6

Multilevel Secure Transaction Processing: Status and Prospects

Vijayalakshmi Atluri[1], *Sushil Jajodia*[2], *Thomas F. Keefe*[3], *Catherine McCollum*[4], *and Ravi Mukkamala*[5]
[1]*Rutgers University*
MS/CIS Department, Newark, NJ 07102, USA.
atluri@andromeda.rutgers.edu
[2]*George Mason University*
Center for Secure Information Systems and Department of Information and Software Systems Engineering, Fairfax, VA 22030–4444, USA. jajodia@gmu.edu
[3]*Pennsylvania State University*
Department of Computer Science, University Park, PA 16802, USA. keefe@cse.psu.edu
[4]*The MITRE Corporation*
1820 Dolley Madison Boulevard, McLean, VA 22102-3481, USA.
mccollum@mitre.org
[5]*Old Dominion University*
Department of Computer Science, Norfolk, VA 23529–0162, USA.
mukka@cs.odu.edu

Abstract

Since 1990, transaction processing in multilevel secure database management systems (DBMSs) has been receiving a great deal of attention from the database research community. Transaction processing in these systems requires modification of conventional scheduling algorithms and commit protocols. These modifications are necessary because preserving the usual transaction properties when transactions are executing at different security levels often conflicts with the enforcement of the security policy. Considerable effort has been devoted to the development of efficient, secure algorithms for the major types of secure DBMS architectures: kernelized, replicated, and distributed. An additional problem that arises uniquely in multilevel secure DBMSs

P. Samarati et al. (eds.), *Database Security*

is that of secure, correct execution when data at multiple security levels must be written within one transaction. Significant progress has been made in a number of these areas, and a few of the techniques have been incorporated into commercial trusted DBMS products. However, there are many open problems remain to be explored. This paper reviews the achievements to date in transaction processing for multilevel secure DBMSs. The paper provides an overview of transaction processing needs and solutions in conventional DBMSs as background, explains the constraints introduced by multilevel security, and then describes the results of research in multilevel secure transaction processing. Research results and limitations in concurrency control, multilevel transaction management, and secure commit protocols are summarized. Finally, important new areas are identified for secure transaction processing research.

Keywords

Commit Protocol, Concurrency Control, Database Management, Multilevel Security, Serializability, Transaction Processing

1 INTRODUCTION

This paper is based on a panel discussion on Transaction Processing in Multilevel Trusted Database Management Systems (DBMSs) that was held at the Tenth IFIP WG11.3 Working Conference on Database Security. The objectives of the panel were to reflect on the progress that has been made in the six years since work in this area began, to recognize the achievements to date, and to explore technical challenges that require future attention.

We begin in Section 2 with a brief discussion of the needs that are addressed by transaction processing, illustrated with some simple examples of anomalies that can occur in the absence of transaction controls. We summarize the concurrency control techniques and commit protocols that are generally used in transaction processing to avoid these problems by providing the transaction properties of atomicity, consistency, isolation, and durability. Section 3 describes the constraints on transaction processing that are introduced in a trusted DBMS as a result of its multilevel security policy. It explains why conventional transaction processing techniques can conflict with the multilevel security constraints, discusses how they must be modified to comply with the security policy, and notes the challenges of doing so with acceptable responsiveness and with little or no trusted code.

In Section 4, we review the accomplishments of the last six years. The published solutions adopted by commercial vendors in their trusted DBMS products are examined briefly, that is, the extent to which they have succeeded in meeting the competing needs of multilevel transaction processing and efficiency. The solutions emerging from research to date are then considered. Concurrency control algorithms developed for replicated and kernelized trusted DBMS architectures are presented and assessed. Both locking and multiversioning-based algorithms are discussed, and their respective resolutions of the security-efficiency tradeoffs noted. Most of this work has assumed that while multilevel DBMSs schedule transactions that write at different security levels, each individual transaction writes at only one security level. The notion of having individual transactions be able to write data at multiple security levels leads to an additional tradeoff between security and atomicity. We describe research in multilevel transaction correctness, which selectively relaxes atomicity to allow secure execution. We then turn our attention to distributed

multilevel secure DBMSs and describe the research into the impact of multilevel security on commit protocols for coordinating the execution of distributed transactions.

Section 5 concludes the paper by identifying technical challenges in multilevel transaction processing that have yet to receive significant attention, including algorithms for multilevel security in advanced transaction models, multilevel index management, potential optimizations in recovery logging and buffer management, performance analysis, and data repair.

2 TRANSACTION PROCESSING IN A NON-MLS ENVIRONMENT

2.1 Concurrency Control: The Problem

The primary purpose of a database management system is to carry out transactions. A transaction is a sequence of database operations either to query or manipulate data in a shared database. The goal of a concurrency control mechanism is to preserve database integrity, even in the presence of concurrent data accesses by multiple users, by properly synchronizing simultaneous executions of transactions. This goal is best illustrated by a simple example.

Consider a database that maintains the information of customer accounts in a bank, where the balance in each account is maintained. Suppose a customer, Mr. Smith, owns two accounts: a checking account (account A) and a savings account (account B), the latter account being jointly owned by Mr. Smith and his wife. Suppose the current balances in A and B are \$1,000 and \$10,000, respectively. Consider the following scenario: A transaction T_1, initiated by Mr. Smith, moves \$100 from account A to account B, while another transaction T_2, initiated by Mrs. Smith, deposits \$10,000 into account B. If these two transactions take place at about the same time, as shown in the execution given below, there is the potential for a problem.

T_1	T_2
Read A	
A = A - 100	
	Read B
Write A	
	B = B + 10,000
Read B	
	Write B
	Commit T_2
B = B + 100	
Write B	
Commit T_1	

In this concurrent execution, interference between T_1 and T_2 causes the amount deposited by T_2 to be lost (even though T_2 has successfully completed). Such concurrent executions are incorrect and should not be allowed by the DBMS.

2.2 ACID Requirements

Traditionally, transactions are expected to satisfy *atomicity, consistency, isolation,* and *durability* properties (known as ACID properties). Atomicity requires that either the effects of all operations of a transaction have to be made permanent (when the transaction commits) or the effects must be removed (when the transaction aborts). Consistency requires that each transaction be correct; i.e., if executed alone, the transaction maps the database from one consistent state to another consistent state. Isolation requires that no transaction should view the intermediate results of other transactions. Finally, durability requires that once a transaction commits, all of its effects must be preserved permanently in the database even in the presence of failures.

A proper concurrency control protocol ensures isolation properties of transactions even in the presence of other concurrently running transactions. A proper failure recovery protocol ensures the atomicity and durability properties. In a distributed database, an atomic commit protocol is additionally required to ensure atomicity. Distributed transaction processing requires proper integration of atomic commit and concurrency control protocols.

2.3 Concurrency Control Protocols

Though there are a number of concurrency control algorithms to produce serializable execution of transactions (Bernstein, Hadzilacos & Goodman 1987), the two most widely used protocols are *two-phase locking* (2PL) and *timestamp ordering* (TO).

