ISESCO Science and Technology Vision
Volume 3 - Number 4 - November 2007 (43-50)

Damage Assessment and Recovery from Malicious Transactions Using Data Dependency for Defensive Information Warfare

Ramzi A. Haraty and Arda Zeitunlian

Lebanese American University
Division of Computer Science and Mathematics
Beirut, Lebanon

Abstract

With the advancement of Internet technology, securing information systems from electronic attacks have become a significant concern. With all the preventive methods, malicious users still find new methods that overcome the system security, and access and modify the sensitive information. To make the process of damage assessment and recovery fast and efficient and in order not to scan the whole log, researchers have proposed different methods for segmenting the log, and accordingly presented different damage assessment and recovery algorithms. Since even segmenting the log into clusters may not solve the problem, as clusters/segments may grow to be humongous in size, this is in case of high data/transaction dependency, we suggest a method for segmenting the log into clusters and its sub-clusters; i.e, segmenting the cluster; based on exact data dependency [12], into sub-clusters; based on two different criteria: number of data items or space occupied. In this work, we also present damage assessment and recovery algorithms, and show the performance results.

1. Introduction

Nowadays, the productivity of any organization depends on the information it shares with and protects from the rest of the world. The disadvantage of this connectivity is that it gives way for malicious users to access and possibly damage the sensitive information. In spite of different defensive and security measures as firewalls, authentication, access controls, data encryption, still hackers/intruders continuously find new ways to overcome the system security.

During a system failure, the effects of all write operations of non-committed transactions that are already written into the stable database are undone. The effects of all write operations of committed transactions, which are not in the stable database, are redone, which and the log is purged [13] [14] [20] [21].

This approach does not work with electronic attacks, since the system treats the attacker as any other valid transaction and makes the update permanent. Whenever an electronic attack is detected, all the updates of the attacker must be undone including the updates of the transactions that directly or indirectly read from the attacker. Then, these valid transactions must be re-executed to return the database to a consistent state. For this reason, the log is modified to store the read-from relationship as well; and not allow purging.

The disadvantage of adding the read from relationships and not allowing purging, is that, the log grows in size, and if an attack is found that has occurred for example a month ago, the damage assessment and recovery will take significant amount of time; to undo and then redo all the transactions that have directly or indirectly read-from the attacking transaction, thus; leading to denial-of-service, which might be the attackers aim. Therefore, developing an efficient algorithm to recover the system from an electronic attack is quite essential, at the same time maintaining the system integrity and availability.

During recent years, there have been many researches in this area. In [2], Jajodia et al have discussed recovery issues for defensive information warfare. Ammann et *al.* [7] used color scheme for identifying the damaged and corrupted data, while the database remains available during the recovery process. In [11] Liu et al. have presented algorithms that re-write transaction history by moving the attacking transaction and all effected transactions.

Later researchers proposed clustering the log using transaction dependency approach for accessing only one of the clusters during damage assessment. Then instead of traditional transaction dependency researchers have used data dependency.

Ragothaman and Panda in [19] have proposed a simpler model to build the log by simply implementing limitations to clusters; like number of committed transactions, the size of clusters, or a time window for clusters. This logging technique was used, in [20], where the already segmented log is segment based on transaction dependency to achieve much faster damage assessment and hence recovery.

To minimize the I/O access, researches have proposed different methods. In [22] Panda and Lala, damage assessment time was reduced by adding auxiliary data structures, that keeps track of the transaction dependency relationships. In [25], the authors suggested using matrices for damage assessment, while, in [24], the model proposed is based on the knowledge of data relationships that is mined during normal database use.

In our work, we present an algorithm to segment the log into clusters, based on exact data dependency [15][18], thus overcoming the limitations of the data dependency. Our logging algorithm suggests that each cluster in itself be segmented into sub-clusters according to two different approaches: number of data items or space occupied. In addition, based on this clustered log, we develop a damage assessment and recovery algorithm. The aim being fast and efficient damage assessment and recovery, with our approach, even if the clusters grow in size, the sub-clusters size will be controlled, and only the required sub-clusters will be scanned, instead of scanning the whole cluster.

The rest of the work is organized as follows. In **Section 2**, we give our log segmentation method, as well as the damage assessment and recovery algorithms, while the experimental results are in **Section 3**; and finally we give our conclusion in **Section 4**.

