Chapter 11

SECURE REAL-TIME TRANSACTION PROCESSING

Jayant R. Haritsa

Database Systems Lab, Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore 560012, INDIA.

haritsa@dsl.serc.iisc.ernet.in

Binto George

Dept. of Computer Science
Western Illinois University, Macomb, IL 61455, USA.

b-george@wiu.edu

1. INTRODUCTION

Many RTDBS applications arise in electronic financial services, safety-critical installations and military systems where enforcing **security** is crucial to the success of the enterprise. For example, consider the environment of an electronic (open bid) auction on the World-Wide-Web with online auctioneers and bidders. Typically, the auction database contains "secret" information such as bidders personal details including private keys, credit-worthiness and past bidding patterns; the purchase price and ownership history of the items that are being auctioned; the list of "in-house bidders" – these are bidders planted by the auction house to provoke other bidders by artificially hiking the maximum bid; etc. The database also contains "public" information such as bidder public keys and authentication certificates; the starting bid price, the minimum bid increment and the time for delivery for each item; the sequence and state of bids for items currently under auction; etc. It is expected that the secret information is known only to the auctioneers whereas the public information is available to both bidders and auctioneers.

In the above environment, the auction service provider faces a problem of *three dimensional* complexity: (1) There is a considerable amount of data to be consulted, processed and updated, and while doing so the *database consistency* should be maintained; (2) There are *time constraints* associated with various operations – for example, a bid is valid only if registered in the database within a pre-specified time period after submission (in the Flash Auction at *http://www.firstauction.com*, bids that arrive more than *five minutes* after the previous bid is registered are invalidated); (3) During every stage of the bidding process, *data security* must be ensured — unauthorized access to the secret information by bidders may help them gain unfair and financially lucrative advantages over other competitors.

We investigate here the design of information systems that can simultaneously and effectively meet the above three challenges, that is, on the design of **Secure Real-Time Database Systems (SRTDBS)**. In particular, we focus on the development and evaluation of high-performance secure real-time *concurrency control (CC)* protocols.

A number of requirements have been identified in the literature for a DBMS to be considered secure [6]. We restrict our attention here to **secrecy**, that is, the prevention of unauthorized knowledge of secret data, which is an especially important requirement for RTDBS due to the sensitive nature of their application domains. More precisely, we wish to enforce *multilevel secrecy* wherein, given a set of transaction security classes, the data associated with a particular class should not be revealed to transactions of lower security classes. In the sequel, we use the term security synonymously with multilevel secrecy.

Our study is carried out in the context of real-time applications with "firm-deadlines" [18] – for such applications, completing a transaction after its has expired is of no utility and may even be harmful. Therefore, transactions that miss their deadlines are considered to be worthless and are immediately "killed" – that is, aborted and permanently discarded from the RTDBS without being executed to completion. Accordingly, the performance metric is *KillPercent*, the steady-state percentage of killed transactions.[1]

2. SECURITY MECHANISMS

Most secure DBMS attempt to achieve the secrecy objective by incorporating access control mechanisms based on the well-known Bell–LaPadula model [22]. 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.

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

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.