The protocols based on locking delay the execution of conflicting* operations by setting locks on data items for read and write operations. 2PL requires that all transactions be well-formed and two-phase. By definition, a well-formed transaction must acquire a shared-lock (S-lock) on a data item before reading it and an exclusive lock (X-lock) before writing it. A transaction is two-phase if it does not acquire any more locks once it releases a lock on some data item.

The protocols based on timestamping assign a unique timestamp ($ts(T_i)$) to each transaction (T_i). Each data item (x) is associated with the timestamp of the latest transaction that read or wrote it ($rts(x)$ and $wts(x)$, respectively). A timestamp ordering protocol permits conflicting operations to execute only in increasing order of timestamps as follows: (1) T_i is not allowed to read x unless $wts(x) \leq ts(T_i)$, and (2) T_i is not allowed to write x unless $rts(x) \leq ts(T_i)$.

2.4 Commit Protocols

Often, information is physically distributed over several sites for reasons such as increased availability, survivability, reliability, performance, or convenience. Such a distributed database consists of a collection of independent computational units connected via communication links. Transactions executing in these systems may require to access (either to update or retrieve) information from a number of remote sites. When a distributed transaction is divided (into subtransactions) for execution at individual sites, we must ensure that they either all commit or all abort together. That is, if a subtransaction fails for any reason, we must abort all the subtransactions and roll back the overall distributed transaction.

Consider once again the bank example described above. Assume Mr. Smith's checking account

*Two operations conflict if both refer to the same data item and at least one of them is a write.

(account A) is maintained in San Francisco (site 1) and his savings account (account B) in New York (site 2). Suppose he wishes to move $1,000 from savings to checking. Assume a distributed transaction is initiated at site 1 to perform these operations. The following set of operations must be performed at the two sites:

Site 1	Site 2
	Read B
	B = B − 1,000
	Write B
Read A	
A = A + 1,000	
Write A	

The distributed transaction must commit only after successful execution of these operations at both the sites. Otherwise, it should abort. It is not desirable to execute operations at site 1 alone or at site 2 alone. An atomic commit protocol ensures the atomicity of the distributed transaction. The two-phase commit protocol (2PC) is one of the most popular atomic commit protocols.

Although many distributed systems rely on 2PC for ensuring atomic commit, most commercial systems use several optimizations of 2PC to reduce the number of message exchanges (Stamos & Cristian 1993). In the *presumed commit (abort)*, the acknowledgment is not sent from the participants if the subtransaction commits (aborts). Most commercial systems (e.g., IBM's LU6.2 and Digital's DECdtm) use the *early prepare* (EP) rather than 2PC as the commit protocol because these systems do not support a request/response paradigm. Thus, an implementation of 2PC in these systems requires 6n messages (where n is the number of sites participating in the commit protocol), compared to 4n messages required by EP.

3 TRANSACTION PROCESSING IN AN MLS ENVIRONMENT

3.1 Why Traditional Protocols Do Not Work

The basic model of multilevel security was first introduced by Bell & LaPadula (1975). The Bell-LaPadula (BLP) model is stated in terms of *objects* (that hold data such as a file or a record) and *subjects* (active entities that manipulate objects). Every object is assigned a classification and every subject a clearance. Classifications and clearances are collectively known as security classes (or levels) and are partially ordered.

In the MLS database context, each transaction as well as each data item is assigned a security class. Transactions are subject to the following restrictions: (1) *simple-security property:* a transaction is allowed to read a data item only if the former's security level is identical or higher than the latter's security level, and (2) *restricted $\star$-property:* a transaction is allowed to write a data item only if the former's security level is identical to the latter's security level. (Unlike in the BLP model, transactions are not allowed to write up for integrity reasons.) Although these restrictions prevent direct legal flow of information from a level to another nondominated level (Denning 1982), they are not sufficient to ensure that security is not compromised since it could

high:	$\text{lock}_2[x], r_2[x]$	$\text{lock}_2[y], w_2[y]$
low:	$w_1[x]$ will be delayed	

Figure 1 A signaling channel with two-phase locking protocol

be possible for leakage of information to occur through indirect means via *covert channels*. Covert channels are paths not normally meant for information flow that could nevertheless be used to signal information.

Unfortunately, a covert channel can easily be established with conventional concurrency control algorithms such as 2PL and TO. In both locking and timestamping techniques, whenever there is contention for the same data items by transactions executing at different access classes, a lower level transaction may be either delayed or suspended in order to ensure correct execution. In such a scenario, two colluding transactions executing at high and low levels can establish an information flow channel from a high level to a low level by accessing selected data items according to some agreed-upon code.† The following example illustrates how a covert channel can be created by malicious transactions under 2PL.

Consider a database that stores information of two types: *low* and *high*. Any *low* information is made accessible to all users of the database by the DBMS; on the other hand, *high* information is available only to a select group of users with special privileges. In accordance with the security policy, a transaction execution on behalf of a user with no special privileges would only be able to access (read and write) low data elements, while a high transaction (initiated by a high user) would be given full access to the high data elements and read-only access to the low elements.‡

Imagine a "conspiracy" of two transactions: T_1 and T_2. T_1 is a transaction confined to the *low* domain; T_2 is a transaction initiated by a high user and, therefore, able to read all data elements. Suppose that only these two transactions are currently active. If T_2 requests to read a low data element x, a lock will be placed on x for that purpose. Suppose that next T_1 wants to write x. Since x has been locked by another transaction, T_1 will be forced by the scheduler to wait. T_1 can measure such delays, for example, by going into a busy loop with a counter. Thus, by selectively issuing requests to read low data elements, transaction T_2 could modulate delays experienced by transaction T_1, effectively sending signals to T_1. Since T_2 has full access to high data, by transmitting such signals, it could pass on to T_1 information that the latter is not authorized to see. The information channel thus created is a covert channel and is shown in figure 1.

The TO protocol also suffers from the same secrecy-related flaw. Suppose $ts(T_1) < ts(T_2)$. Since T_1 attempts to write x after T_2 has read, T_1's write operation is rejected because the read timestamp of x $(rts(x)) > ts(T_1)$ and therefore T_1 must be aborted. Since a high transaction can selectively cause a (cooperating) low transaction to abort, a covert channel can be established.

The security properties of other popular concurrency control protocols are considered in (Keefe,

†Often, we use the terms high and low in our discussion to represent two security levels, where high is greater than low in the partial order.

‡It is necessary to disallow a high transaction from writing into low data elements in order to guard against *Trojan horses*. When a high user wants to update a low data item, he or she must do so by first logging in as a low user and then initiating a low transaction (Denning 1982).

Tsai & Srivastava 1993). This paper also provides a general framework for evaluating the security of concurrency control protocols.