2. Hybrid sub clustering for damage assessment and recovery

In our work, in order to reduce the I/O access, we suggest an algorithm that segments the log into clusters, based on exact data dependency; thus, overcoming the limitations of the data dependency. In addition, each cluster in itself is segmented into sub-clusters according to two different approaches: number of transactions or space occupied; by keeping two additional data structures (lists), in order to facilitate damage assessment process.

2.1 Assumptions

Our logging model is based on the assumption that the attacking transaction has been detected by one of the intrusion detection techniques. We also assume that the scheduler is modified to produce different types of read and write operations, similar to that described in [12], in addition, the history is rigorous serializable [21]. The transactions are kept in temporary sequential log during the execution of the transactions, this is so that the clustered log will store only the committed transactions; thus, will require no undo operations in case of transaction or media failures. Also we assume that the transaction ID is sequentially incremented; i.e, if T_2 commits then the only transaction committed before T_2 is T_1.

2.2 Transaction model

A transaction could be defined as a program unit that accesses and possible updates various data item; it could be formalized in terms of predicates, a precondition that accordingly different statements are executed, and statements [21].

In our model, which agrees with the models described in [15] and [18], the representations of a transaction and the different read and write operations of the transactions, could be explained in the following transaction example:

```
T_1: if (z < 5) then      1
         a := x;          1.1
Else b := y;              1.2
```

Where value of z = 1, then the history is: $pr_1^1[z]$, actual-$read_1^{1.1}[x]$, actual-$write_1^{1.1}[a]$, overlooked-$read_1^{1.2}[y]$, overlooked_$write_1^{1.2}[b]$; the superscripts representing the block numbers, while the subscripts transaction number.

The definitions that our proposed model relies on are:

Definition 1: A write operation $w_i[x]$ of a transaction T_i is dependent on a read $r_i[y]$ operation of T_i if $w_i[x]$ is computed using the value obtained from $r_i[y]$.

Definition 2: A data value v_1 is dependent on data value v_2 if the write operation that wrote v_1 was dependent on a read operation v_2, where v_1 and v_2 may be two different versions of the same data item.

Definition 3: A write operation $w_i[x]$ of a transaction is dependent on a set of data items I, if $x = f(I)$; i.e, the values of data items in I are used in calculating the new value of x. There are the following three cases for the set of data items I.

Case 1) $\mathbf{I} = \Phi$, This means that no data item is used in calculating the new value of x, and is denoted as a fresh write. If $w_i[x]$ is a fresh write and if the previous value of (before this write operation) is damaged, the value of x will be refreshed after this write operation.

Case 2) $x \neq I$, the operation is called a blind write. In this case if the previous value of x (before this write operation) is damaged and none of the data items in I are damaged, then the value of x will be refreshed after this write operation.

Case 3) $x = I$, If the previous value of x (before the write operation $w_i[x]$) is damaged, then x remains damaged. Otherwise, if any item in I is damaged, then x is damaged.

Definition 4: A *predicate-read* of a data item in a transaction T_i is the data item the transaction read for determining the execution path in its program when T_i was executed. This is denoted by $pr_i[x]$, where p before r represents that it is a predicate read, subscript i represents the transaction to which this *predicate-read* operation belongs to and x in the bracket represents the data item that is read by this transaction.

Definition 5: An *actual-read* (or *actual-write*) of a data item in a transaction T_i is the data item transaction read (or wrote) for executing a statement (not logical or relational) when T_i was executed. This is denoted by $ar_i[x]$ (or $aw_i[x]$) where a before r (or w) denotes that this is an actual read (or write) operation and subscript i repesents the transaction to which this *actual-read* (or *actual-write*) operation belongs to, *x* in the bracket represents the data item that is read (or wrote) by the transaction.

Definition 6: The *overlooked-read* (or *overlooked-write*) or data item in a transaction T_i is the data item the transaction would have read (or wrote) if this transaction had followed a different execution path of its program. This is denoted by $or_i[x]$ (or $ow_i[x]$) where the character *o* before *r* (or *w*) denotes that this is in overlooked read or write operation, subscript *i* represents the transaction to which this *overlooked-read* (or *overlooked-write*) operation belongs to, *x* in the bracket represents the data item that would have read (or write) by this transaction.

Definition 7: [18] A transaction T_i is a partial order with ordering relation $<_i$, where:

1. $T_i \leq (pr_i\ [x], ar_i[x], or_i[x], aw_i[x], ow_i[x]$ | x is a data item) $\cap$ $\{E_i\}$;

   where E_i = {ai, ci}.