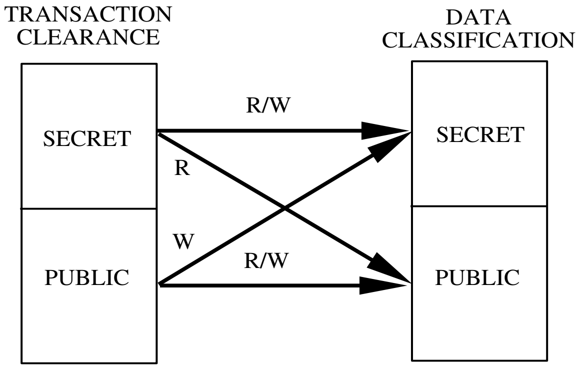

Figure 11.1 Bell-LaPadula access restrictions

The Bell-LaPadula conditions effectively enforce a "read below, write above" constraint on transaction data accesses (an example is shown in Figure 11.1), and thereby prevent *direct* unauthorized access to secure data. They are 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 [21]. 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 that use the DBMS's *physical resources* such as the CPU or the disk as the medium for passing on information can be tackled by introducing "noise" in the form of dummy transactions that make use of these resources (similar to the "pump" scheme proposed in [19]). However, this approach is impractical for covert channels that use *data* as the medium (for example, presence or absence of a lock on a pre-determined data item). This is because, unlike physical resources which are typically few in number, the number of data items is usually enormous, especially in a DBMS. In fact, in heavily loaded systems, noise at the physical resources may be generated "for free", but this will probably never be the case for data since it is trivial to insert an additional data item that is of relevance only to the conspiring transactions. Therefore, *explicitly making data access covert-channel-free is more vital than doing the same for resource access*.

Covert channels based on data can be prevented by ensuring that low security transactions do not "see" high security transactions – this notion is formalized in [16] 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 implementing a **concurrency control (CC)** mechanism that supports the non-interference feature. We address this issue here.

3. INTEGRATING SECURITY AND CONCURRENCY CONTROL

A variety of challenging problems, ranging from priority assignment to ensuring performance fairness, arise when we attempt to integrate security into the RTDBS framework – these issues are discussed in this section.

3.1 PRIORITY ASSIGNMENT

An SRTDBS 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 [17]. 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 an SRTDBS. In this case, priority assignment becomes difficult since assigning a high priority may cause a security violation whereas assigning a low priority may result in a missed deadline.

One solution, predicated on the notion that security is of paramount importance, is to assign transaction priorities based primarily on clearance levels and only secondarily on deadlines. In this scheme, priorities are assigned as a vector $\text{P} = (\text{LEVEL}, \text{INTRA})$, 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 [23], wherein transactions with earlier deadlines have higher priority than transactions with later deadlines. For this choice, the priority vector would be $\text{P} = (\text{LEVEL}, \text{DEADLINE})$ – this priority assignment is used in most of our experimental evaluations.

3.2 SUPPORTING NON-INTERFERENCE

In conjunction with the above priority assignment policy, it would seem at first glance that, in principle, any real-time CC protocol could be used in an SRTDBS and that the actual choice of protocol would be based only on the relative performance of these protocols. However, *not all the previously proposed real-time CC algorithms are amenable to supporting security requirements*. For example, consider the 2PL Wait Promote algorithm proposed in [1]: This protocol, which is based on 2PL, incorporates a *priority inheritance* mechanism [26] wherein, whenever a requester blocks behind a lower-priority lock holder, the lock holder's priority is promoted to that of the requester. In other words, the lock holder *inherits* the priority of the lock requester. The basic idea here is to reduce the blocking time of high priority transactions by increasing the priority of conflicting low priority lock holders (these low priority transactions now execute faster and therefore release their locks earlier).

The Wait Promote approach is *not* suitable for SRTDBS. This is because it permits the blocking of high priority transactions by low priority transactions which violates the requirement of *non-interference* between the transactions of different security levels.

To generalize the above observation, a real-time CC protocol that permits, to even a limited extent, high priority transactions to be adversely affected by low priority transactions, a phenomenon known as *priority inversion* in the real-time literature [26], cannot be used in a secure RTDBS. Apart from Wait

Promote, other examples of real-time CC algorithms that fall into this category include 2PL-CR [1], 2PL-OS/BI [3] and WAIT-50 [18].

3.3 RESOURCE ALLOCATION

Apart from the noise approach mentioned earlier, covert channels can also be eliminated by *static* resource allocation policies, wherein each transaction security class has a set of pre-assigned resources. This approach is taken, for example, in *replicated secure architectures* wherein a multi-level secure DBMS is built by "stacking" single-level database systems. As pointed out in [5], the drawback of such strategies, however, is that they may result in an inefficient and inflexible use of resources.

