Reducing Damage Assessment Latency in Survivable Databases

Indrakshi Ray[1], Ross M. McConnell[1], Monte Lunacek[1], and Vijay Kumar[2]

[1] Department of Computer Science
Colorado State University
Fort Collins, CO 80523-1873
`{iray, rmm, lunacek}@cs.colostate.edu`
[2] School of Computing and Engineering
University of Missouri-Kansas City
`kumarv@umkc.edu`

Abstract. Traditional recovery mechanisms are not adequate in protecting databases from malicious attacks. A malicious transaction by virtue of writing on to the database can corrupt one or more data items; benign transactions reading these data items and writing on other data items can help spread the damage. To prevent the damage from spreading, it is important to assess the damage and confine it as quickly as possible. Algorithms providing fast damage assessment are needed. In this paper we look at two existing techniques for damage assessment and analyze their complexity. We also propose a new technique that improves upon the existing techniques by reducing the time required for damage assessment.

1 Introduction

Traditional recovery techniques [2] are not adequate in protecting a database from malicious attacks. A malicious transaction can write one or more data items and corrupt it. A good transaction reading corrupted data items can spread the damage to the data items that it writes. Such a good transaction is said to be *affected*. A malicious transaction together with affected transactions can corrupt a significant portion of the database.

When an attack is detected, it is important to find out which transactions and data items were affected by this attack. This process is known as *damage assessment* [1, 4, 9]. Liu et al. [7, 8] acknowledge that unless damage assessment proceeds at a faster rate than normal processing of transactions, it is possible that the damage assessment process will never terminate. The time interval between the detection of an attack and damage assessment is known as the *damage assessment latency*. To prevent the damage from spreading, it is important to reduce the damage assessment latency.

To assess the damage caused by a malicious transaction, there are two approaches: transactional dependency and data dependency approach In transactional dependency approach [1, 6, 5], all the good transactions that depend on the malicious transaction are identified. Panda and Giordano [9] realized that some of the transactions operations are innocent and may not have been affected by malicious transactions. The data dependency approach identifies only those statements within a transaction that are affected.

Since this method suffers from transaction integrity issues unless some other actions are taken [10], we focus on transactional dependency based approaches.

In order to repair damage, it is important to get a list of the affected transactions in an order such that, if the transactions are repaired in that order, it is always the case that when it is time to repair transaction T_j, all of the prior transactions on which it depends have already been repaired. Let us call such an ordering a *repair schedule*. An important ingredient in damage assessment is to provide a list of affected transaction, together with a suitable repair schedule.

In this paper, we analyze the asymptotic running times of Lala and Panda's algorithm [5], as well as of Ammann, et. al's [1]. We propose a damage assessment algorithm that is similar to Lala and Panda's in that the dependency information is stored in a separate data structure that represents a directed acyclic graph on the transaction. The running time of our approach improves upon theirs; ours finds the affected transactions and a suitable repair schedule in time that is proportional to the sum of sizes of the log-file records of the affected transactions.

The rest of the paper is organized as follows. Section 2 analyzes the existing damage assessment algorithms. Section 3 presents our damage assessment algorithm. Section 4 concludes our paper with some pointers to future directions.

2 Analysis of Existing Damage Assessment Algorithms

2.1 Ammann, Jajodia, and Liu's Damage Assessment

Ammann, Jajodia, and Liu have given a set of algorithms for assessment and repair. In most of these algorithms the damage assessment and repair phases are not distinct, and so we do not discuss them. We present the algorithm in which the damage assessment phase is distinct.

The algorithm begins by scanning the log backward to find the entry corresponding to the start record of the malicious transaction. Then the log is processed forward to identify all the transactions that directly or indirectly depend on the malicious transaction.

One disadvantage of this approach is that the log file is usually quite large, and is larger than conventional log files which do not record dependency information. When a malicious transaction is detected, the entire portion of the log file that comes after the malicious transaction must be scanned. If malicious activity occurs, this may not be detected for some time, and there is some urgency to repair the effects of a malicious transaction once it is discovered. If an attack involves more than one transaction, and not all of them are discovered at the same time, then fixing them all can involve multiple repeated scans of a large interval of the log file.

