Architectural Impact on Performance of a Multilevel Database System

Myong H. Kang and Judith N. Froscher
Naval Research Laboratory
Information Technology Division
Washington, D.C. 20375

Ravi Mukkamala
Old Dominion University
Department of Computer Science
Norfolk, VA 23529

Abstract

Since protection and assurance are the primary concerns in MLS databases, performance has often been sacrificed in some known MLS database approaches. Motivated by performance concerns, a replicated architecture approach which uses a physically distinct backend database management system for each security level is being investigated.

This is a report on the behavior and performance issues for the replicated architecture approach. Especially, we compare the performance of the SINTRA[1] MLS database system to that of a typical conventional (non-secure, single-level) database system. After observing the performance bottlenecks for the SINTRA, we present solutions that can alleviate them.

1 Introduction

The multilevel data management security summer study in 1982 [1] recommended three near-term approaches to solving the multilevel secure (MLS) database management system (DBMS) problem: Integrity lock, Kernelized, and Distributed DBMS.

The integrity lock approach [Den85] attempts to combine encryption techniques with off-the-shelf database management systems. The trusted frontend applies an encrypted checksum to data in an untrusted database. The integrity lock approach is computationally intensive and has a potential covert channel. Since the trust is in the frontend filter, this architecture is susceptible to Trojan horse attack. Hence this approach cannot be used for highly assured MLS-DBMS (e.g., B3 system in terms of Trusted Computing Systems Evaluation Criteria (TCSEC) [3]).

The kernelized approach [11] relies on decomposing the multilevel database into single level databases which are stored separately, under the control of a security kernel enforcing a mandatory access control policy. The kernelized approach can yield reduced performance due to the need for recombining single level data to produce multilevel data.

Two basic architectural approaches were suggested under the distributed approach: (i) Each DBMS has data at a single security level, and (ii) Each DBMS contains data at a given security level and all data at lower security levels (replicated approach).

The first approach has been investigated [6, ?]. This approach has inherent problems because higher level queries have to be propagated to lower level untrusted backend to request data. Hence, we don't expect this approach to be used for highly assured MLS-DBMS. This approach also can yield reduced performance because it may require fragments to be transferred from a low to a high backend DBMS to present a multilevel relation to users.

The replicated architecture approach [5] uses a physically distinct backend database management system for each security level. Each backend database contains information at a given security level and all data at lower security levels. The system security is assured by a trusted frontend which permits a user access to only the backend database system which matches his/her security level.

Even though all the above approaches have been prototyped, few performance results have been published. In this paper, we concentrate on the replicated architecture approach. We study the behavior of this approach and compare the performance of the SINTRA database system which is based on this approach to that of a typical conventional (non-secure, single-level) database system. In this paper, main performance metrics is throughput. We have not considered price/performance. We believe those metrics are meaningful only after commercial venders produce B3/A1

[1] Secure INformation Through Replicated Architecture

MLS DBMSs because the comparison between the price of extra hardware and that of B3/A1 software will not be conclusive until that time.

This paper is organized as follows. A general concept of the replicated architecture, and its advantages and potential problems are discussed in section 2. Section 3 presents security and transaction models of the SINTRA system. The simulation model and the relevant parameters are described in section 4. Section 5 describes experiments that we performed. We summarize the lessons learned in section 6.

2 The Replicated Architecture

The SINTRA database system, which is currently being prototyped at the Naval Research Laboratory, is a multilevel trusted database management system based on the replicated architecture. The SINTRA database system consists of one trusted frontend (TFE) and several untrusted backend database systems (UBD). The role of a TFE includes user authentication, directing user queries to the backend, maintaining data consistency among backends, etc. Each UBD can be any commercial off-the-shelf database system. Figure 1 illustrates the SINTRA architecture.

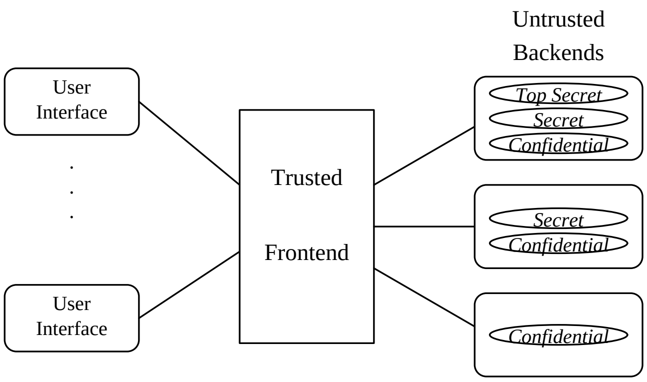

Figure 1: The SINTRA Architecture.

Since each UBD in a replicated architecture contains data at a given security level and all data from lower security levels, updates have to be propagated to higher security level databases to maintain data consistency among replicas. There are some problems which are related to this propagation.

