Journal of Science and Technology
ISSN: 2456-5660 Volume 7, Issue 02 (MAR-APR 2022)

www.jst.org.in *DOI: https://doi.org/10.46243/jst.2022.v7.i02.pp332-340*

ESMVCC: AN ENHANCED SECURE REAL-TIME MULTIVERSION CONCURRENCY CONTROL ALGORITHM

Ebrahim Abduljalil[1]*, Pritam R. Patil[2] , Fursan Thabit[3] , S.B.Thorat[4]

[1] Ph.D candidate, School of Computational Science, SRTM University, Nanded, Maharashtra, India

[2]Assistant Professor Institute of Technology and Managment, Nanded, Maharashtra, India

[3] Postdoctoral Researcher in Department of Computer Engineering Faculty of Engineering, Ege University,Turkey

[4] Director, Institute of Technology and Managment, Nanded, Maharashtra, India

* Corresponding author: Tel: 00919052703361, Email: EBMEKHLAFI@GAMIL.COM

To Cite this Article
Ebrahim Abduljalil, Pritam R. Patil , Fursan Thabit , S.B.Thorat "**ESMVCC: AN ENHANCED SECURE REAL-TIME MULTIVERSION CONCURRENCY CONTROL ALGORITHM**", *Journal of Science and Technology, Vol. 07, Issue 02, Mar-April 2022.*

Article Info
Received: 26-03-2022 *Revised: 18-04-2022* *Accepted: 26-04-2022* *Published: 30-04-2022*

Abstract

In addition to ensuring database integrity, secure concurrency control techniques must be free of covert channels emerging from data conflicts between transactions. Transactions with higher access classes are mistreated by current secure concurrency control techniques. Because it solves the priority inversion problem more effectively. This paper introduces and evaluates a novel secure multiversion concurrency control technique. These experiments show that when comparing the proposed concurrency control algorithm to the SMVCC concurrency control protocol, the proposed algorithm outperforms the SMVCC protocol in terms of total average response time. ESMVCC has the lowest overall average transaction response time and the highest level of fairness among secure concurrency control protocols.

Keywords: Multiversion, Secure Concurrency Control Protocol, Multilevel Security.

1. Introduction

Concurrency control is the method of managing the execution of transactions in a MLDBSs at the same time without interfering with one another. The basic purpose of concurrency in traditional databases is to assure serializability. The database's integrity and consistency are among the other objectives.

Local databases/users at separate sites share multilevel secure databases which handle transactions at different security levels. As a result, secure access to data items shared by several transactions is part of the concurrency requirements.

The Bell Lapadula model [1] presents an information flow that prevents direct information flow from high to low levels. However, there may be some indirect information flow through covert channels, which are developed during the information flow without being in the ones' knowledge.

This resulted in the following extra needs for concurrency control mechanisms in secure databases, as stated by Kaur et al. [2] :

- Confidentiality Requirements: Free of covert channels
- Integrity Requirements: must guarantee serializability
- Availability Requirements: no infinite delays or repeated abort means no starvation should occur.

In this paper, the researcher proposes a secure CC algorithm that meets the above-mentioned requirements.

This remaining sections in this paper are: Section 2 outlines the Concurrency Control Protocols for MLS/DBSs. Section 3 discusses the secure concurrency control protocol requirement. Section 4 presents the architecture of multilevel secure DBMS . Section 5 presents the proposed secure multiversion concurrency control algorithm. Section 6. we evaluate the performance of secure concurrency control protocols through detailed simulation study. Then, the paper is wrapped up with the paper summary in section 7.

2. Concurrency Control Protocols

Concurrency control is required for MLS/DBs in order to avoid a covert channel, which may be simply built by integrating multilevel secure transactions with traditional concurrency control protocols. An MLS/concurrency DB's control protocol must ensure that no covert channels exist between transactions at different security levels. When applied to multilevel secure transactions, traditional concurrency control protocols such as 2PL and Timestamp Ordering protocols cause problems such as covert channels, too much delay or repeated aborts of high security level transactions, and retrieval anomaly.

3. Secure Concurrency Control Protocol Requirements

Concurrency control in MLS/DBs must also ensure security in addition to providing correct concurrent execution of transactions. There are three security requirements for the multilevel secure concurrency control protocol[3].

