A NOVEL DEADLOCK DETECTION ALGORITHM FOR NEIGHBOUR REPLICATION ON GRID ENVIRONMENT

NORIYANI BINTI MOHD ZIN

MASTER OF SCIENCE (SOFTWARE ENGINEERING)

UNIVERSITI MALAYSIA PAHANG

A NOVEL DEADLOCK DETECTION ALGORITHM FOR NEIGHBOUR REPLICATION ON GRID ENVIRONMENT

NORIYANI BINTI MOHD ZIN

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Master of Science (Software Engineering)

Faculty of Computer Systems & Software Engineering
UNIVERSITI MALAYSIA PAHANG

JULY 2012

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's full name : _____

Date of birth : _____

Title : _____

_____

_____

Academic Session : _____

I declare that this thesis is classified as:

☐ **CONFIDENTIAL** (Contains confidential information under the Official Secret Act 1972)*

☐ **RESTRICTED** (Contains restricted information as specified by the organization where research was done)*

☐ **OPEN ACCESS** I agree that my thesis to be published as online open access (Full text)

I acknowledge that Universiti Malaysia Pahang reserve the right as follows:

1. The Thesis is the Property of University Malaysia Pahang
2. The Library of University Malaysia Pahang has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified By:

_____ _____

(Student's Signature) (Signature of Supervisor)

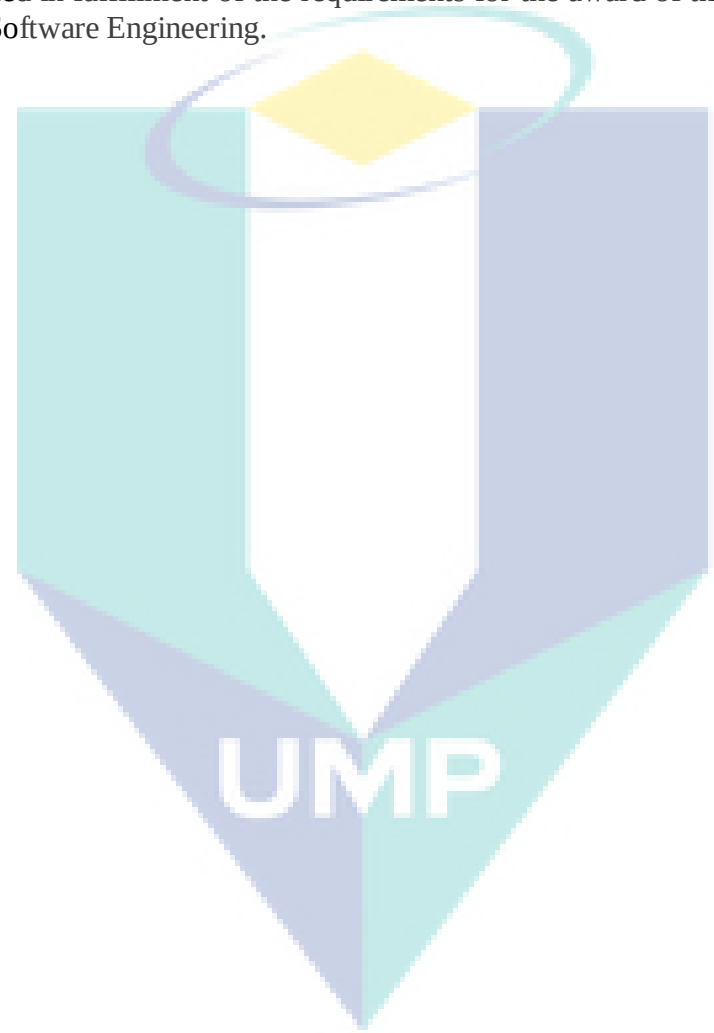
_____ _____

New IC / Passport Number Name of Supervisor
Date : Date :

STATEMENT OF AWARD

Master of Engineering (by Research)

Thesis submitted in fulfillment of the requirements for the award of the degree of Master of Science in Software Engineering.



SUPERVISOR DECLARATION

I hereby declare that I have checked this thesis and in my opinion this thesis is satisfactory in terms of scope and quality for the award of the degree of Master of Science (Software Engineering).

Signature :

Name of Supervisor : ASSOC. PROF. DR.NORAZIAH BINTI AHMAD

Position : ASSOCIATE PROFESSOR

FACULTY OF COMPUTER SYSTEMS & SOFTWARE ENGINEERING, UNIVERSITI MALAYSIA PAHANG

Date :

STUDENT DECLARATION

I hereby declare that the work in this thesis is my own except for quotations and summaries which have been duly acknowledged. The thesis has not been accepted for any degree and is not concurrently submitted for award of other degree.

Signature :

Name : NORIYANI BINTI MOHD ZIN

ID Number : MCS09002

Date :

This thesis is dedicated to

My beloved parents

Mohd Zin Bin Rapie and Rasimah Binti Othman

&

My siblings

For their endless care and support

ACKNOWLEDGEMENTS

I would like to express my most sincere gratitude to the supervisory committee Associate Professor Dr. Noraziah Binti Ahmad for her continuing support, professional guidance and for giving me an opportunity to learn what research is all about.

Sincerely thanks should be forwarded to The Vice Chancellor of Universiti Malaysia Pahang (UMP), Professor Dato' Dr. Daing Nasir Ibrahim for the Graduate Research Scheme (GRS) scholarship.

Special gratitude also to my family, especially to my father, Mohd Zin Bin Rapie; my mother, Rasimah Binti Othman; my brother Mohd Fadli Azahar; and my other siblings for their patience and morale support.

Finally, I thank to all my friends especially Asmahani Binti Ab. Rahman, Ainul Azila Binti Che Fauzi, Rozita Binti Mohd Yusuf, Abul Hashem Beg and Khandaker Fazley Rabbi who have contributed this research.

ABSTRACT

Deadlock occurs when each of the transaction involves is waiting to grant the data that has been locked by other transactions. This can lead to a circular wait called Wait-for Graph (WFG). Deadlock can make the transaction become an inactive, so other transaction is not able to perform any action and further cause unavailability of resources. Therefore, an action must be taken to detect and solve this problem. A new framework and algorithm called Neighbour Replication on Grid Deadlock Detection (NRGDD) has been developed to handle deadlock cycles that exist during the transaction in Neighbour Replication on Grid (NRG) environment. The aim of this research is to handle the deadlock problem in NRG to preserve the consistency of data and increase the throughput. The NRGDD simulation model has been developed to test the algorithm on NRG. Two experiments have been conducted to test the correctness of NRGDD algorithm. The first experiment is to detect two cycles of deadlock while the second experiment is to spot deadlock by using different number of transaction, from three to five transactions. The use of three to five transactions is in NRG the data will be replicated into three to five sites. Each site is locked by different set of transaction. Then, the transaction can send request to other site that is held by another transaction. So, circular wait is formed. Through this experiment, the NRGDD simulation model is able to detect multiple cycles of deadlock which exist on NRG. The NRGDD is compared with Multi-Cycle of Deadlock Detection and Recovery (MC2DR) algorithm based on the time required for both models to detect two deadlock cycles and using different numbers of transactions. The NRGDD achieved 27.5% improvement from MC2DR. From the experimental result, it is clearly shown that handling deadlock on NRG using NRGDD is able to preserve the data consistency and increase the throughput by maximizing the availability of resources.