1. If the propagation of update transactions is not carefully controlled, inconsistent database states among backend databases can be created. Consider this example. Two confidential level update transactions T_i and T_j are serialized in the order of (T_i, T_j) at the confidential level backend database system. Since these two transactions are update transactions, these transactions have to be propagated to the secret level. If these two transactions are serialized in the order of (T_j, T_i) at the secret level, an inconsistent database state between confidential and secret level backend databases may be created. Therefore, the serialization order introduced by the local scheduler at the user's session level must be maintained at the higher level UBDs. This condition ensures data consistency for the complete lattice [7].
2. Since lower level update transactions have to be propagated to higher level databases, high-level databases can be overloaded with lower level update transactions. Hence, the SINTRA system potentially suffers from performance degradation due to uneven workloads at backend computers.

A solution to problem 1 has been proposed in [7]. A brief description of the proposed solution is as follows:

> Since each backend DBMS does not guarantee the serialization order of transactions be the same as their submission order, an external control of serialization order is necessary. Therefore, update projections are sent to a backend DBMS one after another. Specifically, if T_i is serialized before T_j, then send T_i and wait until T_i is committed at the backend DBMS, and then send T_j.

In this paper, we study the behavior of the SINTRA system, especially problem 2, in detail and suggest solutions.

3 The Model

In this section, brief descriptions of security and transaction models are presented.

3.1 Security Model

The security model used here is based on that of Bell and LaPadula [2]. The database system consists of a finite set **D** of *objects* (data item) and a set **T** of *subjects* (transactions). There is a lattice **S** of security classes with ordering relation **<**. A class S_i *dominates* a class S_j if $S_i \geq S_j$. There is a *labeling function* **L** which maps objects and subjects to a security class:

$\mathbf{L}$: $\mathbf{D} \cup \mathbf{T} \cup \mathbf{C} \rightarrow \mathbf{S}$

where **C** is a set of backend database systems. Security class **u** *covers* **v** in a lattice if **u** > **v** and there is no security class **w** for which **u** > **w** > **v**.

We consider two mandatory access control requirements:

1. If transaction T_i reads data item x then L(T_i) $\geq$ L(x).
2. If transaction T_j writes data item x then L(T_j) = L(x).

3.2 Transaction Model

We adopt a layered model of transactions, where a transaction is a sequence of queries, and each query can be considered as a sequence of reads and writes. For example, `replace` and `delete` queries can be viewed as a read operation followed by a write operation, `insert` can be viewed as a write operation, and `retrieve` can be viewed as a read operation. A layered view of two transactions T_1 and T_2 is shown in figure 2.

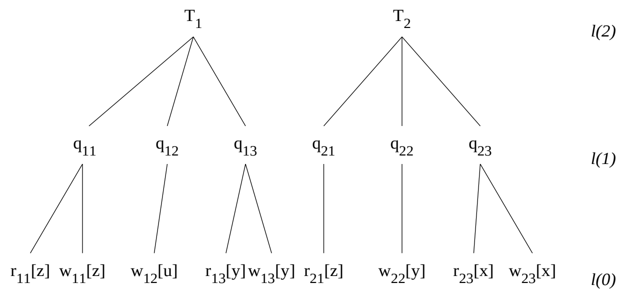

Figure 2: Layered model of two transactions.

Definition 1. A **transaction** T_i is a sequence of queries, i.e., T_i = <q_{i1}, q_{i2}, ..., q_{in}>. Each query, q_{ij}, is an atomic operation and is one of `retrieve`, `insert`, `replace`, or `delete`.

To model the propagation of updates produced by a given transaction to higher security level databases, *update projection* is defined.

Definition 2. An **update projection** U_i, which corresponds to a transaction T_i, is a sequence of update queries, e.g., U_i = <$q_{i2}, q_{i5}, ..., q_{im}$> obtained from transaction T_i by simply removing all `retrieve` queries.

4 Simulation Model

We have developed a simulation model for the SINTRA database. Our simulator is written in MODSIM II which is an object-oriented, discrete-event simulation language. The simulator can be described in three parts: (i) External user model, (ii) SINTRA global view which describes how backend DBMSs are connected, and (iii) each DBMS.

4.1 External User and Transaction Modeling

There are a fixed number of terminals at each security level. There are also a fixed number of data objects at each level. For example, if there are *n1* data objects at level 1 then there are (*n1* + *n2*) data objects at level 2 where *n1* data objects are replicas from level 1 and *n2* data objects are from level 2 itself.

Simulation parameters that are related to users and transactions are presented in table 1. A transaction that originates from a terminal consists of a series of queries. Each query, in turn, consists of read and write sequences. Each read or write sequence consists of a set of data objects. We believe that most of the transactions will be submitted by application programs. Hence, our interactive workloads model has a thinking period between transactions. *Think time* is defined as the time between the commit time of one transaction and the start time of the next transaction at each terminal. `ThinkTime` parameter is the mean of an exponentially distributed think time.