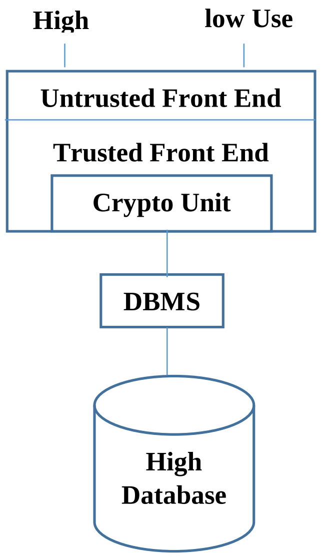
1. Integrity requirements: A secure concurrency control protocol must ensure that transactions are executed correctly (serializability).
2. Secrecy requirements: A secure concurrency control protocol must be free of covert channels.
3. Availability requirements: A secure concurrency control protocol should not cause starvations (indefinite delays).

4. Multilevel Secure DBMS Architectures

There are three architectures for MLS/DBSs, according to "The Woods Hole Report"[30], which protect classified information from unauthorized users based on the classification of the data and the clearance level of users.

Several classification levels of data are integrated in the same database in the integrity lock architecture [4], as shown in figure 1, and encryption is used to handle data item secrecy. It's a front-end and back-end architecture in basic. The front-end is further divided into two categories: untrusted and trusted. Query parsing, optimization, and computation are all used by the untrusted front-end. Authenticating users, installing integrity locks, and testing integrity locks are all handled by the trusted front-end (TFE). The trusted front-end is essentially a filter that encrypts the data item's sensitivity level before it is placed in the database. It then stamps a checksum or integrity lock to that data item. The integrity lock is determined by the sensitivity level of the data item as well as the data item itself. The integrity lock is saved in the front-end or back-end database system, along with the matching data item. The TFE decrypts the integrity lock of the data item being queried and compares it to the one recently computed when a data item is retrieved. It also checks whether the data item's sensitivity level is lower than the user's, after which the retrieved data item is forwarded to the querying user.

The retrieved data item is provided to the querying user if the integrity lock matches and the data item's sensitivity level is less than the user's. The retrieval procedure will be denied if this is not the case. This architecture is also known as spray paint architecture since each level of data item is actually painted with a specific colour dependent on the cryptographic integrity lock. The benefit of this architecture is that the technology required to build it (encryption/decryption) is widely available. This architecture's primary flaw is that it is subject to Trojan horse hacks and covert channels.



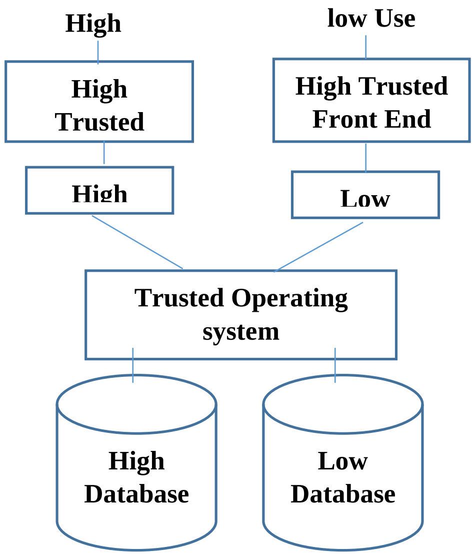


Figure 1: Integrity Lock Architecture Figure 2: Kernelized Architecture

The kernalized architecture [5], as illustrated in figure.2, works by breaking down a multilevel database into single-level databases that are kept separately and are controlled by a security kernel that enforces a required access control policy. The trusted front end ensures that user queries are sent to the DBMS with the same security level as the user’s, while the trusted backend ensures that a DBMS accesses data without violating the mandatory security policy at a specified level (Bell-LaPadula restrictions). Because different levels of data are stored independently, processing a user's query that accesses data from multiple security levels involves expensive joins that may reduce speed. However, because this architecture uses different DBMSs for each security level, the scheduler in responsibility of concurrency control can be separated for each level as well.

The third architecture known as the replicated architecture [6], as shown in figure. 3, employs a physically distinct backend database management system for each security level. Each backend database stores information at a given security level as well as all data from lower levels. The system's security is ensured via a trusted front end that allows users to access just the backend database system that corresponds to their security level.