Similar problems are encountered even in distributed MLS systems. Especially when lock-based protocols are used for concurrency control, in order to eliminate covert channels, read locks acquired by a high level subtransaction on a low level data item must be relinquished when a low subtransaction attempts to write that low data item. Unfortunately, this requirement has grave implications on the commit protocols, especially the early prepare, since to guarantee serializability, it requires not only that each subtransaction must be two-phase, but that the distributed transaction as a whole must be two-phase as well (Bernstein et al. 1987). To satisfy the latter requirement, all subtransactions must hold all their locks until the commit of the transaction, which unfortunately cannot be met in MLS systems (refer to section on secure commit protocols for an example).

Thus, a multilevel secure DBMS, besides ensuring serializability, must also eliminate all illicit flow of information via covert channels. In order to meet these two requirements, sometimes transactions may be subjected to indefinite delays or suspended again and again (especially high transactions, due to requests from low transactions). This problem is known as starvation. Therefore, the requirement of ensuring serializability while preserving security leads to an additional requirement for multilevel secure DBMSs: they must also avoid starvation. Another desirable feature of a multilevel secure DBMS is to use as little trusted code as possible. Since verification of the code for the trustworthiness is very expensive, it is always desirable to have a scheduler that can be implemented with untrusted code.

In summary, the scheduler that implements the concurrency control protocol in a multilevel secure database management system must possess the following key properties: it must ensure correct execution of transactions, it must preserve security (i.e., it must be free of covert channels), it must be implementable with untrusted code, and it must avoid starvation.

4 ACCOMPLISHMENTS

4.1 Commercial Solutions

Three major vendors produce trusted DBMSs: Sybase Secure SQL Server (Sybase 1993), Trusted Oracle (Oracle 1992), and Informix-OnLine/Secure (Informix 1993*a*, Informix 1993*b*). For concurrency control, Sybase uses the usual 2PL, which was shown above not to be secure. Trusted Oracle, on the other hand, uses a combination of locking and multiversioning techniques. Although this hybrid algorithm is secure, it does not produce one-copy serializable histories (Jajodia & Atluri 1992). Finally, Informix uses an approach by which a low-level transaction can acquire a write lock on a low data item, even if a high-level transaction holds a read lock on this data item. Thus, a low-level transaction is never delayed by a high-level transaction. The high-level transaction simply receives a warning that a lock on a low data item has been "broken." However, despite the broken lock, the high-level transaction is still committed, unless the programmer has explicitly added some code to perform rollback of the transaction. A major drawback of the Informix approach is that no information is provided to the high-level transaction specifying which lock has been broken if the transaction has acquired locks on several low data items. Thus, only very primitive handling of broken lock exceptions is possible at the application program level.

4.2 Research Solutions

Concurrency Control for the Replicated Architecture.

The replication-based approach (Air Force Studies Board 1983) provides an elegant way of reusing existing technology. The idea is to construct an MLS DBMS from single-level DBMSs. The DBMS at level l contains a replica of every data item that a subject at level l can access. The challenge is then to design a replica control protocol that will ensure one-copy serializability (Bernstein et al. 1987). In the architecture most commonly used, transactions are submitted to a global transaction manager (GTM). The GTM routes the transactions to their sites of origin and propagates the update projections to each of the dominating containers in turn.

The first concurrency protocol for this architecture was proposed by Jajodia & Kogan (1990). The idea behind the protocol is as follows: The GTM receives transactions and delivers them to the appropriate container database. Transactions completing at level l are submitted to the container at the level covering l, call it l'. Two transactions that conflict at level l must be submitted to level l' in the order in which they commit. However, if a local transaction is not serialized, then its update report can be sent to the GTM in an arbitrary order. To ensure serializability, the protocol assumes that each local scheduler preserves, in its serialization order, the order in which it receives conflicting transactions. Examples of such schedulers are conservative 2PL and conservative TO (Bernstein et al. 1987). Unfortunately, the protocol has a subtle flaw: there is no control over the relative serialization order of non-conflicting update projections. As is shown in (Kang & Keefe 1995), this lack of control can result in nonserializable histories. Fortunately, the problem can be easily solved by executing update projections serially.

Since the Jajodia-Kogan protocol works correctly for only those security posets that form a total order, Costich (1992) presents a variation of the Jajodia-Kogan protocol for security posets that are only partially ordered. In addition, the amount of trust required to implement the GTM is further reduced. However, along with the improvements came a new problem first described by McDermott, Jajodia & Sandhu (1991). The problem is that for some security posets, the protocol will deadlock, blocking any further progress by update projections, and produce nonserializable histories. Following this paper, several other papers attempted to characterize the set of "problem" security posets (Ammann & Jajodia 1993, Ammann & Jajodia 1994*b*, Kang & Keefe 1995). In (Ammann, Jajodia & Frankl 1996) the set of problematic security lattices is shown to be exactly those that embed a crown. An example of a crown is shown in figure 2.

Assume that the containers at levels $p_1, \ldots, p_n$ have executed transactions $T_1, \ldots, T_n$ respectively, and propagate the corresponding update projections to sites $p_{n+1}, \ldots, p_{2n}$. These sites are free to serialize the projections as they arrive in any convenient order. Let us assume then that T_1 is serialized before T_2 at site p_{n+1}, T_2 is serialized before T_3 at site $p_{n+2}, \ldots, T_i$ is serialized before T_{i+1} at site p_{n+1}, and finally T_n is serialized before T_1 at site p_{2n}. If there is a site that dominates all levels in the crown, it must now choose an order for the transactions $T_1, \ldots, T_n$ which is consistent with the decisions made at the levels it dominates. However, upon consideration, we see that choosing an order is impossible. Thus, the history is nonserializable and update projections will make no further progress. For a more detailed explanation of this problem, see (Ammann & Jajodia 1993) or (Kang & Keefe 1995).

The results of (Ammann, Jajodia & Frankl 1996) apply to protocols that attempt to construct an order on transactions dynamically as they are propagated through the poset. A method of avoiding this problem is to determine the order using timestamps generated when the transaction commits for the first time. Update projections arriving at a container are dispatched in timestamp

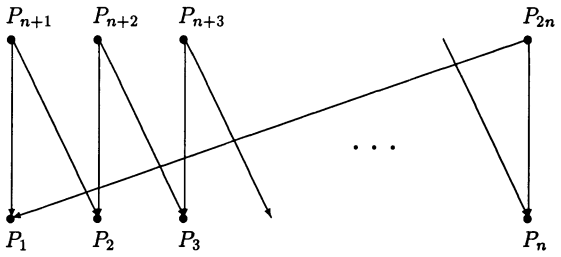

Figure 2 A Crown

order to the local transaction manager. This is done without aborting update projections by following the conservative timestamp protocol discussed in (Bernstein et al. 1987). Solutions based on this approach are presented in (McDermott et al. 1991) and (Kang & Keefe 1995). These solutions suffer from starvation as does the conservative timestamp protocol on which they are based.

The advantage of the replicated architecture approach is its simplicity and its ability to reuse DBMSs in their entirety. The major disadvantages of the replication-based schemes are the inefficient and inflexible use of resources: inefficient, because updating replicas incurs a significant overhead with no clear benefit; and inflexible, because resources are allocated among the DBMSs when the system is configured. Thus, the system cannot respond effectively to dynamic workload variations.