2. $a_i \in T_i$ *iff* c_i e T_i

3. if $OP_i \in T_i$, (any operation (OP_i other than a_i or c_i) then $OP_i <_i (a_i$ or $c_i)$.

4. if $r[x]$, $w[x]$ e T_i. (any combination of read and write operation) then

   $r[x] <_i w[x]$ or $w[x] <_i r[x]$, and

5. if a conditional block is present in the transaction then *predicate-read* precedes all read and write operations in that block.

2.3 Logging, damage assessment and recovery algorithms

2.3.1 Data structures

The data structures used in our sub-clustering model are:

Transaction Sub Cluster List (TSC): This list is used to store the transaction ID and the corresponding Sub Clusters in which the data items of this transaction are stored. This table is referred to obtain the sub clusters that are affected, once the malicious transaction is identified.

Sub Cluster Data List (SCD): This is used to store the sub cluster IDs, the transaction ID and the corresponding data item, whether read/write, actual/overlooked, predicate or statement. This table is referred to identify the data items of the operations of the malicious transaction as the sub cluster is obtained.

2.3.2 Hybrid sub-clustering algorithm

In our model, clusters are determined periodically, once a list of all committed transactions from the temporary log is obtained. (Periodically the temporary log could be purged; i.e, when the updates till that transaction; are guaranteed to be in the stable database and are clustered to their appropriate sub-clusters).

2.3.2.1 Hybrid sub-clustering algorithm based on fixed number of transactions

In this approach, a fixed number of committed transactions are grouped together to form the sub-clusters, shown in ***Table 2.1***.

In the above algorithm, whenever a read operation (actual, overlook read, predicate read) on any data item is encountered, the SCD and TSC lists are checked to determine the sub-cluster in which the data item resides. If both the data item and the transaction are found in the sub-cluster then the operation is added to the log (sub-cluster) and the SCD list is updated. If the data item is found in the sub-cluster, but not the transaction, then the TSC is updated, as well as the transaction counter of the sub-cluster, and the operation is added to the log. In case the operation is a write (actual, overlooked), then the SCD and TSC lists are checked to determine the sub-cluster; in addition data dependency is checked, if data dependency is established then all the data items are merged; i.e, put in the same cluster; and accordingly the SCD and TSC lists are updated.

2.3.2.2 Hybrid clustering algorithm based on fixed size

In this approach, the criterion for dividing the cluster is the size of the sub-cluster. Operations of committed transactions that are dependent are added to the same sub-cluster until there is no more space for the next transaction to fit into it. Though, this method will result in the wastage of disk space, it makes damage assessment simple.

2.3.3 The damage assessment and recovery algorithm

Once the malicious transaction/s is/are identified, Transaction Sub Cluster (TSC) and the cluster/s and sub-clusters (SCD) lists are checked to obtain the affected data items; thus constituting the damage assessment phase, and once damaged data items are identified, recovery becomes easier and instead of scanning the whole scan; only the sub-clusters that contain the damaged data items are scanned. In this section, an example will be described in order to explain the damage assessment and recovery algorithms, and later the damage assessment and recovery algorithms will be presented.

2.3.3.1 A scenario

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;	

Using our sub clustering approach, the Transaction Sub Cluster (TSC) List and the Sub Cluster Data Item (SCD) List for these transactions are given in ***Figure 4.1***. Assuming

TABLE 2.1 - Hybrid Sub-Clustering Algorithm based on fixed number of transactions.