Table 1: Simulation parameters that are related to users and transactions.

Parameter	Meaning
`NumTerms[i]`	Number of terminals at level i
`ThinkTime`	Mean think time at each terminal
`MaxTranSize`	Max # of queries in a transaction
`MinTranSize`	Min # of queries in a transaction
`MaxQuerySize`	Max # of data objects in a read set
`MinQuerySize`	Min # of data objects in a read set
`RetrieveProb`	Probability of retrieve query
`InsertProb`	Probability of insert query
`deleteProb`	Probability of delete query
`ReplaceProb`	Probability of replace query
`WriteProb`	Write Probability in update transactions
`ReadDownProb[i]`	Read-down Probability at level i

A transaction consists of a number of queries which are uniformly distributed between `MaxTranSize` and `MinTranSize`. The type of queries in transactions are determined by the ratio among `RetrieveProb`, `InsertProb`, `deleteProb` and `ReplaceProb`.

A retrieve query consists of only read sequences (i.e., empty write sequences) and an insert query consists of only write sequences. Each read or write sequence consists of a number of data objects that is uniformly distributed between `MaxQuerySize` and `MinQuerySize`. Delete and replace queries consist of non-empty read and write sequences. The read sequences of delete and replace queries consist of a number of data objects that is uniformly distributed between `MaxQuerySize` and `MinQuerySize`. The data objects in the write sequences of delete and replace queries are selected replicas from those of their read sets (i.e., read before write).

`WriteProb` parameter represents the probability of duplicating data objects from the read set to its write set (i.e., `WriteProb`% of data objects in the read set will be copied to write set). `ReadDownProb` parameter specifies the probability of reading data object replicas from lower levels.

4.2 SINTRA Modeling: Global View

The general structure of our simulation model is presented in figure 3.

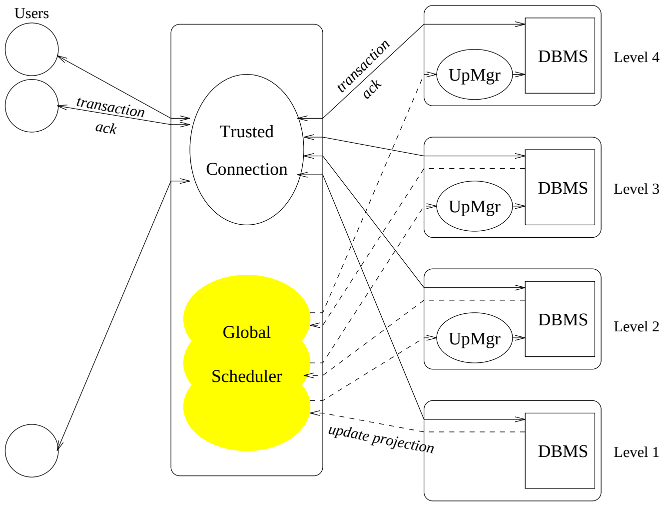

Figure 3: SINTRA simulation model.

There are two components in the TFE: the trusted connection and global schedulers. The trusted connection checks the security level of a user transaction and delivers the transaction to the transaction manager of DBMS at the same level. When a user transaction is committed, an acknowledgement (*ack*) is sent to the terminal through this trusted connection. If the user transaction is an update transaction, then the corresponding update projection is sent to the global scheduler in TFE which in turn sends it to the next level update projection manager (UpMgr).

When an update projection is passed from a backend to the frontend, it is written on disk for recovery purpose [10]. `FrontCPUTime` and `FrontDiskTime` parameters specify the time to receive an update projection, write it on a disk, and send it to the next level update projection manager (see table 2).

The update projection manager receives update projections from the global scheduler and sends them to the transaction manager of DBMS, one by one, to preserve the serialization order which was determined at a lower level DBMS [7]. In other words, an update projection cannot be sent to DBMS if the previously submitted update projection has not been committed. If an update projection cannot be sent to DBMS then it will be stored in a queue.

Simulation parameters that are related to the frontend are presented in table 2. Our simulator assumes that there is one CPU and one disk drive in the frontend machine. The CPU and the disk are defined as resource objects in MODSIM II. A resource object in MODSIM II provides an asynchronous blocking mechanism that allows simulation time to elapse while waiting for a resource and a statistics gathering mechanism to detect potential bottlenecks.

Table 2: Simulation parameters that are related to the frontend.

Parameter	Meaning
`FrontDiskTime`	Disk read/write time
`FrontCPUTime`	Processing time
`Comm`	Time to pass a message between front and back ends

4.3 Backend DBMS Modeling

Each backend DBMS consists of three components: the transaction manager, the concurrency control manager, and the resource manager. A simulation model for each DBMS is presented in figure 4.

The transaction manager is a traffic controller between the concurrency control manager and the resource manager. As far as the transaction manager is concerned, there is no difference between user transactions and update projections (i.e., they are treated in the same manner).