Yet another possibility is to use "snapshot" protocols (e.g. [4]), wherein two versions of data items are provided – for data of a given security level, higher security transactions access the older version whereas transactions of the same level access the current version. A problem with this approach, however, is that higher security transactions, especially long-lived ones, have to base their results on *arbitrarily stale* data [5]. In fact, a performance study in [8] shows instances where the highest security level transactions read data items that have been subsequently overwritten by *fifteen* other transactions.

The above approaches are *especially* problematic in the real-time context since they may result in: (a) A large increase in the number of killed transactions due to poor resource utilization; or (b) Incorrect results since real-time applications typically respond to current situations and therefore should not utilize stale data. Due to these reasons, the challenge in the RTDBS domain is to design "dynamic" and "one-copy" CC policies that are demonstrably secure.

3.4 FAIRNESS

A major problem arising out of the preferential treatment of low security transactions is that of "fairness" – a disproportionately large fraction of the high security transactions miss their deadlines and are killed. 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 "commonfolk", that is, the low security transactions.

Unfortunately, designing dynamic mechanisms for achieving fairness that are completely free of covert channels appears to be *fundamentally impossible* [20]. This is because the fairness-inducing mechanism can *itself become the medium of information compromise*. The issue then is whether it is possible to design fair systems while still guaranteeing that the information leakage bandwidth (from covert channels) is within acceptable levels.

4. SECURE CC PROTOCOLS

In this section, we describe the CC protocols that are evaluated in our study. These include two well-known real-time protocols: 2PL-HP [1], which is based on locking, and OPT-WAIT [18], which is based on optimistic concurrency control. Both these protocols are completely free from priority inversion and can therefore be used to resolve conflicts in an SRTDBS.

Apart from the above real-time algorithms, we also consider S2PL [27], a recently proposed secure locking-based protocol that does not include any real-time-specific features. We include it here, however, for the following reasons: First, it serves as a baseline against which to compare the real-time CC algorithms. Second, we use it in one of the "dual-CC" protocols described in Section 5.. Finally, it has been used in some of the previous research work on SRTDBS (these are discussed in Section 8.).

In the remainder of this section, we describe the 2PL-HP, OPT-WAIT and S2PL protocols.

4.1 2PL HIGH PRIORITY

The 2PL High Priority (2PL-HP) scheme [1] modifies the classical strict two-phase locking protocol (2PL) [10] by incorporating a priority conflict resolution scheme which ensures that 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 (the wait queue for a data item is managed in priority order). 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.[2] Note that 2PL-HP is inherently deadlock-free if priorities are assigned uniquely (as is usually the case in RTDBS).

[2] A new reader joins a group of lock-holding readers only if its priority is higher than that of *all* the waiting writers.

4.2 OPT-WAIT

The OPT-WAIT algorithm [18] modifies the classical forward (or broadcast) optimistic CC protocol (OPT) [25] by incorporating a *priority wait* mechanism. Here, a transaction that reaches validation and finds higher priority transactions in its conflict set is "put on the shelf", that is, it is made to wait and not allowed to commit immediately. This gives the higher priority transactions a chance to make their deadlines first. While a transaction is waiting on the shelf, it is possible that it may be restarted due to the commit of one of the conflicting higher priority transactions. If at any time during its shelf period, the waiting transaction finds no higher priority transactions remaining in its conflict set, it is committed, restarting in the process the lower priority transactions (if any) in its conflict set.

4.3 S2PL

A secure locking-based protocol called Secure 2PL (S2PL) was recently proposed in [27]. The basic principle behind Secure 2PL is to try to simulate the execution of conventional 2PL *without* blocking the actions of low security transactions by high security clearance transactions. This is accomplished by providing a new lock type called *virtual lock*, which is used by low security transactions that develop conflicts with high security transactions. The actions corresponding to setting of virtual locks are implemented on *private versions* of the data item (similar to optimistic CC). When the conflicting high security transaction commits and releases the data item, the virtual lock of the low security transaction is upgraded to a real lock and the operation is performed on the original data item. To complete this scheme, an additional lock type called *dependent virtual lock* is required apart from maintaining, for each executing transaction T_i, lists of the active transactions that precede or follow T_i in the serialization order. The complete details are given in [27].

