy taken by our algorithm for each conflict. Assume T1 is the lock-requesting and T2 is the lock-holding transaction.

1. *Deadline(T1) >Deadline(T2) and Access level(T1)>Access level(T2)*: In this case the requesting transaction is at a lower priority and a higher security level. We can abort or block the requester. There will not be any covert channel or priority violation.

   Block T1 // priority and security maintained
2. *Deadline(T1)>Deadline(T2) and Access level(T1)<Access level(T2):* In this case the requester is at a lower security level and a covert channel will be introduced if it is blocked or aborted. However, if the requester is allowed to proceed and the lock-holder aborted, priority will be violated. In this case, we

compute the CCF if T1 is aborted or blocked. If the CCF is greater than the tolerance then the lock-holder (T2) is aborted, otherwise the lock-requester (T1) is blocked.

```
If CCF>tolerance then
        Abort T2 // security maintained
Else
        Block T1 // priority maintained
```

3. *Deadline(T1)<Deadline(T2) and Access level(T1)>Access level(T2):* In this case, the requester is at a higher priority and at a higher access-level and a covert channel will be opened if the lock holder is aborted. Here we need to compute the CCF if T2 is aborted. If the CCF is greater than the tolerance, then T2 is allowed to proceed and T1 aborted, otherwise T1 is granted the lock and T2 is aborted.

```
If CCF>tolerance then
        Abort T1 // security is maintained
Else
        Abort T2 //priority is maintained
```

4. *Deadline(T1)<Deadline(T2) and Access level(T1)<Access level(T2):* In this case the requester is at a lower security and a higher priority, and we can resolve the conflict by aborting T2. Priority is maintained and no covert channel is introduced.

```
Abort T2 // priority and security maintained
```

5. *Access level(T1) = Access level(T2)*: In this case, two transactions are at the same security level and therefore there is no covert channel. The conflict is resolved according to the 2PLHP algorithm.

```
If deadline(T1)<deadline(T2) then
        Abort T2
Else
        Block T1
```

3.3 Choice of tolerance values

The choice of a tolerance value is very important and it provides a way to control priority and security maintained in the system. The smaller the value of tolerance, the more important it is to maintain security and the greater the number of times a conflict is resolved in favor of security. By choosing an appropriate value for tolerance, the system can be maintained 100% covert channel free with every data conflict resolved in favor of the transaction at the lower security level. In order for that to happen, the condition in the *if-then* in cases 2 and 3 in section 3.2 must be true for every value of CCF. In other words, the minimum CCF should be larger than the tolerance value, i.e., the tolerance should be smaller than the minimum CCF. Any tolerance value greater than that will allow some violation of security.

3.4 Metrics of security maintenance

We now introduce two metrics or security factors to measure the security maintenance of a system. One metric keeps track of the number of times security has been maintained. The other one recognizes the differences between the access levels.

$$\text{Security Factor 1} = \frac{\text{\# of times security is maintained}}{\text{Total number of security conflicts}}$$

Security Factor 2

$$= \frac{\text{Sum of the difference in access levels for conflicts having security maintained}}{\text{Sum of the difference in access levels in all security conflicts}}$$

Both the metrics are suitable for measuring the performance of the system. Depending upon the system, one metric might be more appropriate than the other. If only the number of conflicts maintained is of concern the first factor is appropriate. The second factor is appropriate in systems where difference in access-levels is crucial. In this study, we choose the security factor 2.

3.5 Metric of real-time constraint maintenance

When deadlines are missed, the temporal data is not updated in time and data becomes temporally inconsistent. For this study we will use the percentage of deadlines missed as one of the measures of the maintenance of the real-time constraints.

We also use the priority maintenance factor as a second measure. In order to express the level of priority maintenance in a system, we use the following priority maintenance factor.

$$\text{Priority maintenance factor} = \frac{\text{\# of times priority is maintained}}{\text{Total number of data conflicts}}$$

This metric is used to determine how priority maintenance affects the real-time performance.