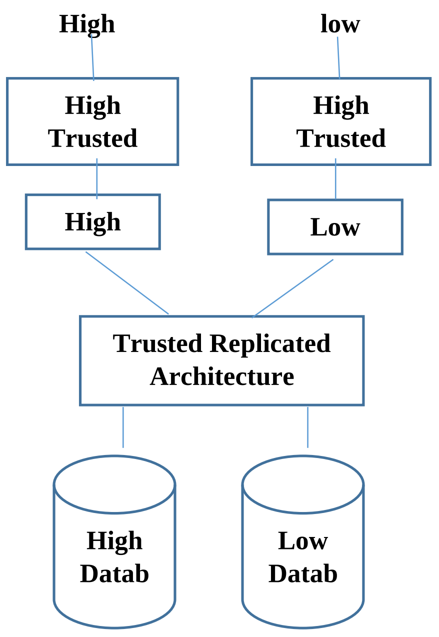


Figure 3: Replicated Architecture

If a high transaction needs to read data from a low level, it will be provided the high container's replica of the low level data. As a result, for a high number of security levels, this architecture is impracticable. Although query processing is not as expensive as in the kernalizied architecture, the essential issue of transaction in this architecture is the secure and proper propagation of lower level data updates to high level DBMSs.

5. Proposed Secure Multiversion Concurrency Control Algorithm

5.1. Background

Kim et al. [7] suggested an approach for safe multiversion concurrency control that is free of starvation. For the correctness of their proposed algorithm, they addressed the following criteria.

1. No possibility for creating a covert channel: Low transaction must not be delayed or rejected while a high transaction is executing, because this can create a covert channel.
2. No starvation of high transactions: Even though low transactions must be executed while high transactions are running, high transactions should not suffer from repeated aborts or re-executions.
3. Serializable schedule for multilevel transactions: In order to maintain the consistent state of the database, a serializable schedule must be produced for multilevel transactions.
4. No retrieval anomaly when multiversion data is employed: High transactions must read consistent versions of data objects in spite that there are new versions created by low transactions while high transactions are suspended.
5. Fast version selection among multiple versions when multiversion data is employed: Since there are a multiple versions of data objects in the database and some of them must not be accessed by certain transactions, the version selection must be fast for performance reason.

They also provided definitions for their algorithm.

Definition 1: A read set (or a write set) of a transaction Ti, denoted as R-setTi (or W-setTi), consists of the data objects that will be read (or written) by Ti. When Ti is submitted to the scheduler, the scheduler receives R-setTi and W-setTi.

Definition 2: An invisible area to a high transaction Tj is an interval from the time when Tj is blocked by the execution of another transaction Ti to the time when Tj resumes its execution. The purpose of defining the invisible area is to prevent Tj from reading new versions created by Ti running within this area; otherwise Tj may suffer from a retrieval anomaly when it resumes. Ti can be the transactions whose security level is either lower than or equal to Tj.

Definition 3: A conflict transaction set of a transaction Tj, denoted as C-setTj, consists of the transactions that enter an invisible area of Tj due to conflicts with Tj. Suppose Ti is submitted to the scheduler with its read and write sets. If R-setTj ∩ W-setTi is not empty, then Ti Є C-setTj,.

Definition 4: To a version created by Ti during the invisible area of Tj, the transaction identifier (TID) of Ti is attached. When Tj terminates, the TID of Ti is detached from that version. They used the term "t-lock" for the attachment and detachment of a TID to and from a version created during an invisible area of some transaction. That is, setting (or releasing) a t-lock to (or from) a version means that a TID that creates that version is attached to (or detached from) that version. Note that a t-lock is actually a version-creator indicator, and it itself is the TID. When Ti enters an invisible area of Tj, its TID is contained in the C-setTj, of the blocked transaction Tj. Therefore, versions with t-locks must be invisible to Tj if C-setTj contains those t-locks, i.e., TIDs.

The algorithm was made up of six steps:

Step 1 The scheduler receives R-setTi and W-setTi, when Ti is submitted.

Step 2 When there is no transaction in execution, the scheduler executes Ti. When Ti commits, the scheduler performs Step 6.

Step 3 When there is a transaction Tj that is currently in execution, the scheduler has three cases according to the security levels of engaging transactions:

Case 1: L(Ti) >L(Tj) . Step 4 is performed.