1.	For every operation O_i of a committed transaction in the temporary log
1.1	Case O_i is read
1.1.1	If ($x_i \in$ Read_Set(Predicate T_i^{jk}) then
1.1.1.1	If ($x_i \notin$ a cluster's sub-cluster (SCD) then
1.1.1.1.1	Assign new cluster ID and new sub-cluster (ID = 1);
1.1.1.1.2	Update the corresponding data structures (SCD & TSC)
1.1.1.2	Else
1.1.1.2.1	If (sub_cluster.transaction_count $\leq$ MAX)
1.1.1.2.1.1	Get the cluster ID and sub-cluster ID;
1.1.1.2.1.2	Update the corresponding data structures (SCD & TSC);
1.1.1.2.2	Else
1.1.1.2.2.1	if (sub_cluster.transaction_count > MAX)
1.1.1.2.2.1.1	Assign new sub_cluster within the same cluster;
1.1.1.2.2.1.2	Update the correspondng data structures (SCD & TSC);
1.1.1.2.3	Record the predicate[predicate-read, T_i^{jk}, x_i, string] in the log (cluster);
1.1.2	If ($O_i \in$ Read_Set(Statement T_i^{jk}) then
1.1.2.1	If ($x_i \notin$ a cluster's sub-cluster) then
1.1.2.1.1	Assign new cluster ID and new sub-cluster (ID = 1);
1.1.2.1.2	Update the corresponding data structures (SCD & TSC);
1.1.2.2	Else
1.1.2.2.1	If (sub_cluster.data_item_count $\leq$ MAX)
1.1.2.2.1.1	Get the cluster ID and sub-cluster ID;
1.1.2.2.1.2	Update the correspondng data structures (SCD & TSC);
1.1.2.2.2	Else
1.1.2.2.2.1	If (sub_cluster.data_item_count > MAX)
1.1.2.2.2.1.1	Assign new sub_cluster within the same cluster
1.1.2.2.2.1.2	Update the corresponding data structures SCD & TSC);
1.1.2.2.3.1	If O_i is actual read
1.1.2.2.3.1.1	Record the operation [read_item, $T_{i,}^{jk}$, x_i, value, string] in the log(cluster)
1.1.2.2.3.2	Else
1.1.2.2.3.2.1	Record the operation [overlook_read_item, $T_{i,}^{jk}$, x_i, value, string] in the log(cluster)
1.2	Case O_i is Write
1.2.1	If ($O_i \in$ write_Set(Statement T_i^{jk}) then
1.2.1.1	If ($x_i \notin$ a cluster's sub-cluster) and (No dependency) then
1.2.1.1.1	Assign new cluster ID and new sub-cluster (ID = 1);
1.2.1.1.2	Update the corresponding data structures (SCD & TSC);
1.2.1.2	Else
1.2.1.2.1	If ($x_i \in$ a cluster's sub-cluster) and (No dependency) then
1.2.1.2.1.1	If (sub_cluster.transaction_count $\leq$ MAX)
1.2.1.2.1.1.1	Get the cluster ID and sub-cluster ID;
1.2.1.2.1.1.2	Update the corresponding data structures (SCD & TSC);
1.2.1.2.1.2	Else
1.2.1.2.1.2.1	If (sub_cluster.transaction_count > MAX)
1.2.1.2.1.2.1.1	Assign new sub_cluster within the same cluster;
1.2.1.2.1.2.1.2	Update the corresponding data structures (SCD & TSC);
1.2.1.2.2.	If (dependency) then
1.2.1.2.2.1	Check all dependent reads
1.2.1.2.2.1.1	For each different cluster MERGE
1.2.1.2.3.1	If O_i is actual-write
1.2.1.2.3.1.1	Record the operation [write_item, $T_{i,}^{jk}$, x_i, new_value, old_value, string] in the log
1.2.1.2.3.2	Else
1.2.1.2.3.2.1	Record the operation [overlook_write_item, $T_{i,}^{jk}$, x_i, new_value, old_value, string] in the log

Cluster (TSC) List

Trans.	Sub Cluster Code
1	1
2	1
3	1
4	2
5	2
6	2
…	…

Cluster	Sub Cluster	Data Item	Trans.	…
1	1	A	1	R
1	1	B	1	W
1	1	A	2	R
1	1	A	2	W
1	1	A	3	R
1	1	C	3	W
1	2	C	4	R
1	2	D	4	W
1	2	D	5	R
1	2	E	5	W
1	2	E	6	R
1	2	F	6	W
1	…	…	…	
1	3	X	8	R
1	3	Y	8	W
1	3	B	9	R
1	3	Z	9	W

Figure 2.1 - The Transaction Sub Cluster (TSC) list and the basic Sub Cluster Data item List (SCD) list, using hybrid sub-clustering on fixed number of transactions.

transaction ID = 1 is found to be malicious. For damage assessment, from the Transaction Sub Cluster List the sub cluster ID is obtained, in this case is 1, and from the SCD List, the affected data items are obtained; {B, Z}, and thus for recovery only the sub clusters, in this case {1,3}, that contain the affected data items will be scanned.

2.3.3.1 The Damage assessment algorithm

Our damage assessment algorithm, shown in ***Table 2.2***, uses additional structures: Damaged_DI and Damaged_PB. The former is the list of data items that are directly or indirectly damaged by the malicious transaction, while the later contains the damaged predicate block of a transaction T_i that has been affected directly or indirectly by the malicious transaction.