4 SIMULATION MODEL

This section outlines the structure and details of our simulation model used to evaluate the performance of our concurrency control algorithms for real-time database systems. Central to our simulation model is a single-site main memory database system operating on a single processor. The database is modeled as a collection of data pages in memory. The simulation consists of three main components: a Transaction Manager (TM), a CPU Manager (CM), and a Log Manager (LM). The TM is responsible for issuing lock requests, CM for granting CPU access, and LM for log disk access. The service discipline used for the queues is Earliest Deadline First (EDF) (Liu 1979) without preemption. Each transaction consists of multiple operations each of which can be either read or write. If the operation is read, then the accessed page is not updated. The write operation updates the accessed page and an entry is written into the log buffer.

When an operation of a transaction makes a data access request, i.e., lock request on a data object to the TM, the request goes through concurrency control to obtain a lock on the data object. If the request is granted, the transaction requests CPU access to the CM. If the CPU is free, the request is granted and the transaction does the CPU computation. After the CPU computation, if there are any more operations left, the transaction proceeds with the next operation and makes a lock request to the TM. If all operations are done, the transaction requests log disk access to the LM and if access is granted, it writes the log buffer to the log disk and commits.

If the request for the lock is denied, the transaction will be placed into a block queue. The blocked transaction will be awakened when the requested lock is released.

4.1 Parameters of simulation model

Table 1 gives the system resource parameters. The parameter CPU_TIME is the time to process a page by a CPU. The simulation does not explicitly take into account the time required for accessing the transaction manager, the CPU manager, and the log manager. It is assumed that those times are included in the time required to access the resources, i.e., the CPU, and the log disk.

Table 1 System resource parameters

Parameter	*Explanation*	*Value*
DBSIZE	Number of data pages in the database	400
CPU_TIME	CPU time for processing a data page	5 msec
WRITE_PROB	Probability that an operation is write	0.5
MAXACCESS	Number of security access levels	6

Table 2 summarizes the workload parameters that characterize the transactions and the system workload. Transactions' inter-arrival rates are exponentially distributed. The *Rate* parameter specifies the mean rate of transaction arrivals. The *TransSize* determines the mean number of operations in the transactions determined from a normal distribution with mean of *TransSize*. The actual data objects or pages accessed by each operation are uniformly distributed across the whole database.

The *LogDelay* is the time required for writing a log buffer to the log disk. The *RestartDelay* is the overhead for roll back when a transaction is aborted. The parameters *MinSlack* and *MaxSlack* are used to set the lower and upper bound of transactions' deadlines. The following deadline assignment formula (Abbot 1992) is used to assign the deadline to the arrived transaction.

*Deadline = Arrival time + Uniform(MinSlack,MaxSlack)*Execution time*

The *Arrival time* is the time of arrival of each transaction. The *Execution time* of a transaction is calculated from the data requirements in all the operations using *TranSize*, *CPU_TIME*, and *LogDelay*.

Table 2 Workload parameters

Parameters	*Meaning*	*Value*
Rate	Arrival rate of the transactions	[5,50]
TransSize	Average transaction size	6
LogDelay	Overhead for log disk access	1 unit of CPU_TIME
RestartDelay	Overhead for restarting	1 unit of CPU_TIME
MinSlack	Minimum slack factor	2
MaxSlack	Maximum slack factor	8

4.2 Experimental setup

The simulation program is written in C++ using the next event simulation strategy and is run for 5000 transactions. Different random seeds are used for different calls to the random number generator to make sure that the arrived transactions are exactly the same for different algorithms. We simulate a firm real-time system in which a value returned after a deadline is useless. Hence, at the beginning of each event, the system is checked to see if there is any transaction that has missed its deadline and if so, it is removed from the system. We perform a detailed simulation study using our proposed algorithm and compare it with an existing non-secure one. We discuss the effectiveness of our algorithms in terms of maintaining real-time constraints and security.

Simulation results

Figure 1 illustrates the deadline miss percentage for the non-secure 2PLHP and the Secure 2PLHP with a tolerance of 0 as the arrival rate increases. Non-secure 2PLHP is priority cognizant and hence has a better performance over the secure 2PLHP algorithm. The non-secure algorithm does not have any deadlines missed with arrival rates below 20. The secure algorithm transactions start to miss deadlines around an arrival rate of 16. The main difference between the performance of the two algorithms is prominent between arrival rates 15 and 25, after which a majority of the transactions start to miss their deadlines in both the algorithms.