Case 2: L(Ti) = L(Tj). Step 5 is performed.

Case 3: L(Ti) <L(Tj). If R-setTj ∩W-setTi≠ ϕ, then the scheduler blocks Tj and lets Ti enter an invisible area of Tj for the whole duration of Ti's execution. The scheduler executes the operations of Ti immediately such that a covert channel should not be created. When Ti commits, Step 6 is performed.

Step 4: Since a low transaction Tj has been running, the scheduler makes the high transaction Ti wait until Tj terminates. When Tj commits, the scheduler performs Step 6.

Step 5: Because they are in the same security level, the scheduler executes them concurrently. If R-setTj ∩W-setTi≠ϕ, then while Tj is running, the scheduler lets Ti enter an invisible area of Tj only for the duration of creating new versions of conflicted data objects. When each of Ti and Tj commits, Step 6 is performed.

Step 6: When Ti is the only transaction executed at Step 2, the scheduler waits for other transactions to be submitted. Otherwise, the scheduler selects and executes those transactions, say Tk, whose security levels are the lowest among the blocked transactions, along with new transactions in the same security level as Tk submitted while Tk has been blocked. Those t-locks set on the new versions created in the invisible areas of Tk must not be released until Tk commits. When Tk terminates, the scheduler releases all the t-locks by obtaining information from C-setTk. The old versions are discarded at this point, if they are not accessed by any other transaction.

The algorithm's primary problems are decreased concurrency, transaction blocking, and the probability of starvation.

5.2. ESMVCC : The Proposed Algorithm

Here, the condition for adding a transaction to the conflict transaction set C-setTj is modified, a counter is attached with every transaction to count how many times the transaction has been already blocked. If it is blocked less than 5 times then it can be blocked and let the new transaction to execute in an invisible area. The transaction will, then, be executed when this counter crosses the maximum limit and the lower transactions will have to wait.

This modification improves concurrency and decreases the blocking time of high level transactions which leads to the improvement of the transactions' response time.

Example I: Suppose that Tj is executing when Ti is submitted and L(Ti) < L(Tj). As at the time of arrival of Ti condition (i) is not satisfied, Ti is added into C-setTj. Both Tj and Ti execute concurrently and Tj reads y1 written by Ti.

Time	1	2	3	4	5	6	7	8
Tj:	r[x0]					r[y1]	w[z1]	C
Ti :		r[x0]	r[y0]	w[y1]	C			

The following is a modified version of the algorithm given in [32]:

Algorithm

Step 1: The scheduler receives R-setTi and W-setTi when Ti is submitted.

Step 2: When there is no transaction in execution, the scheduler executes Ti. When Ti commits, go to Step 6.

Step 3: When there is a transaction Tj that is currently in execution, the scheduler has three cases according to the security levels of engaging transactions:

Case 1: L(Ti) > L(Tj). Go to Step 4.

Case 2: L(Ti) = L(Tj). Go to Step 5.

Case 3: L(Ti) <L(Tj). If R-setTj ∩W-setTi≠ ϕ, then if Bcounter(Tj) <5 then the scheduler blocks Tj and let Ti enter an invisible area of Tj for the whole duration of Ti's execution. Bcounter(Tj)++, The scheduler executes the operations of Ti immediately such that a covert channel should not be created. When Ti commits, Step 6 is performed.

Step 4: Since a low transaction Tj has been running, the scheduler makes the high transaction Ti wait until Tj terminates. When Tj commits, go to Step 6.

Step 5: Because both transactions are in the same security level, the scheduler executes them concurrently if R-setTj ∩W-setTi≠ ϕ, However if (i) is satisfied the new version created by Ti are t-locked and not visible to Tj. When each of Ti and Tj commits, go to Step 6.

Step 6: When T is the only transaction executed at Step 2, the scheduler waits for other transactions to be submitted. Otherwise, the scheduler selects and executes those transactions, say Tk, whose security levels are the lowest among the blocked transactions, along with new transactions in the same security level as Tk submitted while Tk has been blocked. Those t-locks set on the new versions created in the invisible areas of Tk must not be released until Tk commits. When Tk terminates, the scheduler releases all the t-locks by obtaining information from C-setTk. And the old versions are discarded at this point, if they are not accessed by any other transaction.

