Efficient Damage Assessment and Recovery Using Fast Mapping

Ramzi A. Haraty
Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon 1102-2801
Email: rharaty@lau.edu.lb

Hussein Mohsen
School of Informatics and Computing
Indiana University
Bloomington, Indiana, USA
Email: hmohsen@indiana.edu

Abstract

As electronic attacks increase with the advancement of World Wide Web applications, protecting information becomes an essential concern. Clustering and sub-clustering log files have been introduced to save time and prevent searching the log in case any malicious transaction is detected and recovery is needed. In this paper, we introduce new enhanced damage assessment and recovery algorithms with auxiliary data structures and fast mapping for efficient retrieval of information.

Keywords: Recovery, Damage Assessment, Transaction Systems, Data Dependency.

1. Introduction

The information that any organization owns have become a very important factor that it needs to take care of and to protect. As this importance increases, information warfare rages. For this reason, new methods to achieve security and correctness of information have become indispensable as the traditional security and recovery means are still not enough.

Data are retrieved and manipulated by transactions, and these transactions can be highly dependent on each other regarding the data values they read and write. As a result, whenever a malicious transaction is detected, its effects have to be rolled back and the database has to be brought back to a consistent state. For this purpose, damage assessment and recovery are needed to determine affected transactions and data items and to restore valid values while taking the importance of execution time and memory usage into consideration.

In this work, we present an algorithm that segments the log into clusters based on exact data dependency [1]. The algorithm segments every cluster to a number of sub-clusters according to two different approaches: number of data items or space occupied. We also present two other algorithms for damage assessment and recovery is introduced as well. This work adds improvements on and introduces new approaches with respect to the work in [2].

The aim of the work is to minimize execution time while maintaining good results. The rest of the paper is organized as follows. Section 2 contains a literature review. Section 3 overviews the log sub-clustering method. Section 4 introduces the damage assessment algorithm, and Section 5 introduces the recovery algorithm. Section 6 provides the conclusion.

2. Literature Review

Different researchers have introduced their approaches regarding solutions of malicious intrusions. In [3], Giffin and Strivastava describe a preliminary strategy that allows the system to automatically recover from unknown malicious intrusions. Fu and Li in [4] introduce new concepts of damage assessment that deal with different kinds of dependencies (phantom dependency, domain integrity dependency, etc.). They describe their robust damage assessment approach and give examples to show how it works. Both [5] and [6] address information warfare using virtual machines. In [5], Zhang *et al.* write about their recovery strategy, which is most concerned about availability and continuity of the system. They describe a special virtual machine architecture that prevents the system from failing while ensuring recovery of data after malicious intrusions. In [6], Zia *et al.* describe a strategy that uses virtual machines that help in cross-layered damage assessment. Both damage assessment and recovery were under study in the work of Bai and Liu in [7] and [8]. In [7], they describe a light-weight solution called DTQR (Data Tracking, Quarantine and Recovery). They define their own scheme and show experimental results. In [8], the same approach is used for online data systems. Adding auxiliary data structures to help in damage assessment and recovery aims to minimize I/O. Matrices are used in [9] that helps in saving time as accessing data items and their values becomes fast. The authors of [10] and [11] suggest isolation techniques that will eliminate the effect of such intrusions. In [10], Balitanas and Kim present an isolation protocol that defends against any attack, while in [11], Kotla *et al.* present a durable and practical data system called SafeStore. In [12], Xin introduces a new transaction processing model for advanced and complex transactions with the ability of overcoming system crashes and

978-1-880843-96-3 / copyright ISCA, SEDE 2014
October 13-15, New Orleans, Louisiana, USA

malicious attacks.

In addition to all the approaches described, many researchers introduce the concept of clustering the log based on data dependency so that the concerned clusters are accessed and reviewed during damage assessment and recovery. Ragothaman *et al.* in [13] introduce simplified versions of log clustering techniques. They enforce limitation on clusters to control their sizes. These limitations are the number of committed transactions, the size of clusters per lines in the log and the time window for the cluster. The authors of [14] aim to achieve faster damage assessment and recovery. They discuss a hybrid log segmentation strategy that assured a reliable damage assessment. In [15], Zhu *et al.* describe a fine grained transaction log used in recovery from malicious attacks. They then describe the recovery algorithm based on this kind of log. In [16], Haraty and Zeitunian present an approach where the size of the sub-cluster is the used criterion to divide the cluster. Dependent operations of committed transactions are added to the same sub-cluster. These are added until there is no more space for the next transaction to fit into it. This approach may require additional disk space but fastens damage assessment.