Concurrency Control for the Kernelized Architecture.

To improve the situation, it is necessary to share resources (e.g., memory, I/O channels, CPUs) across security levels. This is most easily achieved with the kernelized architecture (Air Force Studies Board 1983). Synchronizing readers and writers in an MLS environment is at the heart of the secure transaction processing problem. The solutions proposed by (Schaefer 1974, Lamport 1977, Reed & Kanodia 1979) are optimistic. High-level readers, before committing, must validate the objects they read. If those objects have been updated, the transaction must abort. To reduce contention between levels and reduce the prevalence of aborts, Keefe & Tsai (1990) proposed a multiversion protocol based on multiversion timestamp ordering (MVTO). The algorithm assigns timestamps such that high-level readers are pushed behind writers in the serialization order. This allows us to control the abort rate of high-level readers by providing additional versions. It also simplifies an untrusted implementation as explained in (Maimone & Greenberg 1990). There are four principal weaknesses of this protocol: (1) A high-level transaction, because it is not always given a current timestamp, can read arbitrarily stale data. The histories are serializable and are therefore consistency-preserving. However, they can produce unexpected results. (2) It is not possible to ensure liveness with a bounded number of versions. However, in practice, it is believed that the problem can be managed by carefully controlling the multiprogramming level and by implementing a policy of "euthanasia" (i.e., abort transactions that live too long). (3) A version

pool, physically separate from the primary segment, can reduce overall physical clustering and thereby degrade performance. Finally, (4) Relatively little is known about implementing DBMSs based on MVTO. 2PL-based solutions have received much more attention in the literature and from the database industry.

Recently, several snapshot algorithms have been proposed. The first, proposed by Ammann, Jaeckle & Jajodia (1992), maintains two complete snapshots of databases at dominated levels for read access by transactions at dominating levels. Ammann & Jajodia (1994*a*) present a multiversion algorithm in which a snapshot of a data item is generated only in cases where the snapshot value differs from the current value. A variation based on locking is presented by Pal (1996) together with a detailed implementation. All three protocols place an upper bound of two on the number of versions that need to be maintained. Transactions reading down, read the snapshot. Transactions accessing data at their own level, access the current state of the database. Periodically, the snapshots are synchronized with the database. Because the snapshot must be commit consistent, the lifetime of a transaction is generally limited to the snapshot period. Pal suggests some cases in which this restriction can be relaxed (Pal 1996). As with schedulers based on MVTO, there is a basic conflict between long-lived transactions and the freshness of the data they read. An open question is whether the snapshot algorithms can provide more precise control over this trade-off.

Weaker Correctness Criteria.

A solution to the problem of secure concurrency control can be obtained by viewing it in a different perspective. One can argue that the traditional notion of correctness is too restrictive for multilevel secure databases (Maimone & Greenberg 1990, Jajodia & Atluri 1992). This is supported by the fact that the integrity constraints in multilevel databases are different from those in conventional databases. Enforcing integrity constraints in MLS databases is difficult or even impossible, especially those constraints that are defined over data at different security levels (Meadows & Jajodia 1988). Since one cannot enforce the integrity constraints, there is no reason to insist on preserving serializability for these systems. Thus, Jajodia & Atluri (1992) argue that serializability requirements can be relaxed for MLS systems and propose three alternative notions of correctness—levelwise serializability, one-item read serializability, and pairwise serializability—for multilevel multiversion databases. These weaker notions of correctness can be used as alternatives for one-copy serializability. They exploit the nature of integrity constraints in MLS databases to improve the amount of concurrency.

Bertino, Jajodia, Mancini & Ray (1996) extend 2PL by providing a powerful set of linguistic constructs to the system programmer which supports exception handling, partial rollback, and forward recovery. The proper use of these constructs can prevent starvation of high-level transactions, and allows the system programmer to trade starvation for degree of transaction isolation.

Multilevel Transactions.

Multilevel update transactions allow a user to read and write information at multiple classification levels. This is done by decomposing a transaction into multiple subtransactions, each of which is assigned a single classification level and whose actions obey the simple and $\star$-properties. Through these multiple subtransactions, the multilevel update transaction is able to perform its task of reading and writing at multiple classification levels, while at the same time satisfying the simple and $\star$-properties. Each subtransaction is a flat transaction in the sense of (Bernstein et al. 1987).

In many applications, it is desirable to ensure atomicity of multilevel update transactions. However, it is not possible to guarantee atomicity without also permitting illegal information flows (Blaustein, Jajodia, McCollum & Notargiacomo 1993, Mathur & Keefe 1993, Jajodia, Smith, Blaustein & Notargiacomo 1996, Smith, Blaustein, Jajodia & Notargiacomo 1996). Blaustein et al. (1993) show that there are two crucial data dependencies which can occur in a multilevel transaction T: a low write followed by a high read of an item and a high read followed by a low read of an item. If all dependencies in T are of the first kind, T can be correctly executed (without affecting the dependencies) using a secure *low-first* algorithm. Similarly, if all dependencies in T are of the second kind, T can be correctly executed using a secure *high-ready-wait* algorithm.[§] However, if T has dependencies of both kinds, T cannot be correctly executed while meeting both security and atomicity requirements. Blaustein et al. (1993) develop a model of multilevel atomicity that defines varying degrees of atomicity and recognizes that low security level operations within a transaction must be able to commit or abort independently of higher security level operations.

All work on multilevel transactions cited above relies on the standard conflict-serializability (Bernstein et al. 1987) as the correctness criterion. Ammann, Jajodia & Ray (1996) propose an alternative notion of semantic atomicity which guarantees that either all or none of the actions of a multilevel transaction are present in any history. The notion of correct executions in this model is based on semantic correctness—that is, maintenance of integrity constraints—rather than serializability. They give a method whereby the developer of an application can statically analyze the set of transactions defined by the application and determine if the set ensures semantic atomicity.

Secure Commit Protocols.

Jajodia & McCollum (1993) study the secure analogs of 2PL and commit protocols. They modify 2PL to give a secure 2PL protocol (S2PL) that yields serializable histories. Every transaction in S2PL must satisfy the following four properties: (1) Well-formed – A transaction must obtain a shared lock (S-lock) before reading a data item and an exclusive lock (X-lock) before writing a data item. (2) Two-Phase – A transaction cannot request additional locks once it has issued an unlock action. (3) Strict – A local transaction holds on to all its locks until it completes. A subtransaction, on the other hand, may release any S-locks once it enters a prepared state. The subtransaction keeps all X-locks until it receives a commit or abort decision from its coordinator. (4) Signaling channel free – A high transaction must release its S-lock on a low data item when a low transaction requests an X-lock on the same data item. In such an event, the high transaction can either abort or start over.

Jajodia, McCollum & Blaustein (1994) recognized that while straightforward modifications to 2PC (and some optimizations of 2PC, e.g., presumed commit) yield a protocol that can be integrated with S2PL without any violation to global consistency, this is not so with EP. We illustrate the difficulty by way of an example.

