Detection and Resolution of Deadlocks in Multi-Level Secure Databases

Pooja Sapra [#1], Suresh Kumar [*2], RK Rathy [#3]

[1]Research Scholar, Manav Rachna International University
Faridabad, India
[2-3]Faculty of Engineering and Technology, Manav Rachna International University
Faridabad, India
[1] mrs.sapra@gmail.com
[2] enthuvs@gmail.com
[3] rkrathy.fet@mriu.edu.in

***Abstract*—The increase in the sharing of databases leads to the increase in the concurrency errors, which are termed as deadlocks. Many concurrency control techniques result in deadlocks, because these techniques require the transactions in the system to wait for one another. At this situation, an outside intervention is often required to resume the normal working. Therefore, some special procedures are needed to resolve the deadlock situation. A multilevel secure database protects the classified information from unauthorized users based on the classification of the data and clearance of the users. If we add multilevel security to databases then the system becomes more complicated and it becomes mandatory for us to resolve the problem like deadlock and concurrency. In these systems, there should be no covert channel and starvation during the coordination of transaction in secure databases. Here we present an algorithm for deadlock detection and recovery in multilevel secure database. The proposed algorithm takes into consideration the requirements and terminates the youngest transaction possible according to the security classes available.**

Keyword-Multilevel security, deadlock detection and recovery, transaction, wait-for-graph.

I. INTRODUCTION

In databases, a deadlock is a state, in which transactions are endlessly waiting for one another [1] and are unable to proceed. To depict the deadlock situation diagrammatically, the directed graphs known as wait-for-graphs are used. These graphs indicate which transactions are waiting for the resources held indefinitely by which transaction. In a wait-for-graph, nodes of the graph represent transactions and edges represent the relationships between transactions. A direct edge is drawn in the wait-for-graph from one transaction to another, if the first transaction is waiting for a resource that is currently held by the second transaction. If the Wait-For-Graph contains a cycle then the system may be in a deadlock state. After detection of a deadlock situation in the system, one of the transactions involved is chosen as the victim transaction and is aborted to resolve the deadlock situation.

In multilevel secure databases (MLS), there are additional requirements for concurrency control. Traditional algorithms, based on locks or timestamps, if applied on these databases are not secure and suffer from starvation. Therefore, a multi version protocol is used to overcome the problems. In this multiple versions of each data item are maintained. Therefore, a *low secure* transaction is never delayed or aborted because of the concurrent execution of a *high* transaction; thus, both signalling channels and starvation are eliminated. The paper is organized as follows: Section 2 highlights the main points on multilevel secure databases, section 3 considers the main issues in deadlock detection and recovery in multilevel secure databases, section 4 and 5 includes the proposed algorithm and section 6 concludes the paper.

II. MULTILEVEL SECURITY

Multilevel Security (MLS) [2] allows the information with different classifications to be available with users having different security clearances and authorizations and at the same time disallowing them from accessing information for which they are not cleared or authorized. In mandatory security models, the subjects and objects are assigned security levels termed as labels. Label of object is called its classification class (o) and for a subject is called its clearance, clear(s).

Security label consists of two components:

- Hierarchical list of sensitivity levels, for e.g. Top-secret > secret > confidential > unclassified
- Non-hierarchical set of categories, which represents classes of objects.

MLS imposes the following two restrictions on all data accesses:

• The Simple Security Property or “No Read Up”: A subject is allowed a read access to an object if and only if the subject’s label dominates the object’s label.

• The *-Property (pronounced the star property) or “No Write Down”: A subject is allowed a write access to an object if and only if the object’s label dominates the subject’s label.

The restrictions given by this model has a drawback of covert channel. Covert channels are the channels through which malicious user can receive any information about the data that is classified beyond the user's clearance.

III. LITERATURE REVIEW

Several approaches have been proposed for centralized secure concurrency control in MLS/DBMSs. These are the extensions of the 2PL protocol or time-stamp based protocol. However none is free from deadlocks and problems such as covert channel, too much delay or repeated aborts of high security level transactions, and retrieval anomaly [3].

Jajodia and McCollum [4], have given a secure locking-based protocol called S2PL which was modified version of strict two phases locking protocol to covert channel free protocol. In this protocol, when a low security level transaction requests a write lock on a data item, then a high security level transaction must release its lock on same data item. This protocol satisfies the requirements of covert channel free and integrity, but results in starvation.