3. The Hybrid Sub-Clustering Algorithm

The algorithm we propose segments the log into clusters and further sub-clusters to reduce I/O and improve damage assessment and recovery.

3.1. Assumptions

The proposed algorithm is based on the following assumptions: First, we assume that the malicious transaction has been detected by a certain intrusion detection technique. Second, histories produced are rigorous serializable. Third, the clustered log stores the details of committed transactions only. Fourth, all details of active transactions are stored in a temporary log and then moved to the clustered log once these transactions commit. Fifth, we assume that the transaction ID is sequentially incremented which means that if T_2 commits then the only transaction committed before T_2 is T_1.

3.2. Transaction Model

The following definitions are essential to the understanding of the algorithms. Interested readers are referred to [2] for more details.

Definition 1: A write operation is dependent on a read operation if the write is computed using the value obtained from the read.

Definition 2: A data value v_1 is dependent on data value v_2 if the write operation that wrote v_2 was dependent on the read operation of v_2.

Definition 3: A write operation *w[x]* is dependent on a set of data items *I*, if *x=f(I)*; for example, the value written on *x* is calculated based (or dependent) on the values of the items in *I*.

There are three cases for the set *I*:

Case 1) *I* is empty. If the previous value of *x* (before the write operation) is damaged, it can be refreshed after the write operation.

Case 2) x != *I*. In case the previous value of *x* is damaged and none of the items in *I* is damaged, the value of *x* can be refreshed after the write operation.

Case 3) *x = I*. If the previous value of *x* is damaged, then *x* remains damaged. Otherwise, if any item in *I* is damaged, then *x* is damaged.

3.3.Sub-Clustering Algorithm

3.3.1. Data Structures

There are four data structures used in the sub-clustering model. They are as follows:

Transaction Sub Cluster List (TSC*):* It is used to store the Transaction ID and the corresponding sub-clusters in which the data items of the transaction are stored. This table is used to obtain the sub-clusters affected by a malicious transaction.

Sub-Cluster Data List (SCD): It is used to store clusters and sub-clusters IDs, the transaction IDs, the corresponding data items, and the operations. This table is used to identify the data item(s) affected or damaged and whose values must be recovered.

Responsible Transaction List (RTL): It is used to determine the transaction that is currently responsible for writing on every data item. On every write operation on a data item, the RTL is updated to indicate the transaction responsible for writing on that item at that time. This list is used to fill the Affected Transactions List (ATL) as discussed next. It also stores old value and timestamp of change) to fasten the recovery process.

Affected Transactions List (ATL): It determines the affected transactions by the writes done by every transaction. In other words, the transactions that read and used values written by the transaction. It also records the data items and the predicate(s)/block(s) in which the operations take place. This data structure is used to determine the affected predicates/blocks and transactions to be recovered. It stores timestamps of some operations but not in the same way as Basic Timestamp Ordering [17].

3.3.2. Hybrid Sub-clustering Algorithm Based on Fixed Number of Transactions

In this algorithm, whenever a read operation on any data item takes place, one or more of TSC, SCD, and ATL is/are updated. If the data item and the transaction reside on the same sub-cluster, the read operation takes place and the data structures and the log are updated. If they do not reside in the same sub-cluster, the log and the SCD are updated. If the operation is a write, RTL is updated as well to determine the transaction responsible for writing on the data item. The log, TSC, SCD and ATL are updated after the operation takes place. The detailed algorithm is shown in Figure 1.

4. The Damage Assessment Algorithm

When the malicious transaction(s) is (are) detected, the STC, SCD, and ATL are checked to determine the affected transactions, data items, predicates/blocks, and sub-clusters. Once these damaged units are identified, recovery becomes easier and faster since only these units will be traversed in the log and not the log as a whole. For example, consider the following transactions:

T1: B := A; T2: A := A+1;
T3: C := B; T4: D := C;
T5: E := D+2; T6: F := E;
T7: X := 1; T8: Y := X;
T9: Z := A;

The four obtained corresponding data structures (TSC, SCD, RTL, and ATL) after the execution of these transactions are shown in Tables 1, 2, 3, and 4, respectively. In the RTL, the responsible transaction is the one that last wrote in the data item. It helps in filling the ATL as described below. At every write on a data item, the RTL row corresponding to this data item is updated.