ABSTRAK

Kebuntuan terjadi apabila setiap traksaksi yang terlibat akan menunggu untuk mendapatkan data yang telah dipegang oleh transaksi yang lain. Ini boleh menyebabkan kitaran menunggu yang dipanggil Wait-for Graph (WFG). Kebuntuan akan membuatkan transaksi menjadi tidak aktif dan transaksi lain tidak dapat melakukan apa-apa kerana tiada sumber. Oleh itu, tindakan perlu diambil untuk mengesan dan menyelesaikan masalah tersebut. Rangka kerja dan algoritma baru yang dipanngil Neighbour Replication on Grid Deadlock Detection (NRGDD) telah dibina untuk mengawal kewujudan kitaran kebuntuan semasa transaksi berlaku dalam persekitaran Neighbour Replication on Grid (NRG). Tujuan penyelidikan ini adalah untuk mengawal masalah kebuntuan dalam NRG bagi memelihara data supaya konsisten dan meningkatkan kadar sumber yang ada. Model simulasi NRGDD dibina untuk menguji algoritma dalam NRG. Dua ekperimen di jalankan untuk menguji ketepatan algoritma NRGDD. Experiment pertama ialah untuk mengesan dua kitaran kebuntuan manakala ekperimen kedua ialah untuk mengesan kebuntuan menggunakan jumlah transaksi yang berbeza, dari tiga hingga ke lima transaksi. Penggunaan tiga hingga ke lima transaksi adalah dalam NRG data akan direplika ke dalam tiga hingga ke lima tempat. Setiap tempat dipegang oleh transaksi yang berbeza. Kemudian, transaksi boleh menghantar permintaan ke tempat yang lain yang telah dipegang oleh transaksi lain. Jadi, kitaran menunggu akan terbentuk. Melalui ekperimen ini, model simulasi NRGDD berkebolehan dalam mengesan lebih daripada satu kitaran kebuntuan yang wujud dalam NRG. NRGDD telah dibandingkan dengan algoritma Multi-Cycle of Deadlock Detection and Recovery (MC2DR) berdasarkan masa yang diperlukan untuk kedua-dua model dalam mengesan dua kitaran kebuntuan dan penggunaan jumlah transaksi yang berbeza. NRGDD telah mencapai 27.5% pembaikan dari MC2DR. Dari keputusan ekperimen, ia menunjukkan dengan jelas bahawa mengawal kebuntuan dalam NRG menggunakan NRGDD boleh memelihara konsisten data dan meningkatkan daya pemprosesan dengan memaksimumkan sumber yang ada.

TABLE OF CONTENTS

CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DDB	Distributed Database
ROWA	Read-One-Write-All
BRS	Branch Replication Scheme
HRS	Hierarchical Replication Scheme
NRG	Neighbour Replication on Grid
MC2DR	Multi-cycle Deadlock Detection and Recovery
WAN	Wide Area Network
ACID	Atomicity, Consistency, Isolation, and Durability
OCC	Optimistic Concurrency Control
PCC	Pessimistic Concurrency Control
2PL	Two Phase Locking
RAC	Resource Allocation Controller
WFG	Wait-For Graph
DST	Distributed Spanning Tree
EDD	Efficient Deadlock Detection
LTS	Linear Transaction Structure
DTS	Distributed Transaction Structure
TM	Transaction Manager
TQ	Transaction Queue
NRGDD	Neighbour Replication on Grid Deadlock Detection

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND OF RESEARCH

Due to the evolution of management in organizations or institutions, distributed database systems and grid systems need to support hundreds or even thousands of sites and millions of clients. Therefore, it will face tremendous scalability challenges with regard to performance, availability (Zhang *et al.*, 2009; He *et al.*, 2009), administrations (Alom *et al.*, 2010), speed and reliability (Mohammed, 2007; Li *et al.*, 2010). Furthermore, there is a tendency of storing, retrieving, and managing different types of data such as experimental data that are produced from many projects. These data play a fundamental role in all kinds of cross-organizational research and collaborations. For example, several scientific applications such as Particle Physics, High Energy Physics(Naseera *et al.*, 2009; Ben Charrada *et al.*, 2010a; 2010b; AL-Mistarihi *et al.*, 2009; Zhao *et al.*, 2008; Allcock *et al.*, 2003) and Genetics, earthquake engineering (Naseera *et al.*, 2009; Barney *et al.*, 2008), climate change modelling (AL-Mistarihi *et al.*, 2009; Zhao *et al.*, 2008), molecular docking, computer micro-tomography (Noraziah *et al.*, 2010a; 2010b) and astronomy (Zhao *et al.*, 2008, Du *et al.*, 2011), to cite a few, manage and generate an important amount of data that can reach terabytes and even petabytes (Li *et al.*, 2010), which need to be shared and analysed. A community of hundreds or thousands of researchers distributed worldwide must share these datasets (Naseera *et al.*, 2009; Ben Charrada *et al.*, 2010a). It is difficult, even

impossible, to store such amount of data in the same location. Moreover, an application may need data produced by another geographically remote application. For this reason, data grid is suitable for the above situation to support the huge amount of data production by researchers.

1.2 DATA GRID ENVIRONMENT

A data grid is composed of hundreds of geographically distributed computers and storage resources usually located under different places, and enables users to share data and other resources. The users can access the information of data easily without knowing the resource position (Fard *et al.*, 2008). The data grid is required because data is being produced at a tremendous rate and volume especially from scientific experiments (Noraziah *et al*., 2010a; 2010b). The grid computing requirements are more complex than distributed computing even though it is quite similar to normal distributed computing.

The aim of the grid computing is to enable resource sharing and coordinated problem solving in dynamic, multi-institutional virtual organization (Foster *et al.*, 2001; 2002; 2008). Furthermore, the concept of the grid computing arose from the need to share computing power, mostly for the jobs that use read-only data sets as input (output from scientific experiments) (Noraziah *et al.*, 2009a). Consequently, the primary design of data management tools for grid computing was used to manage read-only data sets.

The major problem on grid environment is data management. In grid computing, there is no limitation on the number of users, departments or organizations. Besides that, the size of the data managed by data grid is continually growing (Pérez *et al.*, 2010). In the data grid, when a user requests a data, a large amount of bandwidth could be spent to send the data from the server to the client. Furthermore, the delay involved could be high (Bsoul *et al.*, 2011). These problems can be solved through replication.

1.3 DATA REPLICATION

In data replication architecture, the data will be replicated into several sites. If one of the sites has failed, it will fail independently and does not affect other replica site. Data replication is one of the techniques in distributed and grid systems to increase availability and reliability of the data. To speed up data access, the data can be replicated in multiple locations, so that a user can access the data from nearby locations (Sashi *et al.*, 2011). Replication in distributed environment receives particular attention for providing efficient access to data, fault tolerance (Bsoul *et al.*, 2011; Sathya *et al.*, 2010; Noraziah *et al.*, 2010c) and can enhance the performance of the system (Gao *et al.*, 2005; Noraziah *et al.*, 2008; Tang *et al.*, 2006; Latip *et al.*, 2008). The replication strategy can minimize the time access to the file by creating many replicas and storing replicas in appropriate locations. Furthermore, using replication is to reduce bandwidth consumption (Naseera *et al.*, 2009; Ben Charrada *et al.*, 2010a; 2010b; Shorfuzzaman *et al.*, 2010) to achieve efficient and dependable data access in grids, improve access time (Ben Charrada *et al.*, 2010b; Zhao *et al.*, 2008; Du *et al.*, 2011) fault tolerance (Zhao *et al.*, 2008; Bsoul *et al.*, 2011; Shorfuzzaman *et al.*, 2010) and load balancing (Pérez *et al.*, 2010;). Organizations need to provide current data to users who may be geographically remote and request distributed data around multiple sites in data grid (Noraziah *et al.*, 2010a).

Replication strategies determine when and where to create a replica, taking into account of the factors such as the request number of the data, network conditions, storage availability of nodes, and others (Pérez *et al.*, 2010; Sun *et al.*, 2009). Other researchers that discussed on importance of data replication in distributed systems are Sashi *et al.* (2011), Ainul *et al.* (2011), Beg *et al.* (2010), Abdi *et al.* (2010), Li *et al.* (2010), Wong *et al.* (2009), Sun *et al.* (2009), Latip *et al.* (2008), Noraziah *et al.* (2007), and Wang *et al.* (2006).

Managing transaction significantly become important in the replication in order to preserve the consistency of data. Although the transaction may perform all of its actions on the site that it granted, it may also perform actions on other than the granted site. Besides that, concurrent access to the data and deadlock problem are the most

important issues that must be considered when sharing information in distributed systems (Alom *et al.*, 2010), especially in distributed database system (Atreya *et al.*, 2007; Hu *et al.*, 2009). If the transactions have concurrent access to the data, the deadlock condition may occur. Usually, the deadlock also occurs on workflow models (Fan *et al.*, 2010); technology that implement the automation of business processes in whole or part, embedded applications (Xiao *et al.*, 2010; Xiao *et al.*, 2011), automated manufacturing systems (Roszkowska, 2004), multitasking operating systems (Cheung *et al.*, 2009), and streaming computations (Li *et al.*, 2010). The deadlock problem is inherent in a distributed database system which employs locking (Clauss *et al.*, 2010) as its concurrency control algorithm. Several researches have been carried out regarding the handling of deadlock problems which are by Alom et al, 2010; Olson *et al.*, 2005; Wu *et al.*, 2002; Atreya *et al.*, 2007; Srinivasan *et al.*, 2011; Jiang *et al.*, 2008; Mohammed *et al.*, 2007.

