LEBANESE AMERICAN UNIVERSITY

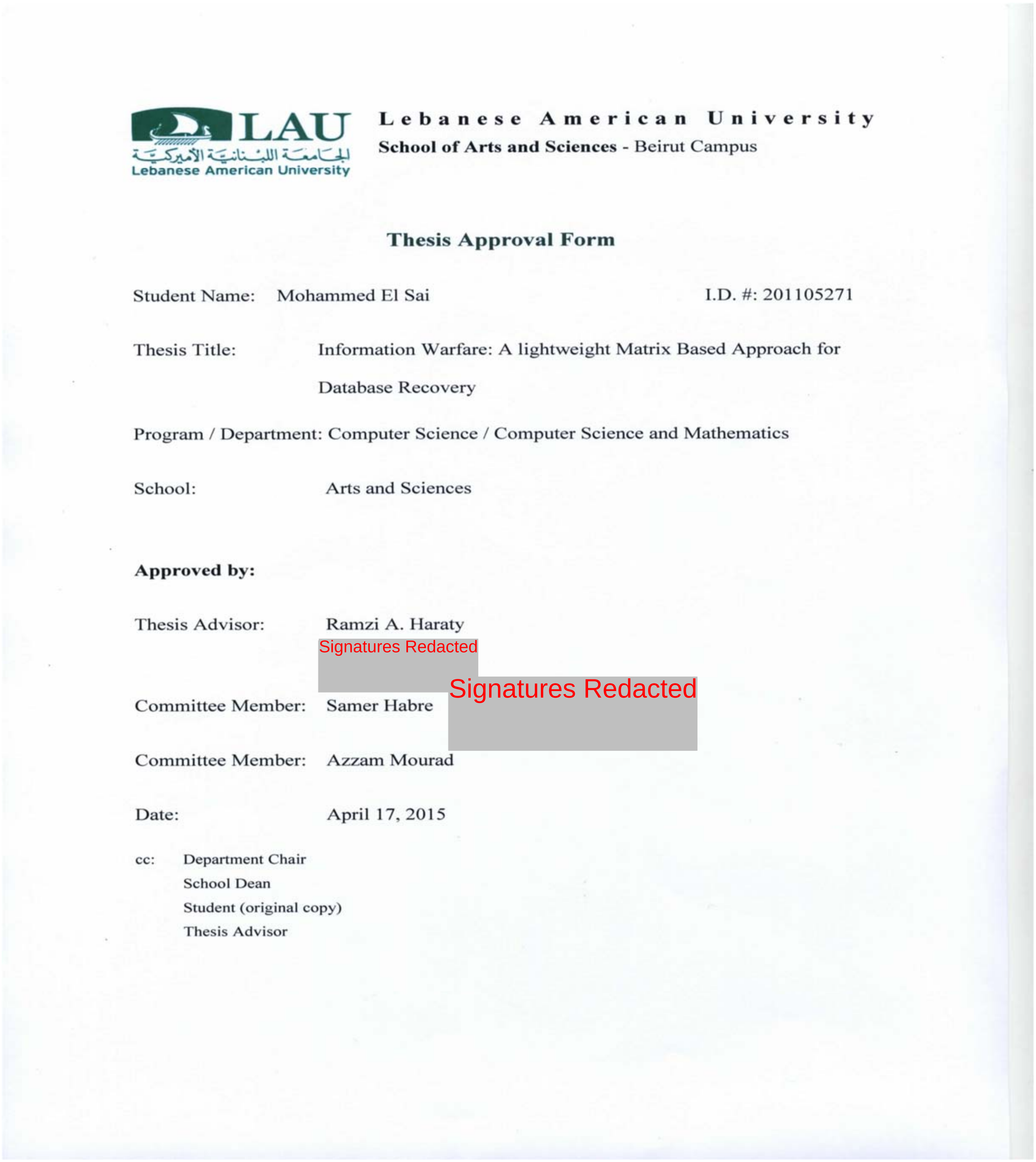
INFORMATION WARFARE: A LIGHTWEIGHT MATRIX BASED APPROACH FOR DATABASE RECOVERY

By

MOHAMED EL SAI

A thesis
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

School of Arts and Sciences
April 2015



الجـامعـة اللبـنانيـة الأميركيـة
LAU
Lebanese American University

Lebanese American University
School of Arts and Sciences - Beirut Campus

Thesis Approval Form

Student Name:	Mohammed El Sai
I.D. #:	201105271
Thesis Title:	Information Warfare: A lightweight Matrix Based Approach for Database Recovery
Program / Department:	Computer Science / Computer Science and Mathematics
School:	Arts and Sciences

Approved by:

Thesis Advisor:	Ramzi A. Haraty
Committee Member:	Samer Habre
Committee Member:	Azzam Mourad
Date:	April 17, 2015

Signatures Redacted

Signatures Redacted

cc: Department Chair
School Dean
Student (original copy)
Thesis Advisor

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

THESIS COPYRIGHT RELEASE FORM

Name: Mohamed El Sai

Signature: Signatures Redacted

Date: 17/04/2015

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that:

- I have read and understood LAU's Plagiarism Policy.
- I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.
- This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Mohamed El Sai

Signature: Signatures Redacted

Date: 17/04/2015

Dedication Page

I would like to dedicate my thesis work to my family. Special thanks for my wife who supported me all the time and provided me with the appropriate environment to be able to accomplish my work. Another special thanks for my brother-in-law who was a role model for me and a big supporter for my Masters plans. In addition, I would like to thank some of my friends who encouraged me and kept me motivated.

And finally, I wish to dedicate this work to a special and unique person, my mom, who was a great inspirational force.

ACKNOWLEDGMENT

The help and assistance of many people made this work possible. I would like to especially thank my advisor, Dr. Ramzi Haraty, for his guidance, support and patience. Thanks to Dr. Samer Haber and Dr. Azzam Mourad for being on my committee.

Finally, I would like to thank everybody who helped in this work especially during the proofreading.

INFORMATION WARFARE: A LIGHTWEIGHT MATRIX BASED APPROACH FOR DATABASE RECOVERY

MOHAMED EL SAI

ABSTRACT

The age of Internet Technology that we are in has introduced new types of attacks to new assets that didn't exist before. Databases that represent Information assets are subject to attacks that have malicious intentions such as steeling sensitive data, deleting records or violating the integrity of the database. Many counter measures have been designed and implemented to protect the databases and the information they host from attacks. While preventive measures could be overcome and detection measures could detect an attack late after damage has occurred, there is a need for a recovery algorithm that will recover the database to its correct state before the attack has taken place. Numerous damage assessment and recovery algorithms have been proposed by researchers. In this work we present an efficient lightweight detection and recovery algorithm that is based on the matrix approach and that can be used to recover from malicious attacks. In addition, we will implement the algorithm and compare its performance to other detection and recovery algorithms.

Keywords: Information Warfare, Transactions, Transactional Dependency, Data Dependency, Malicious Attacks, Recovery.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Overview

The safety of information assets is part of any online system. The data to be protected could include strictly confidential information such as financial transactions, medical records or even military and trade secrets [1]. There are three main approaches that can be used to protect any asset: prevention, detection and correction. Preventive methods include techniques such as authentication, authorization, access control, firewalls, and data encryption. Detection methods include techniques such as intrusion detection systems, checksums, and trend analysis [2]. The two main techniques of corrective methods are logging and backup.

This layered paradigm of protection is used in order to have multiple lines of defense for the system where in case one line is overcome by an attack, the following line will continue to protect the system. Preventive measures are designed to prevent attacks or at least minimize their impact but still malicious users and hackers' manage to overcome these security measures and attack the system.

When the preventive measures fail to protect the system, detection measures are there to assess and quantify the damage that occurred on the system. Since no detection technique could guarantee the immediate capture of an attack, damage would be a very probable result of the attack. The damage could take many forms such as deleted records, reading confidential data or maliciously modifying records and hence violating the integrity of the database. The main aim after detecting that damage has occurred is to clean the database and remove the effects of the attack. In such case a recovery algorithm is needed to do this job.

Due to the fact that we live in highly interconnected world on the internet and online systems level, billions of computers worldwide are connected to each other and are sharing information. The implication of this process is ease of launching an attack from any place in the world to any other place in the world. This in turn makes the idea of preventing an attack nearly impossible. Therefore a big portion of the effort of fighting back is to detect attacks (malicious transactions) as soon as possible, assessing the damage that took place and finally recovering from the attack by removing malicious transactions and their ripple effects on the database. Many approaches have been developed to enhance the recovery by new ways of integrating tables and structured data such as the one [3] named "Silk".

In the recent times, information systems along with its technologies have flooded the globe with data. One of the most important critical success factors for any organization is the ease and speed of accessing, processing and making use of information [4]. Information assets need to be defended in the domain of information warfare in a similar way to classical types of war where human beings are to be protected.

So what is meant by information warfare? In [4] it is defined as the war for dominating the ‘info-sphere’. Information warfare covers the full range of competitive information operations from destroying IT equipment to subtle perception management and from industrial espionage to marketing. Two main objectives in information warfare are:

• To use your own systems and the information associated with them to gain advantage against competitors and adversaries.

• To protect your information assets and systems from anyone who could damage them either by intent or by mistake.

There are many strategies and techniques in information warfare but all of them move around the fundamental weapon and target ‘information’. Information is leveraged and manipulated by the attacker in order to access the information of the entity being targeted. Some of the strategies that can be used in information warfare include [4]:

• Destroying or deleting data: destruction of data occurs with the physical destruction of the storage medium or the data residing on this medium.

• Denying access to data: it is the attempt to temporarily deny access to data to a point in time where it becomes useless due to the temporal dimension of data.

• Steal data: stealing confidential information from competitors such as trade or product secrets. One important thing that differentiates stealing data from stealing any other physical asset is that stealing a physical asset has a higher probability to get noticed while stealing data has a very low probability to get noticed due to the fact that the victim might still have a copy of the data on the same place from where it was read.

• Manipulation of data: data can be inserted, deleted or updated to accomplish the target of the attack. Fraud is one example of such an attack where the attacker would manipulate the data in such a way to hide his fraud.

Some of the weapons or tools that can be used in information warfare include:

• Logic bombs: These can be Trojan horses that are either present within a normal code or are independent programs. Such a weapon can be used by an employee that left his company on bad terms so s/he puts a code in their system that will get executed when certain conditions are fulfilled to run a virus or do other malicious actions.

• Computer viruses: These are code fragments that insert themselves into a program to modify it and harm the hosting system. Such weapons can be used in today's smart phones and might cause phone system failure.

• Information collection: The more the information we have about our adversary will make our position stronger. This information will inform us ahead of time about the intentions or plans of the other entity [5].

• Information degradation and denial of service: This is used to prevent the adversary from getting complete or correct information at the time the information is needed for processing [5].

• SQL injections: SQL injection is the most common methodology employed by a hacker to exploit vulnerabilities in software applications. By vulnerabilities we mean weak links in the software that exposes unauthorized data or information to a user. SQL injection occurs when the user input is incorrectly filtered for embedded SQL statements. The technique is powerful enough not only to expose the information to the user but also modify and delete the data [6].

In order to be in stronger position in the domain of information warfare, two important concepts should be understood: the information-in-warfare and the information based process [7]. To be able to defend your data and to be able to exploit the data of others, one should have a full understanding of information warfare strategies and tactics. One should not only know how to attack or defend himself/herself, but also how to gain and exploit. If we attack a system and do not gain access to it or do not exploit it, then the attack is useless.

1.2 Problem Statement

The database should be recovered as soon as an attack is detected. Unfortunately, some time would have elapsed till we detect that an attack took place. Therefore transactions following the point where the attack happened may have been affected by simply reading data written by the malicious transaction. To be on the safe side, all transactions from the point of the attack and onwards should be assessed whether they are affected or not.

There are two main paradigms that the damage assessment algorithms are based upon: transactional dependency [8] and data dependency [9]. To illustrate the idea of damage recovery, let’s consider two committed transactions, Tx and Ty where Tx is a malicious transaction and Ty as a transaction that read from Tx (i.e. Ty read a data item that was last written by Tx). Since Ty reads from Tx then we categorize Ty as an affected transaction. The recovery mechanism should retain the integrity of the database by not only deleting Tx, but also redoing Ty.