In our implementation of S2PL, we have had to make some modifications since the algorithm (as described in [27]) does not eliminate interference under all circumstances – the details of the security loopholes and our fixes for these loopholes are given in [14].

5. THE DUAL-CC APPROACH TO SECURE CONCURRENCY CONTROL

In this section, we move on to discussing our new *dual-CC* approach to secure real-time concurrency control. Our design is based on the observation that in the secure environment there are *two* categories of conflicts: *inter-level* and *intra-level*. Inter-level conflicts are data conflicts between transactions belonging to different security clearance levels whereas intra-level conflicts are data conflicts between transactions of the same level. The important point to note here is that *only inter-level conflicts can result in security violations*, not intra-level conflicts. This opens up the possibility of using *different* CC strategies to resolve the different types of conflicts. In particular, we can think of constructing mechanisms such that inter-level conflicts are resolved in a secure manner while intra-level conflicts are resolved in a timely manner. For example, S2PL could be used for inter-level conflicts while OPT-WAIT could be used to resolve intra-level conflicts.[3]

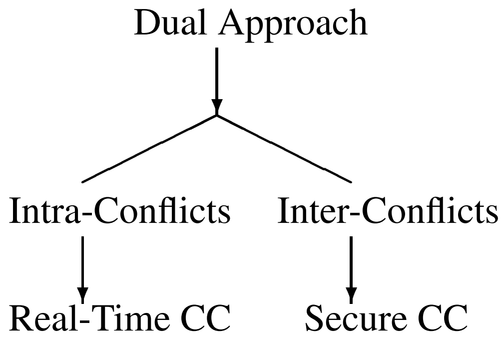

Figure 11.2 Dual approach

The advantage of the dual-CC approach, pictorially shown in Figure 11.2, is that the RTDBS can maximize the real-time performance, by appropriate choice of intra-level CC protocol, without sacrificing security. This is in marked contrast to the *tradeoff* approach suggested in [8] which requires the application to *compromise* on security in order to achieve enhanced real-time performance (a more detailed assessment of the tradeoff approach is presented in Section 8.). Another advantage of the dual-CC approach is that the separation of security and timeliness concerns makes it possible to use even *unsecure* real-time CC algorithms (e.g., Wait-Promote, WAIT-50) for resolving intra-level conflicts! *The dual-CC approach therefore empowers the use, even in the secure RTDBS domain, of the rich set of real-time CC algorithms developed during the last decade.*

[3]A similar, but unrelated, dual-protocol strategy has been used earlier for the purpose of *task scheduling* in the Secure Alpha project [17].

5.1 ENSURING SERIALIZABILITY

At first glance, it may appear that concurrently using multiple CC mechanisms could result in violation of the transaction serializability requirement. This could happen, for example, if the serial orders enforced by the individual mechanisms were to be different. A detailed study of a generalized version of this problem is presented in [31], wherein the transaction workload consists of a mix of transaction classes and the objective is to allow each transaction class to utilize its preferred CC mechanism. They propose a database system architecture wherein intra-class conflicts are handled by the class's preferred CC manager while inter-class conflicts are handled by a new software module called the Master Concurrency Controller (MCC) that interfaces between the transaction manager and the multiple CC managers. The MCC itself implements a complete concurrency control mechanism and ensures a single global serialization order in the entire database system by using a *Global Ordering Scheme* [31].