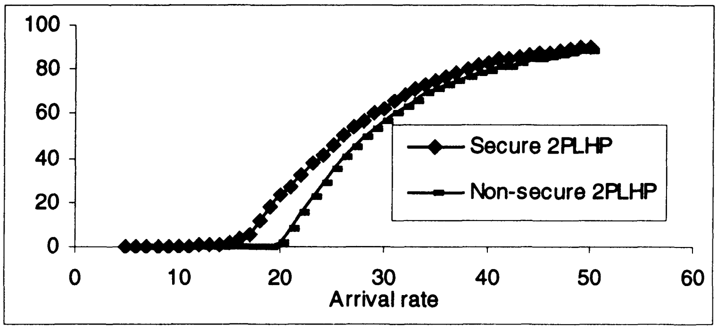

Figure 1 Deadline miss percentage.

The Secure 2PLHP algorithm recognizes covert channels and, therefore, the security factor of a system is improved when the algorithm is used. This result is illustrated in the Figure 2, which compares the security factors for the secure and

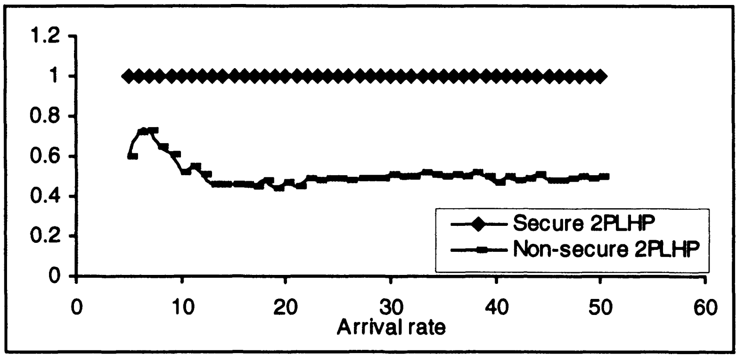

Figure 2 Security factors.

non-secure algorithm. In the secure algorithm, the security factor is 1; in contrast, in the non-secure algorithm, it is inconsistent and remains around 0.5. It is interesting to notice in Figure 1 that although we enforce security with the Secure 2PLHP, we do not necessarily have to sacrifice the maintenance of the real-time constraints a great deal. This is because there are many data conflicts for which security enforcement does not violate priority based on deadline. Increasing security means more data conflicts resolved in favor of lower security transactions, irrespective of their deadlines. If a lower security transaction has a later deadline, enforcing security means loss of priority. However if it happens to have an earlier deadline, then priority is maintained as well. Therefore, achieving complete security (i.e. security factor 1) does not mean that we fail to meet all real-time constraints. Figure 3 supports this claim. It shows the priority maintenance factor at different arrival rates when the security factor is 1. In this case the priority maintenance factor is not zero but instead varies between 0.2 to 0.6. That is the reason why the real-time performance with the Secure 2PLHP Figure 1 is close to the non-secure 2PLHP. This is a very important feature of our algorithm.