The concurrency control manager maintains `Locktable` and uses strict two phase locking protocol. Hence, all locks will be released when a transaction commits. The unit of granting locks is a query. If all locks for read and write sets within a query can be granted then it grants locks and returns the query to the transaction manager. Otherwise it will keep the query in the `Blocked queue` until locks

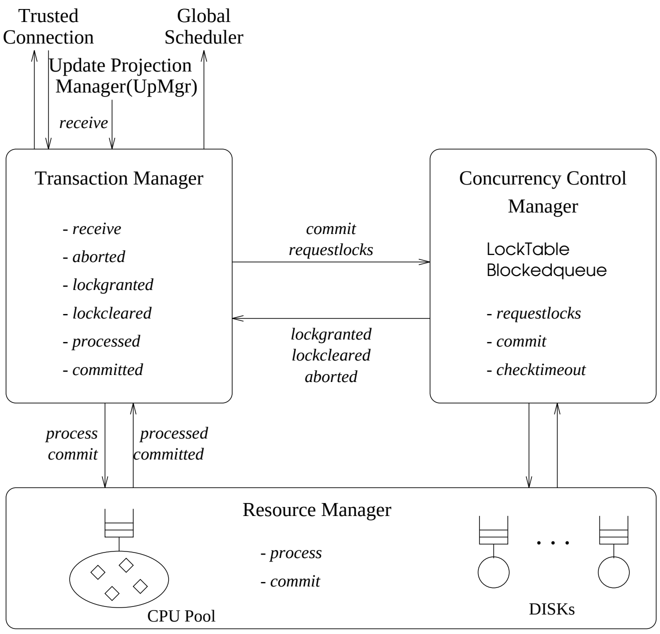

Figure 4: DBMS simulation model.

can be granted. A *waits-for* graph is maintained to find deadlock.

The resource manager manages CPUs and disks, and processes queries. Each backend may have multiple CPUs and disks. All CPUs and disks are defined as resource objects in MODSIM II. We assume that all CPUs are identical and disks are different depending on the data objects on these (see figure 4). For simplicity, we assume that data objects are uniformly distributed on disk. For example, i-th data object is located at the disk ($i \bmod m$) where m is the total number of disks at the backend.

Simulation parameters related to each backend DBMS is presented in table 3. `ProcessLockTime` parameter specifies the fixed CPU time to process the lock request per query by the concurrency control manager.

Table 3: Simulation parameters related to backends.

Parameter	Meaning
`NumDataObj[i]`	Number of data objects at level i
`ProcessLockTime`	Lock processing time for each query
`NumCPU[i]`	Number of CPUs in backend i
`NumDisk[i]`	Number of disks in backend i
`HitRatio`	Buffer pool hit probability
`WriteonMemTime`	Time to write a data object on memory
`InitWriteCPU`	Time to initiate a disk write
`MaxDiskTime`	Maximum disk read/write time
`MinDiskTime`	Minimum disk read/write time
`MaxCPUTime`	Maximum processing time
`MinCPUTime`	Minimum processing time

When the resource manager processes queries, it will read necessary data either from memory or disk. `HitRatio` parameter specifies the ratio between reading from memory and reading from disk. Processing time for a data object is uniformly distributed between `MaxCPUTime` and `MinCPUTime` if the data object is in memory. If the data object is in disk then it takes extra disk access time that is uniformly distributed between `MaxDiskTime` and `MinDiskTime`.

All modified data will be temporarily written on memory. `WriteonMemTime` parameter specifies a fixed time to write a modified data object on memory. The resource manager writes modified data on disk when it is asked to commit a transaction. The time to write a data object on disk is also uniformly distributed between `MaxDiskTime` and `MinDiskTime`. `InitWriteCPU` models the CPU overhead associated with initiating a disk write.

Let us trace how a typical transaction is executed.

1. The transaction manager receives a transaction.

2. The transaction manager dispatchs a query to the concurrency control manager.

3. If all the required locks for read and write sets of the query can be granted then those will be granted. Otherwise it will be put in the `Blocked queue` until those locks can be granted. Once locks for a query are granted, it is sent back to the transaction manager.

4. The transaction manager receives a query whose locks are just granted and sends it to the resource manager to execute it.

5. The resource manager executes a query if resources (i.e., CPU and disks) are available. If more than one resource is available then objects in read (write) set will be processed simultaneously. If the necessary resources are not available temporarily, then the query has to wait for resources. When the execution of the query is done, it will be returned to the transaction manager.

6. The transaction manager repeats steps (2) – (5) until all queries in a transaction are executed. Once all queries have been executed then the transaction manager asks the resource manager to commit the transaction and asks the concurrency control manager to release locks.