For our study, we adapt the above architecture to the secure real-time environment. In our implementation, each security level has a Local Concurrency Controller (LCC) that resolves intra-level data conflicts. The LCCs can use *any* high performance real-time CC protocol without risking security violations. The Master Concurrency Control mechanism, however, is chosen to be a secure protocol and is responsible for handling all inter-level data conflicts. Figure 11.3 shows an example of this implementation for a two security level system. The proof of correctness and other details of our adaptation of the MCC architecture are given in [14].

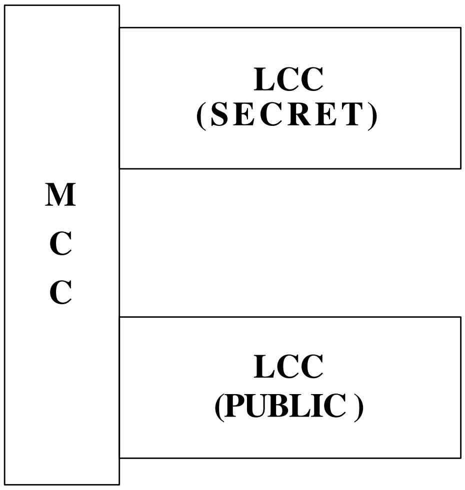

Figure 11.3 MCC architecture

5.2 DUAL-CC PROTOCOLS

We have developed and evaluated three protocols, based on the dual-CC approach: WAIT-HP, HP-WAIT and S2PL-WAIT. These protocols are described below:

WAIT-HP: Inter-level data conflicts are resolved by OPT-WAIT and intra-level conflicts are resolved by 2PL-HP.

HP-WAIT: This protocol is the "mirror" of WAIT-HP – inter-level data conflicts are resolved by 2PL-HP while intra-level conflicts are resolved by OPT-WAIT.

S2PL-WAIT: Inter-level data conflicts are resolved by S2PL and intra-level conflicts are resolved by OPT-WAIT. The scheme operates as follows: After the intra-level conflicts are resolved by OPT-WAIT, the transaction enters the "virtual commit" state, and subsequently the inter-level data conflicts are resolved by S2PL. After all conflicts are resolved, a transaction "really" commits. Recall that S2PL is *not* a restart-oriented algorithm and allows a high security transaction to continue regardless of the virtual commit of a low security conflicting transaction.

6. PERFORMANCE RESULTS

Using a detailed simulation model of a firm-deadline SRTDBS, we evaluated the real-time performance of the various secure CC protocols described in the previous sections. Further, to help isolate and understand the performance cost that occurs due to having to eliminate covert channels, we also simulated the performance achievable for each of these protocols if only "direct" unauthorized access was prevented (by using the Bell–LaPadula conditions) – for this scenario, a priority assignment of (P = DEADLINE) was used.

Our experiments, which covered a variety of transaction workloads and system configurations, demonstrated the following:

1. Under "normal" loads (KillPercent values in the range of 0 to 20 percent), the overall performance of the secure system is worse than that of the direct system, whereas under heavy loads (KillPercent greater than 20 percent), it is the other way around. Within the secure system, the performance of high-security transactions is significantly worse than that of the low-security transactions. Among the secure (individual) CC protocols, OPT-WAIT performed the best in minimizing the kill percentages on both an overall basis and on a per-clearance-level basis. These

results show that OPT-WAIT, which provided excellent performance in traditional (unsecure) real-time concurrency control [18], continues to perform well even in the secure real-time domain.

2. The dual-CC combinations of HP-WAIT and WAIT-HP generally perform worse than pure OPT-WAIT. However, the dual combination of *S2PL-WAIT* performs better than OPT-WAIT, especially at lower kill percentage levels. This is because S2PL is a non-restart-oriented algorithm unlike both OPT-WAIT and 2PL-HP, and therefore ensures reduction of the harm done to high-security transactions.

   The above results highlight the power and flexibility that is provided by the dual-CC approach. In fact, it may be possible to develop dual protocols that perform *even better than S2PL-WAIT* by appropriately choosing the constituent protocols.