6. Performance Evaluation

Simulation is used to evaluate the performance of the proposed secure concurrency control algorithm and SMVCC [7] at two levels of security (high and low). The workload and system model used in our simulations, as well as the metrics used to access the performance of secure concurrency control algorithms, are described in this section. Then the results are described.

The system in this model consists of a shared-memory multiprocessor DBMS operation on disk-resident data. The database is represented as a collection of pages that are evenly distributed across all disks. A Poisson stream generates transactions, and each transaction has a security clearance level connected with it. A transaction consists of a sequence of page read and write operations. A read access begins with a concurrency control request to get access permission, then a disc I/O to read the page, and then a period of CPU use to process the page. Except for disc I/O, which is delayed until the transaction commits, write requests are handled similarly. I will assume the DBS has enough buffer space to save updates until commit time. When a transaction is resumed due to a data conflict, it uses the same data access pattern as before. The workload generation process and the hardware resource configuration are described in the next two subsections.

6.1. Workload Model

The workload model categorizes transactions based on their level of security clearance and data access. The key parameters of the workload model are summarized in Table 1. The RoArri parameter determines the transaction arrival rate. Any of the LClear security clearance levels is as likely to apply to a transaction.

Table 1: Workload Parameters

Symbol	Description	Value
RoArri	Rate of the transaction arrival	Varies (0- 100)
LClear	Number of clearance Levels	2
STrans	size of average transaction	10
PWrite	Write probability	30

A normal distribution with mean STrans determines the number of pages that are accessed by a transaction, while the actual pages to be accessed are determined uniformly from the database. Every transaction is able to only access data from a specific part of the database for security reasons. The database is divided into ClassLevels security classification levels in an even distribution. Bell-LaPadula standards are carefully followed when granting transaction access to data pages. A transaction, in other words, cannot read pages categorized higher than its clearance level. Furthermore, a transaction is not allowed to write to pages with security classifications lower than or higher than its own clearance level. The chance that a transaction operation is a write is determined by the PWrite parameter.

6.2. System Model

Multiple CPUs and disks are used in the database system. The CPUs share a single queue, but the disk each have their own queue.

The system's resource parameters are shown in Table 2 The DBSize parameter specifies the database's number of pages. The Pr.No and Dsk.No parameters define the number of hard disks, respectively, whereas the Cpu.PT and Dsk.PT parameters represent the CPU and disc processing times per data page. The number of classification levels in the system is specified by the L.class parameter.

Table 2: System Parameters

Symbol	Description	Value
DBSize	Number of pages in the database	500
Pr.No	Number of processors	2
Dsk.No	Number of disks	4
Cpu.PT	CPU time for processing a data page	12ms
Dsk.PT	Disk service time for a page	35ms
L.class	Number of classification levels in the system	2

6.3. Performance Metrics

The average response times of transactions at each security level for different arrival rates are the key performance metric used in this simulation experiments. The transaction's response time is the time from the beginning of a transaction to its commitment, regardless of how many times it has been aborted and resumed.

6.4. Results and Discussion

We simulated the proposed algorithm using C#. A database was created using MSSQL. The simulation is run with the same parameters for six different random number seeds in each experiment. At least 2,000 transactions of each security level has committed for each simulation ran terminated. The figures represent the average of four runs. The confidence intervals for all of the results in this paper are 95 percent. The workload and system parameter settings in the simulation experiments are set so that there is significant data and resource contention in the system, which helps to bring out the differences. Tables 1 and 2 show the parameter settings used in the experiments.