Let T_1 and T_2 be two distributed transactions as follows:

$$\begin{aligned} T_1 &= r_1[x]r_1[y]w_1[z], \quad L(T_1) = high \\ T_2 &= w_2[x]w_2[y], \quad\quad\ L(T_2) = low \end{aligned}$$

[§]High-ready-wait has no storage channels, but has a limited bandwidth timing channel.

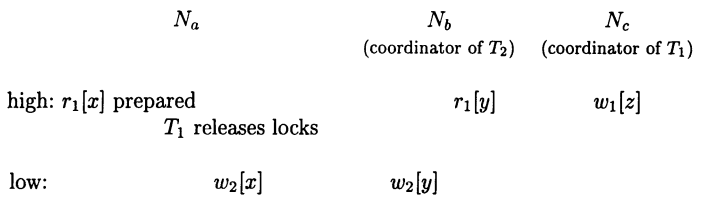

Figure 3 The distributed history D

Suppose T_1 is initiated at node N_c, and T_2 is initiated at node N_b. Furthermore, assume that data item x is stored at node N_a, y is stored at N_b, and z is stored at N_c. The coordinators, N_c and N_b, generate the subtransactions, and send them to the corresponding remote nodes. Accordingly, N_c divides T_1 into three subtransactions, $T_{1,a}$, $T_{1,b}$ and $T_{1,c}$, and then sends $T_{1,a}$ and $T_{1,b}$ to N_a and N_b, respectively. Similarly, N_b, upon dividing T_2 into two subtransactions $T_{2,a}$ and $T_{2,b}$, sends $T_{2,a}$ to N_a.

The execution of these subtransaction at each of the nodes may result in a distributed history D, as shown in figure 3. At N_a, the following sequence of events takes place: After successful execution, the subtransaction $T_{1,a}$ votes **yes** and enters the prepared state. At this point, the low-level subtransaction $T_{2,a}$ arrives. $T_{1,a}$ releases its S-lock on x, enabling $T_{2,a}$ to acquire an X-lock on x. At N_b, the subtransaction $T_{1,b}$ is successfully executed after the commit of $T_{2,b}$. At N_c, T_1 is committed after the coordinator receives the **yes** vote from all the participants.

Clearly, the distributed history D is not serializable since T_1 is serialized before T_2 at N_a, while the serialization order is reversed at node N_b. A moment of reflection shows that this inconsistency arises because the distributed transaction T_1 is not two-phase, although it is well-formed.

Atluri, Bertino & Jajodia (1995) give a different modification to EP, called Secure Early Prepare (SEP), which ensures global consistency of data. It modifies EP as follows: Assume T_i is decomposed into n subtransactions, and $T_{i,j}$, the subtransaction at the originating node N_j, is one among them. In the prepare phase, the coordinator generates subtransactions $T_{i,1}, T_{i,2}, \ldots, T_{i,n}$ and sends them to the participating nodes $N_1, N_2, \ldots N_n$, respectively. The coordinator also sends the security level s with each subtransaction. At each participant N_k, $T_{i,k}$ must first acquire an S-Lock (X-lock) on an item before it is read (written). S-locks as well as X-locks are long locks. If $T_{i,k}$ completes successfully, the participant augments its **yes** vote to the coordinator with a *read-low* indicator. A one-bit *read-low* indicator is added whenever $T_{i,k}$ has read an item from a lower level. Otherwise, it sends a **no** vote.

During the decision phase, if the coordinator receives **yes** votes from all the participants and if no subtransaction has read data from lower levels, it commits T_i and then sends **commit** messages to all its participants. An extra round of messages is required between the coordinator and all those participants N_j that sent the read-low indicator with their **yes** vote as follows: The coordinator first sends to each N_j a **confirm** message to confirm the commit. If N_j has not released its S-locks on any lower level data item during the time it has been in the prepared state, it responds with a **confirmed** message; otherwise, it sends a **not-confirmed** message. If the coordinator receives a **confirmed** message from all N_j's to which the coordinator has sent the additional round of messages, then it sends **commit** messages to all its participants; otherwise

it sends **abort** message. On the other hand, if the coordinator receives at least one no vote or if it times out waiting for a vote, it aborts the transaction, and sends **abort** messages to all participants.

SEP as described above has two drawbacks. First, if we compare the number of messages required by SEP to that required by EP, EP always requires about 4n messages (where n is the number of participants), while SEP sometimes requires more than 4n messages. An extra round of messages is required between the coordinator and those nodes where the subtransactions have read data from the lower levels. Second, SEP is overly pessimistic. Any high subtransaction that reads low data is aborted if any of its S-locks on the low data are broken while it waits for the confirm message from the coordinator. Thus, SEP aborts a transaction if there is a possibility of a violation of the two-phase requirement. It is entirely possible that the transaction is two-phase, even though some of the S-locks are broken. Atluri et al. (1995) propose an optimization to SEP, called OSEP, that reduces both the number of messages and the number of transactions being aborted. OSEP assumes that the clocks in the distributed system are synchronized.

Ray, Bertino, Jajodia & Mancini (1996) propose an *advanced secure commit protocol* (ASEP), a modification of SEP. Like SEP, ASEP sends an additional round of messages to all subtransactions which have read low data items. However, in ASEP, subtransactions can roll back to a prespecified save point and reexecute the subtransaction (as in (Bertino et al. 1996)) if they release their read locks on low data items. If a subtransaction successfully completes, then it sends a **yes** message. ASEP checks this by sending messages to these participants repeatedly until it receives all **yes** responses. However, if any of the participants sends a **no** message, the coordinator aborts the transaction.

5 TECHNICAL CHALLENGES

This paper has attempted to summarize our current understanding of multilevel secure transaction processing. Research in this area has yielded many interesting results, which, taken as a whole, show that this area represents an exciting and dynamic discipline. As we have indicated throughout the discussions in the previous section, there are still gaps in our understanding that require continued investigation. Below we list additional technical challenges that require future attention.

5.1 Advanced Transaction Models

Despite the substantial research activity in the area of secure transaction processing, the work in this area has been limited to the traditional transaction concepts; moreover, the existing solutions are not completely satisfactory.

Driven by the need to use databases in advanced application areas such as Office Information Systems (OISs), Computer Aided Design/Computer Aided Manufacturing (CAD/CAM), and software development and publication environments, traditional transactions have been extended in several directions by relaxing the rigid ACID requirements. These include, among others, (1) nested transaction models (which directly support the representation of nested computations), (2) workflow and extended transaction models (which represent processes in manufacturing and office environments and heterogeneous database management systems and are capable of capturing

the inter-relationships among different operational activities), and (3) long-duration transactions (which support environments such as CAD/CAM and OIS, where transactions typically last for days or weeks). To incorporate multilevel security into advanced database application areas, there is a need to investigate how multilevel security constraints impact transaction processing with these extended transaction models. Recently, Atluri & Huang (1996) have proposed a multilevel secure workflow transaction model and presented a Secure Petri Net (SPN) model to detect and prevent covert channels.