3. At light loads when virtually all transactions make their deadlines, all the CC protocols are (trivially) fair. As the loading increases, however, the protocols become increasingly unfair since they selectively miss the deadlines of Secret transactions to accommodate the Public transactions. With regard to the relative fairness of the protocols, OPT-WAIT is the best among the individual CC protocols, whereas S2PL-WAIT beats OPT-WAIT on this metric also.

7. FAIRNESS ACROSS SECURITY LEVELS

As discussed above, our experimental results showed that even with the S2PL-WAIT protocol, a marked lack of fairness exists with respect to the real-time performance of the high security classes, especially in environments with significant resource contention. In this section, we discuss the fairness issue in more detail.

Policies for ensuring fairness can be categorized into static and dynamic categories. Static mechanisms such as resource reservation can provide fairness while still maintaining full security. However, they may considerably degrade the *real-time* performance due to resource wastage arising out of their inability to readjust themselves to varying workload conditions.

Dynamic mechanisms, on the other hand, have the ability to adapt to workload variations. However, as observed in [20], providing 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*. For example, consider the following situation: A conspiring Secret process submits a workload such that the Secret class performance degrades. The fairness mechanism subsequently tries to improve the Secret class performance by allocating more resources for the Secret class. A collaborating Public transaction could now feel the reduction in the availability of system resources and thereby sense the presence of the Secret process in the system. Therefore, this mechanism could be exploited for covert signaling.

In summary, for full-secure applications, unfairness can only be mitigated to an extent by a judicious choice of CC protocol, but not completely eliminated. Therefore, it is interesting to consider the more tractable problem of whether it is possible to dynamically provide fairness while guaranteeing an *upper bound* on the bandwidth of the covert-channel introduced by the fairness mechanism.

We investigated the above issue in [14] and proposed a simple feedback-based transaction admission control mechanism for achieving fairness. The bandwidth of the covert channel introduced by this mechanism was bounded by appropriately setting the feedback period. In particular, the bandwidth bound was set to *one bit per second*, an acceptable level as per the so-called "Orange Book" [7], which defines US military security standards. Experimental evaluation of this system indicates that it *provides close to ideal fairness at little cost to the overall real-time performance*.

8. RELATED WORK

The material discussed in this chapter is based on our earlier papers [11, 12, 14]. Apart from this, the only other work that we are aware of on SRTDBS concurrency control is a series of papers by Son et al [9, 24, 28, 29, 30]. In this section, we briefly summarize these papers and contrast them with our study.

A concurrency control protocol called Adaptive 2PL (A2PL) that attempts to *balance* the dual requirements of security and timeliness was presented in [9, 28, 29]. In their scheme, transactions *dynamically* choose between an unsecure version of 2PL-HP and the secure S2PL described in Section 4.. The goal of the A2PL protocol is to *tradeoff* security for real-time performance with the tradeoff depending on the state of the system and the application's requirements.[4] In contrast, in our work, we have assumed that full security is a *fundamental* requirement and that it is not permissible to improve the real-time performance at the cost of security.

There are also a few design and implementation difficulties associated with the A2PL protocol:

[4]The tradeoff approach, and alternative schemes to implement the tradeoff, have also been considered in the Secure Alpha project [17], which investigated the interactions between security and timeliness in the context of a distributed real-time operating system.

1. In A2PL, the decision to use 2PL-HP or S2PL is a function of the system state and the application requirements. The priority assignments in these protocols are independent and may therefore work at cross purposes. For example, consider the case where a tight-deadline Secret transaction requests a data item held by a slack-deadline Public transaction. Assume that the A2PL protocol chooses the real-time option to resolve this conflict and therefore grants the lock to the requester aborting the Public holder. Assume that another slack-deadline Public transaction now makes a conflicting request on the same data item. If the A2PL protocol decides to follow the secure option to resolve this conflict, the Secret holder will have to be restarted. This means that a transaction that was earlier assisted based on deadline considerations now gets hindered on security considerations, negating the priority assignment objective.