The first step of the algorithm initializes the Damaged_DI and Damaged_PB data structures, by creating them and setting them to null. Then every transaction in the Transaction Sub Cluster (TSC) list s checked, from the first point of attack to identify all affected transactions. Step 2.2.1 checks whether the record is predicate read, if so, then if the Data Item (x) belongs to the Damaged_DI then the block to which this predicate belongs is also damaged;thus, the block is added to the Damaged_PB list. In step 2.2.2, if the record is actual-read or overlooked-read then if the operation does not belong to the malicious transaction but the Data Item x is damaged then the block is checked, if the block is not damaged, it should be added to the Damaged_PB list. Step 2.2.3 checks if the record is write operation, if so, then if the transaction is malicious but data item does not belong to the Damaged_DI list then it is added, else in step 2.2.3.2 if the transaction is not malicious and the block is not damaged, but the Data Item (x) is affected, then this operation is clean write, thus; the the data item is removed from the Damaged_DI list. In step 2.2.3.2.2.2 if the transaction is malicious and the Block is damaged then the Data Item is added to the Damaged_DI list.

2.3.3.2 The recovery algorithm

For recovery, the data structures used in damage assessment are used, described in ***Table 2.3***.

In the first step of our recovery model, is to scan all the Damaged_PB records that were obtained from the damage assessment algorithm. In step 1.1., the sub cluster in which

TABLE 2.2 - Damage Assessment

1.	Create Damaged_DI List and initialize it to null; Create Damaged_PB List and initialize it to null;
2.	For every Malicious Transaction
2.1.	Get the Sub Cluster ID from the Transaction Sub Cluster (TSC) List
2.2.	Scan the Sub cluster Data Item (SCD) List starting with the minimum sub cluster ID; having transaction ID >= the malicious transaction ID.
2.2.1.	If (Record = = Predicate_Read_Item) then
2.2.1.1.	If (($x \in$ Damaged_DI) then
2.2.1.1.1.	Add the predicate block number ($j.k$) & transaction ID (i) of this operation into the Damaged_PB
2.2.2.	Else If ((Record = = Read_item) OR (Record = = Overlooked_Read_item) then
2.2.2.1.	If (T_i is not Malicious) AND ($x \in$ Damaged_DI) then
2.2.2.1.	If ($T_i S^{j.k} \notin$ Damaged_PB) then
2.2.2.1.1.	Add the block of number number ($j.k$) & transaction ID (i) of this operation into the Damaged_PB
2.2.3.	Else If ((Record = = Write_Item]) OR (Log Record = = Overlooked_Write_Item) then
2.2.3.1.	If (T_i is Malicious) then
2.2.3.1.1.	If ($x \notin$ Damaged_DI) then
2.2.3.1.1.1.	Add x to the Damaged_DI
2.2.3.2.	Else
2.2.3.2.1.	If ((T_i is not Malicious) AND ($T_i.S^{j.k} \notin$ Damaged_PB)) then
2.2.3.2.1.1.	If ($x \in$ Damaged_DI) then Remove the entry for x from the Damaged_DI,
2.2.3.2.2.	Else
2.2.3.2.2.1.	If ((T_i is not Malicious) AND ($T_i.S^{jk} \in$ Damaged_PB)) then
2.2.3.2.2.1.1.	Add x to the Damaged_DI

TABLE 2.3 - Recovery

1.	For every block in the Damaged_PB
1.1	Get the Sub-Cluster ID from the Transaction Sub Cluster (TSC) List
1.2	Call ReEvaluate_PB (T_i,$P^{j,k}$)
2.	Flush the updated data items to the stable database
3.	Delete Damaged_DI and Damaged_PB
	Procedure ReEvaluate_PB
1.	reconstruct the Predicate Block from the sub-cluster (with the overlooked-reads and writes)
2.	Reevaluate the Predicate
2.1	For every statement in the predicate Block
2.1.1	Read all the data items x
2.1.1.1	if ($x \notin$ Damaged_DI) then
2.1.1.1.1	Read the current value from the entry
2.1.1.2	Else
2.1.1.2.1	Scan the sub-cluster to find the update record that modified x and get the old_value
2.1.1.2.2	if not found then read the value from the database
2.1.2	for write x
2.1.2.1	if ($x \in$ Damaged_DI) then
2.1.2.1.2	Update the entry with the new fresh_value
2.1.2.2	Add x to the Damaged_DI with the new fresh_value

the transaction belongs is obtained from the Transation Sub Cluster (TSC) list, then in step 1.2., each block is re-evaluated, then in step 2, the updated data items are flushed back to the database, and in step 3, the Damaged_DI and Damaged_PB are cleared and released.