Concerning the ATL, for the second column titled (Aff. T, DI, Blk, ts), or (Affected Transaction, Data Item, Block, Timestamp), every set of values in a row states the affected transaction (which reads from the data item last written by the row's transaction in the RTL), the data item that was read, the block of code where the operation took place where it may be used in recovery, and finally the timestamp that records the time at which the operation took place. We assume that code blocks are numbered before the algorithm starts. For example, (3, B, 3, t5) in the first row of ATL means that T3 read the value written by T1 in B in block 3 at t5.

```
For every operation O_i of a committed transaction in the temporary log
  Case O_i is read
    If x_i ∈ Read_Set(Predicate T_i^jk)
          If x_i ∉ a cluster's subcluster (SCD)
          Assign new cluster ID and new sub-cluster (ID=1)
          Update corresponding data structures (SCD & TSC)
          Else If  sub_cluster.transaction_count ≤ MAX
          Get the cluster ID and sub-cluster ID
          Update corresponding data structures (SCD & TSC)
          Else If sub_cluster.transaction_count > MAX
          Assign new sub-cluster within same cluster
          Update corresponding data structures (SCD & TSC)
          Record predicate [pred-read, T_i^jk, x_i,  string] in log
          (cluster)

    If O_i ∈ Read_Set(Statement T_i^jk)
          If x_i ∉ a cluster's subcluster (SCD)
          Assign new cluster ID and new sub-cluster (ID=1)
          Update corresponding data structures (SCD & TSC)
          Else If sub_cluster.transaction_count ≤ MAX
          Get the cluster ID and sub-cluster ID
          Update corresponding data structures (SCD & TSC)
          Else If sub_cluster.transaction_count > MAX
          Assign new sub-cluster within same cluster
          Update corresponding data structures (SCD & TSC)
          Record operation [read-item, T_i^jk, x_i, value, string] in log
          Update corresponding data structures (SCD & ATL)

  Case O_i is write
    If O_i ∈ Write_Set(Statement T_i^jk)
          If x_i ∉ a cluster's subcluster & No dependency
          Assign new cluster ID and new sub-cluster (ID=1)
          Update corresponding data structures (SCD & TSC)
          Else If x_i ∈ a cluster's subcluster & No dependency
            If sub_cluster.transaction_count ≤ MAX
            Get the cluster ID and sub-cluster ID
            Update corresponding data structures (SCD & TSC)
            Else If sub_cluster.transaction_count > MAX
            Assign new sub-cluster within same cluster
            Update corresponding data structures (SCD & TSC)
            Else If Dependency
            Check all dependent reads

  For each different cluster MERGE
    Record op. [write-item, T_i^jk, x_i, new_value, old-value, string] in
          log
    Update corresponding data structures (SCD & ATL & RTL)
```

Figure 1. The sub-clustering algorithm

Table 1. The TSC

Transaction	Sub-Cluster Code
1	1
2	1
3	1
4	2
5	2
6	2
7	3
8	3
9	3

Table 2. The SCD

Sub-Cluster	Data Item	Transaction	Operation
1	A	1	R
1	B	1	W
1	A	2	R
1	A	2	W
1	B	3	R
1	C	3	W
2	C	4	R
2	D	4	W
2	D	5	R
2	E	5	W
2	E	6	R
2	F	6	W
3	X	7	W
3	X	8	R
3	Y	8	W
3	A	9	R
3	Z	9	W

Table 3. The RTL

Data Item	Responsible Transaction	Old Value	Timestamp of Change
A	2	3	t19
B	1	2	t22
C	3	5	t27
D	4	8	t29
E	5	7	t32
F	6	0	t36
X	7	7	t40
Y	8	4	t42
Z	9	2	t45

Table 4. The ATL

Transaction	(Aff. T, DI, Blk, ts)	(Writt. DI, Blk, ts)
1	(3, B, 3, t5)	(B, 1, t1)
2	(1, A, 1, t1) (2, A, 2, t3) (9, A, 9, t16)	(A, 2, t4)
3	(4, C, 4, t7)	(C, 3, t6)
4	(5, D, 5, t9)	(D, 4, t3)
5	(6, E, 6, t11)	(E, 5, t10)
6		(F, 6, t12)
7	(8, X, 8, t17)	(X, 7, t13)
8		(Y, 8, t15)
9		(Z, 9, t17)