1.4 PROBLEM STATEMENT

Ensuring efficient access to such a huge network and widely distributed data is a challenge to those who design, maintain and manage the distributed database. In this system, several characteristics are considered such as: (1) provides an interface for the user which is transparent to where the data actually resides; (2) ability to locate the data; (3) network-wide concurrency control and recovery procedures; (4) translation of queries and data between heterogeneous systems (Bhushan *et al.*, 2007). Replication in distributed environment receives particular attention for providing efficient access to data, fault tolerance and enhance the performance of the system (Gao *et al.*, 2005; Tang *et al.*, 2006).

Even the replication gave more advantages; it still becomes a problem when the concurrent access happens to the data. The lock mechanism is used when the transaction makes request to get a data. If the data is available, the transaction that makes a request will get a lock for that data, otherwise it will wait until the data is unlocked or released then it can be acquired again. In this situation, a deadlock may occur in which every transaction involved in the deadlock is waiting to grant the data that has been locked by another transaction that make a circular wait until an action is taken to detect and

resolve deadlock problems. A deadlock can reduce the throughput by minimizing the available resources, so it becomes an important resource management problem in distributed systems (Srinivasan *et al.*, 2011). In order to manage the deadlock problems, a new deadlock detection and resolution algorithm will be proposed to preserve the consistency of data replication in a distributed environment. The proposed algorithm will be applied to Neighbour Replication on Grid (NRG) replication model. Before this, the researcher does not implement the deadlock detection and resolution on NRG.

The NRG replication model is chosen because it can maximize the write availability with low communication cost due to the minimum number of quorum size required compared to other techniques such as Read-One-Write-All (ROWA), Branch Replication Scheme (BRS), and Hierarchical Replication Scheme (HRS). In ROWA technique (Noraziah et al., 2010d) read operation has low communication cost. This technique restricts the availability of write operations since they cannot be executed at the failure of any copy. In BRS technique (Pérez et al., 2010), the replicas are created as close as possible to the clients who request for the data file. The root replica grows towards the clients in a branching way, slip replicas into several sub replicas (Noraziah et al., 2010c). In this technique, the replica tree is grown based on the client needs. In HRS technique, a hierarchical replication consists of a root database server and one or more database servers organized into a hierarchy topology (Pérez et al., 2010). Using this technique, the data will be replicated or copied at all sites and has the highest storage of use. Besides that, the proposed algorithm will be done in order to preserve the data consistency and maximize data availability when the transactions concurrently want to update the data.

1.5 AIM OF RESEARCH

The aim of this research is to handle deadlock problem in replication data through Neighbour Replication on Grid environment in order to preserve the data consistency and increase the throughput by maximizing the availability of resources.

1.6 OBJECTIVES OF RESEARCH

The objectives of the research are as follows:

i. To propose a new framework to manage deadlock problems during transaction execution through Neighbour Replication on Grid (NRG) model.
ii. To develop a new algorithm to manage deadlock problems during transaction execution through Neighbour Replication on Grid (NRG) model.
iii. To compare Neighbour Replication on Grid Deadlock Detection (NRGDD) and Multi-cycle Deadlock Detection and Recovery Algorithm for Distributed System (MC2DR)
iv. To test the new algorithm to ensure their correctness using two case studies.

1.7 SCOPE OF RESEARCH

The scope of this research is as follows:

i. Design a new framework by extending the Multi-cycle Deadlock Detection and Recovery (MC2DR) algorithm in Neighbour Replication on Grid (NRG) replication model.
ii. Develop a Neighbour Replication on Grid Deadlock Detection (NRGDD) algorithm for deadlock detection implemented on NRG.
iii. Consider only non-failure cases.

1.8 ORGANIZATION OF THESIS

This thesis has been prepared to give details on the definitions, facts, observations, arguments and procedures in order to meet its objectives. Chapter 1 generally describes the brief background of data grid environments, data replication, the problem statement, objectives and scope of the research. Chapter 2 presents the literature review of replication, replication strategies, replication technique on grid, transaction handling, concurrency control, deadlock mechanism, deadlock detection model and comparison between the existing of replication model. Chapter 3 presents the new proposed algorithm, Neighbour Replication on Grid Deadlock Detection (NRGDD) to handle deadlock on Neighbour Replication on Grid environment. Framework, flowchart, development of NRGDD simulation model, example cases, comparison between NRGDD and MC2DR and correctness will also be presented in this chapter. Chapter 4 addresses the simulation results for detecting the deadlock existing cycles and compares the executing time taken to detect deadlock based on cycles and the number of transactions used. The conclusions of the present research are summarized and presented in Chapter 5. Research objective achievements and suggestion and recommendation for the future work are also presented in this chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

This chapter reviews literatures on replication including its general information, strategies and techniques implemented on grid. Besides that, the review is also done on transaction handling, concurrency control, and deadlock mechanism including its techniques in detecting deadlock occurrence. Figure 2.1 below shows the general to detail of the review that has been done.

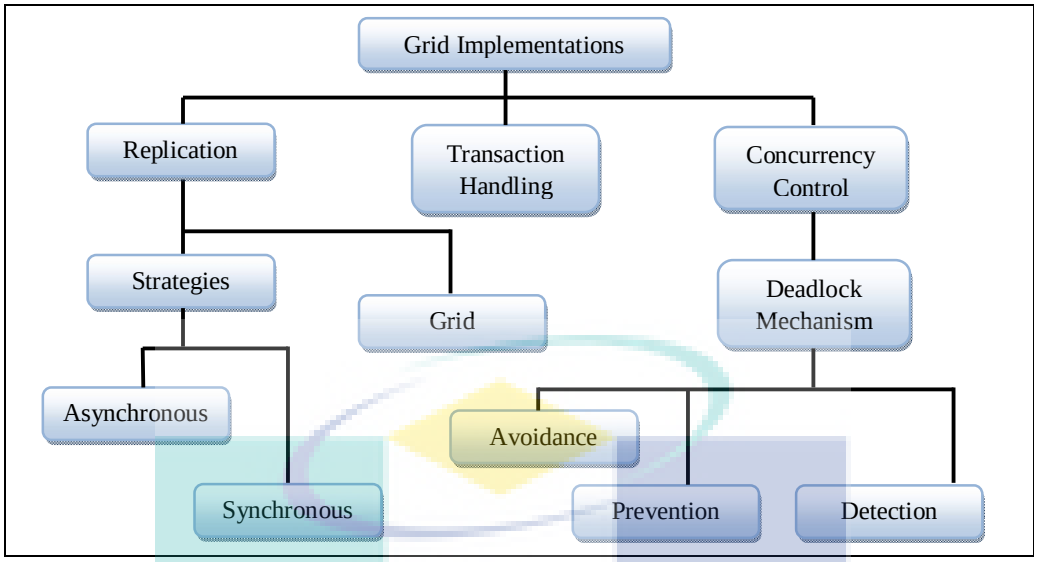


Figure 2.1: General to detail of literature review

2.2 REPLICATION

Replication receives particular attention for providing high data availability, fault tolerance and performance enhancement of the system (Noraziah *et al.*, 2007; Amjad *et al.*, 2012). In replication, the identical data copies can be replicated to another site or place. Replication means that 100% of the same data is on other locations (Bost *et al.*, 2009). It can enable organizations to provide users with access to current data where and when they need it. If one of the sites has failed, they can obtain data from an identical data from other sites and the failure of the system can be transparent for users and applications.

In minimizing communication costs during data access, replication of data from the primary site to other locations can be an important optimization step to reduce the frequency of remote data access (Abdi *et al.*, 2010; Zhang *et al.*, 2009). Furthermore, it can improve performance by scaling the number of replicas with demand and by offering nearby copies of services distributed over the network (Noraziah *et al.*, 2007). Huge datasets are collected and stored in different geographic locations, but are organized through a network with certain topological structures to provide reliable resources (Zhang *et al.*, 2009).