In the re-evaluate procedure, the Damaged_PB is scanned, so that every block that was marked as damaged be re-evaluated. In step 2.1.1, if the operation is read (actual or overlooked) and is not affected then the current value is obtained from the log (i.e, from the sub-cluster record) step 2.1.1.1.1, else the sub-cluster is scanned to find the update record that modified x and get the old_value from that record, step 2.1.1.2.1, if there are no updates on this data item in the sub-clustered log, then the value is obtained from the database. If the operation is actual-write or overlooked-write, then if the data item is affected the Damaged_DI structure is updated with the new value, else, step 2.1.2.2, is added into the Damage_DI list with its new value, so it will be updated accordingly.

3. Experimental results

We tested the performance of our models by means of simulated environment. The simulated program develops a temporary log that holds all the committed transaction, each transaction having unique sequential ID. Then the program clusters this temporary log based on exact data dependency; in addition to sub-clustering according to the criteria (fixed number of transactions, or space occupied), and generates the Transaction Sub Cluster (TSC) and Sub Cluster Data Item (SCD) lists.

Once an attack is detected the ID of that transaction is provided to the model, the damage assessment algorithm is executed and a procedure to count the page I/O access time is started for damage assessment, and another procedure to count page I/O access time for recovery.

3.1 Performance analysis for damage assessment

The total page I/O time calculation is performed by checking the total number of pages read during damage assessment and then multiplying this number with the time required to read each page. In order to calculate the total I/O time for traditional not clustered the counting procedure considers the bytes scanned from the starting point of attack till the end of the traditional log, for traditional clustered based on data dependency; the bytes scanned from the staring point of the attacking transaction till the end of the cluster. The parameters used are shown in ***Table 3.1***.

To compute page I/O for our damage assessment algorithm, each record in the Sub Cluster Data item (SCD) List is scanned. Each Sub Cluster Data item (SCD) List record consists of the sub-cluster id, the data item id, transaction id and block number which are identified as a number; the space used for each is 4 bytes. In addition a bit is used to store the read/write, overlooked/actual and predicate/statement.

TABLE 3.1 - Parameters and their values used in page I/O Calculation

Parameters	Values
Space taken by a read operation of a transaction in the log	40 bytes
Space taken by write operation of a transcation in the log	60 bytes
Page Size	2KB
Page I/O Time (in milliseconds)	15

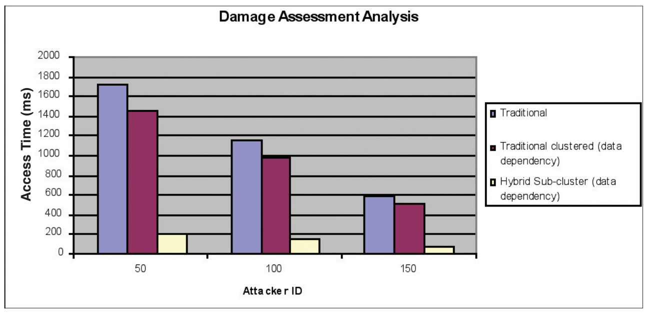

Figure 3.1 - Damage assessment time comparison, when total number of transactions is 200, total number of data items is 5000, and maximum number of data accessed by a transaction is 45.

The comparison analysis, shown in ***Figure 3.1***, confirms that our model accesses less page I/O during damage assessment, thus improving performance. To construct the graph depicted above, the attacker id was varied to 50, 100 and 150.

3.2 Performance Analysis for Recovery

To calculate the page I/O by our recovery algorithm, each record; i.e, each operation that was tagged as affected, in the sub-cluster is scanned, using the parameters shown in ***Figure 3.1***.

Figure 3.2 shows the performance of the recovery process on the log sub-clustered based on the number of committed transactions. To construct the graph the values of fixed number of the committed transactions were varied from 5 to 30 with increment of 5, the attacker id is 100 and using the parameters shown in ***Table 5.1***. Sub-clustering having a small number of committed transactions has the advantage of skipping them during recovery, thus as the number of committed transactions increases, the performance decreases.