With the decreasing cost of hardware nowadays and the needs for large amounts of data to be captured for many reasons such as financial modeling and weather predictions, databases are hosting huge amount of data. This makes the process of damage assessment and recovery more time consuming. Therefore, the complexity and efficiency of the recovery algorithm is the main interest in this work. Sometimes the attack is not direct, for example what could happen after the attack finishes is the target of the attack instead. To illustrate this idea we can think of a log file of a very large size (due to the fact that it belongs to a large database with a large number of transactions taking place on it). The recovery process will have a bottle neck when trying to use this tremendous log file to recover the database. As a result, this unrealistic time to check the log file would by itself lead to denial of service. Thus, we are interested in finding an algorithm that prevents such drawbacks or at least one that reduces them. For this purpose, some researchers [8],[10], [7], and [9], have proposed using auxiliary structures for keeping track of dependencies while other researchers proposed using clusters or matrices.

1.3 Scope of the Work

In this thesis, we present a damage assessment and recovery algorithm that is based on matrices. The suggested approach keeps a single matrix data structure along with the log file of the database. This matrix saves the dependency between the transactions. When a transaction depends on some other transaction, both of these transactions will be part of the matrix. Only transactions that have a dependency on other transactions are stored in the matrix since they are the only part of the log file that is needed for the damage detection and assessment. All the needed information during the recovery phase of the process will be retrieved from this matrix. The aim of this work is for efficient and lightweight database recovery model. The matrix has been optimized in order to have a very low memory footprint by storing only the information that could be used during the detection and assessment phase of the algorithm. In addition, during the recovery phase of the algorithm we only scan part of the log file in order to clean the database from the malicious transactions and their effects.

The main contribution in our model is the use of a single matrix without any additional data structures, thus requiring less space to be stored and less time to be scanned when compared with graphs or clusters. In addition, the use of sorted linked lists in our algorithm requires less time to detect the thread of affected transactions to be recovered later in the recovery stage. Moreover, the use of only one matrix with linked lists rather than using two matrices for each of the read and write operations, makes the algorithm more scalable for the recovery of larger databases. Since the dependency between transactions is only used to detect the affected transaction, the model of dependency used in the algorithm is transactional dependency instead of data dependency. This makes the algorithm require even less space and perform better than if we use data dependency. The data stored in the matrix are only integers signifying the primary keys of the transactions that depend on each other.

1.4 Definitions

The following definitions will be used throughout the thesis:

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 $r_i[y]$ [11].

Definition 2: A *blind write* is when a transaction T_i writes data item x without reading the previous values of x [12].

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. If $x \neq I$, 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 [11].

Definition 4: If X is *totally ordered* under $\leq$, then the following statements hold for all a, b and c in X:

If $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry)

If $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity)

$a \leq b$ or $b \leq a$ (totality)

Definition 5: If a *read(x)*/*write(x)* is to be executed in a *strict execution,* it will be delayed until all *write(x)* operations are either committed or aborted [5].

Definition 6: A transaction management mechanism guarantees *rigorous*ness if the following two conditions hold [13]:

1. It guarantees strictness, and

2. No data item may be written until the transaction which previously read either commits or abort

1.5 Organization of the Thesis

The remaining chapters of the thesis are organized as follows: chapter 2 will present a review of the literature of the related research work in the area of information warfare and recovery from attacks. Chapter 3 will present the new detection assessment and recovery algorithm that we propose with an example of how it works. In chapter 4 we present and analyze the results of the experiments done with the algorithm after its implementations and doing simulations of database recovery on sample data. Our conclusion and the potential proposed future work will follow in chapter 5.

CHAPTER TWO

LITERATURE REVIEW

Due to the importance of the topic of database recovery, many studies have been done by researchers in order to find an efficient solution to detect and recover the damage caused by malicious attacks. There are two main types of algorithms used for database attack detection and recovery. The first type uses transactional dependency where there is no capture of the exact data item that causes the dependency such as the model presented in [14]. The second type uses data dependency where the data item in concern is captured. In each type several models have been proposed such as the cluster based, graph based and matrix based algorithms. An overview of the available literature about these models will be provided in this section.

2.1 Traditional Methods

The main approach that is used in traditional approaches is to scan the log file from the point where the attack happened until the end of the file to do the recovery by undoing malicious transactions and redoing the affected ones and this approach focuses on the complete rollback of the database. Bai and Liu [15] presented a new traditional recovery based algorithm. Unlike other traditional recovery paradigms, in this paper they proposed a lightweight dynamic Data Damage Tracking, Quarantine, and Recovery (DTQR) method which can sustain an excellent data service while healing the database server when it is under a malicious attack.

To overcome the limitations of other approaches such as substantial run time and long system downtime, they have suggested a new recovery approach named ‘Trace’ which is a zero system downtime database data damage tracking quarantine and recovery solution. This approach is based on the following three pillars:

1. Cleaning the compromised data on-the-fly
2. Avoid blocking read only transaction by having multiple copies
3. For the read-write transactions, the detection and recovery are done simultaneously in order to minimize the overall recovery time

Trace uses an advanced tagging technique that is proved to be efficient in tracking affected transactions without the need for logging the read operations.

The Trace approach has two working modes: standby mode and cleansing mode. The chosen mode of the Trace depends on the current situation of the database and more precisely on the fact that there is an attack on the database or not. In both case the Trace depends on an intrusion detection to notify it about any attack or data corruption risk present on the database. In case no attack is reported by the intrusion detection system, Trace works in the standby mode. Whenever an attack is reported, Trace will be activated and shifts to the cleansing mode where it executes the quarantine, assessment, and cleansing procedures.

2.2 Using Graphs for Recovery

In [12] Zou and Panda developed a set of graph based models for damage assessment. These models assume that the log files are not damaged in an attack and there are no blind writes in the database system. The result that is generated from these models is the set of affected transactions that will later be submitted to the recovery model that will do the necessary steps to roll back the database to the previous state of integrity. While the paper focuses on the detection part (leaving the recovery to other papers), it uses a more general database that is distributed in nature and has many sites such that each one of these sites has a local manager to coordinate with the other sites or the central coordinator (depending on the specific model used). The two main assessment models presented in [12] are the centralized and the peer-to-peer.

In the peer-to-peer model there is no central coordinator to help in the detection process. Here, each local manager is responsible for its local log file and scanning this local file in case there is a need for that. In the case where a local manager suspects about affected transactions at the global level, it will multicast the identifiers of these transactions to each site that has a sub-transaction of the affected transaction executed. After that, each receiving site will scan its local log file from the point where the first affected transaction happened to detect if there are any other affected transactions. Then the newly discovered information (about any new detected affected transactions) will be sent to all the relevant sites to continue with the detection process.

On the other hand the centralized model requires the presence of a central coordinator for assessing the damage after an attack. A voting exercise can be used to select the coordinator where this coordinator to be elected it should have these features:

1. Located at the most convenient location distance wise from the other sites.
2. Equipped with processing capabilities to enables it to perform its coordination role.

3. Connected to the other distributes sites using a fast link that enable the fast transmission of messages for database damage detection purposes.
4. There should be some sort of cluster or back of the coordinator machine is case the machine fails for some reason or another.

The centralized model is divided further into three sub models that are listed below:

1. Receive and forward model
2. Local dependency graph model
3. Central graph repository model

In [10] Ammann, Jajodia and Liu present a graph based model that focuses more on the recovery and repair of databases after an attack happens. This recovery model can be divided into two categories, a cold-start approach where the database would be unavailable during the recovery and a warm-start approach where normal execution may continue with some degradation in the quality of service. The recovery of the database in [10] assumes that the history of transactions is Serializable with a classical transaction processing model. In [10] there are two sets of transactions, set B $\{B_1, B_2 \ldots B_n\}$ of bad or undesirable committed transactions and set G $\{G_1, G_2 \ldots G_n\}$ of good or desirable committed transactions. The main data structure used in this model is the dependency graph where the previously identified bad and good transactions are put into a hierarchal shaped graph to illustrate which transactions depend on each other to facilitate the recovery process.

2.3 Clusters and Sub clusters

The main idea behind clusters is the segmentation of the log file. The sub clustering approach takes the previously clustered log file a step further by clustering it more mainly for size considerations. In [16] Ragothaman and Panda suggested a limitation on the size of the cluster. These limitations could be the number of committed transactions, the size of the cluster and the time window of the cluster.

In [7] Haraty and Zeitunlian presented a log clustering algorithm based on exact data dependency. The further sub clustering of clusters is done in two ways: number of data items or space occupied by the cluster. The proposed model assumes the following in order to be able to work properly:

1. The existence of an intrusion detection system that detects the attacking/malicious transactions.
2. The database schedule is modified to produce customized types of read and write operations.
3. The execution history is rigorously Serializable.
4. The clustered log will contain only committed transactions
5. Transaction ID's are sequentially incremented starting from T_1 and incrementing by 1 for each next transaction.

The two data structures used in [7] are the *transaction sub cluster list* and the *sub cluster data list*. The first list is used to store the transaction ID and the sub cluster in which the data items of the transaction are stored. On the other hand, the second list is used to store the sub cluster ID, the transaction ID and the corresponding data item that could be a read, write, actual read, overlooked read, predicate or statement. In this model the clusters and sub clusters are determined periodically once the list of committed transactions is obtained from the temporary log. This is done as mentioned before either by the number of committed transactions in the cluster or the size of the cluster.

Once the intrusion detection system detects a set of malicious transactions, the *transaction sub clusters list* and the *cluster/sub cluster data list* are checked in order to detect the affected transactions. This step of checking the lists represent the detection part of the algorithm where after it the recovery phase can start by scanning only the sub clusters that contain the damaged data items. The detection part of the model uses two additional data structures: Damaged_DI, which is the list of data items that are affected by the malicious transactions and Damaged_PB, which contains the predicate block of a transaction T_i that has been affected by a malicious transaction. These two data structures are first initialized to null. Later on, every transaction in the *transaction sub cluster* starting from the point of attack is checked to detect the affected transactions.

In the recovery phase of the algorithm these are the steps to be done:

1. Scan all the Damaged_PB records that resulted from the assessment part of the model
2. The sub cluster in which the transaction belongs is obtained from the *transaction sub cluster list*.
3. Each block is re-evaluated
4. The updated data items are flushed back to the database
5. Damaged_DI and Damaged_PB are cleared and released.

Another cluster based algorithm is presented in [17], where Sobhan and Panda proposed a new damage assessment and recovery algorithm that is based on a new logging protocol that records all needed information for the repair of the database. This model defines a cluster based on predicates and their following statements and calls it *Predicate Statement Block*. A predicate is a precondition with a set of statements where the predicate condition that must evaluate to true for the statements to be executed. The predicate block can be either conditional or unconditional where in the first type there are two explicit predicate one on each branch, while in the second type there is only one predicate.

The model in [17] has a set of assumptions in order to be able to perform assessment and recovery:

1. The model uses a log sequence number (LSN)
2. The model follows the write-ahead-logging protocol
3. The database system follows the Steal/No Force protocol
4. The use of cache consistent check pointing
5. The history produced by the scheduler is rigorous Serializable
6. The log cannot be modified by the users and no part of this log is purged
7. Nested transaction are not allowed
8. A transaction writes a data item into a stable database only

2.4 Before Images

In [18], Xie et al presented a recovery approach that can track the damage in a more complete way by using “before image” tables. In their model they adopted a new type of before image table which is in general a table that is transparent to the users and is similar to the normal database tables. The before image table is a replica of the base table in the database but without the integrity constraints that are set on the base table. The before image tries to preserve the history of the transactions so that at any point in time we can roll back to the last state of integrity. Whenever a certain row is deleted or updated in the base table, a copy of the row’s previous/old data is added to the before image table of that base table. In order to prevent the before image table to grow exponentially in size, there is a defined time window for the data in the before image to be out of date and hence deleted.