The resource manager makes use of as much parallelism as possible. It uses the *fork*-and-*join* construct. Consider the backend that has two disks and three read operations from disk that read data objects 1, 2, and 4. Data object 1 should be read from disk 1 and data objects 2 and 4 should be read from disk 0. Even though there is no conflict to read data objects 1, 2, and 4 in parallel, there is a resource conflict. Hence, even if all three read operations are forked at once, it actually will be executed as in figure 5.

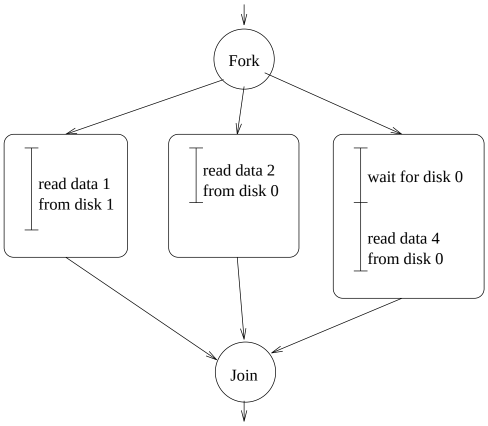

Figure 5: Execution pattern for reading data objects 1, 2, and 4.

The parallelism among queries are not exploited in our simulation (only parallelism among transactions and parallelism within read (write) set have been exploited). Hence the exploitation of parallelism can be controlled by the number of queries in a transaction and the number of data objects in read and write sets.

5 Experiments

This section reports selected results of the performance comparison between SINTRA and a typical conventional (single-level) DBMS. The conventional DBMS that we used is one of the backend DBMS which is described in section 4.3.

Since update projections are executed in serial while many user transactions may be executed concurrently, the execution of update projections can be delayed depending on user workload. In our experiment, the time to propagate update projections to the next level has been measured (i.e., from the commit time of an update projection at one level to the commit time of the update projection at the next level). Also the length of the update projection queue has been monitored to make sure our results are steady state results.

The SINTRA DBMS consists of four security levels. Table 4 shows the values of the fixed simulation parameters for the SINTRA. Table 5 shows the values of varying simulation parameters for the SINTRA (i.e., unless specified otherwise, the following values have been used). The specific values of simulation parameters that apply only to a particular experiment will be specified in an appropriate section. Hence, the values in each section overwrite the values of parameters in table 5.

Table 4: Simulation parameter settings I.

Parameter	Setting
`MaxTranSize`	5
`MinTranSize`	1
`MaxQuerySize`	7
`MinQuerySize`	3
`WriteProb`	50 %
`ReadDownProb[i]`	20% when i = 2, 30% when i = 3, 40% when i = 4
`NumDataObj[i]`	800 when i = 1, 1400 when i = 2, 1700 when i = 3, 2000 when i = 4
`FrontDiskTime`	12 ms
`FrontCPUTime`	1 ms
`Comm`	0.3 ms
`ProcessLockTime`	0.5 ms
`HitRatio`	50 %
`WriteonMemTime`	2 ms
`InitWriteCPU`	2 ms
`MaxDiskTime`	36 ms
`MinDiskTime`	12 ms
`MaxCPUTime`	10 ms
`MinCPUTime`	2 ms

Table 5: Simulation parameter settings II.

Parameter	Setting
`NumTerms[i]`	8 when i = 1, 6 when i = 2, 4 when i = 3, 2 when i = 4
`ThinkTime`	0.0 sec
`RetrieveProb`	40 %
`InsertProb`	20 %
`deleteProb`	20 %
`ReplaceProb`	20 %
`NumCPU[i]`	1 for all i
`NumDisk[i]`	2 for all i

Since the SINTRA DBMS provides a single MLS database management service, some of the corresponding simulation parameters for the conventional (non-secure, single-level) DBMS are the sum of 4 different backends of the SINTRA DBMS. For example, if there are w, x, y, z terminals at 4 security levels in the SINTRA then there are $(w + x + y + z)$ terminals in the corresponding conventional DBMS. In our experiment, we use 20 terminals for conventional DBMS. Multiprogramming level 20 was chosen because the throughput is maximized around this value in the simulation of our conventional DBMS (see figure 6).

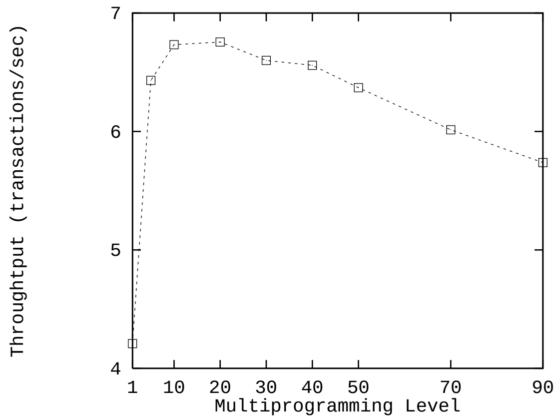

Figure 6: Throughput vs. Multiprogramming level for a conventional DBMS.