For the third column in ATL, (Writt. DI, Blk, ts) or (Written Data Item, Block, Timestamp), it records the data item(s) that the transaction wrote on, in which block of code and at which time (used to determine the initial write by this transaction on that item, where this value is used in the recovery algorithm). For example, (B, 1, t1) in the first row of ATL means that T1 wrote on B in block 1 at t1. Now, after filling these data structures while sub-clustering and writing to the log, the damage assessment algorithm shown in Figure 2 is executed. It has three data structures: A queue of transactions used to track the (directly or indirectly) affected transactions from the detected malicious one(s), Damaged_DI which is a list of data items that are (directly or indirectly) damaged, and Damaged_PB which is a list that contains the damaged blocks that have to be recovered. Given that the three data structures do not contain duplicates, they contain each needed item once, since one pass of the recovery algorithm over every affected item will bring the database to a stable state.

```
Create queue q, Damaged_DI and Damaged_PB and initialize them to null
Add detected malicious transaction(s) T_i to q and its ID to Damaged_DI
Set Initial_Write_timestamp(x) = ts in col.3 of T_i

While (q not empty)
        T_temp = poll from q
        Loop over transactions T_aff affected by T_temp (frol col. 2 in ATL)
                If T_aff reads item x (2nd col. of ATL) & x written by
T_temp with                                     timestamp >
Initial_Write_timestamp (col. 3 of ATL)
                                        Add T_aff to q
                                        Add x and Trans. ID of T_aff to
Damaged_DI
```

Figure 2. The damage assessment algorithm.

In the algorithm, the data structures are firstly initialized to null. Then, the malicious transaction(s) is(are) added to the queue. After that, we traverse the queue. For every affected transaction obtained from the ATL, (if its read has been after the write of the malicious transaction), which may directly or indirectly read an incorrect value, the transaction ID and its data item(s) is(are) added to the Damaged_DI and the block(s) is(are) added to the Damaged_PB. Then, the affected transaction(s) is(are) added to the queue. While the queue is not empty, the loop keeps on iterating, since that means that more transactions have been directly or indirectly affected by the malicious transaction(s). If transaction T1 in the example scenario is detected as malicious, and after running the damage assessment algorithm, the data items that have to be recovered (found in Damaged_DI) are B, C, D, E, and F. The affected transactions are T1, T3, T4, T5, and T6 (found in Damaged_DI). The blocks to be recovered (found in Damaged_PB) are 1, 3, 4, 5, and 6. The corresponding sub-clusters are 1 and 2.

When T1 is detected as malicious, the algorithm first adds T1 to the queue, *q*, and its ID to Damaged_DI and

then goes to the row of T1 in ATL. It checks the second column cell of this row and recognizes that T3 is affected by T1. Since T3 read data item B in block 3 at t5 (as second column first row cell tells), which is after t1, when T1 first wrote B (initial write timestamp from third column), then T3 read a maliciously written value by T1, so the transaction ID of T3 and item B are added to Damaged_DI, ID of block 3 is added to Damaged_PB, and T3 is added to *q*. Then, transaction T3 is pulled from the queue, and the algorithm goes to its corresponding row in ATL. The same checks for affected transactions, blocks, and data items (in this case for T4, block 4, and data item C), depending on timestamps (in this case for t6 and t7) are applied and Damaged_PB, Damaged_DI, and *q* are filled accordingly. While the queue *q* is not empty, or in other words, while more transactions are being detected as affected by the malicious attack, these checks are repeated based on the ATL. The used *q* does not contain duplicates to avoid getting stuck in cycles.

5. The Recovery Algorithm

After Damaged_DI and Damaged_PB are filled by the damage assessment algorithm, they are used for recovery. The recovery algorithm is shown in Figure 3. The importance of sub-clustering appears in this recovery algorithm. Instead of scanning the whole log for a certain value of a data item, we scan the sub-cluster in which the item resides only. That improves the execution time. Also, an old value is saved on every new write so that it can be checked if it is valid (if the read operation took place before the malicious write) so that it can be used instead of looking to that value in the sub-cluster even. That saves time especially with using the *O(1)* mapping described in the next section. In the algorithm, the damaged blocks marked in Damaged_PB have to be re-evaluated and that re-evaluation must bring the database to a consistent state without any malicious effects. For every damaged data item in Damaged_DI, the valid value (before any malicious write if found) has to be brought back either from the RTL or from the sub-cluster in the log or from the stable database. The written values after re-evaluation are flushed then to the stable database to ensure getting rid of the malicious transactions effects.