The approach used in [18] is based on the inter-transaction dependency graph that stores the inter-transaction dependency. In addition to the graph, the model adds two data items for each data item in the database. These additional data items are *x.ins_tran* that captures the last transaction that wrote the data item x and *x.del_tran* that represents the last transaction that deleted the data item. An additional table is used to capture the inter-transaction dependencies. This table is called *TranDepTab* and has these three columns:

1. *commit_ord*: the order in which *dependent_tran* has committed
2. *dependent_tran*: represents the transaction depending on other transaction,
3. *precursor_tran*: represents the transaction being depended by *dependent_tran*

The algorithm in [18] is invoked whenever an intrusion is detected. The recovery of the database is done in two phases:

1. detect all the transactions that need to be undone
2. delete the effect these transactions had on the database

The major contribution of this model is that the detection of the affected transactions is much easier. This is due to the presence of the *TranDepTab* table that captures all the dependencies between the transactions in the form of a graph.

2.5 Column Dependency

In [19] the authors presented a column-dependency based model that is used to detect the affected transactions in order to recover the database from the affected and malicious transactions. This model is based on the transactional dependency approach. Moreover, the authors presented two versions of the algorithms a static recovery scheme and an online recovery scheme. The proposed static recovery algorithm has a set of inputs and a set of outputs and performs the recovery in two phases. The inputs of the algorithm include the following:

1. The set of committed transactions
2. The schedule of execution of the above committed transactions
3. The list of malicious transactions which could be provided for example by an intrusion detection system

The output of the recovery algorithm is a consistent database that has been cleaned from the effects of the malicious transaction that were part of the input of the algorithm.

While the static recovery algorithm halts the database during the recovery period, the online recovery version allows the active transactions in the database to continue execution and the new transaction to start executing. One of the main differences between the two recovery schemes is that the vulnerability window in the online version is not fixed because transactions continue to execute and hence the vulnerability of having new malicious transactions is larger. The online algorithm performs the recovery in three phases (instead of two in comparison to the static scheme). These three phases are:

1. Assessment phase
2. Recovery phase
3. Confinement phase

2.6 Matrices in Recovery

In [4], Haraty and Zbib presented a matrix based damage assessment and recovery algorithm that is used to recover from malicious attacks that are part of the information warfare. This model assumes that there is an external intrusion detection system that is responsible for the detection of the malicious transactions a reporting them to the proposed model and that the execution history is rigorously Serializable. The authors of [4] use check points on the version of the log file in order to prevent the log file from growing tremendously in size and hence lowering the performance of their model.

The main data structure in this model is the dependency matrix that is treated as a log file where it will be flushed at certain check points when the size grows above a certain limit. This matrix is a two dimensional array where the columns represents the set of data items present in all the tables of the database and the rows represent all the committed transactions that occurred up to certain point in time. Any data item x could have three types of interactions with a transaction T:

1. x could be blindly written by T
2. x could be modified according to one or more committed transactions
3. x could be left unmodified by T (for example T read x)

According to the above scenarios, the dependency matrix is filled by 00, 01 or $-T_i$ to reflect how the transaction interacted with the data item. In order to solve the problem of saving the information about the set of transactions that affected a data item (case 2 above), an additional data structure will be used to store this data since the dependency matrix has only one entry per each transaction and data item. This data structure is also a two dimensional array that holds the transactions that affected a certain data item to be used in the detection phase.

The detection part of the algorithm works by traversing the dependency matrix row by row from the transaction that followed the first malicious transaction. While scanning the matrix, if the algorithm finds a 01 it will check if it is one of the malicious or affected transactions. In case a negative value is found in the matrix the respective transactions are retrieved from the secondary matrix and checked to in the same way to see it they have malicious or affected transactions. Any transaction that is found to be affected is added to a third data structure that will hold all the affected transactions to be recovered from in the recovery phase.

In the recovery phase of the model there will be an additional data structure that will be used to read the log file. When the detection algorithm finds all the affected transactions in the database, the recovery phase is started by sending the set of malicious and affected transactions found. The recovery algorithm will make sure to delete (undo) the malicious transactions from the log file and to redo the affected transactions to make that they execute properly without any interference from the malicious transactions that would have been deleted by that time. In [4], the authors presented a fast detection and recovery algorithm that is based on a simple matrix data structure but on the other hand the algorithm uses more than one data structure to do the detection and recovery and hence consumes a big amount of memory that can be reduced.

In [20], Zhou presented a new database recovery model the uses pre-developed data structures affected transactions and data items without accessing the log file. The data structures used in [20] are built using bit vectors which are manipulated using the logical AND and OR operations. Zhou presented two models in his paper; a base model that uses transactional dependency to identify affected transactions and later on uses this information to identify the damaged data of those affected transactions. The second model uses the previous base model and adds parallelism to it in order to enhance the performance of the damage assessment process.

The base model assumes the following in order to function correctly:

1. Presence of some sort of intrusion detection system that forwards the malicious transactions list to the base model
2. The execution history is rigorously Serializable

3. The database log cannot be corrupted
4. Any data item is not updated twice by any transaction
5. The dependency relationships between transactions will not change during recovery

The four data structures of the base model are:

1. Read_Matrix: it stores information about the data items that have been read by transactions
2. Write_Matrix: it stores information about data items that have been written by transactions. The values in the first two matrices are bit number (0's and 1's).
3. Damaged_Data_Vector (DDV): it stores the data items that have been identified as damaged during the assessment phase
4. Damaged_Transaction_List (DTL): it stores all the transactions that have been identified as damaged

The parallel damage assessment version of the model partitions the transactions into clusters based on their relationships with each other. This will help in making the damage assessment and recovery more efficient because in large databases many transactions are unrelated to each other.

2.7 Fuzzy dependency

The fuzzy dependency approach presented in [21] is a loose dependency relationship between two sets of attributes. Zou in his paper [21] defined the fuzzy dependency as follows:

"For two sets of attributes X and Y of a relation R, Y is fuzzily dependent on X (or X fuzzily determines Y) if and only if for every value a_i in the domain of X, a_i belongs to Domain(X), there is an uniquely determined subset S_i' in the domain of Y, Si' is a subset in Domain(Y), such that a tuple T_i in a relation instance of R with value a_i for X should have a value b_i belong Si' for Y".

The fuzzy dependency model can be used in three different ways:

1. Constraint specification: specifying a constraint on possible tuples that can form an instance r of a relation R and to enforce database policies such as checking the integrity of the fuzzy relationship. This will ensure that database users don't mistakenly enter illegal data.
2. Intrusion detection: fuzzy dependency can be run periodically on a database to check the integrity of the database and detection any intrusion by malicious transactions.
3. Minimizing the DOS (denial of service): By generating fuzzy values for the damaged data in the database due to malicious transactions, fuzzy dependency minimizes the possibility of having a denial of service due to an attack.

The suggested recovery algorithm consists mainly of the *"Fuzzy Value Generator"* that interacts with the database and the "Fuzzy dependency storage". This architecture consists of fuzzy check, data fetch unit, fuzzy reasoning unit, supplemental fuzzy rule unit, usage identifier, value finalizer and a supplemental reasoning unit. The key requirement of the model is that the fuzzy value generator should be very fast in comparison to other conventional database recovery techniques.

The main contribution of the authors' model in [21] is that it allows a fast recovery after an attack due to the fact that unlike other traditional database recovery models, the proposed model does not require intensive search of the database logs based on data or transactional dependencies. On the other hand, the proposed fuzzy model must be supplemented with semantic reasoning because it cannot guarantee total accuracy.

2.8 Transaction Fusion

In [22] Chen et al presented a new database recovery model that skips the unnecessary steps done in the traditional models. While the traditional models will undo all the malicious and affected transactions and then redo all the affected transactions again, the model presented in [22] will assess the need for recovery through a proposed recovery approach and then later on fuse malicious transaction and valuable affected transactions. By skipping unnecessary steps, the proposed model will have a better execution time. The aim of the Real-Time Database Recovery Algorithm with Transaction Fusion (RTDBS) is to achieve partial and timely information more than to gain absolutely correct but outdated information**.**

The model presented in [22] significantly reduce the number the total number of transactions that will be undone or redone. This way the recovery can be done more efficiently by removing unnecessary steps and minimizing the I/O operations and log scanning. Even though the RTDBS model reduces the time for the recovery, it does that at the cost of having partial recovery instead of recovering the whole damaged scope since it uses an assessment approach to select which affected transactions to redo.

2.9 Selective Recovery

A new database recovery model is presented in [13]. Xia et al in [13] presented a self-healing model that is based on transactional dependency and performs selective recovery by undoing only suspect transactions in order to reduce the effect of those suspect transactions as much as possible. The proposed model performs the following steps in order to recover the database:

1. Find the suspected transaction
2. Generate the Undo Transaction Set (UTS)
3. Undo each transaction in the UTS
4. Redo unfinished transactions

The model uses a transaction reverse dependency log as (DepID, TranNum, TranID1, TranID2, TranID3, … TranIDn) and it logs the transactional dependency between the database transactions. Here DepID is the ID of the transaction that the other transactions (TranID1 → TranIDn) depend on and TranNum is the number of those transactions. For example if we have (T1, 2, T2, T3), it means that there are two transactions (T2 and T3) that depend on T1. In addition, the model uses the transaction operation log which logs the different data modification operations done by the transactions. This log is represented by this structure (TranID, TabID, Attrs, TupleNo, OldVal, NewVal) where TranID is the transaction numbers and the TabID is the ID of the table in which the transaction operated and modified the fields listed in Attrs of the row with number TupleNo from the previous value OldVal to the new value NewVal. The transaction reverse dependency log (TRDL) is used to generate the undo transaction set (UTS) through two approaches; a recursive algorithm or a stack based algorithm. The concept that this model presents describes the TDRL as a set of trees and the UTS as a member tree and performing recovery on the database is like tree traversal.

2.10 Distributed Recovery

A recent research in [17] presented a new model for the recovery of the heterogeneous distributed databases. The methodology of the proposed model is on a hard limit of failure tolerant databases and is named *failure evaluation and patch* (FEP). The responsibility of failure evaluation and patch (R-FEP) is to find the malicious and affected transactions and recover the database from their effects. The authors' definition of a distributed database is that it's simply a collection of data which belong logically to the same system but are spread over a dispersed network of computers. In this architecture, transactions have global unique identifiers that indicate takes into consideration each site. The model has the following assumptions:

1. The execution history is Serializable with two-phase locking (2PL) protocol
2. For the atomicity of the transactions, the authors use the two-phase commit (2PC) protocol

The failure can be only caused by all committed transactions. Therefore, the set of committed malevolent transactions is M = {M1, M2, M3... Mn} and the set of good conditioned transactions is C= {C1, C2, C3….Cm}. At any site s, a committed transaction T_i is dependent upon T_j if a data item 'd' is stored at that site such that T_i reads 'd' after T_j updates 'd', and no other transaction updates 'd' between the time. In order for the FEP to work, the authors assume the presence of a local transaction manager (LTM) at each site and a FEP module at each site. The recovery is done through local operations at the sites where the FEP module instructs the LTM to perform some critical operations. The FEP is responsible for knowing the local log in order to locate the sub transactions that are affected by a malicious transaction and to perform the cleaning process for these kinds of transactions. On the other hand, the LTM is responsible for only coordinating the FEP process for the distributed transactions which are having their coordinators at the sites. This coordination between the FEP and the LTM modules is what makes the proposed model work. The patching process is done by the LTM modules where the FEP module only sends the transactions to be patched to the LTM module.

CHAPTER THREE

THE MODEL

3.1 Overview

In this chapter, we present a new detection and recovery model that has been optimized to have high accuracy and efficiency in the process of assessment and recovery from malicious transactions resulting from attacks. The first stage of this algorithm is to detect the effects of the malicious transactions after being initiated with a call that has the list of malicious transactions as input to the call. This step mainly tries to find all the affected transactions (transactions that read from malicious or affected transactions). Finally, the algorithm will recover the database to a consistent state by undoing the malicious transactions and redoing the affected transactions.

3.2 Assumptions

As mentioned in the previous section, the proposed model will receive a set of malicious transactions that will trigger the execution of the algorithm. So the model assumes that there is some sort of an intrusion detection system that determines the set of malicious transactions. After receiving the set of malicious transactions, the algorithm will guarantee the efficient detection and recovery of the database.

Serial execution ensures a Serializable history. In such a history, every transaction is assumed to be correct as it would be depending on the committed transactions only.