In our simulation, the conventional system has 2000 data objects, which is the same number of data objects as in level 4 of the SINTRA system. Ambiguous parameters for the conventional DBMS will be specified at each section.

5.1 Think time

First, we simulate an interactive environment with different think times between transactions. Remember that the think time is defined as the time between the commit time of one transaction and the start time of the next transaction at each terminal. The result of experiments is shown in figure 7.

We could not report the throughput of SINTRA when the think time is less than 3.3 sec because the SINTRA DBMS never reached the steady state. In other words, if the think time is less than 3.3 sec then the rate of generating update projections is greater than the rate of processing those at higher level backends with our input profile (i.e., the length of update projection queue is growing). If there exists enough think time (3.3 sec or more in our experiment) then update projections can be processed between user transactions (i.e., during think time). Hence, we could obtain the steady state throughput.

The two systems (i.e., SINTRA and single-level DBMS) show almost the same throughput. Especially,

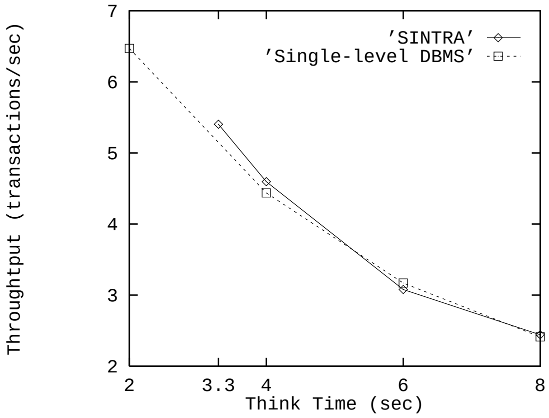

Figure 7: Throughput vs. Mean think time.

if the think time is longer than 6 seconds, these systems display almost identical throughput. This experiment shows that if the think time is long (i.e., if the think time is much longer than the average execution time of transactions), the throughput of the systems become identical because the think time becomes the dominating factor (i.e., the system is underloaded).

This experiment reveals an important fact: The SINTRA system needs a strategy to prevent the lower levels from overwhelming the higher levels with their updates. That strategy can be a flow control mechanism.

In the following sections when we simulate batch environment, SINTRA uses a simple dynamic flow control mechanism that inserts a delay to the response to user transactions. When the trusted connection receives a response from the backend i, it finds the maximum length of update projection queues from level i and higher. Then it postpones the response depending on the maximum queue size (i.e., in our experiment, 500 ms $\times$ maximum length). The reason for searching only its own level and higher level queues is as follows. For example, let long update projection queue occur at level 3. Since level 2 and level 1 are responsible for this long queue, user transactions at level 1 and level 2 need to be slowed down. Also user transactions at level 3 need to be slowed down so that update projections can be processed. However, user transactions at level 4 are not responsible for this long queue, and hence they need not be slowed down. This strategy has been chosen because it works well for our simulation; however, we have not identified the best strategy at this time.

5.2 Query Mix

Since all update projections must be propagated to the higher security levels in the replicated architecture, the ratio among different type of queries (especially retrieve vs. update queries) may affect the performance. To investigate this effect, we have experimented with the following four options for query mix (see table 5):

1. `RetrieveProb` = 0.25, `InsertProb` = 0.25, `deleteProb` = 0.25, `ReplaceProb` = 0.25.
2. `RetrieveProb` = 0.4, `InsertProb` = 0.2, `deleteProb` = 0.2, `ReplaceProb` = 0.2.
3. `RetrieveProb` = 0.5, `InsertProb` = 0.16, `deleteProb` = 0.16, `ReplaceProb` = 0.18.
4. `RetrieveProb` = 0.75, `InsertProb` = 0.08, `deleteProb` = 0.08, `ReplaceProb` = 0.09.

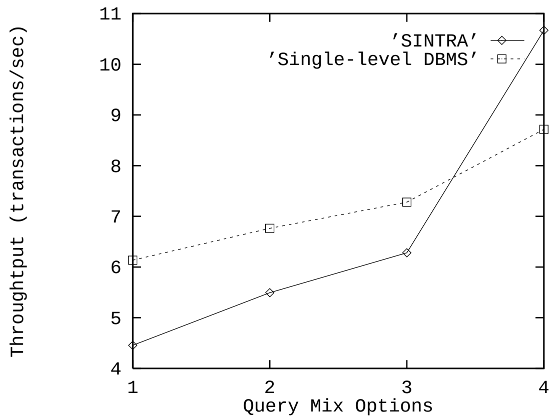

Figure 8: Throughput vs. Different query mixes.

The result demonstrate that SINTRA performs well for retrieve oriented input profiles. However, considering that each SINTRA backend has a lighter workload than the single-level system, the performance of SINTRA did not meet our expectation.