In replication, the value of each logical item is stored in one or more physical data items. Each reads or writes operation on a logical data item must be mapped to corresponding operations on physical copies (Noraziah *et al.*, 2007). Applications of such systems broaden across many domains, including business applications (e.g., Bank transactions, retail transactions, e-commerce, etc.), scientific applications (e.g., NASA's Earth Observing System, Sky Survey, etc.) (Zhang *et al.*, 2009). Furthermore, through replication, various users can use the data in different locations which can decrease the latency and increase the reliability of data (Li *et al.*, 2010). For example, almost every digital enterprise has a compelling reason to employ some form of database replication for disaster recovery or high availability (Wong *et al.*, 2009).

Data replication can drive by programs which transfer data to some other location and then loaded at the receiving location and the data may be filtered and transformed during replication (Beg *et al.*, 2010). The disadvantage of replication is it becomes more of a security risk. Handling security across several locations is more complicated (Alkhatib *et al.*, 1995). But the biggest problem that data replication introduces is that of concurrency control. It isalready known that the concurrency control is an issue even without replicated tables; with replicated tables, it becomes even more complex. How do we ensure data consistency when it is replicated to more than one site? Without properly placing the replicas, the overall availability can be hindered because of data consistency requirements (Zhang *et al.*, 2009).

2.3 REPLICATION STRATEGIES

Replication strategies can use either asynchronous or synchronous replication to copy data.

2.3.1 Asynchronous Replication

In asynchronous replication, changes are made after a certain time with a lot of data from the master site replicated to different other site (Beg *et al.*, 2010). Normally, in asynchronous replication, such a transaction writes its commit record to

the redo logs, releases all locks, waits for an acknowledgement from the storage system, and finally sends an acknowledgement to the user.

2.3.2 Synchronous Replication

Synchronization mechanisms are needed to maintain the consistency and integrity of data among replicas when changes are made by the transactions (Noraziah *et al.*, 2007). In (Beg *et al.*, 2010) mentioned. With synchronous replication, changes are made immediately once some data transaction occurs to the master sites. Once the updated operation occurs on the primary copy, an update on the transaction results immediately replicates the update at all other sites which is identical to the primary copy. It is the suitable solution for organizations which are seeking for the fastest possible data recovery, minimal data loss, and protection against database integrity problems (Beg *et al.*, 2010).

There are many examples of replication schemes in distributed file and database systems that are almost based on synchronous replicating, which deploy quorum to execute the operations with a high degree of consistency and ensure serializibility (Noraziah *et al.*, 2007). Several schemes are used in synchronous replication, i.e., all-data-to-all-sites (full replication) and some-data-items-to-all-sites. The maximum approach of replicating every table at every site (full replication) is great for availability, but it is the absolute worst arrangement regarding concurrency control. Furthermore, it causes high update propagation and high storage capacity. The replication model that uses this scheme is ROWA (Noraziah *et al.*, 2010d). A few studies have been done in partial replication technique based on some data items to all sites using the tree structure technique. However, this technique will cause high update propagation overhead. Therefore, some-data-items-to-all-sites scheme is not realistic. Moreover, in many applications, there is update-intensive data, which should be replicated to very few sites.

In this strategy, a user transaction is not allowed to commit except it is guaranteed that it will be successfully applied on all the replicas. This guarantee can be achieved by relying on the well-known two-phase commit protocol (Alkhatib *et*

al., 1995), or by middleware that imposes a global order on all user transactions via an atomic broadcast service (Wong *et al.*, 2009). The main benefit of synchronous replication is the data can be recovered quickly and automatically handles concurrent conflicting transactions on different replica databases. The main disadvantage is that it greatly reduces the throughput of dependent transactions on each replica. The critical issue is the time period when a transaction is complete and is seeking to commit. The transaction must first broadcast its intention to commit to the other replicas and wait for an acknowledgement. The precise mechanics of this communication depend on the particular implementation, but in all implementations, database locks cannot be released until the minimum time required for one network round-trip has elapsed. This network round-trip means that dependent transactions can commit no faster than the network latency permits.

2.4 REPLICATION TECHNIQUES ON GRID

In managing the replication on the data grid, the replica placement strategy is important due to the limited storage use of data grids. There are three fundamental questions (Ben Charrada *et al.*, 2010b; Zhao *et al.*, 2010) that must be answerable to managing replica placement strategy in data grids:

1. When should the replicas be created?
2. Which files should be replicated?
3. Where should the replicasbe placed?

There is no restriction of users or originations in grid computing. Replication is one of the basic and the key aspects in grid computing. The importance of data replication in grid systems and distributed systems (Du *et al.*, 2011) are as the following:

1. It can effectively increase data access performance from different locations, and thus reduce data access time cost;
2. It enables a system to handle more workload, as more nodes can be served at the same time;

3. It increases system availability;
4. It can be treated as a backup, and thus ensures the dependability of the data.

2.4.1 Neighbour Replication on Grid (NRG)

In NRG model, all sites are logically organized in the form of a two-dimensional grid structure (Noraziah *et al.*, 2006; 2009b). As an example, if NRG consists of sixteen sites, it will logically organize in the form of 4 x 4 grid as shown in Figure 2.1. Each site contains a master data object. A site is either operational or failed and the state (operational or failed) of each site is statistically independent of the others (Noraziah *et al.*, 2006; 2009b). The copy on the site is available when a site is operational, otherwise it is unavailable.

A data at the primary site is replicated to the neighbouring sites. Let $N = n^2$ be a set of all sites that are logically organized in two-dimensional grid structure (GS) form (Noraziah *et al.*, 2009b). The *N* sites are labelled as $n(i, j)$, where $1 \leq i < n$, $1 \leq j < n$ (Noraziah *et al.*, 2009b). The site $n(i, j)$ will connects to its neighbours through two way links as long as there are sites in the grid, which are sites $n(i \pm 1, j)$ and $n(i, j \pm 1)$. The number of data replication is $d \leq 5$. For example from Figure 2.2, data from site 2 will replicate to its neighbours which are site 1, site 3, and site 6. Site 6 has four neighbours, which are sites 2, 5, 7, and 10. As such, site 6 has five replicas. Similarly, site 7, 10 and 11 also has four neighbours and five replicas. Each of the primary sites of any data object and its neighbours is assigned to vote one and if not vote zero. This vote assignment is called binary vote assignment on grid (Noraziah *et al.*, 2006; 2009b). A neighbour binary vote grid assignment on the grid, *B*, is a function such that $B(n(i, j)) \in \{0, 1\}$, $1 \leq i < n$ and $1 \leq j < n$, where $B(n(j, j))$ is the vote assigned to the site $n(i, j)$ (Noraziah *et al.*, 2009b). This assignment is treated as an allocation of replicated copies and a vote assigned to the site results in a copy allocated at the neighbour (Noraziah *et al.*, 2006; 2009b).

That is 1 vote ≡ 1 copy. Let $L_B = \sum_{i=1}^{n} B(n(i,j))$ Where L_B is the total number of votes assigned both to the primary site and its neighbours and it also equals the number of copies of an object allocated in the system.

Let us denote that r and w are the read and write quorums, respectively. The $r + w$ must be greater than the total number of copies (votes) assigned to all sites in order to ensure that the read operations always get an up-to-date value. The following conditions are used to ensure the consistency (Noraziah *et al.*, 2009b):

$$1 \leq r \leq L_B, 1 \leq w \leq L_B \quad (2.1)$$

$$r + w = L_B + 1 \quad (2.2)$$

The conditions (2.1) and (2.2) ensure that there is a non-empty intersection of copies between every pair of read and write operations. Therefore, this condition ensures that a read operation has access to the most recently updated copy of replicated data based on the timestamps (Noraziah *et al.*, 2006). Timestamps are used to decide which copies are most recently updated.

Let $S(B)$ be the set of sites at which replicated copies are stored equivalent to the assignment. Then, $S(B) = \{n(i, j) \mid B(n(i, j)) = 1, 1 \leq i \leq n, 1 \leq j \leq n\}$.