Hence, Serializability provides correctness [5]. An update to any transaction in the system will be represented in our model as if it is a new insert transaction. In our proposed model, we assume we have a rigorous Serializable history. A sequential log file is also maintained in which only committed transactions are saved. This log file cannot be accessed by the users at all times and it will be used during recovery.

In our model we use only one matrix without any other data structure that supports it. Panda and Zhou in [23] had two different matrices one for the write operations and another for the read operations in addition to two other supporting data structure to do the recovery. In addition, they used logical operations between the matrices to discover dependencies. In our algorithm we only use one matrix without any logical operations. In addition Haraty and Zbib in [24] used two data structure in their model. First data structure in the core matrix that stores the dependability of a transaction in relation to a data item. While the second data structure, the secondary structure in used to store the specific transactions 'y' that a transaction 'x' depends on.

Sometimes the attacker has indirect intentions when launching the attack. Maybe s/he does not want to steel confidential data or delete records from the database and s/he plans to target the detection and recovery mechanism by itself. The recovery procedure may take a long time to recover the database and during this time the database may be set to inaccessible mode for the users that has legitimate transactions to execute on the database. As a result a situation of denial of service would occur. In order to prevent or reduce the possibility of such a scenario, our algorithm will be very efficient in terms of time and memory space when recovering the database. Therefore denial of service due to the recovery process would be less probable with a fast recovery algorithm that requires low memory space and uses one data structure to store the dependencies and recover the database.

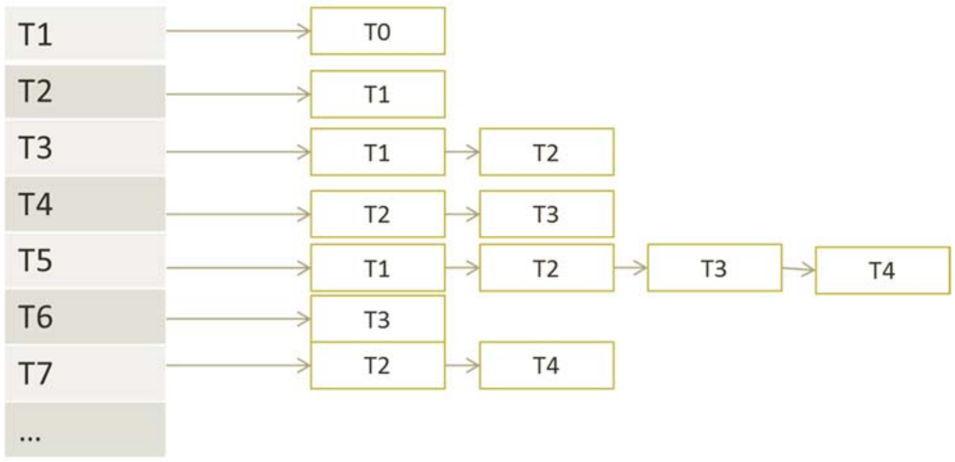
3.3 The Matrix

In this section we will discuss the structure of our algorithm's matrix. We assume that this matrix is built dynamically along with the execution of every transaction. Only committed transactions are inserted into our matrix. The matrix will be an array of sorted linked lists. Each data item will either be blindly written by the transaction, left unmodified by the transaction, modified according to one previously committed transaction, or modified according to a set of previously committed transactions. Our matrix only takes into consideration the transactions that have modified data according to a set of one or more previously committed transactions.

The entries in the array will represent the set of transactions that have modified data items in the database after reading data from other previously committed transaction(s). This is translated to having the ID of the transaction included in the array as an array index. The actual entries that are stored at a specific index of the array are the linked lists of transaction IDs that the transaction with ID equal to the array index have read from. So for example, if we have a transaction with ID = 200 and this transaction modified any data item after reading from transactions with IDs 150 and 151, then the matrix will have at index 200 a sorted linked list containing IDs 150 and 151 respectively.

Using the above approach makes sure that the data needed for detection and recovery is captured in the matrix. In addition, only the needed information is saved in one consolidated data structure without the need for any other secondary or tertiary structure like other algorithms. The proposed algorithm does not take into consideration which specific data item has been read from other transactions it only cares about the fact that transaction y has read data from transaction x and hence y depends on x. The specific details of the dependency between transaction y and x are not needed for the detection or recovery parts because if transaction x is malicious, the recovery phase will undo it and redo transaction y in all cases. The below figure depicts the matrix that is used in our model.

Figure 3.1 Dependency Matrix



3.4 Detection Algorithm

As we have previously mentioned, we assume the presence of an intrusion detection system that will generate the list of malicious transactions. The detection phase starts by finding the malicious transaction with the smallest ID. Since we have a rigorously Serializable history, we will not face the case where we have a transaction T_j such that $j < i$ and T_j depends on T_i. Our algorithm starts the detection from the next transaction after the smallest malicious ID. So for example if the smallest ID in the malicious transactions list is 100, we start checking if a certain transaction is affected from first transaction with ID>100 and present in the matrix. This reduces the detection time since we will not check older transactions that have committed before the first malicious transaction and therefore cannot be affected due to the rigorous Serializable history.

Starting from the first transaction bigger than the smallest ID in the malicious list and till the end of the matrix, the algorithm tries to find all the directly or indirectly affected transactions. A transaction can be affected in two ways directly affected if it has read data from a malicious transaction or indirectly affected by reading data from an affected transaction. The algorithm will traverse the linked list found at the inspected array entry for a number of times equal to the total number of entries in the list of malicious transactions and the list of affected transactions. Since the linked list is a sorted one, the execution will stop when we find in the linked list an ID equal to the ID of the malicious/affected transaction that we are looking for or when we reach an ID greater than the ID of the malicious/affected transaction that we looking for. Having a sorted linked list saves execution time and enhances performance of the algorithm since not every time we are going to go through the entire linked list.

After checking the linked list at the specific array entry and we find the malicious/affected transaction that we are looking for, we add the index of the array entry to the list of affected transactions. If we have not found the malicious/affected transaction we move to the next entry in the matrix array.

For example, if the set of malicious transactions is *{ T_a, T_b, T_c}* and the set of affected transactions is *{ T_d, T_e, T_f}* and the detection algorithm is running and it reached transaction T_m, the algorithm will check if any of the above malicious transactions is present in the linked list at entry T_m in the array. If any of the malicious transactions *{ T_a, T_b, T_c}* is found in the linked list of T_m, them T_m is added to the list of affected transactions. If T_m does not has any of the above mentioned malicious transaction IDs in its linked list then it does not imply that T_m is clean because it may be indirectly affected by other affected transactions. So the next step is to check T_m against the affected transactions list if any of the *{ T_d, T_e, T_f}* IDs is found in T_m linked list then we will add T_m to the affected transactions list.

3.5 Example on the Detection Process

Consider a database for a health management system that mainly manages the process of prescribing medications to patients by doctors. It contains information about the following:

- Doctors: a unique identification number for each doctor (DID), name (DName), major (DMajor), and years of experience (DExperience).
- Patients: a unique identification number for each patient (PID), name (PName), date of birth (PDOB), and address (PAddress).
- Types: a unique identification number for each medication type (TID) and type name (TName).
- Medication: a unique identification number for each medication (MID), medication name (MName), and medication type (MType).
- Prescription: a unique identification number for each prescription (PrID), the patient to whom this prescription belongs (PID), the doctor that gave this prescription (DID), and the date of the prescription (PrDate).
- Prescription_details: a unique number that identifies each Prescription_details entry (PDID), the ID of the prescription that this prescription_detail belongs (PrID), the number of times to take the prescription per day (PDfrequency), and the duration in days of the treatment (PDDays).

Consider the following transactions in the database stated above:

T_1 = Doctors (‘1’, ‘Peter’, ‘Heart & Blood Vessels’,’10’);

T_2 = Types (‘4’, ‘Heart Medication’);

T_3 = Medication (‘11’, ‘Aspirin’, ‘4’);

T_4 = Types (‘8’, ‘Pain Killers’);

T_5 = Medication (‘9’, ‘Panadol’, ‘8’);

T_6 = Doctors (‘7’, ‘Robert’, ‘Internal Medicine’, ‘15’);

T_7 = Patients (‘14’, ‘Helen’, ‘1/1/1980’, “LA”);

T_8 = Patients (‘3’, ‘George’, ‘9/10/1982’, ‘Pennsylvania’);

T_9 = Prescription (‘1’, ‘3’, ‘7’, ‘1/1/2015’);

T_{10} = Prescription (‘2’, ‘14’, ‘1’, ‘31/1/2015’);

T_{11} = Prescription (‘3’, ‘3’, ‘1’, ‘15/2/2015’);

T_{12} = Prescription_details (‘1’, ‘1’, ‘9’, ‘1’, ‘5’);

T_{13} = Prescription_details (‘2’, ‘2’, ‘9’, ‘2’, ‘7’);

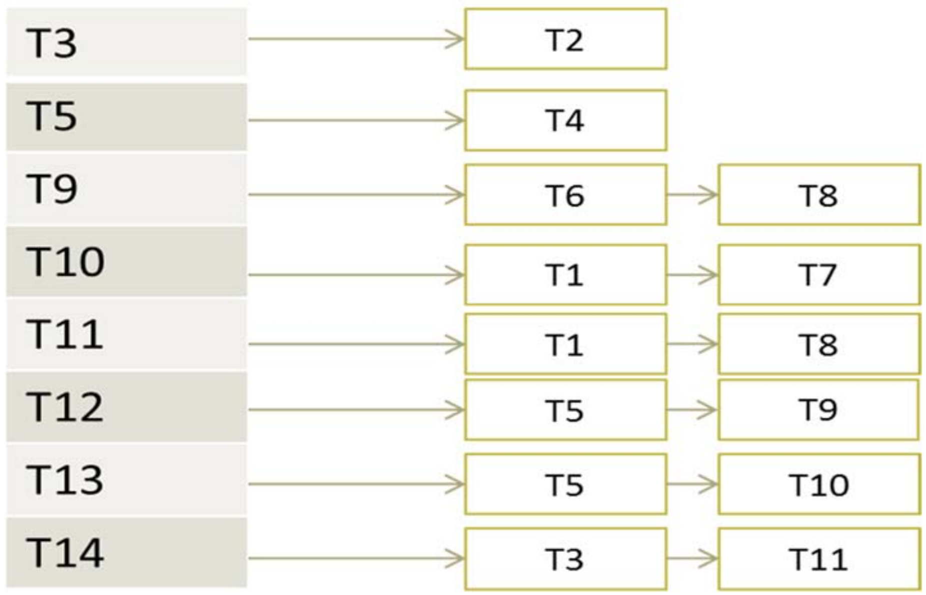
T_{14} = Prescription_details (‘3’, ‘3’, ‘11’, ‘3’, ‘14’);

Let the dependency matrix that corresponds to this database be called *M*. We assume that the transactions occur in the order presented above. This matrix will be made up as the transactions occur one after the other. When T_1 occurs it does not get added to the matrix since it does not depend on any other transaction that happened before it. The same case applies with T_2. When T_3 occurs, it will be the first transaction to be added to *M* since it depends (reads from) other transactions (T_2), therefor both T_3

and T_2 get added to the matrix with T_3 pointing to T_2 since it reads from it (TypeID). T4 is a standalone transaction so it doesn't get added to the matrix. Same case as T_3 happens with T_5 since it reads from T_4, so both T_5 and T_4 are added to the matrix. Each of T_6, T_7, and T_8 are independent transactions so they don't affect the contents of the matrix. The orders table depends on another two tables from which it reads data from (Patients table and Doctors table). T_9 depends on T_8 and T_6, T_{10} depends on T_7 and T_1, and T_{11} depends on T_8 and T_1. As a result we will have three rows added to the matrix; one with T_9, T_8, T_6 and one for T_{10}, T_7, T_1 and one for T_{11}, T_8, and T_1. Prescription_details table depends on both prescription and medication tables. When T12 takes place it reads from T9 and T5, while when T13 occurs it depends on T10 and T5 and when T14 occurs it depends on T11 and T3. We will then have three rows added to the matrix (T12, T9, T5 + T13, T10, T5 + T14, T11, T3).