McDermott and Jajodia [5] discussed a protocol to reduce the amount of starvation. According to this protocol, high level transaction does not entirely aborts itself and roll backs but holds its write locks on high security level data items. This approach, fails to produce the serializable schedules [6].Before submitting your final paper, check that the format conforms to this template. Specifically, check the appearance of the title and author block, the appearance of section headings, document margins, column width, column spacing and other features.

Son and David [7], discussed another secure two-phase locking-based protocol (S2PL), based on the concept of virtual locks, which is used by low security level transactions. The virtual locks are implemented on private versions of the data item and are upgraded to a real lock, when the high security level transaction commits and releases the data item. To make the schedule serializable, this protocol requires maintaining virtual locks.

E. Bertino et al. [8] introduced an approach that uses single- version data items and is based on the use of nested transactions, application-level recovery, and notification-based locking protocols. The notification protocol is based on the use of signal locks. A signal lock is acquired by a transaction whenever it needs to read lower security level data; such a lock does not delay a write lock request by a low security level transaction on the same data item. Hence, timing covert channels arising from synchronization are eliminated. When a data item on which a write lock is acquired by a transaction is modified, all high security level transactions holding signal locks on that data are notified by the trusted lock manager, and thus may perform recovery actions. To better support recovery activity, transactions are organized according to the nested transaction model extended with specific primitives for supporting the notification protocol. The proposed approach satisfies most of the properties pointed out in Atluri et al. [5], as basic requirements for a secure concurrency control mechanism in a multilevel environment: it avoids starvation and timing channels, and guarantees serializability.

IV. PROPOSED ALGORITHM FOR DEADLOCK DETECTION AND RESOLUTION IN MULTILEVEL SECURE DATABASES

For the detection and recovery of deadlocks the proposed algorithm uses the following data structures:

LTSi: local transaction structure for each site i;

DTSi: distributed transaction structure;

SLDCi: sensitivity level local deadlock cycle at each site i;

SGDCi: sensitivity level global deadlock cycle;

LDi: local deadlock cycle;

GDCi: global deadlock cycle;

TQi: transaction queue.

Let us consider two sensitivity levels for the transactions: low (l) and high (h), such that l<h. Further the requests can be made by:

1. Low level secure transactions for the objects held by highly secured transactions.
2. Low level secure transactions for the objects held by less secured transactions.
3. High level secure transactions for the objects held by highly secured transactions.
4. High level secure transactions for the objects held by less secured transactions.

In case 1, the transaction with high security level has to be aborted, so as to avoid the covert channel. In cases 2, 3 and 4, the normal execution of algorithm [16] will be carried out.

For finding out the transaction to be aborted, sensitivity level deadlock cycles are generated, at each local site (SLDCi) and global sites (SGDCi) corresponding to distributed transaction structure.

Algorithm deadlock_detection_and_recovery:

1. Create Linear transaction Structure (LTSi) for each local site i.
2. Detect Local Deadlock cycle LDi.
3. Create SLDCi.
4. If SLDCi contains an edge (l -> h), then abort the transaction with security level high (h).Otherwise, Create Transaction Queue TQi corresponding to each LDi and Abort the youngest transaction.
5. Create Distributed Transaction Structure (DTSi) for global communication.
6. Detect Global Deadlock cycle GDCi.
7. Create SGDCi.
8. If SGDCi contains an edge (l -> h), then abort the transaction with security level high (h).Otherwise, Create Transaction Queue TQi corresponding to each GDCi and Abort the youngest.

V. ILLUSTRATIONS

Let us consider transactions T1, T2, T3…..T7 that are depicted by the Wait-For-Graph in fig.1.

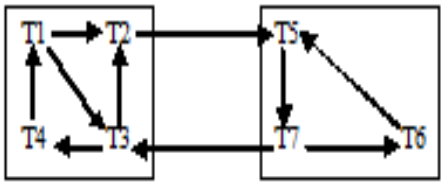

FIG. 1: WAIT FOR GRAPH

A. Illustration 1

The transactions T1, T2, T7 are classified at high (h) level and all others are the less secure transactions. The sensitivity level table is shown in Table I.

TABLE I
Sensitivity Level Table for Transactions

Transaction	T1	T2	T3	T4	T5	T6	T7
Sensitivity level	h	H	l	l	l	l	h

- Create Linear transaction Structure (LTSi) for each local site i.