Let $Q(B, q)$ be the quorum set with respect to the assignment B and quorum q, then $Q(B, q) = \{G | G \subseteq S(B)$ and $|G| \geq q\}$ (Noraziah *et al.*, 2009b). As an example, from Figure 2.2, let site 7 be the primary site of the primary data x. Its neighbours are sites 3, 6, 8, and 11.

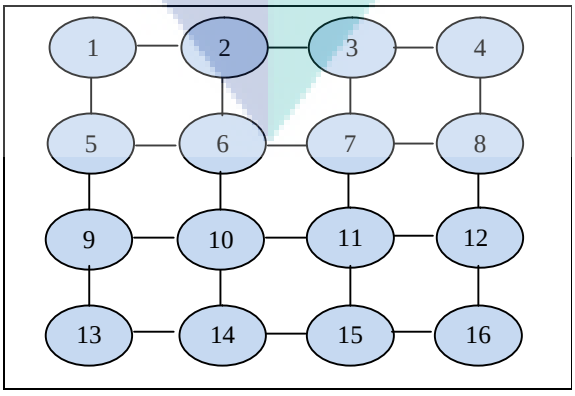


Figure 2.2: A grid organization of 16 copies of an object

Consider an assignment B for the data file x, such that $B_x(5) = B_x(3) = B_x(6) = B_x(8) = B_x(11) = 1$ and $L_{Bx} = B_x(5) + B_x(3) + B_x(6) + B_x(8) + B_x(11) = 5$. Thus, $S(B_x) = \{7, 3, 6, 8, 11\}$. If read quorum for data file x, $r = 2$ and write quorum $w = L_{Bx} - r + 1 = 4$, then the quorum sets for read and write operations are $Q(B_x, 2)$ and $Q(B_x, 4)$, respectively (nrg3), where

$Q(B_x, 2) =$

$$\{\{7,3\},\{7,6\},\{7,8\},\{7,11\},\{7,3,6\},\{7,3,8\},\{7,3,11\},\{7,6,8\},\{7,6,11\},\{7,8,11\},\{3,6,8\},\{3,6,11\},\{3,8,11\},\{6,8,11\},\{7,3,6,8\},\{7,3,6,11\},\{7,3,8,11\},\{7,6,8,11\},\{3,6,8,11\},\{7,3,6,8,11\}\}$$

$Q(B_x, 4)$

$$=\{\{7,3,6,8\},\{7,3,6,11\},\{7,3,8,11\},\{7,6,8,11\},\{3,6,8,11\},\{7,3,6,8,11\}\}.$$

Figure 2.3 shows the process of data replication in NRG. In NRG, a set of transaction start to lock a site by initiate lock. Then, it will propagate lock to request another site. At this stage, what happen if five transactions concurrently want to request or lock at five replica sites? If this situation happen, deadlock will occur. The past researcher do not consider any solution if this situation occurs on NRG. The proposed algorithm is develop to handle deadlock on NRG.

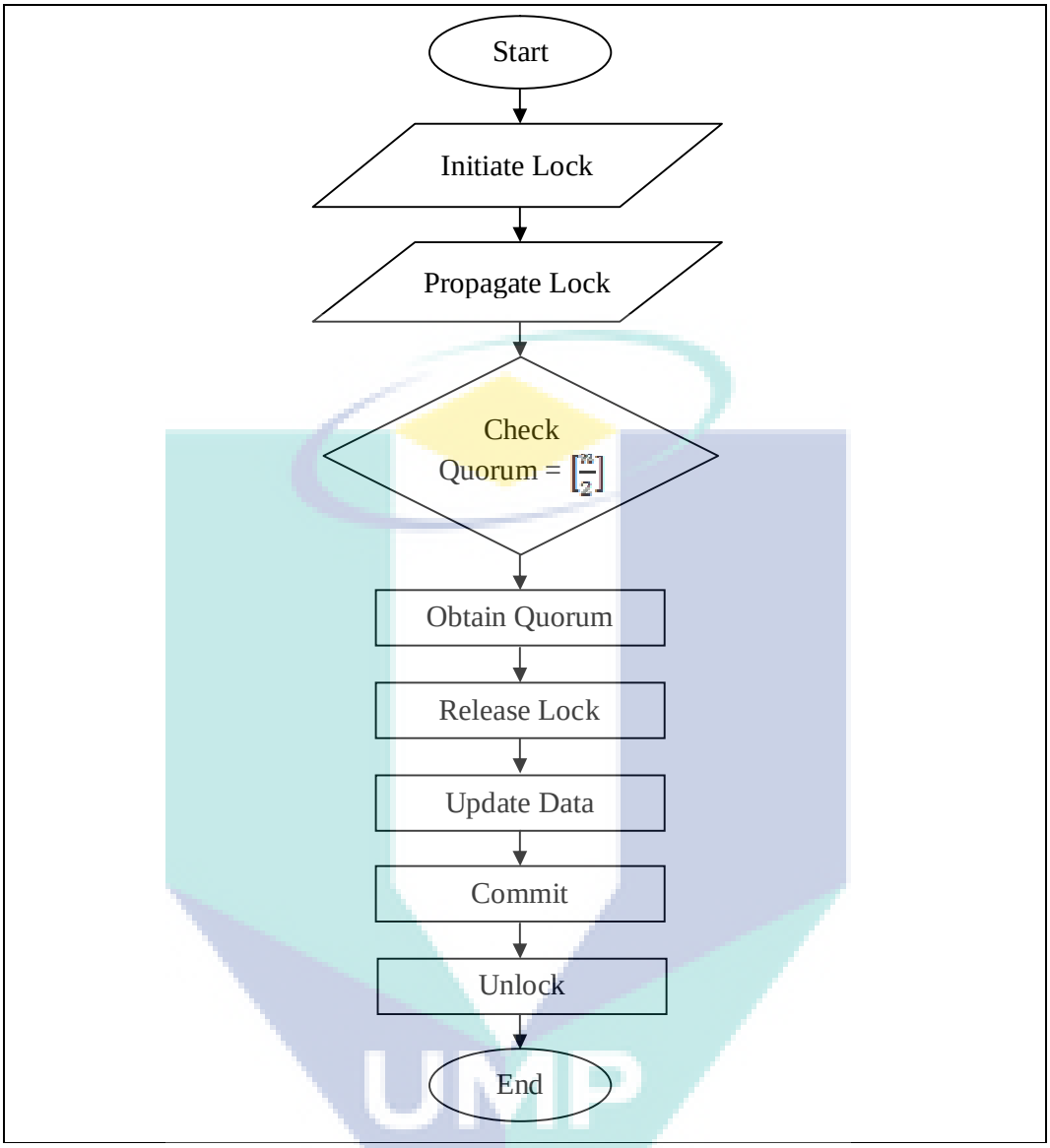


Figure 2.3: Process in NRG

2.5 TRANSACTION HANDLING

A transaction (Enokido *et al.*, 2008; Eya *et al.*, 2011; Khachana *et al.*, 2011; Alkhatib *et al.*, 1995; Xiao *et al.*, 2007; Garcia-Munoz *et al.*, 2007; Zheng *et al.*, 2010; Sanzo *et al.*, 2010) consists of a series or group of operations performed on a distributed system. The examples of such distributed applications involving frequent transactions are distributed databases, distributed-agent based systems and the numerical analysis as well as simulation applications such as gene analysis or climate

modelling systems (Bagchi, 2011). There are thousands of distributed nodes which are connected through network in a very large scale distributed systems such as the cloud computing and grid computing platforms.

A transaction consists of four properties that lead to the consistency and reliability of a distributed database. These are Atomicity, Consistency, Isolation, and Durability, also known as ACID (Garcia-Munoz *et al.*, 2007; Alkhatib *et al.*, 1995; Khachana *et al.*, 2011). Table 2.1 shows the description of ACID.

Table 2.1: The descriptions of ACID

ACID	DESCRIPTION
Atomicity	• Atomicity means all the actions related to a transaction are complete or none of them is carried out. • The recovery of transaction can be split into two types which correspond to the two types of failures: o The transaction recovery – due to the system terminating one of the transactions because of deadlock handling. o The crash recovery – it is done after a system crash or hardware failure.
Consistency	• Consistency means that the committed data must be left in a consistent manner when the transaction has run its course. • Refer to its correctness that deals with maintaining consistent data in a database.
Isolation	• Isolation means that transactions taking place at the same time may show other transactions only the committed data. • Each transaction must maintain the consistency of database at all times. Consequently, no other transaction can read or modify data that is being modified by another transaction. • If this property is not maintained, one of two things which are lost updates and cascading aborts could happen to the database.