Matrix *M* is presented in the below figure:

Figure 3.2 Matrix *M* for the presented example



As we can see in the above figure that the entries in each row are sorted in increasing order of transaction number. This will enhance the performance of our algorithm by enhancing the detection part where not all entries in each row will be checked. For example if we are searching for T_6 in the row headed by transaction T_{10}, we first come across T_1 which is not equal to T_6 or greater than it so we continue forward to reach T_7 which is not equal to T_6 but is greater than T_6 so we stop the execution of the loop here because we are sure that T_6 is not present in the specific rows we are searching at. In a nutshell, the matrix used in the detection algorithm is made of rows and columns where the first column (column-1) represents all the transactions that depend on other transaction, while the successive columns (columns-2 $\rightarrow$ column-n) represent the specific transactions that column-1 depends on.

Now that we have seen the matrix structure and how it is filled from a sample execution of database transactions, we can move to the part of using the populated matrix in the detection of malicious transactions effects. The trigger for the start of our algorithm is the occurrence of a malicious attack that is detected by an external intrusion detection system. The detection part of the algorithm uses the populated matrix in order to find the list of affected transactions that needs correction in order to roll back the database to a state of integrity. This will insure that later on in the recovery phase of the algorithm, malicious transactions are deleted (as if they never happened) and affected transactions are redone.

For illustration purposes, consider that the first transaction in the list of malicious transactions is T_6. As we have said before, there is an intrusion detection system that will notify our algorithm about the list of malicious transactions. We have chosen one transaction to be malicious for simplicity and in case there was more than one transaction, the detection steps to be presented below will be repeated for each malicious transaction in the list. Due to our assumption of a rigorous Serializable history, only transactions that happened after T_6 could be affected and therefore need to be examined. The detection algorithm will start the execution by examining the next transaction greater the T_6 and since the current matrix includes neither T_7 nor T_8 in its first column, we start the search at the first transaction that is greater than T_6 and exists in the matrix. This transaction is T_9. The detection algorithm will check the list of transactions on which T_9 depends on to see if T_6 is one of them. T_6 is one of the transactions that T_9 depends on so T_9 is an affected transaction and we add it to the list of affected transactions that need to be redone in the recovery stage of the algorithm. An important thing to mention here is that our algorithm enhances the detection phase performance by starting its detection from the first transaction that is greater than the smallest malicious transaction. In our example we skipped transactions smaller than T_6 (T_1 to T_5). In real databases our algorithm is highly scalable by skipping unnecessary rows checks.

There are two main data structures used in the algorithm to keep track of the malicious transactions and the affected transactions. The malicious transactions data structure will remain fixed but the affected transactions data structure will most probably grow while running the detections part of the algorithm. For the detection purposes, both the malicious and the affected transactions will be treated the same. This means that during the detection phase the algorithm takes each transaction in the malicious or affected sets and check the potential transactions if they affected or not. The malicious set is $\{T_6\}$ and the affected transactions set is $\{T_9\}$. Continuing with our detection part we check the remaining part of the database and confirm that no other transaction depends on T_6. This leaves the affected transactions set as $\{T_9\}$. Other transactions could be indirectly affected; this case happens when a transaction does not read directly from a malicious transaction but from an affected one. Our algorithm only uses the matrix structure to check for affected transactions and no need for other data structures like other algorithms.

The algorithm will continue the execution till the end of the database to find all the affected transactions. Now we move to transaction T_9 to check if any transaction depends on it. We start from the first transaction that is greater than T_9, T_{10} but T_{10} does not depend on T_9 so we move forward in the database till we reach T_{12} which depends on T_9. So T_{12} is now added to the list of affected transaction to become $\{T_9, T_{12}\}$. No other transaction reads from T_9 except T_{12}, so we move to the next affected transaction in the list T_{12}. Searching for other transaction based on T_{12} leaves us with no new discoveries (no new transaction depends on T_{12}) and hence finalizing the list of affected transaction to be $\{T_9, T_{12}\}$. At this stage we can say that the detection part of the algorithm is complete and we can forward the set of malicious transaction $\{T_6\}$ and the set of affected transactions $\{T_9, T_{12}\}$ to the recovery part of the algorithm to do what is necessary to recover the database from both the malicious and affected transactions.

3.6 Recovery Algorithm

After the detection phase of the algorithm identifies all the affected transactions as explained in the previous sections, the recovery phase starts by rolling back the database to the last consistent state. This is done mainly in two steps undoing the malicious transactions and redoing the affected transactions.

In order to enhance the performance of the recovery algorithm, an array data structure will be used to read the log file into it. This array will include only the transactions that occurred from the ID of the smallest malicious transaction till the end of the log file. This way we only copy to the array transactions that could be either undone or redone because the transactions that took place before the first malicious transaction are neither malicious nor affected and hence are not part of the scope of the recovery phase. The array is indexed by the ID of the transaction, so for example if transaction T_i is part of the array it will be found at array[12]. This indexing approach will enhance the performance of the recovery algorithm by decreasing the access time for transactions to be redone or undone. Whenever a transaction is to be referenced, we do not go through the whole log file but we only need the ID of the transaction and we reference it directly from the array.

Our recovery algorithm starts with the malicious transactions and undoes them and their effects and then moves to the affected transactions and redoes them in a database where the malicious transactions do not exist anymore because they have been deleted in the previous step. What is actually done in this step is that for each malicious transaction in the input to the recovery algorithm, we go to the array and check the details of the transaction and reverse its actions. So for example if a malicious transaction added a record to the database, undoing the transaction means deleting the record that have been maliciously added. On the other hand, when we have an affected transaction the action to be done is to redo the transaction. So if we have an update transaction we will try to re-execute the transaction to make sure that it runs properly without any interference from malicious transactions what so ever.

When the recovery algorithm finishes with the last affected transaction in the list, we would have reached the end of the recovery phase and rolled back the database to a consistent state. The array will be deleted since its role in referencing the details of the needed transactions is completed. Moreover, undoing malicious transactions and redoing affected ones will have updated the database log automatically and we will end up with a log file that reflects a consistent database.

3.7 Example on the Recovery Process

This section of the thesis report will present the recovery part of the example that was started in section 3.5. The detection part of the algorithm resulted in the following:

1. The set of malicious transaction is {T6}
2. The set of affected transactions is {T9, T12}

Our recovery algorithm will start with the first malicious transaction (we have only one in our case), T6. In our example T6 = Doctors (‘7’, ‘Robert’, ‘Internal Medicine’, ‘15’). Undoing T6 means that we need to reverse the insert statement by using a delete statement instead. As a result the Doctor with DID = 7 is deleted from the Employees table. This way the malicious transaction and its effects have been removed from the database. The next step is to deal with the affected transactions. The first affected transaction on the list is T9 where T9 = Prescription (‘1’, ‘3’, ‘7’, ‘1/1/2015’). Redoing T9 means that we have to add a new prescription for patient with ID equals to 3 and by doctor with ID equals to 7. We just deleted the doctor with ID = 7 because it was added by a malicious transaction, therefor we cannot add the Prescription (‘1’, ‘3’, ‘7’) and hence redoing this transaction means deleting the prescription in order to preserve the referential integrity in the database because doctor with ID = 7 does not exist anymore. T12 is the next affected transaction on the list. T12 = Prescription_details (‘1’, ‘1’, ‘9’, ‘1’, ‘5’). It has a similar case to T9 and we will have to delete the prescription_detail with ID = 1 because it tries to reference prescription with ID = 1 that is non-existing at the current time.

The model presented in this report uses a single matrix for the detection and recovery. In addition to some auxiliary data structure such as the array holding the log file, our algorithm depends solely on the matrix for the detection of the affected transactions. The used matrix is fast and simple to access unlike other data structures that are used in other recovery models such as graphs. Moreover, the algorithm skips transactions that are obviously clean in the detection phase and uses check points in order not to make the log file grow tremendously. In the recovery part of the algorithm only the transactions that are needed in the recovery are copied to the array and hence enhancing the performance and memory consumption of the algorithm.

CHAPTER FOUR

PERFORMANCE ANALYSIS

4.1 Introduction

In order to analyze the performance of our proposed algorithm, we simulated the model in a simulation environment that is very similar to the real environment where the algorithm should be used. As described in previous sections, we assume the presence of an intrusion detection system that will forward a list of malicious transactions to our model. Therefore, our model is initiated as soon as there is an intrusion detected by the intrusion detection system. The main data structure that the model depends on is the dependency matrix that is built before the algorithm is initiated and is fed as an input to the detection algorithm. More precisely, the dependency matrix is built after understanding the dependency relationships between the database tables and what transactions read data items and what transactions write data items. Another data structure that is built during the execution of the algorithm is a version of the log file that is used in the recovery phase and is built from reading a sample of the execution history.

The main assumption of our model is that the execution history is rigorously Serializable. This will ensure that no transaction can depend (read data items) from any transaction that occurs after it. Therefore, we cannot find two transactions T_i and T_j such that $i < j$ and T_j happens before T_i ($T_j < T_i$). This assumption will highly enhance the performance of our model because it makes sure that any malicious transaction that happened at time t can only affect transactions that happened after t not before it and hence we do not search for affected transaction that happened before time t.

The detection algorithm which is the first phase of the model is initiated by a call from the intrusion detection system with a list of one or more malicious transactions that have infiltrated the database and violated its integrity. The malicious transaction with the smallest ID is the most important one because it determines from where the detection algorithm will start its search for affected transactions. Once the detection phase is done, the list of affected transactions detected is forwarded to the recovery phase of the model which undoes the malicious transactions and redoes the affected ones. In our simulation we use more than one value for the smallest malicious ID in order to show how our algorithm performs when it scans different portions of the log. Moreover, the analysis results will show how our algorithm we perform due to its low memory consumption as a result of the lightweight matrix that it uses. The algorithm will also be tested on database of different sizes in order to evaluate its performance on larger databases and how much it is scalable.

In our experiment we used the "Northwind Database" that has been tweaked in order to serve our experiment. This database is provided as a template in the Microsoft Access or Microsoft SQL Server. The data will be exported to .sql format and then imported to MySQL. The server that was used is WampServer 2.0 (that is available online and is downloadable for free) with the following configuration: Apache Version 2.4.9, PHP Version 5.5.12 and MySQL Version 5.0.11. The simulated environment was developed on a system with an AMD Dual-Core Mobile M620 2.5GHz with a 4 GB RAM.

In section 4.2 we present and analyze the performance results of the detection part of the model, in section 4.3 we present and analyze the performance results of the recovery part of the model and in section 4.4 we present the analysis of our model in terms of memory consumption.

4.2 Performance results and analysis of the Detection algorithm

The recovery of the database consists of two phases the detection phase where the affected transactions are detected and the recovery phase where the malicious transactions are undone and the affected transactions are redone. In this section we simulate and analyze the performance of the detection phase of the model.

The NorthWind database that was used in the experiment has been tailored slightly in order to fit the requirements of the algorithm. To simplify the input data to the detection algorithm we used a subset of the tables found in the original North Wind database. The tables that have been used throughout the simulation are:

1. Categories Table: represents the categories of the products that are sold and it has 5 columns.
2. Customers Table: represents the customers with their details and contact information and it has 12 columns.
3. Employees Table: represents the employees that sell products to the customers and it has 18 columns.
4. Order_details Table: represents the details (such as quality) of the orders that have been made by the customers and it has 7 columns.

5. Orders Table: represents the high level information about the order (the related customer and employee) and it has 15 columns.
6. Products Table: represents the items/products that can be bought in the North Wind database and it has 11 columns.
7. Suppliers Table: represents the suppliers/providers of the products and it has 13 columns.