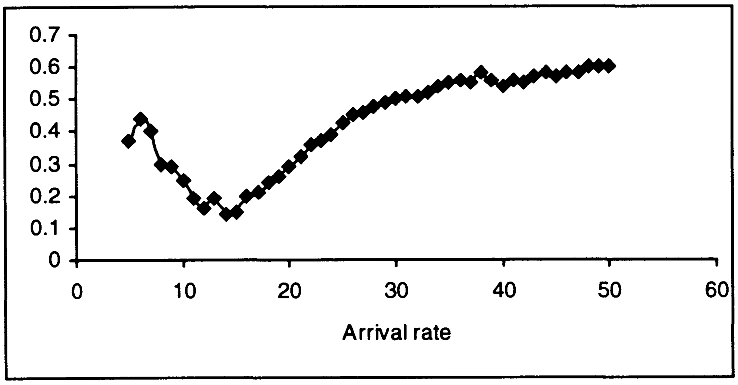

Figure 3 Priority maintenance factor

As the arrival rate increases, the number of data conflicts also increases. When tolerance = 0, data conflicts are always resolved in favor of security, and therefore, the number of data conflicts resolved in favor of priority does not increase at the same rate as the number of total data conflicts. As illustrated in Figure 3, the priority maintenance factor decreases with the increase of arrival rate. However when the arrival rate increases beyond 16-17, the system starts to miss deadlines and transactions are removed from the system as soon as they are late. As a result, the number of data conflict decreases with any subsequent increase in arrival rate. However because of the increased arrival rate, the number of conflicts resolved in favor of priority still increases, which in turn increases the priority maintenance factor. Therefore, once the system starts to miss deadlines, the priority maintenance factor increases.

Figure 4 shows how security *tolerance* affects the security factor and the priority maintenance factor. The number of access levels in our study is 6 and therefore, the minimum possible CCF is 0.2 (section 2.2). As a result, when the tolerance is 0, the value of CCF in any security conflict is higher than the tolerance. In this situation our algorithm resolves a conflict in favor of security. Therefore the security factor stays at 1 until the tolerance increases to larger than 0.2. As

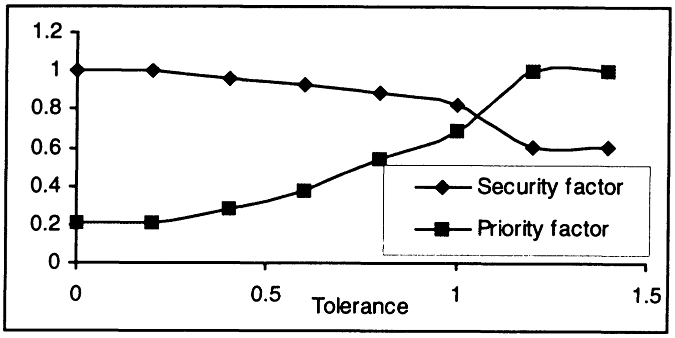

Figure 4 Variation of security and priority factors with tolerance.

indicated earlier in section 3.3, the smaller the tolerance, the higher the number of times conflicts are resolved in favor of security and vice versa. Therefore, the security factor decreases with subsequent increase of tolerance values. As the security factor decreases, more and more conflicts are resolved in favor of priority, and as a result, the priority factor increases. The maximum value of CCF is 1. Therefore when the tolerance is greater than 1, any CCF will be smaller than the tolerance in which case every conflict is resolved in favor of priority. Thus the priority maintenance factor stays at 1 for tolerance greater than 1.

Figure 5 shows a comparison of the restart ratios between the non-secure and the Secure 2PLHP algorithms. If the arrival rate is low, there are fewer conflicts and consequently fewer restarts. However as the transactions start to miss their deadlines, some of the aborted transactions may be already late and hence may not even restart because they are removed from the system. With an increased arrival rate, the number of late aborted transactions increases, which in turn decreases the number of restarts. Thus, the restart ratio increases with the increase in arrival rate until the system starts missing deadlines, after which the restart ratio begins decreasing and it continues to decrease with the subsequent increase in arrival rate. Figure 5 also illustrates that the restart ratio changes with the change of tolerance. This is explained by cases 2 and 3 of the Secure 2PLHP algorithm described in section 3.2 where tolerance is used to resolve the conflict. In case 3, no matter what the value of tolerance is, both the options abort transactions and therefore, do not change the restart ratio. Only in case 2 are there both block and abort options. If the value of tolerance is large, the CCF is more likely to be smaller than the tolerance, in which case the requester will be blocked yielding fewer restarts. If tolerance is very large (>1), number of restarts is usually small compared to low tolerance cases, and therefore the change in restart ratio is also not as sharp as in other cases.