5.3 User Mix

Consider an environment where very few users have top secret or secret clearances but many have unclassified or confidential clearances. In this case, there are a large number of users at lower security levels and a small number of users at higher levels.

In this section, we have used different user (transaction) mix options. The number of terminals at different security levels are (Note that level 1 is the lowest security level):

1. `NumTerms[1]` = 5, `NumTerms[2]` = 5, `NumTerms[3]` = 5, `NumTerms[4]` = 5.
2. `NumTerms[1]` = 8, `NumTerms[2]` = 6, `NumTerms[3]` = 4, `NumTerms[4]` = 2.
3. `NumTerms[1]` = 10, `NumTerms[2]` = 6, `NumTerms[3]` = 2, `NumTerms[4]` = 2.
4. `NumTerms[1]` = 16, `NumTerms[2]` = 2, `NumTerms[3]` = 1, `NumTerms[4]` = 1.

The results of our experiments are shown in figure 9.

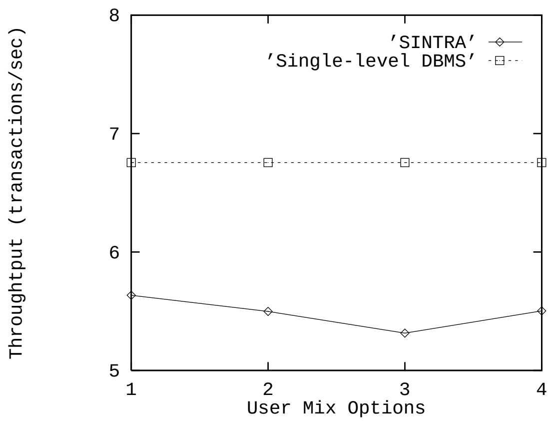

Figure 9: Throughput vs. Different user mixes.

Considering that no SINTRA backend has higher workload than a single-level DBMS, this was an unexpected result. In the case of option 4, in specific, despite that the workloads at higher level backends are lighter than that of the level 1 backend, the performance has not improved. Hence, the workload of update projections from lower levels cannot be the reason for this unexpected result.

A careful analysis of the simulation shows that the performance bottleneck was not the lack of resources but rather lack of concurrency. In section 2 and 4.2, we explained that update projections will be submitted to DBMS one by one to maintain the order of serialization that was determined at the lower level DBMS. Hence, there is no concurrency among update projections, and this results in update projections staying in the update projection queue for long time. This makes sense from figure 6, where we observed that the performance suffers when the level of concurrency is too low.

For a general security lattice, update projections must be submitted one by one to maintain the serialization order that was determined at the lower level [7]. However, if the security classes form a completely ordered set, then update projections that do not conflict with each other can be submitted to DBMS simultaneously [7]. In the following section, we take advantage of this fact and show that the performance of SINTRA system can be improved.

5.4 Using Data Dependence Analysis

It is well known that the serialization order among non-conflicting update projections need not be maintained if the security classes form a completely ordered set [7]. Hence, if there are update projections that do not conflict with each other then those can be submitted to the DBMS simultaneously. The data dependence analysis [9] that detects conflicts among transactions without the knowledge of data or the concurrency control mechanism of backend DBMSs has been used to increase parallelism among update projections.

Suppose that the update projection manager receives a sequence of update projections (U_1, U_2, U_3, ... , U_n). Since U_1 is the first update projection, it will be submitted to the backend DBMS. The dependence analysis is applied between U_1 and U_2. If there is dependency then U_2 and the remaining update projections have to wait until U_1 is committed. If there is no dependency between U_1 and U_2 then U_2 can be submitted to the backend DBMS before U_1 is committed. In general, U_i can be submitted only if there is no dependency between U_i and all update projections that have been submitted but have not committed.

The dependence analysis can remove all dependence relationships between U_1 and other update projections as soon as U_1 is committed. Suppose that at a later time an additional transaction arrives, making the sequence (U_2, U_3, ... , U_n, U_{n+1}). All existing dependence relationships are still valid; only the dependence relationships between U_{n+1} and other update projections need to be analyzed.

We perform the same experiments that were performed in sections 5.1, 5.2 and 5.3. Data dependence analysis is performed by the update projection manager to submit as many non-conflicting update projections as possible at once. Figure 10 shows performance comparison with different think times.

We observed steady state behavior of SINTRA when the think time is greater or equal to 2.1 second. In other words, the concurrent execution of update projections helped to reduce the queue life of update projections.

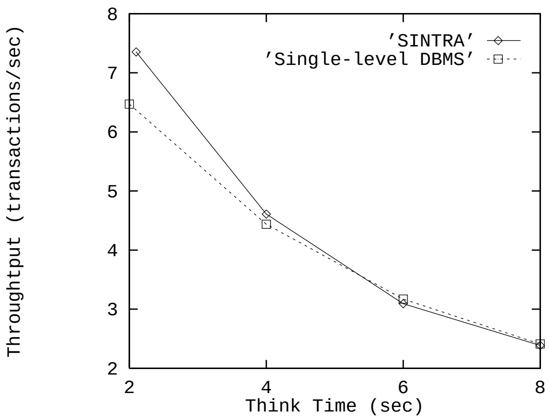

Figure 10: Throughput vs. Mean think time.