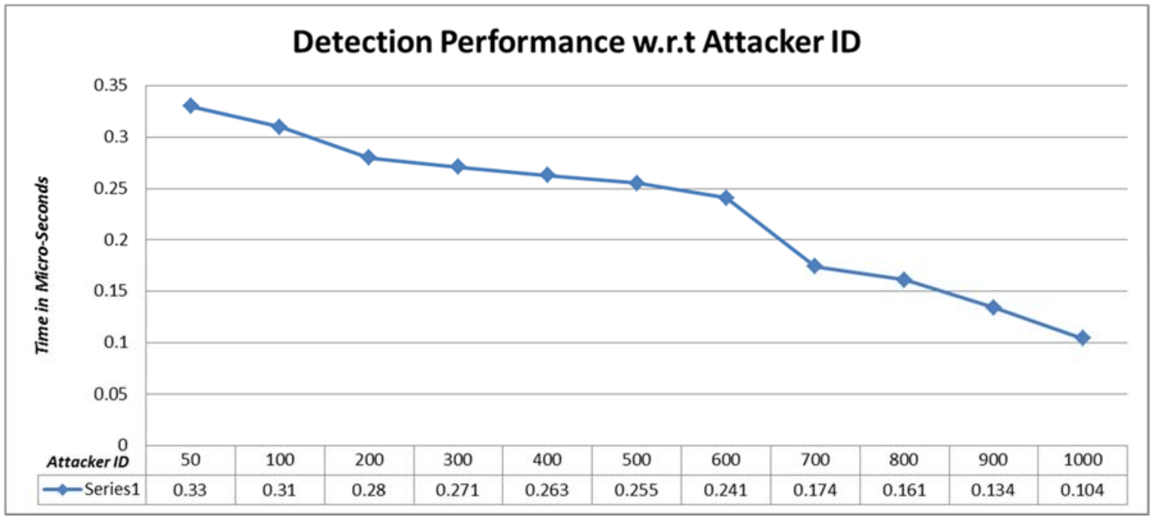
In addition to the previous assumptions that were used to develop our recovery model we have the below two assumptions for the simulation part:

1. The maximum number of data items accessed by any transaction is equal to the number of data items present in the table to which the transaction belongs. So for example if a transaction is adding a new category to the categories table, it can access a maximum of five data items when doing so (these are the 5 fields/columns present in the categories table).
2. In order to have global uniqueness of the database rows, we added one additional column to represent the global primary key of table records. The previous primary key used in the table is unique only for the scope of the table in concern but not to other tables. This will help us in identifying transactions by global ID's. So for example when we say transaction with ID = 9, there is only one transaction with ID = 9 in the database where before altering the database there could be two rows in different tables with primary keys = 9 and hence making confusion on which one to pick.

Since the performance of our algorithm depends mainly on the number of transactions to be scanned to check them if they are affected or not, we analyze the performance of our detection algorithm with respect to the ID of the smallest malicious transaction. The ID of the smallest malicious transaction determines the scope of scanning since the detection starts directly at the first transaction after the smallest malicious transaction and continues onwards to the end of the log file.

Figure 4.1 below shows the performance of the Detection phase of our model with respect to Attacker ID. What is meant by the Attacker ID is the smallest ID present in the list of malicious transactions sent to the detection algorithm. From the first look at the figure we can deduce the correctness of our model assumption that as the Attacker ID increases, the performance of our detection algorithm enhances because it will have to scan less transactions to see if they are affected or not.

Figure 4.1 Performance of the detection algorithm with respect to Attacker ID



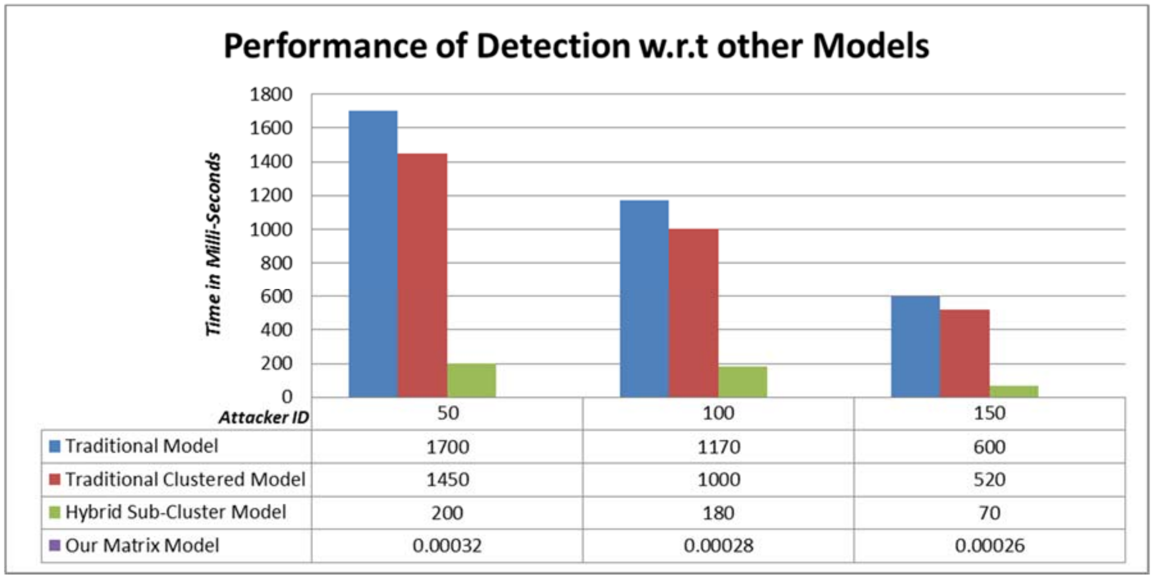
Attacker ID	50	100	200	300	400	500	600	700	800	900	1000
Series1	0.33	0.31	0.28	0.271	0.263	0.255	0.241	0.174	0.161	0.134	0.104

The previous analysis was performed on a database snapshot consisting of 1080 records. Any transaction can access a maximum of 18 data items at a time (since the largest number of columns in any of our tables is 18 and it is the employees table).

Due to our assumption that the execution history is rigorously Serializable, we cannot find two transactions T_i and T_j such that $i<j$ and T_i happens after T_j. As a result, there cannot be a transaction that happened before a malicious transaction and is affected by this malicious transaction. This is shown in the results in figure 4.1. When the attacker ID is 50, the detection algorithm will have to scan 1030 transactions (1080 minus 50) to check if any one of them is infected from the malicious transaction with ID 50. On the other hand, when the attacked ID is 1000 the detection algorithm will have to scan only 80 transactions (1080 minus 1000). The time decreases from 0.33 microseconds to 0.104 microseconds when the attacker ID changes from 50 to 1000, which is more than threefold performance enhancement. Therefore our detection algorithm proves to be very efficient when the attacker ID decreases and still performs very well when the attacker ID is small.

In order to measure the performance of our damage assessment/detection algorithm we have compared it with other models present in [7]. The three other models are the traditional, traditional clustered, and the hybrid sub-cluster models. The traditional model was presented by Bai and Liu in [15]. The previous experiments were done on a database with 200 records. Therefore, we have sampled another snapshot of our version of the northwind database in order to have proper comparison between our matrix based model and the previously mentioned models.

Figure 4.2 Performance results of our model compared to other models



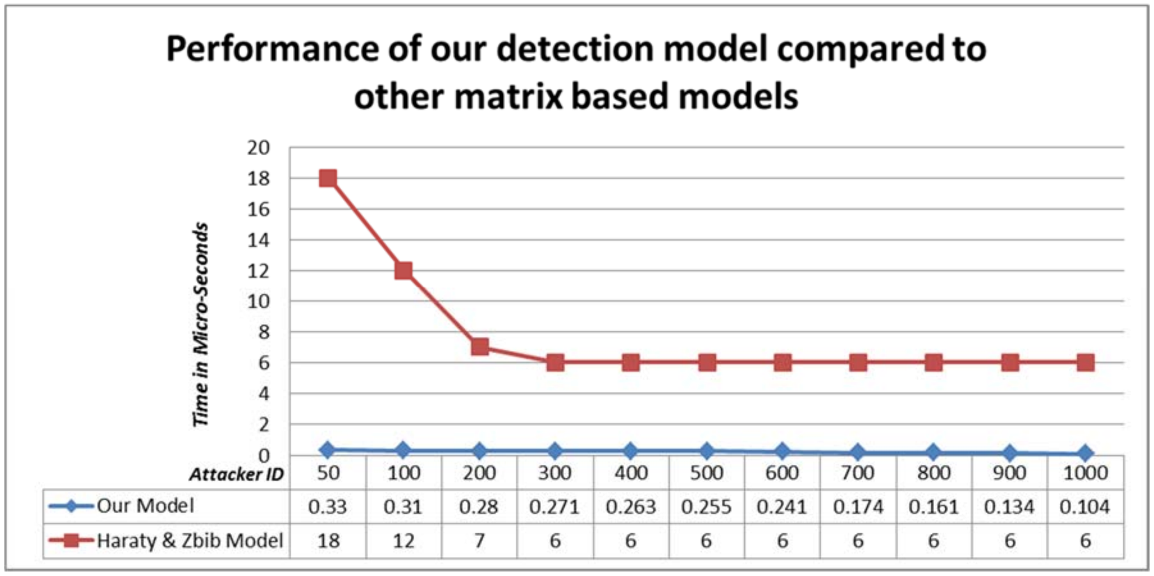
Attacker ID	50	100	150
🟦 Traditional Model	1700	1170	600
🟥 Traditional Clustered Model	1450	1000	520
🟩 Hybrid Sub-Cluster Model	200	180	70
🟪 Our Matrix Model	0.00032	0.00028	0.00026

Figure 4.2 shows that efficiency of our matrix based model compared to other models. So for example in the worst case scenario (when the attacker ID is 50), comparing our model to the fastest of the other models, hybrid sub-cluster model shows that our model is at least 625,000 times faster. In the best case scenario (when the attacker ID is 150), our model is at least 270 times faster. On the other hand, if we take the slowest of the three other models, the traditional model, our model is faster by 5,312,000 times. The high performance of our model is due mainly to the dependency matrix that only stores the information needed for the damage assessment in the form of linked lists that are sorted to decrease the assessment time by removing unnecessary check to transactions that are not affected. Our matrix is easily indexed unlike other complex data structure used in other models such as graphs. Moreover our matrix only stores the IDs of the transactions that depend on each other. Another important difference is that our model does not read the entire execution history or even the entire dependency matrix, but it

only reads the information from the dependency matrix starting from the first transaction after the smallest malicious transaction.

A recent matrix based algorithm presented by Haraty and Zbib in [24] showed promising results in terms of running times. Therefore, in order to know the performance of our model in comparison to similar models that are matrix based, we analyzed the performance of our model with the model in [24]. Figure 4.3 shows the results of comparing our model to [24].

Figure 4.3 Performance of our detection model compared to other matrix based models



Attacker ID	50	100	200	300	400	500	600	700	800	900	1000
Our Model	0.33	0.31	0.28	0.271	0.263	0.255	0.241	0.174	0.161	0.134	0.104
Haraty & Zbib Model	18	12	7	6	6	6	6	6	6	6	6

Figure 4.3 shows clearly that our model is more efficient than the most recent matrix based models. This is evident from the big difference in the running time between the two models. There is a ratio of 55 times when the attacker ID is 50 and there is a ratio of 58 when the attacker ID is 1000. This difference in performance is due to the following:

1. The fact that our algorithm uses only one light weight matrix instead of two

2. Our model stores only the dependency between transactions instead of storing an n*m matrix (n is the number of transactions in the database and m is the number of data items in the database)

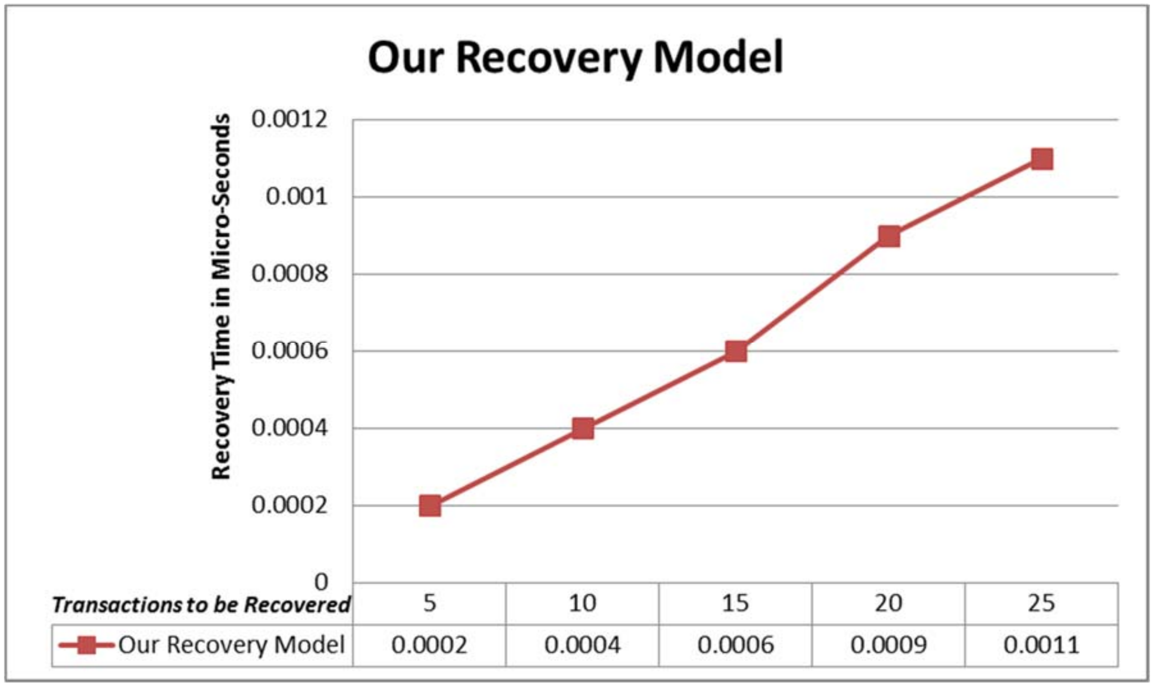
3. Our model uses Sorted Linked Lists that saves more space and makes the algorithm lighter and faster. Moreover, by being sorted, the linked lists removes many unnecessary checks that are done in the matrix of [24] since in our model whenever we reach a transaction that is greater than the one we are looking for we skip the rest of the linked list and move to the second row in the dependency matrix.