2.2 Damage Assessment by Lala and Panda

To assess damage faster, Lala proposes to create dependency lists and transaction information lists by analyzing the information stored in the log. The dependency list stores the dependency information of transactions. In the damage assessment phase an *affected list* is created that gives a valid repair schedule.

Specifically, let a *transaction* be an operation on the database that reads data items $R = \{r_1, r_2, ..., r_r\}$ and uses these to compute values that are written to data items $W = \{w_1, w_2, ..., w_w\}$. A transaction is *committed* if it has completed successfully and carried out its write operations on the database. A committed transaction T_j is *directly dependent* on a committed transaction T_i if R_j contains some element that was last written to by T_i.

Let $(T_1, T_2, ..., T_n)$ be the sequence of transactions in the log, in the order in which they are committed, and let the *timestamp* of T_j be denoted by j.

Initially Lala and Panda's damage algorithm for assembling an affected list consists only the malicious transaction T_j. It then iteratively removes an entry T_j from the front of the affected list, finds all the neighbors of T_j that are unmarked, and marks and inserts them to the affected list "in sorted order". That is, they maintain the invariant that the timestamps of vertices in the affected list are in ascending order, which is a valid repair schedule.

Let V be the set of affected transactions, and let $v = |V|$, and let s denote the sum of sizes of the transaction records corresponding to V in the log file. Lala and Panda do not discuss data structures for implementing the affected list. If it is implemented with a linked list, then it is easily shown that the running time of their repair algorithm is $O(v^2 + s)$. This time bound would not be practical, because of the quadratic behavior in the size of v, as v might be large. However, a fairer assessment of their running time is obtained by observing that the affected list is accessed in accordance with an abstract data type called a *priority queue*. A priority queue allows elements to be inserted into a list in $O(\log n)$ time and the minimum element to be removed in $O(\log n)$ time, where n is the size of the list.

Under this assumption, Lala and Panda's algorithm can be shown to run in $O(v \log v + s)$ worst-case time, using the most efficient implementations of priority queues known [3]. Though this is a practical time bound, because of the $\log v$ factor in the running time, this is not linear in the sum of lengths of the affected transactions. This is due to the use of a priority queue, a data type whose use we show can be eliminated from the problem altogether, improving the worst-case efficiency and simplifying the task of implementing it.

3 Our Approach

We can model direct dependencies with a directed *dependency graph* whose vertex set is the set of transactions, and whose edge set is $\{(T_i, T_j) |\ T_j$ is directly dependent on $T_i\}$. Since (T_i, T_j) is an edge only if T_j is committed after T_i, it follows that the dependency graph is a directed acyclic graph. This structure is similar to that of Lala and Panda, although we would propose using a standard adjacency-list representation of the graph [3], rather than the more cumbersome representation given in Lala and Panda's paper.

Instead of using a priority queue, we propose using *depth-first search*. This is a recursive strategy that marks a node u as *discovered* when a recursive call is made on u. The call on u then generates recursive calls on the unmarked neighbors of u. When all of these have been completed, all neighbors have been marked, u is marked as *finished* and the recursive call on u returns.

An initial call is initiated at any malicious transaction. After it returns, new calls may be initiated at unmarked transactions. It is not hard to show by induction that after all malicious transactions have been marked, so have the affected transactions.

Let us now address the question of how to produce a repair schedule. Every time a node is marked as *finished*, we prepend it to an affected list. We claim that when all malicious transactions and affected nodes have been marked, our affected list is a valid repair schedule, even though it does not necessarily list the transactions in ascending chronological order.

The reason that the repair schedule is that it constitutes a *topological sort* of the subgraph induced by the affected transactions. A topological sort is an ordering of the affected transaction such that all directed edges in the subgraph that they induce point from earlier to later elements of the ordering. It is well-known that taking vertices of a directed acyclic graph in descending order of finishing time produces a topological sort [3].

If the transactions are repaired from left to right, then when a transaction T_j is reached, all transactions on which it depends, namely, all transactions that have a directed edge to T_j, are earlier in the list, and have therefore already been repaired.