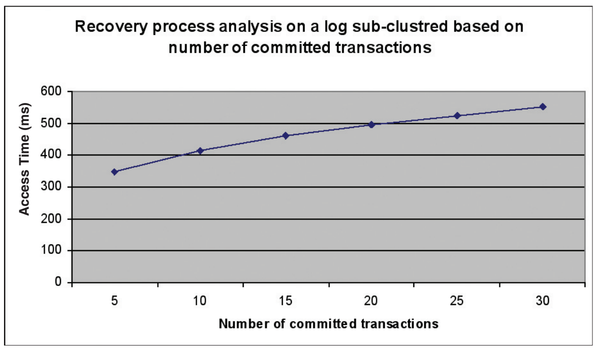

Figure 3.2 - Results of the recovery algorithm on the log sub-clustered based on fixed number of transaction.

Figure 3.3 shows the performance of the recovery process on the log sub-clustered based on the size of the sub-cluster. The values to construct the graph are the same as the previous, except that the size of the cluster varies from 5000 to 30000 with increment of 5000. As before, the performance decreases as the size of the sub-cluster increases.

Figure 3.4 shows comparison analysis of the recovery using the traditional log, traditional clustered log based on data dependency, and sub-clustered log (based on the number of committed transactions and the size of the sub-cluster). The values used to construct the graph are the same as in Table 5.1, for sub-cluster based on number of committed transaction is 20, for sub-cluster based on size is 20000 and the attacker ID is varied to 50, 100 and 150.

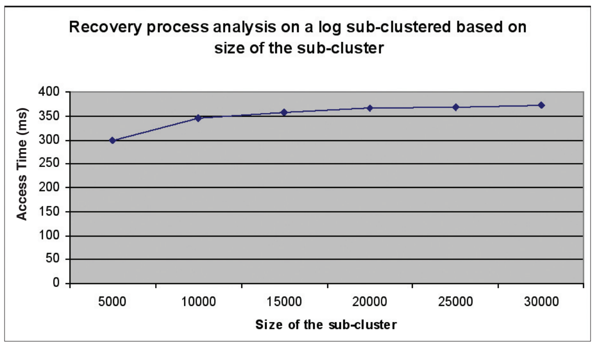

Figure 3.3 - Results of the recovery algorithm on a log sub-clustered by the size of the sub-cluster.

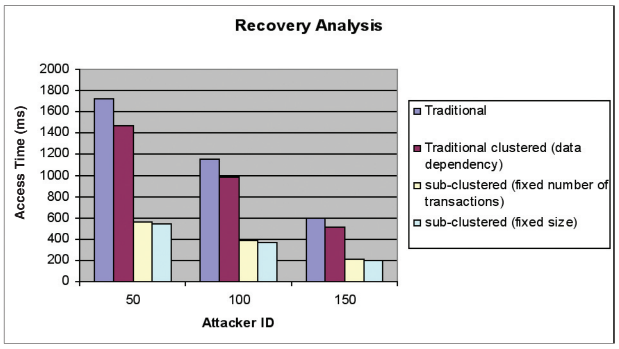

Figure 3.4 - Recovery time comparison, when total number of transactions is 200, total number of data items is 5000, and maximum number of data accessed by each transaction is 45.

4. Conclusion

Since it is extremely difficult to have security measures that guarantee the prevention of information system from attacks, it is crucial to have fast and efficient damage assessment and recovery methods. In our work, we have focused on clustering the log; based on exact/extended data dependency, into sub-clusters for faster damage assessment and recovery; also ensuring that the size of each sub-cluster was under control so the sub-clusters would not grow to humongous proportions. Through simulation we compared the performance of our logging technique; sub-clustering the clusters, with the traditional methods and the results confirm that our proposed method accelerates both the damage assessment and the recovery processes considerably. In addition, sub-clustering the clusters by size proved to be slightly better than sub-clustering with a fixed number of transactions.

As future work, our model could be combined with the matrix approach in order to achieve better results.

References

[1] Jajodia S.,McCollum C.D. and Ammann P. (1999, July). Trusted Recovery *Communications of the ACM*. 42(7). 71-75.

[2] Panda B. and Giodano J. (1999, July). Defensive Information Warfare *Communications of the ACM*. 42(7). 31-32.

[3] Elbirt A.J. (2003). Information Warfare: Are you at Risk? *IEEE Technology and Society Magazine*. 13-19.