2. Since the A2PL protocol takes only data conflicts into account, it cannot control the conflicts at other resources. Therefore, here again, a transaction that was assisted to make its deadline at the data may be hindered at the physical resources.

3. For applications where full security is a fundamental requirement, the A2PL protocol reduces to plain S2PL, which does not have any real-time features.

In [24], a concurrency control protocol that ensures *both* security and timeliness is proposed. For this scheme, however, the RTDBS is required to maintain *two* copies of each data item. Further, transactions are required to obtain *all* their data locks *before* starting execution (i.e., conservative locking). These requirements limit the applicability of the protocol. In our work, we have considered more general database environments where all data items are single-copy and transactions acquire data locks dynamically.

Another feature of their work is that it is primarily addressed towards "soft-deadline" applications, that is, real-time applications in which there is value to completing tasks even *after* their deadlines have expired. In contrast, we have concentrated on firm-deadline applications. The type of deadline has a significant impact on both the performance evaluation model and on the interpretation of the results, as observed earlier for (unsecure) real-time transaction concurrency control [1, 18].

9. SUMMARY AND FUTURE WORK

In this chapter, we have attempted to characterize the state-of-the-art with regard to secure real-time transaction processing. We first identified that, in order to satisfy the requirement of non-interference, only those real-time concurrency control protocols that are free from priority inversion can be used in a secure RTDBS. This requirement ruled out several previously proposed real-time CC protocols, including algorithms such as 2PL Wait Promote and WAIT-50. Then, we studied the relative performance of the secure versions of the 2PL-HP and OPT-WAIT real-time concurrency control algorithms as well as S2PL, a non-real-time secure algorithm. OPT-WAIT performed best in these experiments across all metrics.

We also proposed a novel dual-CC approach to secure concurrency control wherein different concurrency control algorithms are used to resolve inter-level conflicts and intra-level conflicts. The dual combination of S2PL-WAIT performed better than OPT-WAIT in our experiments. Another advantage of the dual-CC approach is that the separation of security and timeliness concerns makes it possible to use even unsecure real-time CC algorithms for resolving intra-level conflicts.

Finally, we showed that secure RTDBS are inherently biased against high security transactions. This problem can be addressed by implementing a simple feedback-based transaction admission control mechanism, whose information leakage bandwidth can be bounded by appropriately setting the feedback period.

We now outline a variety of issues that appear appropriate for further research: First, we have only considered environments wherein the security levels are *fully-ordered* – in general, however, security hierarchies may be *partially ordered*, typically in the form of lattices. It is therefore imperative to extend the CC protocols discussed here to this more general framework. Further, additional security requirements such as data integrity and availability should also be considered.

Second, the performance characteristics of snapshot-based concurrency control algorithms and the performance tradeoffs involved in replicated architectures need to be assessed.

Third, we considered only one class of covert channels in our study, namely, *timing channels*, wherein information transfer is by modulation of response times. A complementary class of covert channels is known as *storage channels*, wherein information transfer is through modulation of the success or failure of access of storage locations [2]. Solutions for plugging security loopholes based on storage channels need to be devised.

Finally, high-performance secure protocols for other real-time transaction processing components, including *buffer management* and *distributed commit*, need to be designed. A first step in this direction was taken in our recent study of real-time secure buffering [13, 15].

Acknowledgements Binto George worked on this project when he was a graduate student at the Database Systems Lab, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India. Jayant Haritsa's work was supported in part by research grants from the Dept. of Science and Technology and the Dept. of Bio-technology, Govt. of India.

References

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

[2] M. Abrams, S. Jajodia and H. Podell, *Information Security*, IEEE Computer Society Press, 1995.

[3] D. Agrawal, A. El Abbadi and R. Jeffers, "Using Delayed Commitment in Locking Protocols for Real-Time Databases", *Proc. of ACM SIGMOD Conf.*, June 1992.