Lemma 1. *The proposed assessment algorithm produces a repair schedule in time proportional to the sum of lengths of the affected transactions.*

It remains to give asymptotic bounds for the total time spent keeping the dependency graph up-to-date as new transactions are committed and added to the log file. The transaction vertices are stored in an array, a Transaction Array; the numberings or timestamps serve as their indices into the array. The number of transactions in the array grows as new transactions are committed. When the array is full, it can be copied to a newly allocated array that is twice as large to create more room for nodes. By an *amortized analysis*, the total cost of reallocating and recopying is bounded by the number of elements in the array [3].

In addition to this, each data element of the database carries a *last-write record*, which is the index of the last transaction that wrote to it. This allows lookup of this transaction in the transaction array in $O(1)$ time.

When a new transaction T_n is added to the log file, the last-write records of the data elements read by T_n gives a multilist of transactions on which T_n belongs. This can be reduced to a a simple list, where no transaction appears more than once, by traversing the list, accessing the transactions in $O(1)$ time, marking each occurrence of an element in the list that is not already marked, and deleting any occurrences that were previously marked. This removes duplicates from the list. A second traversal of the list serves to remove the marks so that they do not interfere with later operations. An edge can then be installed from each T_j on which T_n belongs in $O(1)$ time, because we use an adjacency-list representation of the graph.

To finish updating the structure, T_n must replace the last-write records of the set of elements that it writes to with its index n. This takes time proportional to the number of elements that it writes to. The total time inserting T_n is proportional to the number of data elements that it reads and writes, which is proportional to the length of the transaction. This gives the following:

Lemma 2. *The total time to construct the dependency graph is proportional to the sum of lengths of transaction records in the log file.*

Because of this, the time to maintain the graph is on the order of the time simply to write the transaction records to the log file.

4 Conclusion

In this paper we analyzed two damage assessment algorithms and proposed a new algorithm that can reduce the time taken for damage assessment. The first algorithm [1] that we analyzed reads the log to find out the transactions that were affected by the malicious transaction. The log file, containing records of all operations of transactions, is typically very large and accessing it will involve significant disk I/Os. The second algorithm [5] that we analyzed minimizes log accesses by storing the transaction dependency information in a list structure. This information is captured while the transactions execute. We show how to store this dependency information in a data structures such that the speed for damage assessment is minimized. We also compare our approach with others and show that our approach requires the minimum time for damage assessment.

This paper looked at damage assessment algorithms based on transaction dependencies. This approach may result in a lot of false positives. Such false positives can be reduced by data dependency approaches. In future we plan to study existing data dependency approaches and improve upon them, if possible. Reducing the time taken for damage assessment is important, but it is also necessary to minimize the time taken for damage repair. In future, we also plan to optimize, if possible, existing damage repair algorithms.

References

1. P. Ammann, S. Jajodia, and P. Liu. Recovery from malicious transactions. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1167–1185, 2002.
2. P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison–Wesley, Reading, MA, 1987.
3. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw Hill, Boston, 2001.
4. Sushil Jajodia, Catherine D. McCollum, and Paul Ammann. Trusted recovery: An important phase of information warfare defense. *Communications of the ACM*, 42(7):71–75, July 1999.
5. C. Lala and B. Panda. Evaluating damage from cyber attacks. *IEEE Transactions on Systems, Man and Cybernetics*, 31(4):300–310, July 2001.
6. Chandana Lala and Brajendra Panda. On achieving fast damage appraisal in case of cyber attacks. In *Proceedings of the IEEE Workshop on Information Assurance*, West Point, NY, 2000.
7. Peng Liu and Sushil Jajodia. Multi-phase damage confinement in database systems for intrusion tolerance. In *Proc. 14th IEEE Computer Security Foundations Workshop*, 2001.

8. Peng Liu, Sushil Jajodia, and Catherine D. McCollum. Intrusion confinement by isolation in information systems. In *IFIP Workshop on Database Security*, pages 3–18, 1999.
9. Brajendra Panda and Joseph Giordano. Reconstructing the database after electronic attacks. In *Proceedings of the 12th Annual Working Conference on Database Security*, Chalkidiki, Greece, 1998.
10. Brajendra Panda and Kazi Asharful Haque. Extended data dependency approach - a robust way of rebuilding database. *ACM Symposium on Applied Computing*, pages 446–452, 2002.