The previous figures (4.1, 4.2, and 4.3) show that our model is faster from other models that may not necessarily use a matrix and models that are based on matrices like the one in [24]. Moreover, our model is lightweight and does not require a lot of memory in order to execute efficiently and hence making it more scalable for larger databases.

4.3 Performance results and analysis of the Recovery algorithm

In order to analyze the performance of the recovery phase of our model, we have also simulated it on the modified version of the northwind database. Since it is logical for the recovery algorithm to take more time to recover as the number of transactions to be recovered increases, we analyzed the performance of our algorithm with respect to an increasing number of recovered transactions. The experiment was done on a database with 200 records where any transaction can access a maximum of 18 data items at a time. The results of the simulation are presented in figure 4.4.

Figure 4.4 Performance of our recovery model with respect to increasing number of recovered transactions

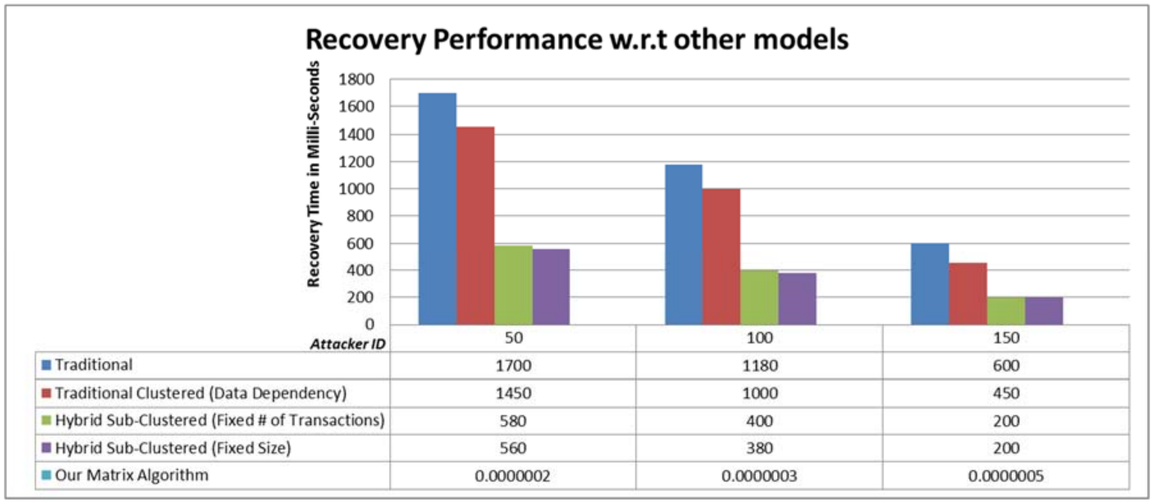


Transactions to be Recovered	5	10	15	20	25
Our Recovery Model	0.0002	0.0004	0.0006	0.0009	0.0011

The transactions to be recovered could include both the malicious transactions and the affected transactions. As it appears from figure 4.4, the time taken by our recovery algorithm increases with the number of transactions to be recovered. This increase in time is quite acceptable since when the transactions to be recovered increased by 5 folds, the time to recover increased from 0.0002 micro-seconds to 0.0012 micro-seconds.

In figure 4.5 below we analyze the performance of our matrix based recovery algorithm with four other models as presented in [7]. These four other models are the traditional, traditional clustered (with data dependency), hybrid sub-clustered (with a fixed number of transactions), and hybrid sub-clustered (with a fixed size of the sub-cluster).

Figure 4.5 Performance of our recovery model with respect to other models

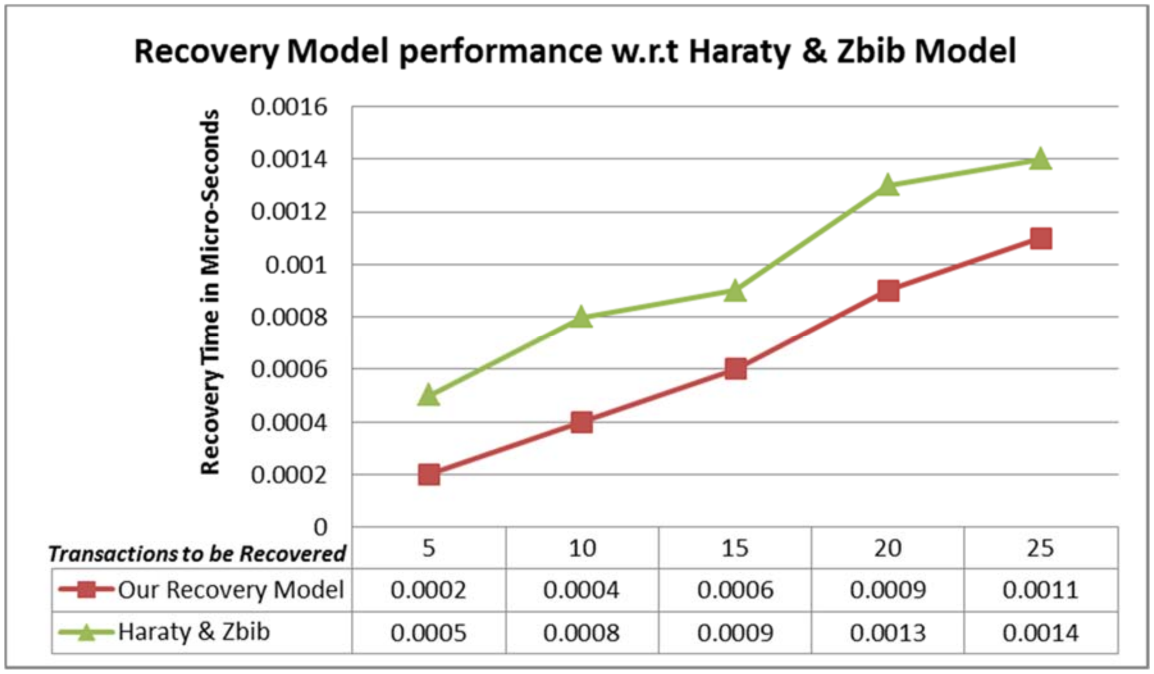


Attacker ID	50	100	150
🟦 Traditional	1700	1180	600
🟥 Traditional Clustered (Data Dependency)	1450	1000	450
🟩 Hybrid Sub-Clustered (Fixed # of Transactions)	580	400	200
🟪 Hybrid Sub-Clustered (Fixed Size)	560	380	200
■ Our Matrix Algorithm	0.0000002	0.0000003	0.0000005

Figure 4.5 shows that our recovery algorithm out performs all of the other models whatever was the Attacker ID. For example in the case when the Attacker ID is 50, our recovery model is faster by 8,500,000,000 times than the slowest model (Traditional model) and is faster by 2,800,000,000 times than the fastest model (Hybrid Sub-Clustered with fixed size). The main contribution of our model which is the use of a light weight matrix is the main reason behind its high performance. The entries of matrix are sorted linked lists which highly decrease the assessment time by removing unnecessary checks in the model. Moreover, out recovery model converts the log file into an array (this conversion step is done only once throughout the recovery phase) which is a lot faster to access than the log file itself. So instead of accessing the log every time we want to access a certain transaction (which is time consuming due to the high cost of I/O involved), we access them as entries of an array which is indexed by nature. While converting the log file to an array, we only take into consideration the part of the log file that starts after the first malicious transaction not the whole log file. This will decrease the size of the array to only store the entries that are needed for recovery because the transactions that happened before the first malicious transactions need not to be recovered because they cannot be affected to our assumption of rigorous Serializable execution.

In order to analyze the performance of our recovery model with respect to other matrix based models, we used the model in [24] which is a recent matrix based model. In the below figure we compared the performance of our recovery algorithm to Haraty and Zbib matrix based model in [24] with respect to an increasing number of transactions to be recovered. The database that we used consists of 200 transactions where each transaction can access a maximum of 18 data items at any time.

Figure 4.6 Performance analysis of our recovery model with respect to Haraty and Zbib model



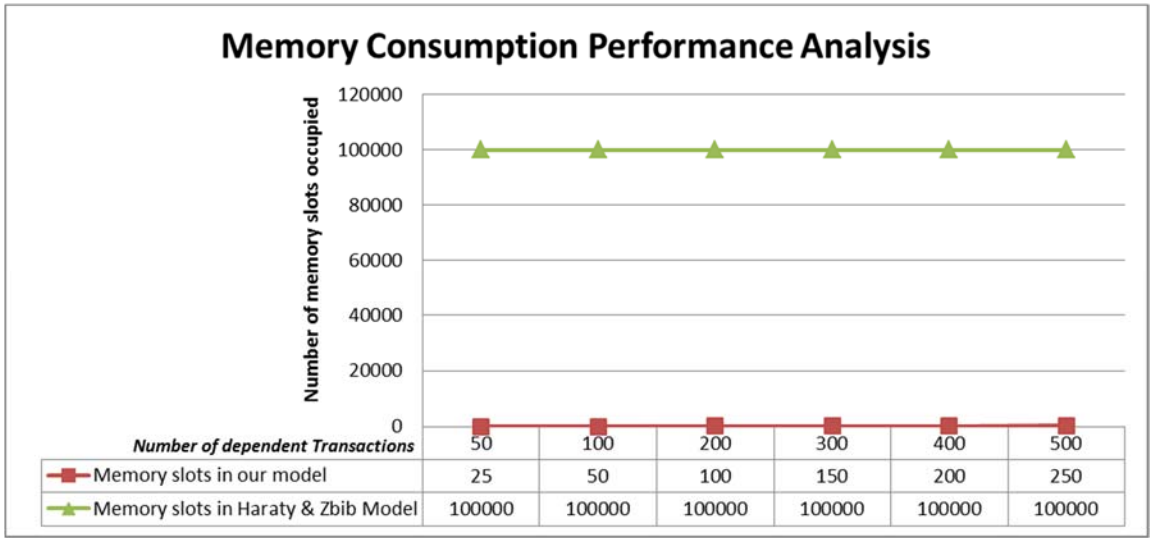
Transactions to be Recovered	5	10	15	20	25
🟥 Our Recovery Model	0.0002	0.0004	0.0006	0.0009	0.0011
▲ Haraty & Zbib	0.0005	0.0008	0.0009	0.0013	0.0014

As we can see from figure 4.6, our recovery model is around two times as fast as the other matrix model. The high performance of our algorithm is due to the fact that we only read the needed part from the log file and we populate it into an array that reduces the access time when we need to recover any transaction.