[4] P. Amman, F. Jaeckle and S. Jajodia, "A Two-Snapshot Algorithm for Concurrency Control in Secure Multilevel Databases", *Proc. of IEEE Symp. on Security and Privacy*, 1992.

[5] V. Atluri et al, "Multilevel Secure Transaction Processing: Status and Prospects", *Database Security, X: Status and Prospects*, Chapman & Hall, 1997, eds. Pierangela Samarati and Ravi Sandhu.

[6] S. Castano, M. Fugini, G. Martella and P. Samarati, *Database Security*, Addison-Wesley, 1995.

[7] "DOD Trusted Computer System Evaluation Criteria", *Department of Defense Standard, DoD 5200.28-STD*, December 1985.

[8] R. David, "Secure Concurrency Control", *Master's Thesis*, Univ. of Virginia, May 1993.

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

[10] K. Eswaran et al, "The Notions of Consistency and Predicate Locks in a Database System", *Comm. of ACM*, November 1976.

[11] B. George and J. Haritsa, "Security Protocols for Real-Time Database Systems", *Proc. of Natl. Conf. on Software for Real-Time Systems*, January 1996.

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

[13] B. George and J. Haritsa, "Secure Buffering in Firm Real-Time Database Systems", *Proc. of Intl. Conf. on Very Large Data Bases*, August 1998.

[14] B. George and J. Haritsa, "Secure Concurrency Control in Firm Real-Time Database Systems", *Journal of Parallel and Distributed Databases*, (Special Issue on Security), 8(1), January 2000.

[15] B. George and J. Haritsa, "Secure Buffering in Firm Real-Time Database Systems", *Intl. Journal on Very Large Data Bases (VLDB)*, 8(3), February 2000.

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

[17] I. Greenberg et al, "The Secure Alpha Study (Final Summary Report)", *Tech. Report ELIN A012*, SRI International, June 1993.

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

[19] M. Kang and I. Moskowitz, "A Pump for Rapid, Reliable, Secure Communication", *Proc. 1st ACM Conf. on Computer and Communications Security*, November 1994.

[20] T. Keefe, W. Tsai and J. Srivastava, "Multilevel Secure Database Concurrency Control", *Proc. of 6th IEEE Intl. Conf. on Data Engineering*, February 1990.

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

[22] L. LaPadula and D. Bell, "Secure Computer Systems: Unified Exposition and Multics Interpretation", *The Mitre Corp.*, March 1976.

[23] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", *Journal of the ACM*, 20(1), 1973.

[24] R. Mukkamala and S. Son, "A Secure Concurrency Control Protocol for Real-Time Databases", *Proc. of Annual IFIP WG 11.3 Conference of Database Security*, August 1995.

[25] J. Robinson, "Design of Concurrency Control Protocols for Transaction Processing Systems", *PhD Thesis*, Computer Sciences Dept., Carnegie Mellon University, 1982.

[26] L. Sha, R. Rajkumar and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *Tech. Rep. CMU-CS-87-181*, Depts. of CS, ECE and Statistics, Carnegie Mellon University, 1987.

[27] S. Son and R. David, "Design and Analysis of a Secure Two-Phase Locking Protocol", *Proc. of Intl. Computer Software and Applications Conf.*, November 1994.

[28] S. Son, R. David and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems", *Proc. of Annual IFIP WG 11.3 Conference of Database Security*, August 1995.

[29] S. Son, R. David and B. Thuraisingham, "Improving Timeliness in Real-Time Secure Database Systems", *SIGMOD Record*, Special Issue on Real-Time Database Systems, March 1996.

[30] S. Son and B. Thuraisingham, "Towards a Multilevel Secure Database Management System for Real-Time Applications", *Proc. of IEEE Workshop on Real-Time Applications*, May 1993.

[31] S. Thomas, S. Seshadri and J. Haritsa, "Integrating Standard Transactions in Real-Time Database Systems", *Information Systems*, 21(1), March 1996.