LTS1:

P	Q
1	2
1	3
3	2
3	4
4	1

LTS2:

P	Q
5	7
7	6
6	5

- Detect Local Deadlock cycle LDi.
  LD1: {1→3, 3→4, 4→1}
  LD2: {5→7, 7→6, 6→5}
- Create SLDCi:
  SLDC1: {h→l, l→l, l→h}
  SLDC2: {l→h, h→l, l→l}
- Abort the victim transactions T4 and T5.
- Create Distributed Transaction Structure (DTSi) for global communication.
  DTS1:

p	Q
2	5
5	7
7	3
3	2

- Detect Global Deadlock cycle GDi.
  GDC1: {2→5, 5→7, 7→3, 3→2}
- Create SGDCi:
  SGDC1: {h→l, l→h, h→l, l→h}
- Abort the transaction T3.
  Total no. of transactions aborted: 3

B. Illustration 2

Now let us consider that the transactions T1, T2, T7 are classified at unclassified (u), which is the least secure level and T3,T4 are at secret level (s), T5 is assumed to be at top secret level (ts) and T6 be at confidential (c) level. The levels are in the ordering of u < c < s < ts. The sensitivity level table is shown in Table II

TABLE III
Sensitivity Level Table for Transactions

Transaction	T1	T2	T3	T4	T5	T6	T7
Sensitivity level	u	u	s	s	ts	c	u

- Create Linear transaction Structure (LTSi) for each local site i.

LTS1:

P	Q
1	2
1	3
3	2
3	4
4	1

LTS2:

P	Q
5	7
7	6
6	5

- Detect Local Deadlock cycle LDi.
  LD1: {1→3, 3→4, 4→1}
  LD2: {5→7, 7→6, 6→5}
- Create SLDCi:
  SLDC1: {u→s, s →s, s→u}
  SLDC2: {ts→u, u→c, c→ts}
- Abort the victim transactions T1 and T7.
- Create Distributed Transaction Structure (DTSi) for global communication.
  DTS1:

p	Q
2	5
5	7
7	3
3	2

- Detect Global Deadlock cycle GDi.
  GDC1: {2→5, 5→7, 7→3, 3→2}
- Create SGDCi:
  SGDC1: {u→ts, ts→u, u→s, s→u}
- Abort the transaction T2.
  Total no. of transactions aborted: 3

C. General Case

Next, let us take a situation where the total number of sites are S and N the total number of transaction concurrently executing on these sites.

Let N1, N2, N3…….Ns be the total number of transactions at sites S1, S2, S3……..Ss.

Therefore, N1+N2+N3….Ns=N

Let there be four classification levels l1, l2, l3, l4; such that l1<l2<l3<l4.

At any site Si,

Number of transactions with security level l1 = m1

Number of transactions with security level l2 = m2

Number of transactions with security level l3 = m3

Number of transactions with security level l4 = m4

Therefore,

For site S1, m1+m2+m3+m4=N1.

For site S2, m1+m2+m3+m4= N2…….. and so on…

Total number of operations required to detect the deadlock and recover = S + 2LD +S(S-1)(1+3GD)/2

Where, LD is the total number of local deadlock cycles in the Wait-For-graph and GD be the total number of global deadlock cycles.

D. Proof

Number of operations required for:

Step number 1= S,

Step number 2= d1+d2+…+ds= LD, where d1, d2,…..ds are the number of deadlocks cycles at site S1, S2,…..Ss.

Step number 3= LD.

Step number 4= LD

Step number 5= S*(S-1)/2

Step number 6= (G1+G2+……..GS)*S*(S-1)/2 = GD*S*(S-1)/2

Step number 7 = GD*S*(S-1)/2

Step number 8= GD*S*(S-1)/2

Therefore Total Number of Operations= S + 2LD +S*(S-1)*(1+3GD)/2.

As LD and GD depends on the total number of transactions linearly so LD=GD=cN, Where c is any constant.

So, from the calculated result of total number of operations we can conclude that deadlock detection and recovery process has the complexity $\Theta(NS^2)$.

E. Performance

In the given illustrations, initially we have taken two security levels; in this case the number of transactions aborted is 3. If we increase the security levels to four then also the number of transactions aborted is 3. In general we can conclude that number of transactions aborted is directly proportional to the number of transactions and number of sites rather than the number of security levels.