An additional issue in need of investigation is whether exploiting the properties of advanced transaction models (such as nested, split, or flexible models) yield better secure concurrency control solutions for traditional databases.

5.2 Index Methods

Database systems give users access to large amounts of data based upon key values. Therefore, mapping users' requests to the physical data (index management) is critical to performance. One typical method of doing this is using B+-trees. Because contention for the index can be high, specialized synchronization techniques are often used (Shasha & Goodman 1988, Gray & Reuter 1993). Maintaining B+-trees for database systems using a multiversion scheduling algorithm introduces additional issues by providing the option to map keys to logical data objects or to a particular version of that data object (Bober & Carey 1994). Furthermore, index management must be done in a way that ensures recoverability. It is not clear how to adapt existing methods to an MLS environment.

5.3 Recovery Methods

Efficient recovery algorithms are critical for good performance in systems that support transactions. Write-Ahead Logging (WAL) is a popular recovery technique (Gray & Reuter 1993). The method provides good performance because transactions' updates are not forced to disk on commit. Only the log records written by the transaction must be on stable storage when the transaction commits. This can be done more efficiently because: (1) The log records describing an update are generally quite compact, allowing the log records of several transactions to be written using a single operation; and (2) the I/O to the log is largely sequential, avoiding seek delays.

Recovery has received less attention from researchers than concurrency control. The main reason for this seems to be that most recovery algorithms do not require trust (privilege). It is possible to maintain a separate recovery log for each active security level. Recovery can proceed at each security level independently. The only constraint is that new transactions cannot begin execution until recovery is completed at all dominated security levels. Kang & Keefe (1992) propose an Undo/No Redo protocol based on this observation. This protocol was adopted in the experimental prototype described in (Warner & Keefe 1995).

This approach, while simple, is not very efficient. Its simplest implementation would employ a separate disk for each security level. Because the log file is generally duplexed, this would require 2n disks, where the system includes n security levels. In some systems, this may still be possible. However, in few cases would it be a good use of resources.

A less expensive solution would be to place the log files from multiple security levels on the same disk drive. While this would reduce the number of disk drives needed, it could result in increased head movement and a drastic increase in the effective access time. More work is needed

to develop good solutions to this problem because logging plays an important part in database system performance.

5.4 Buffer Management

The choice of concurrency control and recovery protocols is important for good performance. However, several other system aspects that are crucial for good performance are not apparent at this level of abstraction. An example of this is buffer management. The buffer manager allocates memory to competing transactions and provides methods to synchronize access to the shared memory. Both of these functions can have a large impact on performance. In addition, introducing security constrains the ways in which these problems can be addressed.

The simplest allocation policy divides the memory statically among the active security levels in the system. While simple, this is not very efficient, as it does not adjust to the dynamic requirements of the transactions at the various security levels. Transactions at one security level may thrash while memory is available at a security level below. The free movement of memory among levels must be ruled out for security reasons. Otherwise, knowledge of the allocation policy would allow covert communications using the page fault frequency. Now consider the problem of synchronization. From a security standpoint, the easiest solution is to let each level maintain its own buffer pool. However, this may result in the same page of data being replicated at many different levels. This will require some way of maintaining consistency among the replicas. A drawback of this approach is its poor use of memory. Maintaining a single copy of each page is desirable, but it makes in-place access more difficult. A range of solutions to these problems is discussed in (Warner, Li, Keefe & Pal 1996).

5.5 Performance

The performance of MLS transaction processing systems is influenced by several design choices as well as the user profiles. Among the design choices, the decomposition method adopted for relation decomposition and the transaction management techniques have the greatest impact. Similarly, the ratio of read-only to read-write transactions and the characteristics of database operations such as SELECT (e.g., selectivity factor) and PROJECT (e.g., duplication factor) influence the performance.

Kernelized architectures such as SeaView (Denning, Lunt, Schell, Shockley & Heckman 1988) include both horizontal and vertical partitioning of relations based on tuple and attribute class. While this partitioning of a multilevel relation into single-level relations satisfies the security requirements, its impact on storage and processing overhead during the reconstruction of a relation is significant. For example, if a large relation needs to be reconstructed during a query, then the processing overhead is significantly high. Another decomposition suggested by Jajodia & Sandhu (1991) partitions relations only horizontally and gives better performance than the SeaView decomposition in these cases. On the other hand, when a query requires only a small portion of a large relation, then the SeaView decomposition has better performance. An open question is whether we can optimize the decomposition method for a given set of queries so as to maximize performance.

Transaction management algorithms also have a significant impact on the performance of MLS systems. Some transaction management algorithms insist on aborting a high-level transaction when it accesses a relation that is subsequently (but before the high user committed) accessed for

update by a low user. Obviously, this affects the performance of the high-level user. In fact, there may be a problem of starvation. While multiversion concurrency control algorithms address this problem, they may give rise to additional storage costs, disk I/O, and processing overhead for managing multiple versions (Warner & Keefe 1995). However, by taking the semantics of queries in a specific application into account, it may be possible to customize the transaction management (see for example (Ammann, Jajodia & Ray 1996)). However, no performance data is currently available for these systems.

As in a nonsecure DBMS, the user profile, including the types and frequency of queries, the size and data redundancies of relations, and the ratio of update to read-only queries, have a significant impact on the performance of a secure MLS database. In fact, the impact of these parameters is felt much more in an MLS database.

First, consider the ratio of updates to read-only queries. The number of attributes and the percentage of tuples that a high-level user updates (thereby creating an additional copy at the high level) has significant impact on the overall system performance. When these are high, both storage and processing overhead are found to be significant. These increases have been quantified in several studies (Jajodia & Mukkamala 1992, Mukkamala & Jajodia 1994, Kang, Froscher & Mukkamala 1994, McDermott & Mukkamala 1994). Second, the properties of a query such as the selection factor in a SELECT operation or the duplication factor in a PROJECT operation have significant impact on the system performance. Often the impact is felt much more by the high-level users and to a lesser extent by the low-level users. As discussed above, the impact also depends on other design factors such as the decomposition method and the transaction management scheme (Jajodia & Mukkamala 1992, Mukkamala & Jajodia 1994).

5.6 Integrity/Repair

General techniques for database recovery compensate for the effects of occasional unavoidable but benign failures, either failures of individual transactions or more global failures affecting a system, device, or network. These recovery techniques share the philosophy that, once successfully completed and committed, a transaction is permanent and need not be considered further. However, this philosophy needs to be reexamined when it is assumed that the systems in question may be subjected to intentionally malevolent behavior. Systems that may suffer malicious attacks may be asked to execute transactions that complete normally but create inaccurate or misleading data. This could happen as a result of the system being broken into by outsiders, the subversion of another system with which it cooperates, or simply misbehavior on the part of its own legitimate users. In such a case, the possibility must be considered that the corrupted data may not be discovered immediately. Unless it fails to satisfy explicit integrity constraints, the malicious transaction is indistinguishable from a normal transaction at the time of execution. Other transactions may continue to read and act upon the bad data it has written, spreading integrity problems to the data they innocently write. By the time a problem is noticed, recovering is no longer a matter of backing out the effects of recent, uncommitted transactions. Nor is it appropriate to restore a whole device or database to a previous correct state, such as a checkpoint, since the results of other work not affected by the integrity problems would also be lost. Furthermore, the temporary halting or loss of other critical work could in itself be the object of a malicious attack. Techniques must be developed for repair of bad data that allow continued availability of the database while recovery proceeds, contain the effects of malicious transactions from propagating further, repair the effects of the original transaction and corruption propagated

through intervening transactions, and operate efficiently without undue impact on the continuing workload and performance of the DBMS.