[4] Jajodia S., Ammann P. and McCollum C.D.(1999, April) Surviving Information Warfare Attacks. IEEEComputer. 32(4). 57-63.

[5] Choy M., Leong H.V. and Wong M.H. (2000, November). Disaster Recovery Techniques for Database Systems. *Communication of the ACM*. 43(11). 272-280.

[6] Hu Y. and Panda B. (2003). Identification of Malicious Transactions in Database Systems. *Proceedings of the 7th International Database Engineering and Applications Symposium (IDEAS)*. 329-335.

[7] Ammann P., Jajodia S., Collum C. D. and Blaustein B.T.(1997, May 04_07) , Surviving Information Warfare Attacks on Databases. *Proceedings of the 1997 Symposium on Security and Privacy (S&P)*. 164- 174.

[8] Panda B. and Patnayk S. (1998, December). A recovery Model for Defensive Information Warfare. *Proceedings of the 9th International Conference on the management of Data*, 359-368.

[9] Patnaik S. and B.Panda B.(1999, November) Dependency Based Logging for Database Survivability From Hostile Transactions. *Proceedings of the 12th International Conference on Computer Applications in Industry and Engineering*.

[10] Liu P., Ammann P. and Jajodia S.(2000, January). ReWriting Histories: Recovering from Malicious Transactions. *Distributed and Parallel Databases*. 8(1). 1-14.

[11] Panda B. and Yalamanchili, R.(2001).Transaction Fusion in the Wake of Information Warfare. *Proceedings of the 2001 ACM Symposium on Applied Computing*. 242-247.

[12] Ammann P., Jajodia S. and Liu P. (2002, September-October). Recovery From Malicious Transactions. *IEEE Transactions on Knowledge and Data Engineering*. 14(5). 1167-1185.

[13] Panda B. and Giordano J. (1999) Reconstructing the Database After electronic Attacks. *Database Security XII: Status and Prospect Kluwer Academic Publishers*, 143-156.

[14] Panda B. and Giordano J. (1998, February). An Overview of Post Information Warfare Data Recovery. *Proceedings of the 1998 ACM Symposium on Applied Computing*. 253-254.

[15] Sobhan R. and Panda B. (2001, July). Reorganization of the Database Log for Information warfare Data Recovery. *Proceedings of the 15th Annual IFPI WG 11.3 Working Conference on Database and Application Security*. 121-134.

[16] Panda B. and Tripathy S. (2000, March). Data Dependency Based Logging for Defensive Information Warfare. *Proceedings of the 2000 ACM Symposium on Applied Computing*. 361-365.

[17] Tripathy S.and Panda B.(2001 June 5-6). Post-Intrusion Recovery Using Data Dependency Approach. *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*. 156-160

[18] Panda B. and Haque K.A. (2002). Extended Data Dependency approach- A Robust Way of Rebuilding Database. *Proceedings of the 2002 ACM Symposium on Applied Computing*. 446- 452.

[19] Ragothaman P. Ragothaman and B. Panda and Panda B. Analyzing Transaction Logs for Effective Damage Assessment. (2002, July). *Proceedings of the 16th Annual IFPI WG 11.3 Working Conference on Database and Application Security*. 121-134

[20] Ragothaman P. and Panda B. (2003). Hybrid Log Segmentation for Assured Damage Assessment. *Proceedings of the 2003 ACM Symposium on Applied Computing*. 522-527

[21] El Masri E. and Navathe S.B. (2000). Fundamentals of Database Systems. Third Edition. Addison-Wesley.

[22] Lala Ch. And Panda B. (2001, July). Evaluating Damage from Cyber Attacks: A Model and Analysis. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans. 31(4). 300-310.

[23] Valsankgar A. and Panda B.(2003, June). An Architecture for Making Data Available Ceaselessly During Recovery. *Proceedings of the 2003 IEEE: Workshop on Information Assurance*. 196 - 202.

[24] Hu Y. and Panda B. (2004 June 10-11).Mining Data Relationships for Database Damage Assessment in a Post Information Warfare Scenario. *Proceedings of the 2004 IEEE Workshop on Information Assurance*. 401- 409.

[25] Panda B. and Zhou J. (2003, July 16-18). Database Damage Assessment Using A Matrix Based Approach: An Intrusion Response system. *Proceedings of the 7th International Database Engineering and Applications Symposium (IDEAS '03)*. 336- 341.