VI. CONCLUSION

Adding multilevel security to databases makes the system more complicated and it becomes difficult to resolve the problems like deadlock and concurrency. The deadlock detection and resolution in these databases has different requirements as that of the unsecure databases. There should be no covert channel and starvation during the coordination of transaction in secure databases. The proposed technique is an initiative taken in the direction of deadlock detection in multilevel secure databases.

Here if we increase the number of security levels then also the number of transactions aborted remains the same with some extra overheads of maintaining the more security levels. The elimination of covert channel is achieved through the technique. In future we can extend the solution to achieve the starvation free execution of transactions.

REFERENCES

[1] A.K. Elmagarmid, "A Survey of Distributed Deadlock Detection Algorithms," *SIGMOD RECORD*, vol. 15: 3, pp. 37-45, 1986.

[2] Dawyer P., Jelatis G. D. and Thuraisingham B., "Multilevel Security in Database Management Systems", *Computer and Security*, vol. 6, no. 3, pp. 252-260, June 1987.

[3] S. Jajodia and V. Atluri, "Alternative correctness criteria for concurrent execution of transactions in multilevel secure databases," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 839-854,Oakland, California, 1992.

[4] S. Jajodia and C. McCollum, "Using two-phase commit for crash recovery for federated multilevel secure database management systems," *Dependable Computing and Fault Tolerant Systems*, vol. 8, pp. 365-381, New York, Springer-Verlag, 1993.

[5] J. McDermott and S. Jajodia, “Orange locking: Channel free database concurrency control via locking," *Database Security, VI: Status and Prospects, Database Security*, pp. 267-284, 1995.
[6] S. Jajodia, L. V. Mancini, and I. Ray, “Secure locking protocol for multilevel database management systems," *Proceedings of the Annual IFIP WG 11.3 Conference of Database Security*, pp. 177-194, 1995.
[7] S. H. Son and R. David, “Design and analysis of a secure two-phase locking protocol," *18th International Computer Software and Applications Conference(COMPSAC'94),* pp. 374-379, IEEE Computer Society Press, 1994.
[8] E. Bertino, B. Catania, And E. Ferrari, “ A nested transaction model for multilevel secure database management systems," *ACM Transactions on Information and System Security*, vol. 4, no. 4, pp. 321370, Nov. 2001.
[9] Bertino E. and Sandhu R., “Database Security-Concepts, Approaches, and Challenges”, *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no.1, pp. 2-19, 2005.
[10] S. Jajodia and B. Kogan, “Concurrency control in multilevel secure databases based on a replicated architecture," *Proceedings of the llth IEEE Symposium on Security and Privacy*, pp. 360-368, Oakland, CA, Apr. 1990.
[11] T. F. Keefe and W. T. Tsai, “Multiversion concurrency control for multilevel secure database systems," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 369-383, Oakland, California, 1990.
[12] T. Keefe, W. Tsai, and J. Srivastava, “Multilevel secure database concurrency control," *Proceedings of IEEE International Conference on Data Engineering*, pp. 337-344, Feb. 1990.
[13] T. F. Keefe, W. T. Tsai, and J. Srivastava, “Database concurrency control in multilevel secure database management systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 1039-1055, 1993.
[14] H. T. Kim and M. H. Kim, “Starvation-free secure multi version concurrency control," *Information Processing Letters*, vol. 65, pp. 247-253 pp. 247-253, 1998.
[15] Imran S., Hyder I.,”Security Issues in Databases”, *Proc. of Second International Conference on Future Information Technology and Management Engineering*, pp. 541-545, 2009.
[16] Alom B.M.M., Henskens F., Hannaford M., “Deadlock Detection Views of Distributed Database”, *Proc. of sixth International Conference on Information Technology: New Generations*, pp. 730-737, 2009.
[17] Bertino E. and Sandhu R., “Database Security-Concepts, Approaches, and Challenges”, *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no.1, pp. 2-19, 2005.
[18] Bell D. and La Padula L., “Secure Computer Systems: Unified Exposition and Multics Interpretation”, Technical Report NTIS AD-A023588. Bedford, Mass.: MITRE Corporation, July 1975.
[19] Kaur N., Saini H. S., Singh R., “Design And Analysis Of Secure Scheduler For MLS Distributed Database Systems”, *Proc. of International* Advance Computing Conference, pp. 1400 – 1404, March 2009.