4.4 Performance results and analysis of the memory consumption

Our recovery algorithm is not only efficient in terms or running time, but also it is very efficient in terms of memory consumption. The low memory consumption of our model makes it more efficient with databases of larger sizes due to its increased scalability levels. In order to analyze the performance of our model in terms of memory needs, we compared its memory needs to Haraty & Zbib model [24] both in the best case and worst case scenarios. In the below figures we assume that the memory slots consumed by any algorithm is the number of content entries in all of its data structures. So for example if we have a transaction T_4 that depends on three other transactions (T_1, T_2, T_3), then the memory slots used to represent this dependency relation is **3**. In our model we use only one matrix while in the model presented in [24], they use a matrix and a secondary structure. In our comparison we assumed that we have a database that is made up of 1000 transactions and 100 data items. As a result of this the model in [24] will have a dependency matrix made up of 100,000 entries (1000 transactions * 100 data items) that will be used and populated whatever the dependency between the transactions was. Our model does not have such matrix and on the contrary uses a matrix that depends on the dependency relation between the transactions and not on the number of transactions and data items in the database.

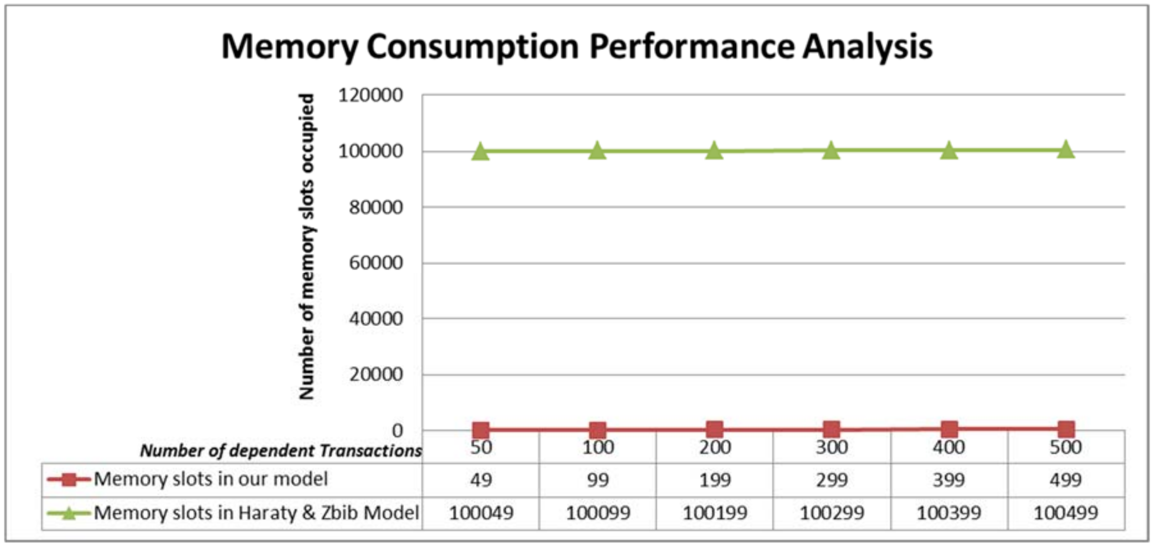
Figure 4.7 Best Case scenario memory consumption analysis



Number of dependent Transactions	50	100	200	300	400	500
🟥 Memory slots in our model	25	50	100	150	200	250
▲ Memory slots in Haraty & Zbib Model	100000	100000	100000	100000	100000	100000

In figure 4.7 we compared the memory consumption under the best case scenario of our algorithm to latest matrix based recovery algorithm presented in [24]. The best case scenario (where the lowest number of memory slots is used) occurs in both algorithms when each transaction depends on only one other transaction not more. As a result, in the case of our algorithm we will use only a number of memory slots that is equal to half of the dependent transactions number and in the case of Haraty's model we will not use the secondary structure to store the dependency relation. In the case of 50 dependent transactions, the matrix used in our model will be an array of length 25 and at each index we have only one transaction (memory slot). On the other hand, the model in [24] will not use any secondary data structures but it will populate the core matrix that will have 100,000 entries. As we can see from figure 4.7, the number of memory slots used by the model in [24] is 4,000 times than the memory slots used in our model.

Figure 4.8 Worst Case scenario memory consumption analysis



Number of dependent Transactions	50	100	200	300	400	500
■ Memory slots in our model	49	99	199	299	399	499
▲ Memory slots in Haraty & Zbib Model	100049	100099	100199	100299	100399	100499

In figure 4.8 we analyze the memory consumption under the worst case scenario (when the highest number of memory slots is consumed by both algorithms). The worst case scenario occurs when one transaction depends on the remaining transactions in the set. So for example when we have 50 related transactions, one transaction depends on the remaining 49 transactions. In this case our model will have a matrix with 49 entries and the model in [24] will use the secondary structure to reflect this dependency relation by using also an extra 49 entries. As a performance indicator when we can take the case when the set of dependent transactions is 50, the model in [24] will consume 100049 slots and our model will consume only 49 slots (i.e. 2,042 times more slots consumed by the model in [24]).

From the previous experiments it is evident that our model is very efficient in terms of its memory consumption. This makes it more scalable for databases with very large sizes. This also helps in indirectly enhancing the running time of our model because it is expected that whenever an algorithm consumes less memory than its peers it will run faster. The matrix used in our model stores only the data needed for the detection and recovery without any additional unnecessary information. Unlike the Haraty and Zbib model in [24], our matrix does not store information about blind writes or explicit information that T_2 does not depend on T_1. What matters for the recovery in our model is the explicit information that T_y depends on T_x and nothing else.

CHAPTER FIVE

CONCLUSION

A layered approach is adopted in most of the information systems protection. We should first prevent or at least reduce the possibility of an attack. Then, we should detect any attempts of an attack before the attack damages any data. Prevention methods do not always work; hackers and attackers always find ways to breach the system. Again detection systems fail to detect the malicious transaction as soon as they happen. Here comes the role of recovery algorithms that will repair the damage of the data by cleaning the database from the malicious transactions and their effects. The algorithm described in this thesis presents a new matrix based approach for database recovery that is lightweight and more efficient than all the other methods currently found in the literature. We implemented the algorithm and compared it with different recovery algorithms available and showed that our algorithm outperforms them in multiple times. These experimental results prove that our algorithm is more efficient and requires less space than the other proposed models including other matrix based ones. In both phases of the recovery process (assessment phase and recovery phase), the presented algorithm has outperformed its counter parts.

As a future work, we will work on a distributed version of the algorithm. Our model works on a centralized database where only one copy of the data exists. The algorithm needs to take into consideration the fact that different parts of the data could be distributed or replicated into different sites creating new challenges that need to be dealt with in order to achieve the correct recovery.

References

[1] R. Christian, M. Ryan and J. Wohlrabe. "Handwashing and respiratory illness among young adults in military training." *American Journal of Preventive Medicine*, vol. 21, pp. 79-83, Aug. 2001.

[2] S. Chalurkar, S. Kakade, N. Khochare and B. Meshramm. "Survey on sql injection attacks and their countermeasures." *IJC EM international journal of computational engineering & management*, vol. 14, pp.2230-7893, Oct. 2011.

[3] J. Gray and A. Reuter. "Transaction processing concepts and techniques." *Department of computational biology-university of Tokyo, Japan*, Morgan Kaufmann, pp.143-144, Jan. 1993.

[4] W. Hutchinson and M. Warren. *Information warfare: Corporate attack and defence in a digital world*. Oxford: Butterworth-Heinemann, 2001, pp. 21-127.

[5] B. Megan. "Information warfare: what and how?" Internet: http://www.cs.cmu.edu/~burnsm/InfoWarfare.html, 1999 [Feb. 25, 2015].

[6] C. Kumar, D. Ramesh and V. Vijay Kumar. "A resilient failure evaluation and patch-up (R-FEP) algorithm for heterogeneous distributed databases." *International journal of computer applications*, vol. 62, pp. 20-25, Jan. 2013.

[7] R. Haraty and A. Zeitunlian. "Damage assessment and recovery from malicious transactions using data dependency for defensive information warfare." *ISESCO science and technology vision*, vol. 3, pp. 43-50, Nov. 2007.

[8] B. Panda and J. Zhou. "Database damage assessment using a matrix based approach: An intrusion response system," in *Proc. 7th International database engineering and applications symposium*, 2003, pp. 336-341.

[9] B. Panda and K.A. Haque. “Extended data dependency approach: A robust way of rebuilding database,” in *Proc. the 2002 ACM Symposium on Applied Computing*, 2002, pp. 445-452.

[10] P. Ammann, S. Jajodia, and P. Liu. “Recovery from malicious transactions,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, pp. 1167-1185, Oct. 2002.

[11] B. Panda and S. Tripathy. “Data dependency based logging for defensive information warfare,” in *Proc. of the 2000 ACM Symposium on Applied Computing*, 2000, pp. 361-365.

[12] B. Panda and Y. Zuo. “Damage discovery in distributed database systems,” in *Research directions in data and applications security XVIII*, 2004 ed., vol. 144. C. Farkas P. Samarati, Ed. Catalonia, Spain: Springer US, 2004, pp.111-123.

[13] X. Qin, J. Sun and J. Zheng. “Data dependency based recovery approaches in survival database systems,” in *Computational Science -- ICCS 2007*, 2007 ed., vol. 4488, Y. Shi, G. Van Albada, J. Dongarra and P. Sloot, Ed. Beijing: Springer Berlin Heidelberg, 2007, pp. 1131-1138.

[14] M. Kaashoek, T. Kim, X. Wang and N. Zeldovich. “Intrusion recovery using selective re-execution,” in *Proc. of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*, 2010, pp. 89-104.

[15] Q. Ji, J. Le, and X. Xia. “Research on transaction dependency mechanism of self-healing database system,” in *Proc. International conference on systems and informatics*, 2012, pp. 2357-2360.

[16] B. Panda and P. Ragothaman. “Analyzing transaction logs for effective damage assessment,” in *Research directions in data and applications security*, 2003 ed., vol. 128. E. Gudes and S. Shenoi, Ed. Cambridge: Springer US, 2003, pp. 121-134.

[17] B. Panda and R. Sobhan. “Reorganization of the database log for information warfare data recovery,” in *Database and application security XV*, 2002 ed., vol. 87. M. Olivier and D. Spooner, Ed. Ontario: Springer US, 2002, pp. 121-134.

[18] Y. Feng, G. Hu, M. Xie and H. Zhu. “Tracking and repairing damaged databases using before image table.” *Japan-China joint workshop on frontier of computer science and technology*, vol., pp. 36-41, Dec. 2008.

[19] A. Chakraborty, A. Majumdar and S. Sural. “A column dependency based approach for static and dynamic recovery of databases from malicious transactions.” *International journal of information security*, vol. 9, pp. 51-67, Nov. 2009.

[20] Y. Hu, B. Panda and J. Zhou. “Succinct and fast accessible data structures for database damage assessment,” in *Distributed computing and internet technology*, 2005 ed., vol. 3347. R. Ghosh and H. Mohanty, Ed. Berlin: Springer Berlin Heidelberg, 2005, pp. 111-119.

[21] B. Panda and Y. Zuo. “Fuzzy dependency and its applications in damage assessment and recovery,” in *Proc. of the 2004 IEEE workshop on information assurance*, 2004, pp. 350-357.

[22] C. Chen, Q. Liu, Y. Liu, and G. Shen. “A recovery approach for real-time database based on transaction fusion,” in *Recent advances in computer science and information engineering*, 2012 ed., vol. 124. Z. Qian, L. Cao, W. Su, T. Wang and H. Yang, Ed. Berlin: Springer Berlin Heidelberg, 2012, pp. 473-479.

[23] K. Bai and P. Liu. “A data damage tracking quarantine and recovery (DTQR) scheme for mission-critical database systems,” in *Proc. of the 12th International*

Conference on Extending Database Technology: Advances in Database Technology (EDBT '09), 2009, pp. 720-731.

[24] R. Haraty M. Zbib. “A matrix-based damage assessment and recovery algorithm.” *Innovations for community services (I4CS)*, vol., pp. 22-27, Jun. 2014.