Acknowledgement

The work of Vijayalakshmi Atluri was partially supported by National Science Foundation under grant IRI-9624222. The work of Sushil Jajodia was partially supported by National Science Foundation under grants IRI-9303416 and IRI–9633541 and by National Security Agency under grants MDA904-96-1-0103 and MDA904-96-1-0104. The work of Thomas Keefe was partially supported by National Security Agency under grant MDA904-94-C-612. The work of Catherine McCollum was partially supported by U.S. Air Force Rome Laboratory under contract number F19628-94-C-0001.

REFERENCES

Air Force Studies Board (1983), *Multilevel Data Management Security*, National Research Council, National Academy Press, Washington, DC.

Ammann, P., Jaeckle, F. & Jajodia, S. (1992), A two snapshot algorithm for concurrency control in secure multi-level databases, *in* 'Proc. Symp. on Research in Security and Privacy', Oakland, CA, pp. 204–215.

Ammann, P. & Jajodia, S. (1993), 'Distributed timestamp generation in planar lattice networks', *ACM Trans. on Computer Systems* **11**(3), 205–225.

Ammann, P. & Jajodia, S. (1994*a*), An efficient multiversion algorithm for secure servicing of transaction reads, *in* 'Proc. of the 1st ACM conference on Computer and Communication Security', Fairfax, VA, pp. 118–125.

Ammann, P. & Jajodia, S. (1994*b*), Planar lattice security structures for multilevel replicated database, *in* T. F. Keefe & C. E. Landwehr, eds, 'Database Security VII: Status and Prospects', North-Holland, Amsterdam, pp. 125–134.

Ammann, P., Jajodia, S. & Frankl, P. (1996), 'Globally consistent event ordering in one-directional distributed environments', *IEEE Transactions on Parallel and Distributed Systems* **7**(6), 665–670.

Ammann, P., Jajodia, S. & Ray, I. (1996), Ensuring atomicity of multilevel transactions, *in* 'Proc. IEEE Symp. on Security and Privacy', Oakland, CA, pp. 74–84.

Atluri, V., Bertino, E. & Jajodia, S. (1995), Degrees of isolation, concurrency control protocols, and commit protocols, *in* M. Morgenstern, J. Biskup & C. E. Landwehr, eds, 'Database Security, VII: Status and Prospects', North Holland, pp. 259–274.

Atluri, V. & Huang, W.-K. (1996), An extended petri net model for supporting workflows in a multilevel secure environment, *in* 'Proc. of the 10th IFIP WG 11.3 Workshop on Database Security', pp. 199–216.

Bell, E. & LaPadula, L. J. (1975), Secure computer systems: Unified exposition and multics interpretations, Technical Report MTR-2997, The Mitre Corporation, Burlington Road, Bedford, MA 01730, USA.

Bernstein, P. A., Hadzilacos, V. & Goodman, N. (1987), *Concurrency Control and Recovery in Database Systems.*, Addison-Wesley, Reading, MA.

Bertino, E., Jajodia, S., Mancini, L. & Ray, I. (1996), 'Advanced transaction processing in multilevel secure file stores', Accepted for publication in *IEEE Transactions on Knowledge and Data*

Engineering .

Blaustein, B. T., Jajodia, S., McCollum, C. D. & Notargiacomo, L. (1993), A model of atomicity for multilevel transactions, *in* 'Proc. IEEE Symposium on Security and Privacy', Oakland, California, pp. 120–134.

Bober, P. & Carey, M. (1994), Indexing alternatives for multiversion locking, *in* 'Proc. Int'l. Conf. on Extending Database Technology', pp. 145–158.

Costich, O. (1992), Transaction processing using an untrusted scheduler in a multilevel database with replicated architecture, *in* C. Landwehr & S. Jajodia, eds, 'Database Security V: Status and Prospects', North-Holland, Amsterdam, pp. 173–190.

Denning, D. E. (1982), *Cryptography and Data Security*, Addison-Wesley, Reading, MA.

Denning, D. E., Lunt, T., Schell, R., Shockley, W. & Heckman, M. (1988), The Seaview security model, *in* 'Proc. IEEE Symp. on Security and Privacy', Oakland, CA, pp. 218–233.

Gray, J. & Reuter, A. (1993), *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Mateo, California.

Informix (1993*a*), *Informix-OnLine/Secure Administrator's Guide*, Informix Software, Inc., Menlo Park, CA.

Informix (1993*b*), *Informix-OnLine/Secure Security Features User's Guide*, Informix Software, Inc., Menlo Park, CA.

Jajodia, S. & Atluri, V. (1992), Alternative correctness criteria for concurrent execution of transactions in multilevel secure databases, *in* 'Proc. IEEE Symposium on Security and Privacy', Oakland, California, pp. 216–224.

Jajodia, S. & Kogan, B. (1990), Integrating an object-oriented data model with multilevel security, *in* 'Proc. IEEE Symposium on Security and Privacy', Oakland, California, pp. 76–85.

Jajodia, S. & McCollum, C. (1993), Using two-phase commit for crash recovery in federated multilevel secure database management systems, *in* C. E. Landwehr, B. Randell & L. Simoncini, eds, 'Dependable Computing and Fault Tolerant Systems, Vol. 8', Springer-Verlag, New York, pp. 365–381.

Jajodia, S., McCollum, C. D. & Blaustein, B. T. (1994), Integrating concurrency control and commit algorithms in distributed multilevel secure databases, *in* T. F. Keefe & C. E. Landwehr, eds, 'Database Security, VII: Status and Prospects', North-Holland, Amsterdam, pp. 109–121.

Jajodia, S. & Mukkamala, R. (1992), Effects of seaview decomposition of multilevel relations on database performance, *in* C. E. Landwehr & S. Jajodia, eds, 'Database Security V: Status and Prospects', North-Holland, Amsterdam, pp. 203–225.

Jajodia, S. & Sandhu, R. (1991), A novel decomposition of multilevel relations into single-level relations, *in* 'Proc. IEEE Symp. on Security and Privacy', Oakland, California, pp. 300–313.

Jajodia, S., Smith, K. P., Blaustein, B. T. & Notargiacomo, L. (1996), Securely executing multilevel transactions, *in* S. K. Katsikas & D. Gritzalis, eds, 'Information Systems Security', Chapman & Hall, London, pp. 259–270.