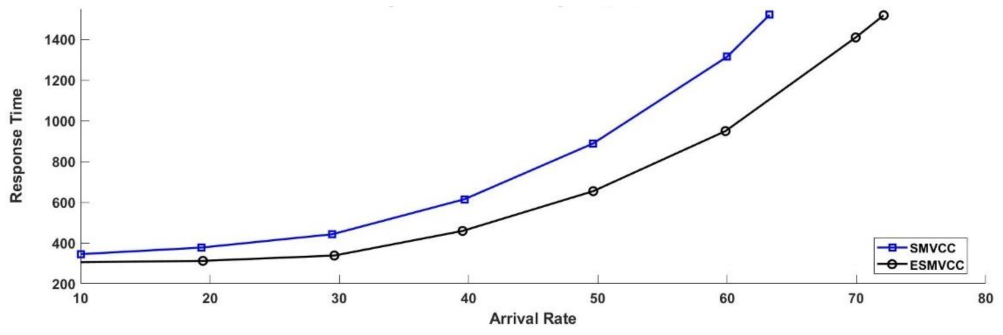


Figure 3 Overall Average response times

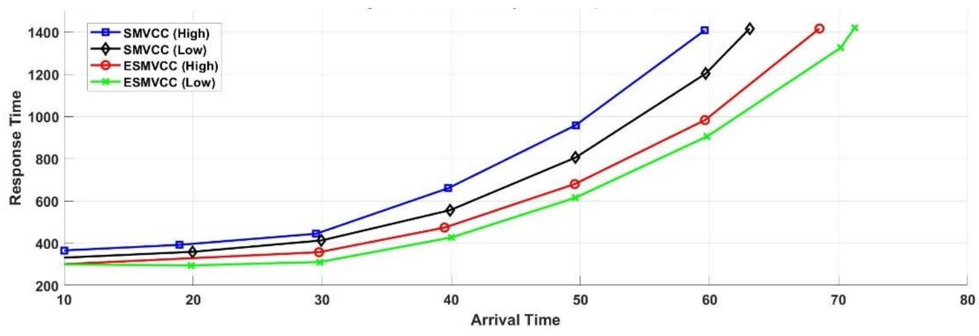


Figure 4 Pre-Security level Response Time

For different arrival rates, the overall response times of proposed (ESMVCC) and SMVCC [7] protocols are measured in Figure 3.

The difference in response times of both concurrency control protocols is less at low arrival rates, as seen in this figure. This is due to low contention levels, with the majority of time spent on disk and CPU access instead of resource queues, lock queues, or transaction aborts. As the arrival rates increase, the impact of these factors goes up as well, and performance differs depending on how much they increase in each concurrency control protocol. As the rate of arrival increases, the contention level for data items increases and the difference in response times of both protocols is increases. However, the response time of ESMVCC protocol is always less than SMVCC protocol. This is due to the fact that in the SMVCC protocol, more high security level transactions are blocked in order to give low security level transactions a higher priority than in the proposed algorithm.

Response times at each security level for various arrival rates can be used to evaluate the fairness of both systems. Figure 4 shows the resulting graph. The proposed algorithm has a low performance margin at both security levels (high and low), indicating a better measure of fairness than SMVCC.

7. Conclusion

Despite efforts by researchers to make traditional concurrency control protocols more secure, existing protocols suffer from starvation of high security level transactions. Here, a new secure multiversion concurrency control protocol is proposed and evaluated in this paper.

Comparing the proposed concurrency control algorithm to the SMVCC concurrency control protocol, these experiments show that the proposed algorithm performs better in terms of overall average response time. Furthermore, among secure concurrency control protocols, ESMVCC provides the lowest overall average transaction response time and the best level of fairness.

References

[1] N. Dobrinkova, "Information Security-Bell-LaPadula Model. Institute of Information and Communication Technologies." Sofia, 2010.

[2] N. Kaur, R. Singh, and H. K. Sidhu, "Secure concurrency control algorithm for multilevel secure distributed database systems," *ICEIS 2005 - Proceedings of the 7th International Conference on Enterprise Information Systems*, pp. 267–272, 2005, doi: 10.5220/0002516002670272.

[3] M. D. Abrams, S. G. Jajodia, and H. J. Podell, *Information security: an integrated collection of essays*. IEEE computer society press, 1995.

[4] D. E. Denning, "Commutative filters for reducing inference threats in multilevel database systems," in *1985 IEEE Symposium on Security and Privacy*, 1985, p. 134.

[5] D. E. D. T. F. L. R. R. Schell, W. R. Shockley, and M. Heckman, "The Sea View Security Model," *IEEE Trans. Software Eng*, vol. 16, no. 6, pp. 593–607, 1990.

[6] J. N. Froscher and C. A. Meadows, "Achieving a Trusted Database Management System Using Parallelism.," in *DBSec*, 1988, pp. 151–160.

[7] H. T. Kim and M. H. Kim, "Starvation-free secure multiversion concurrency control," *Information Processing Letters*, vol. 65, no. 5, pp. 247–253, 1998, doi: 10.1016/s0020-0190(98)00014-3.