Durability	• Durability means that the committed data have to be made permanent. • Once a transaction commits, its results are permanent and cannot be erased from the database • This means that whatever happens after the COMMIT of a transaction, whether it is a system crash or aborts of other transactions, the results already committed are not modified or undone.

2.6 CONCURRENCY CONTROL

The transaction of process normally concurrently has access to the shared data. When multiple transactions are executed concurrently and involved in accessing to the data, data consistency (Bagchi, 2011; Clauss *et al.*, 2010) can be affected because of mutual interference of concurrent transactions (Xiao *et al.*, 2007). Only one process can be accessed to one data. Therefore, the concurrent communication in a group of distributed processes requires concurrency control as well as message ordering mechanisms, which should be scalable and fault tolerant in nature (Bagchi, 2011).

The concurrency control (Garcia-Munoz *et al.*, 2007; Zheng *et al.*, 2010; Sanzo *et al.*, 2010; Sanzo *et al.*, 2008) protocols can avoid the mutual interference of concurrent transactions. It can be done by controlling executing orders of concurrent data operations which are employed to guarantee logical consistency of shared data. There are two basic types of concurrency control mechanisms which are optimistic concurrency control (OCC) (Garcia-Munoz *et al.*, 2007; Bai *et al.*, 2008; Zheng *et al.*, 2010) and pessimistic concurrency control (PCC) (Garcia-Munoz *et al.*, 2007; Zheng *et al.*, 2010). In OCC, it is assumed that the transaction conflicts are improbable to occur when shared data are accessed. Consequently, remote server resources can stay mainly unused until transactions commit time. If these conflicts do occur, then transactions are aborted without further ado, and maybe it will try to retrieve or access again. Besides that, in PCC, the conflicts are expected to occur,

and remote resources must be ready to be used on demand at any time during transaction time. Unless a deadlock occurs, the transactions will terminate successfully by pessimistic concurrency control. The PCC policy has been used in locking based. There are two types of locking based used in distributed systems for handling concurrency control which are Two Phase Locking (2PL) (Sanzo *et al.*, 2010; Bai *et al.*, 2008; Zheng *et al.*, 2010), and Strict 2PL.

In detecting and resolving conflict among transactions in distributed systems, the locking based protocols usually combine two phase locking (2PL) with a priority scheme. However,some intrinsic problems of 2PL such as the possibility of deadlocks (Zheng *et al.*, 2010) and long blocking times make transactions difficult to meet their deadlines (Bai *et al.*, 2008). Nevertheless, by using locking manner, when a client updates data, other clients who update the data will not be affected. Thus, data loss can be avoided, as well as read data (Zheng *et al.*, 2010).

In order to ensure serializability scheduling, locking protocol must be observed. The transaction occurs in the specific data object to grant or lock that resource. If the application is successful, operations may continue, or else it will have to wait for the corresponding transaction release of lock resources. In handling concurrency control in transactions, ensuring the consistency of data is important in order to provide the real data to the user.

If a transaction runs across two sites, it may commit at one site and may fail at another site, leading to an inconsistent transaction. Two-phase commit protocol (Eya *et al.*, 2011; Alkhatib *et al.*, 1995; Singh *et al.*, 2009; Khachana *et al.*, 2011**)** is most widely used to solve these problems. The two phase commit policy has become a standard for distributed systems. The commit protocols are implemented in distributed database system to ensure the transaction atomicity. The two phase commit protocol consists of two phases as shown in Table 2.2.

Table 2.2: Descriptions of two phases consisted in the two phase commit protocol

Phase	Descriptions
Prepared phase	• The coordinator asks all participating sites to send a commit or abort vote for the transaction which has been executed (not committed).
Decision phase	• If coordinators receives the commit vote or yes vote from all the participants' sites then it issues an instruction to commit to all the participants. • If it receives abort vote or no vote from any of the participants then it sends abort decision to all the participating sites. • Prepares participant after getting decision from the coordinator and releases the data resources pertaining to the transaction in order to preserve the atomicity of the distributed transaction.

The commit protocol ensures the transaction atomicity (Eya *et al.*, 2011). Global transactions may consist of multiple sub transactions that may execute on different remote sites. Commit protocol forces sub transaction to agree on a single outcome which means that a global transaction will commit if and only if all the sub transactions commit. In case if any of the subtransaction fails, the global transaction aborts and forces successfully executed (not committed) to abort and the previous state of the system is restored (Eya *et al.*, 2011).

The transaction management deals with the problems of keeping the database in a consistent state even when concurrent accesses and failures occur, (Alkhatib *et al.*, 1995). Atomic commit protocol is used to ensure the consistency of data during transaction occurrence especially when update data has happened. Besides that, it is also used for data integrity (Eya *et al.*, 2011).

2.7 DEADLOCK MECHANISM

Deadlock is defined as a system state in which every process in a set is waiting for an indefinite period for another process on the same set. This will continuously happen for their requests to be satisfied.

In a distributed environment, when a thread or a process needs a resource on another site for its computations, a message (Nyo, 2009; Lee *et al.*, 2005; Clauss *et al.*, 2010) will be sent to the requested site through a communication network to grant access to its resource. If the resource is available, it will be granted to the requesting process; otherwise the requesting process needs to wait until the resource is released then it can require reaccess to that resource. In this situation, deadlock (Nyo, 2009; Thiare, 2009) may occur in which processes involved in the deadlock are waiting indefinitely in a circular fashion until a special action is taken (Abd El-Gwad *et al.*, 2009; Mohammed, 2007). Unfortunately, it will reduce the throughput by minimizing the available resources (Srinivasan *et al.*, 2011; Hu *et al.*, 2009; Lee *et al.*, 2005); therefore it becomes an important problem for resource management in distributed systems. It is a highly undesirable situation at which the entire or partial system is crippled (Hu *et al.*, 2009) and become stuck so that restricts the system to operate or run as usual.

Deadlock may arise since the resources are limited such as a fixed-size pool of threads or locks protecting mutually exclusive regions and multiple processes have been spawned at different sites (Sanchez *et al.*, 2007). There are four common conditions necessary for a deadlock to occur among concurrent processes (Gomez *et al.*, 2010; Mayer *et al.*, 2010) as shows in Table 2.3:

Table 2.3: Four common conditions for deadlock to occur

Conditions	Descriptions
Mutual exclusion	• Processes require the exclusive use of resources
Hold while waiting	• Process hold onto resources while waiting for additional

	required resources to become available.
No pre-emption	• Processes holding resources determine when they are released.
Circular waiting	• Closed chain of processes in which each process is waiting for a resource held by the next process in the chain.

The deadlock becomes a hot topic for the past few years and also today the researcher still try to find the best solution to handle this problem in distributed environments. The researchers still try to find the way to implement the semantics of synchronizing merges without deadlock happening to the system (Fan *et al.*, 2010). There are three techniques traditionally used to deal with deadlocks: i) avoidance, ii) prevention () and iii) detection (Xiao *et al.*, 2010; 2011; Hu *et al.*, 2009; Mohammed, 2007; Cheung *et al.*, 2009; Sanchez *et al.*, 2007; Fan *et al.*, 2010) which will be discussed in the following section.

2.7.1 Deadlock Avoidance

Deadlock avoidance is a method that takes a middle route which is a run-time protocol implementing a resource allocation controller (RAC) (Sanchez *et al.*, 2007). Moreover, it is also an event driven and avoids actions that may cause the system to be in a deadlock (Xiao *et al.*, 2011). Based on the resource availability and possible future requests, it will decide whether to grant a request or not. A resource is granted only if it is safe (Sanchez *et al.*, 2007). When a process enters the system it must inform the protocol about its resource utilization. Since the controller has such strategy, all processes can complete. In addition, deadlock avoidance is used as a part of scheduling algorithm to prefer at least one possible execution path where no deadlock will occur (Cheung *et al.*, 2009).