Kang, I. E. & Keefe, T. F. (1992), On transaction processing for multilevel-secure replicated databases, *in* 'Proc. of the European Symposium on Research in Computer Security', pp. 329–347.

Kang, I. E. & Keefe, T. F. (1995), 'Transaction management for multilevel secure replicated databases', *Journal of Computer Security* **3**, 115–145.

Kang, M., Froscher, J. & Mukkamala, R. (1994), Architectural impact on performance of a multilevel database system, *in* 'Proc. 10th Annual IEEE Computer Security Applications Conf.', pp. 76–85.

Keefe, T. F. & Tsai, W. T. (1990), Multiversion concurrency control for multilevel secure database systems, *in* 'Proc. IEEE Symposium on Security and Privacy', Oakland, California, pp. 369–383.

Keefe, T. F., Tsai, W. T. & Srivastava, J. (1993), 'Database concurrency control in multilevel secure database management systems', *IEEE Trans. on Knowledge and Data Engineering* **5**(6), 1039–1055.

Lamport, L. (1977), 'Concurrent reading and writing', *Comm. of ACM* **20**(11), 806–811.

Maimone, W. T. & Greenberg, I. B. (1990), Single-level multiversion schedulers for multilevel secure database systems, *in* 'Proc. 6th Annual Computer Security Applications Conf.', Tucson, Arizona, pp. 137–147.

Mathur, A. G. & Keefe, T. F. (1993), The concurrency control and recovery problem for multilevel update transactions in mls systems, *in* 'Proc. IEEE Computer Security Foundations Workshop', Franconia, NH, pp. 10–23.

McDermott, J., Jajodia, S. & Sandhu, R. (1991), A single-level scheduler for replicated architecture for multilevel secure databases, *in* 'Proc. 7th Annual Computer Security Applications Conf.', San Antonio, Texas, pp. 2–11.

McDermott, J. & Mukkamala, R. (1994), Performance analysis of transaction management algorithm for the sintra replicated-architecture database system, *in* T. F. Keefe & C. E. Landwehr, eds, 'Database Security VII: Status and Prospects', North-Holland, Amsterdam, pp. 215–234.

Meadows, C. & Jajodia, S. (1988), Integrity versus security in multi-level secure databases, *in* C. E. Landwehr, ed., 'Database Security, Status and Prospects', North-Holland, pp. 89–101.

Mukkamala, R. & Jajodia, S. (1994), A performance comparison of two decomposition techniques for multilevel database systems, *in* T. F. Keefe & C. E. Landwehr, eds, 'Database Security VII: Status and Prospects', North-Holland, Amsterdam, pp. 199–214.

Oracle (1992), *Trusted Oracle Administrator's Guide*, Oracle Corp., Redwood City, CA.

Pal, S. (1996), A locking protocol for multilevel secure databases providing support for long transactions, *in* D. L. Spooner, S. A. Demurjian & J. E. Dobson, eds, 'Database Security IX: Status and Prospects', Chapman & Hall, London, pp. 183–198.

Ray, I., Bertino, E., Jajodia, S. & Mancini, L. (1996), An advanced commit protocol for mls distributed database systems, *in* 'Proc. Third ACM Conference on Computer and Communications Security', New Delhi, India, pp. 119–128.

Reed, D. P. & Kanodia, R. K. (1979), 'Synchronization with eventcounts and sequencers', *Comm. of ACM* **22**(5), 115–123.

Schaefer, M. (1974), Quasi-synchronization of readers and writers in a secure multi-level environment, Technical Report TM-5407/003, System Development Corp.

Shasha, D. & Goodman, N. (1988), 'Concurrent search structure algorithms', *ACM Trans. on Database Systems* **13**(1), 53–90.

Smith, K. P., Blaustein, B. T., Jajodia, S. & Notargiacomo, L. (1996), 'Correctness criteria for multilevel transactions', *IEEE Trans. on Knowledge and Data Engineering* **8**(1), 32–35.

Stamos, J. W. & Cristian, F. (1993), 'Coordinator log transaction execution protocol', *Distributed and Parallel Databases* **1**, 383–408.

Sybase (1993), *Sybase Secure SQL Server Security Administrator's Guide*, Sybase, Inc., Emeryville, CA.

Warner, A. & Keefe, T. (1995), Version pool management in a multilevel secure multiversion transaction manager, *in* 'Proc. IEEE Symposium on Security and Privacy', Oakland, California, pp. 169–182.

Warner, A., Li, Q., Keefe, T. & Pal, S. (1996), The impact of multilevel security on database buffer management, *in* 'Proc. of the European Symposium on Research in Computer Security', pp. 266–289.

BIOGRAPHY

Vijayalakshmi Atluri is an Assistant Professor of Computer Information Systems in the MS/CIS Department at Rutgers University. She received her BTech in Electronics and Communications Engineering from Jawaharlal Nehru Technological University, Kakinada, India, in 1977, MTech in Electronics and Communications Engineering from Indian Institute of Technology, Kharagpur, India, in 1979, and PhD in Information Technology from George Mason University, USA, in 1994. Her research interests include Information Systems Security, Database Management Systems, Workflow Management and Distributed Systems.

Sushil Jajodia received his PhD from the University of Oregon, Eugene. He is Director of Center for Secure Information Systems and Professor of Information and Software Systems Engineering at the George Mason University, Fairfax, Virginia. His research interests include information security, temporal databases, and replicated databases. He has published more than 150 technical papers in the refereed journals and conference proceedings and has edited or coedited ten books, including *Multimedia Database Systems: Issues and Research Directions*, Springer-Verlag Artificial Intelligence Series (1996), *Information Security: An Integrated Collection of Essays*, IEEE Computer Society Press (1995), and *Temporal Databases: Theory, Design, and Implementation*, Benjamin/Cummings (1993). He received the 1996 Kristian Beckman award from IFIP TC 11 for his contributions to the discipline of Information Security. The URL for his web page is http://www.isse.gmu.edu/~csis/faculty/jajodia.html.

Thomas F. Keefe received the BS degree in electrical engineering and the MS and PhD degree in computer science from the University of Minnesota in 1980, 1985 and 1990, respectively. He is currently an Associate Professor in the Department of Computer Science and Engineering at the Pennsylvania State University. His areas of interest are database security, transaction processing, object-oriented systems and software engineering. He is a member of the editorial board of the Journal of Computer Security.

Catherine McCollum is assistant department head in Secure Information Technology at the MITRE Corporation. Her main research interests are in database and information systems security.

Ravi Mukkamala is an Associate Professor in the Department of Computer Science at the Old Dominion University. He received the BS degree in electronics and telecommunications engineering from Osmania University in 1976, and the MTech degree in computer systems from Indian Institute of Technology, Kanpur in 1978. He earned the PhD in computer science from the University of Iowa in 1987. Before moving to an academic environment, he worked as a systems analyst at Telco-Pune, India, from 1978–81 and as a systems consultant at ACCI-Hyderabad, India, from 1981–83. His research interests include distributed database systems, data security, high-speed data communications, and performance analysis. He has published over 75 technical papers in conference proceedings and journals in these areas.