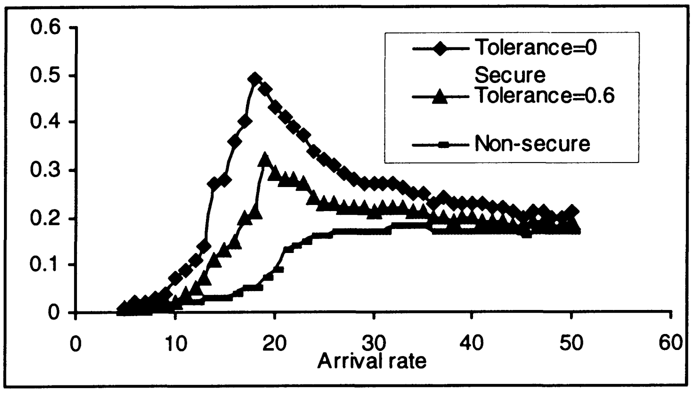

Figure 5 Restart ratio.

5 CONCLUSION

In this paper we proposed a new secure 2PL concurrency algorithm for real-time databases. The algorithm can use security as a correctness criteria where security must be enforced. It can also be thought of as a compromising criteria with real-time constraints where security can be sacrificed to maintain more real-time constraints. We have implemented the secure algorithm and a non-secure algorithm and studied their performance using a firm real-time database system simulation model. We have also introduced metrics to measure security in real-time database systems. Results clearly show that our algorithm performs fairly well in terms of maintaining real-time constraints and security compared to the non-secure algorithm. We also have shown that achieving security does not necessarily mean a great deal of sacrifice in maintaining real-time constraints. A system can be made 100% covert channel free, but can still have a low deadline miss percentage for an arrival rate as high as 20. In the future we will examine new measures for temporal consistency, design suitable concurrency control algorithms and study their performance.

6 REFERENCES

Abbott, R. and Garcia-Molina, H. (1992) Scheduling Real-Time Transactions: A Performance Evaluation, *ACM Transactions on Database Systems*, 17, 513-560.

Bell, D.E. and LaPadula, L.J. (1974) Secure Computer Systems: Mathematical Foundations and Model, in *Technical Report*, MITRE Corporation.

David, R.,Son, S.H., and Mukkamala, R. (1995) "Supporting Security requirements in Multilevel Real-Time Databases," in *Proceedings IEEE Symposium on Security and Privacy*, Oakland, CA, 199-210.

Denning, D.E. (1988) The Sea View Security Model, in *Proceedings IEEE Symposium on Security and Privacy*, Oakland, Ca.

George, B. and Haritsa, J. (1997) Secure Transaction Processing in Firm Real-Time Database Systems, in *Proceedings SIGMOD,* 462-473.

Jajodia, S. and Sandhu, R. (1991) Toward a Multilevel Secure Relational Data Model, in *Proceedings ACM SIGMOD*, Denver, Colorado, May, ACM, New York, 50-59.

Liu, C. L. and Layland, J.W. (1979) Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Journal of the ACM*, 20(1), 46-61.

Moskowitz, I.S. and Miller, A.R. (1994) Simple Timing Channels, in *Proceedings of the IEEE Symposium on Security*, 56-64.

Ozsoyoglu, G. and Snodgrass, R.T. (1995) Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4), 513-532.

Qian, X. (1994), Inference Channel-Free Integrity Constraints in Multilevel Relational Databases, *Proceedings of the IEEE Symposium on Security*, 158-167.

Ramamritham, K. (1993) "Real-Time Databases," *International Journal of Distributed and Parallel Databases*, 199-226.

Son, S.H., David, R. and Thuraisingham, B. (1995) An Adaptive Policy for Improved Timeliness in Secure Database Systems, in *Proceedings of Annual IFIP WG 11.3 Conference of Database Security.*

Wolfe, V.F. and DePippo, L.C. (1997) Real-Time Database Systems, in *Database Systems Handbook* (ed. P. J. Fortier).

7 BIOGRAPHY

Quazi N. Ahmed is a Ph.D. candidate in Computer Science at the University of Alabama. He has an MS degree in Computer Science and Systems Engineering from Muroran Institute of Technology, Japan. His current research interests include real-time database systems, temporal databases and database security.

Susan V. Vrbsky is an Associate Professor of Computer Science at the University of Alabama, Tuscaloosa, AL. She received her Ph.D. in Computer Science in 1993 from the University of Illinois, Urbana-Champaign and an MS. from Southern Illinois University, Carbondale, IL. Her research interests include real-time database systems, database security, approximate query processing and temporal databases.