2.7.2 Deadlock Prevention

Deadlock prevention ensures that one of the necessary conditions for deadlock is broken (Sanchez *et al.*, 2007). It is possible if particular resource allocation policies are applied (Cheung *et al.*, 2009) and it is also sufficient to assure that at least one of the four necessary conditions of deadlock is not fulfilled to avoid the deadlock to happen (Mayer *et al.*, 2010). In Hu *et al.* (2009), deadlock prevention refers to a group of static rule imposing restrictions on the interactions among resource requested that may lead to deadlock. In Xiao *et al.* (2011), deadlock prevention utilizes system designs and mechanisms which disallow the system from continually entering a deadlock state.

2.7.3 Deadlock Detection

Deadlock is allowed to occur while a monitoring mechanism is deployed for detecting their correctness and a recovery procedure is initiated for convenient resolution (Hu *et al.*, 2009). This approach is applicable only when deadlock states temporarily exist. In Sanchez *et al.* (2007), deadlock detection is an optimistic method for concurrency control, where deadlocks are detected and corrected at a run-time, such as the rollback of transactions. Deadlock detection is an approach commonly used in databases but usually not applicable in embedded systems (especially systems that interact with physical devices) (Sanchez *et al.*, 2007). In deadlock detection, a resource allocation graph or state graph is normally used to analyse and identify deadlock situations (Cheung *et al.*, 2009). At least two major deficiencies (Clauss *et al.*, 2010) created in the system to ensure the deadlock is present are:

1. The resources will not be available to other processes when it is held by deadlock processes.
2. Each process involved in the deadlock will add the deadlock persistence time to its response time.

A directed graph called Wait-For Graph (WFG) (Clauss *et al.*, 2010; Lee *et al.*, 2005; Mitchell *et al.*, 1984) is used to show the dependent relationship between processes in distribution systems. A cycle in this graph indicates the presence of a deadlock in the system. Each node in this graph corresponds to a process and an edge directed from one node to another indicates that the first process is waiting for a resource which was held by another process.

2.8 DEADLOCK DETECTION MODEL

This section describes the existing algorithm to handle the deadlock problems in distributed system that was proposed by Selvaraj *et al.* (2011), Razzaque *et al.* (2007) and Alom *et al.* (2009). Besides that, at the end of this section comparison is made for each algorithm.

2.8.1 Decentralized Algorithm for Detection Generalized Deadlock in Distributed Systems

A new decentralized algorithm for detection generalized deadlock in distributed systems was proposed in Selvaraj *et al.* (2011). It can handle the concurrent executions of the algorithm. Based on this algorithm, the initiator builds the Distributed Spanning Tree (DST) of Wait-For Graph (WFG) through a propagating probe (CALL) messages along the outgoing edge of WFG in the forward phase. In the WFG, each node represents a process and an arc represents dependency relations between the processes. The initiator receives the backwards replies (REPORT) in the backward phase; the algorithm determines the reducibility of a blocked node. Until it receives a reply in response to all probes (CALL messages), the reducibility of a blocked node is arbitrarily delayed. An unblocked process initiates the reduction of distributed snapshots by eliminating all the reducible nodes during the backward phase. Then, deadlock processes are declared through the processes that have not been reduced in the snapshot. Figure 2.4 shows the WFG for the cycle of deadlock in the form of tree structure.

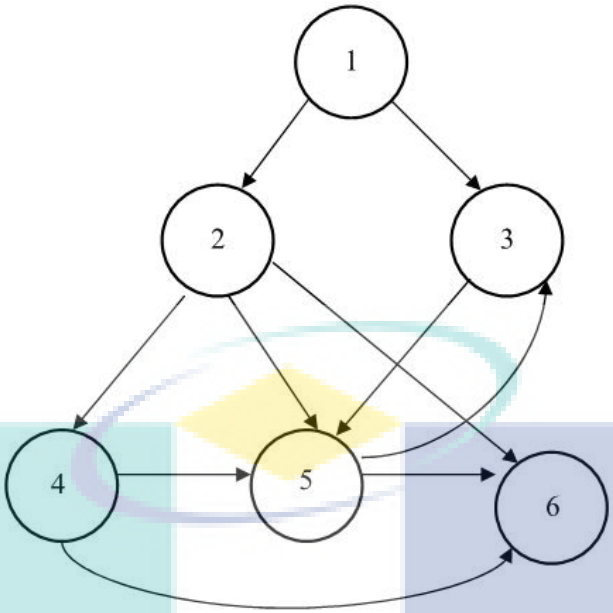


Figure 2.4: The Wait-for Graph

The advantages of this algorithm are the unblocking conditions which were not carried by the replies. The reducibility of node is not delayed until the termination of algorithm. Besides that, the initiator does not construct WFG partially to find out the victim. The DST was built from distributed WFG when the node or process initiates the deadlock detection algorithm as shown in Figure 2.5. The initiator propagates the CALL message along the edges in the WFG. After the successors of the initiator received the CALL message it is then sent to their successor until the end stage of tree. From end of edge, it then sends REPORT message to their predecessors until initiator receives REPORT message from its own successors. Then the algorithm will declare a deadlock. When the initiator detects a deadlock, it sends abort signal to the victim directly.

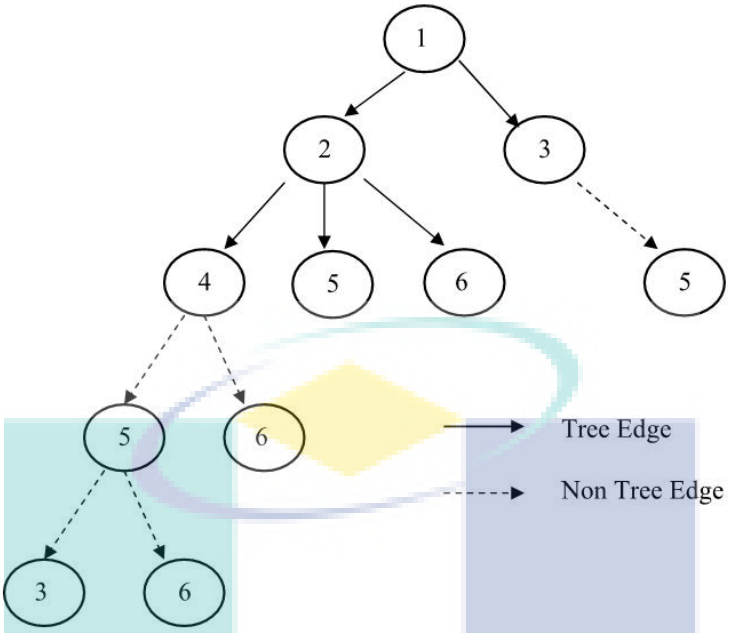


Figure 2.5: The distributed spanning tree

2.8.2 Multi-cycle Deadlock Detection and Recovery Algorithm for Distributed System

Multi-cycle deadlock detection and recovery (MC2DR) algorithm is used to detect the multi-cycle of deadlock problems. The suitable algorithm that will be chosen in order to solve the deadlock problems is important because some of the algorithm cannot detect the presence of deadlock also known as phantom deadlocks and some of them cannot detect deadlocks when the single node or transaction or process is involved in multiple deadlock cycles.

MC2DR were proposed to detect multiple cycle of deadlock and some changes have been made such as a probe message structure, a victim message structure and probe storage structure for each node or transaction or process. Razzaque *et al.* (2007) contributes that MC2DR can:

1. Detect all deadlocks reachable from the initiator of the algorithm in a single execution, even though the initiator does not belong to any deadlock

2. Detect multi-cycle deadlocks i.e., deadlocks where a single process is involved in many deadlock cycles,
3. Decrease the deadlock detection algorithm initiations, phantom deadlock detections, deadlock detection duration and the number of useless messages
4. Provide with an efficient deadlock resolution method.