Figure 11 shows the performance comparison with different query mixes.

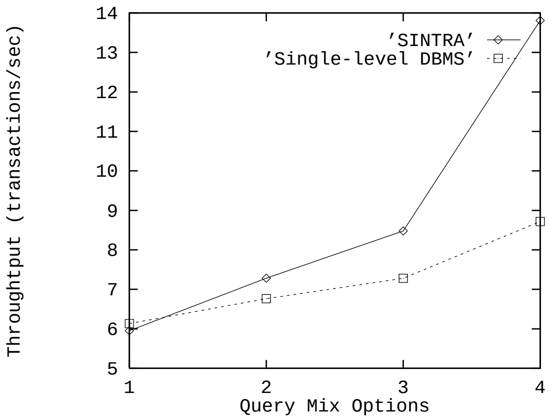

Figure 11: Throughput vs. Different query mixes.

As compared to Figure 8, throughput has increased at all the query mix options.

Figure 12 shows the performance comparison with various user mixes at different security levels.

The results are much more encouraging for SINTRA. Also the above results show that the performance of SINTRA is much more sensitive to the type of queries than the number of users at different security levels.

6 Summary

This paper reports selected results of the simulation study that compares the throughput for the SINTRA

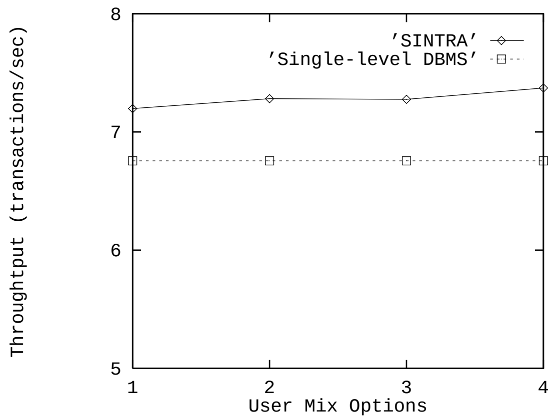

Figure 12: Throughput vs. Different user mixes.

multilevel database system to a non-secure, single-level database system. The SINTRA system uses physical separation and data replication to provide security. Additional benefit of extra hardware of the replicated approach is the performance improvement. Our simulation study shows that extra hardware actually helps to improve performance. Our results also show that the SINTRA performs relatively well for the update oriented input profiles and very well for the retrieve-oriented input profiles.

Our simulation reveals that the main performance bottleneck of SINTRA system is the lack of concurrency among update projections. Data dependence analysis that can detect dependency among update projections can be used to alleviate this problem.

Our future work includes: (1) investigate methods that can increase the parallelism among update projections, (2) investigate dynamic secure flow control strategies. We plan to experiment with the communication pump [8], and (3) investigate the types of hardware configuration (especially, computing power and I/O speeds) that are needed to meet a specific performance and cost requirements of the user.

References

[1] Multilevel Data Management Security, Air Force Studies Board, Commission on Engineering and Technical Systems, National Research Council, National Academy Press, Washington, D.C. (1983).

[2] Bell, D. E., and LaPadula, L. J. Secure computer systems: Unified exposition and *multics* interpretation. The Mitre Corp, (1976).

[3] Department of Defense National Computer Security Center, *Trusted computer system evaluation criteria*, DoD5200.28-STD (1985).

[4] Denning, D. Commutative filters for reducing inference threats in multilevel database systems. The IEEE symposium on Security and Privacy (1985).

[5] Froscher, J. N., and Meadows, C. Achieving a trusted database management systems using parallelism. in Database Security II: Status and Prospects (North-Holland 1989).

[6] Jensen, C., et al. SDDM: A prototype of a distributed architecture for database security. Conference on Data Engineering (1989).

[7] Kang, M. H., Froscher, J. N., and Costich, O. A practical transaction model and untrusted transaction manager for multilevel-secure database systems. The Eighth IFIP Workshop on Database Security (1992).

[8] Kang, M. H., and Ira S. Moskowitz. A pump for rapid, reliable, secure communication. ACM Conference on Computer and Communications Security (1993).

[9] Kang, M. H. Data dependence analysis for an untrusted transaction manager. Conference on Information and Knowledge Management (1992).

[10] Kang, M. H., et el. Achieving database security through data replication: The SINTRA prototype. Submitted for publication (1994).

[11] Lunt, T., et al. The SeaView security model. IEEE Transactions on Software Engineering, 16, 6 (1990).

[12] O'Connor, J., and Gray, J. A distributed architecture for multilevel database security. National Computer Security Conference (1988).