It's worth noting that correctness both damage assessment and recovery algorithms is proved and shown in [2] and thus improvements in this paper focus on algorithms execution time. This is the main reason behind adding two new data structures (ATL and RTL) and a transaction queue *q* along with using the timestamps of transactions. ATL and RTL store useful information among which are old values of data items, timestamps of changing their values, the transaction responsible for the write operation, transactions affected by different write operations. All of these details can be stored and accessed by *O(1)* mapping operations in this paper while they require linear search methods in [2] as most of them were not stored in data structures but searched for. All improvements are proved below by calculating the complexity of algorithms in this paper compared to the previous ones. We will refer to algorithms in this paper as *new* and the ones in [2] as *previous.*

```
For every block in Damaged_PB
        Get the Sub-Cluster ID from the TSC List
        Call ReEvaluate_PB(T_i, P^{j.k})
Flush the updated data items to the stable database
Delete Damaged_DI and Damaged_PB

Procedure ReEvaluate_PB
Reconstruct Predicate Block from sub-cluster (w/ read & write op.s)
Reevaluate the Predicate
For every statement in the Predicate Block
        Read all data items x
        If x ∉ Damaged_DI
        Read current value from the entry
        Else if TimestampOfChange(x) (col. 4 RTL)<Initial Write
        timetamp (x)
        Set value of x as Old Value(x) (col. 3 of RTL)
        Else
        Scan subcluster to get old_value of x before last update record
        there
        If not found then read value from database
        For write x
        If x∈Damaged_DI then update entry with new fresh_value
        Add x to Damaged_DI with new fresh_value
```

Figure 3. The recovery algorithm.

First, the complexity of the *previous* damage assessment algorithm is O(M.|TSC|.|SCD|.|Damage_DI|.|Damage_PB|), where *M* is the number of malicious transactions. For each malicious transaction, TSC and SCD are scanned for Sub Cluster ID of transaction and minimum Sub Cluster ID in $O(|TSC| . |SCD|)$ time, then Damaged_DI and Damaged_PB are searched before adding a new data item and transaction respectively in $O(|Damage\_DI|.|Damage\_PB|)$ time.

The *new* damage assessment algorithm reduced the execution time to $O(|q|.|ATL|)$. This algorithm eliminates the need of searching TSC, SCD, Damaged_DI and Damaged_PB. In addition, $|q| << M.|TSD|$ because the size of queue grows incrementally until it contains all malicious transactions and the ones affected *only* and *not all* possible transactions as *previous* algorithm does. Also, $|ATL| << |SCD|$ since ATL contains the transactions affected by each other transaction, and this is always less than the number of operations performed on data items by each transaction which is stored in SCD. Moreover, the *new* algorithm contains one conditions statement, while the *previous* one contains 13 if-else statements. The reason behind that is the use of timestamps by *new* algorithm. Thus, the *new*

damage assessment algorithm has an improved complexity of $O(|q|.|ATL|) << O(M.|TSC|.|SCD|.|Damaged\_DI|.|Damaged\_PB|)$ by the *previous* algorithm.

Second, the *new* recovery algorithm adds to the *previous* one a step to check the timestamp of the write operation that changed the value of a data item *x* and recovers its old value accordingly. Both the timestamps and old values of data items are stored in RTL, and each row in RTL can be accessed by an *O(1)* mapping operation which can be achieved by one of two ways: either creating a hash map to store the index of each data item in RTL or using a mapping function that calculates the index such as a one that depends on the ASCII code of the name of data item: *index of item = ASCII code of item name – ASCII code of first item name*. The first option obviously requires more space, but both of them run in *O(1)* and may replace the *O(N)* search in the *previous* recovery algorithm where *N* is the size of a whole sub-cluster. Thus, the *new* recovery algorithm replaces *O(N)* searches of sub-clusters in *previous* algorithm by *O(1)* operations, what leads to *N'*-fold execution time reduction, where *N'* is the average sub-cluster size.

6. Conclusion

This paper presents improvements to the work of Haraty and Zbib [2]. A totally new and improved damage assessment algorithm has been introduced. In addition, two new data structures (Responsible Transaction List RTL and Affected Transactions List ATL) and a mapping function are introduced and the sub-clustering algorithm was modified accordingly. Improvement has also been applied to the recovery algorithm. All of the changes aimed to improve the performance and decrease the execution time of the algorithms as time is a very valuable resource in processing transactions.

References

[1] B. Panda, and K.A. Haque, "Extended Data Dependency Approach – A Robust Way of Rebuilding Database," *Proc. ACM Symposium on Applied Computing*, pp. 446-452, 2002.

[2] Ramzi A. Haraty and Mirna Zbib. A Matrix-based Damage Assessment and Recovery Algorithm. *Proceedings of the IEEE 14th International Conference on Innovations for Community Services (I4CS 2014)*. Reims, France, June 2014.

[3] J. Jiffen, and A. Srivastava, "Attribution of Malicious Behavior: Surviving Information Warfare Attacks on Databases," *Proc. 6th International Conference on Information Systems Security*, pp. 28-47, 2010.

[4] G. Fu, H. Zhu, and Y. Li, "A Robust Damaged Assessment Model for Corrupted Database Systems," *Proc. 5th International Conference on Information Systems Security*, pp. 237-251, 2009.

[5] S. Zhang, X. Xiong, X. Jia, and P. Liu, "Availability Sensitive Intrusion Recovery," *Proc. 1st ACM Workshop on Virtual Machine Security*, 2009.

[6] X. Jia, S. Zhang, J. Jing, and P. Liu, "Using Virtual Machines to Do Cross-Layer Damage Assessment," *Proc. 1st ACM Workshop on Virtual Machine Security*, pp. 29-38, 2008.

[7] K. Bai, and P. Liu, "A Damage Tracking Quarantine and Recovery (DTQR) Scheme for Mission-Critical Database Systems)," *Proc. 12th International Conference on Extending Database Technology: Advances in Database Technology*, pp. 720-731, 2009.

[8] K. Bai, and P. Liu, "A Light Weighted Damage Tracking Quarantine and Recovery Scheme for Mission-Critical Database Systems," *Proc. 17th ACM Conference on Information and Knowledge Management*, pp. 1403-1404, 2008.

[9] B. Panda, and J. Zhou, "Database Damage Assessment Using a Matrix Based Approach: An Intrusion Response System," *Proc. 7th International Database Engineering and Applications Symposium*, 2003.

[10] M. Balitanas, and T.H. Kim, "Isolation Solution for Insecure Information Systems," *Proc. 9th WSEAS International conference on Data Networks, Communications, and Computers*, pp. 58-61, 2010.

[11] R. Kotla, L. Alvisi, and M. Dahlin, "SafeStore: A Durable and Practical Storage System," *Proc. 2007 USENIX Annual Technical Conference*, 2007.

[12] T. Xin, "A Framework for Processing Generalized Advanced Transactions," Ph.D. dissertation, Colorado State University, 2006.

[13] P. Ragothaman, B. Ragothaman, and B. Panda, "Analyzing Transaction Logs for Effective Damage Assessment," *Proc. 16th Annual IFPI WG 11.3 Working Conference on Databases and Application Security*, 2002.

[14] P. Ragothaman, and B. Panda, "Hybrid Log Segmentation for Assured Damage Assessment," *Proc. ACM Symposium on Applied Computing*, 2003.

[15] H. Zhu, G. Fu, Y. Zhu, R. Jin, K. Lu, and J. Shi, "Dynamic Data Recovery for Database Systems Based on Fine Grained Transaction Log," *Proc. 2008 International Symposium on Database Engineering & Applications*, pp. 249-253, 2008.

[16] Ramzi A. Haraty and Arda Zeitunlian. *Damage Assessment and Recovery from Malicious Transactions using Data Dependency*. ISESCO Journal of Science and Technology, Volume 3, Number 4. November 2007.

[17] R. El Masri, and S.B. Navathe, "Concurrency Control Based on Timestamp Ordering," in *Fundamentals of Database Systems*, 6th ed., London: Addison Wesley, pp. 788-791, 2011.