The MC2DR used probe message for deadlock detection that consists of four fields as shown in Figure 2.6 (a). The *InitID* contains the identity of the initiator of the algorithm, *VictimID* is the identity of the node to be victimized upon detection of the deadlock, *DepCnt* of a node represents the number of successor for which it is waiting for resources, and *RouteString* contains the node IDs visited by a probe message in order (Razzaque *et al.*, 2007). At each node, there will be a probe message storage structure, named *ProbeStorage*, same as that of the probe message for temporary storage of probes (Razzaque *et al.*, 2007). Only one probe message will store in Probe Storage at a particular time. MC2DR is history independent and upon detection of a deadlock, the respective probe message is erased from storage and the node that detects the deadlock sends a victim message to the node found to be victimized for deadlock resolution (Razzaque *et al.*, 2007). This message will be used for deleting probes from respective storage entries. This short message contains just the first two fields of the probe message as shown in Figure 2.6 (b) (Razzaque *et al.*, 2007).

(a) Probe Message (b) Victim Message

Figure 2.6: Structure of probe and victim message

In Razzaque *et al.* (2007), they did not mention or consider any replication model to simulate the MC2DR algorithm. The Figure 2.7 shows pseudo code of MC2DR.

```
Algorithm_Initiation()
{
        int W; //waiting time for a particular resource
        probe p; allocate memory for p;
        if (W > To && ProbeStorage == NULL)
        {
                p = Create_Probe(i); Send_Probe(i, p);
        }
}

probe Create_Probe(node i)
{
        p.InitID = i.ID;
        p.VictimID = i.ID; p.DepCnt = i.DepCnt;
        p.RouteString = i.ID; return (p);
}

Send_Probe(node i, probe p)
{
        int j = i.DeptCnt;
        while (j)
        {
                //sends probe to all successors, j
                send (j, p); j--;
        }
}

Receive_Probe(probe p)
{
        if (ProbeStorage == NULL)
        {
        if( p.DepCnt < i.DepCnt)
        {
                p.VictimID = i.ID; p.DepCnt = i.DepCnt;
        }
                p.RouteSting = p.RouteString + i.ID;
                Send_Probe(i,p);
        }
        else if( i.RouteString is prefix of p.RouteString)
        {
                Deadlock is detected.
                //send victim message to all successors and simply blocked nodes
                Send_Victim(j, p.VictimID);
        }
        else if (i is the initiator of another probe)
                Exception_Handling(p);
        else
        {
```

```
                Discard (p);
        } //probe message is discarded
}

Receive_Victim(int VictimID)
{
        //forward victim message to all successors
        Send_Victim(j, VictimID);
        if(VictimID == i.ID)
        { // this node is vicitimized
                Release (All locks held by this node);
                Kill (this node);
        }
        else
        {
                Erase Probe message from ProbeStorage;
        }
}

Exception_Handling(probe p)
{
        int Td; //avg. deadlock detection period
        put p in a buffer space;
        wait for Td and check for i's receiving probe
        if(i's probe is received)
        {
                Discard (p);
        }
        else
        {
                p.RouteSting = p.RouteString + i.ID;
                Store(p); //Store p in ProbeStorage
                Send_Probe(i, p);
        }
}
```

Figure 2.7: Pseudo code of MC2DR

2.8.3 Deadlock Detection Views of Distributed Database

Deadlock detection is very difficult in a distributed database system because no controller has completed and current information about the system and data dependencies (Alom *et al.*, 2009). The proposed algorithm shows that the global deadlock is not dependent on the local deadlock (Alom *et al.*, 2009).

A deadlock detection algorithm or technique is correct if it satisfies two conditions: (1) every deadlock is eventually detected, and (2) every detected deadlock really exists, i.e., only genuine deadlocks are detected (Alom *et al.*, 2009). The algorithm is based on creating Linear Transaction Structure (LTS) that used to find the local cycle of deadlock, Distributed Transaction Structure (DTS) is used to find the global cycle of deadlock and deciding priority ID of the transaction will be assigned by the Transaction Manager (TM) and local global abortion. Transaction Queue (TQ) is used to store the priority ID for all transactions which are in local deadlock cycles or in global deadlock cycles; the youngest transactions (priority ID) are aborted to free the system from deadlock cycles.

2.8.4 Comparison Between the Existing of Deadlock Detection Model

The algorithm proposed by Selvaraj *et al*. (2011) shows that only the initiator can detect the node or process as a victim to cause the deadlock to happen. Different with algorithm proposed by Razzaque *et al.* (2007), it mentions that not only the initiator can detect deadlock cycle, but another node or process also can detect the existing of deadlock. In Alom *et al*.(2009), it only shows that local deadlock is not dependent with global deadlock. Besides that, the algorithm proposed by Selvaraj *et al.* (2011), Razzaque *et al.* (2007) and Alom *et al.* (2009) does not mention about the logical data that will be used to test their algorithm. However, the research proposed by Razzaque *et al.* (2007) mentioned that it has simulated the algorithm using fixed sites (20) and only consider write operation to the data objects.

2.9 SUMMARY

This chapter reviews a study on data replication and its strategies such as asynchronous and synchronous replication. This chapter also reviews data replication technique on grid such as NRG. Besides that, a review on transaction handling and concurrency control also will be presented in this chapter. Lastly, deadlock mechanism as well as other researchers' related work on deadlock detection algorithm has been discussed. In this research, deadlock detection is implemented with replication model on NRG. This technique is discussed in the next chapter.

CHAPTER 3

METHODOLOGY

3.1 INTRODUCTION

This chapter focuses on firstly describing the operational framework used in the development of Neighbour Replication Grid Deadlock Detection (NRGDD) then it is followed by a description of The NRGDD Transaction Model. This chapter also includes the flowchart and framework of NRGDD with all possible diagrams, and also the detailed algorithm shown as a pseudo code. Besides that, it also covers the hardware and software specifications, the NRGDD simulation model, and the comparison between NRGDD and Multi-cycle Deadlock Detection and Recovery (MC2DR) (Razzaque *et al.*, 2007). This chapter ends with some examples of cases and correctness.

3.2 OPERATIONAL FRAMEWORK

The methodology used to develop NRGDD has five phases which comprises of literature study, logical design, implementation, testing and analyse the result as shown in Figure 3.1. Every phase in this methodology can be divided into several steps that can be achieved in a suitable time frame.

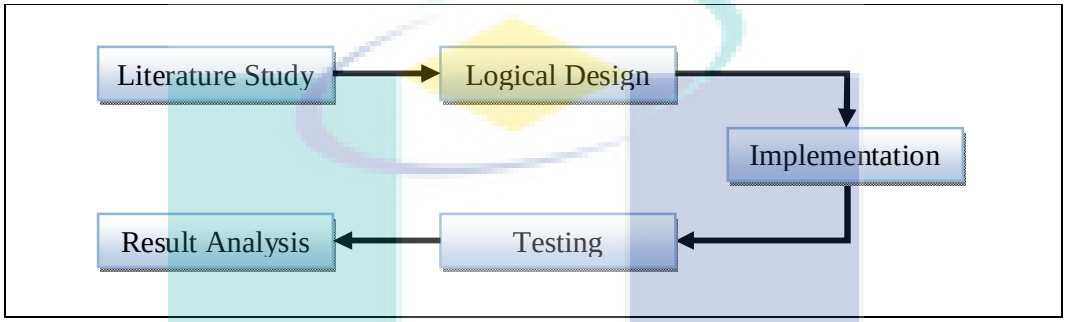


Figure 3.1: Methodology phases

In the literature study phase, a review was made on data replication including its strategies which consist of the asynchronous and synchronous replications. In this phase also, the replication techniques on the data grid were defined. Besides that, a transaction handling and concurrency control and the definition and traditional techniques of deadlock handling including the existing techniques were also reviewed and studied during this phase. Then ultimately, the scope of the research was identified.

In the logical design phase, the proposed framework was designed to support the occurrence and detection of the deadlock on Neighbour Replication on Grid (NRG). Next, the algorithm was designed to manage any deadlock problems that were able to support the deadlock detection and resolution.

During the implementation phase, the NRGDD simulation was developed in order to test the proposed algorithm in handling deadlock problems by using appropriate programming techniques and development tools.

In the testing phase, the algorithm was tested on the NRG replication model in order to ensure the algorithm was correct and well functioned. Correct and well functioned here mean the algorithm can detect deadlock during the time when transactions made their request to grant resource at any sites on NRG. If the algorithm can detect and resolve deadlock on NRG, then the final phase will be implemented. If not, logical design will be revised and followed with the implementation.

At the final phase, the results were analysed. Next, the report was written based on the results of the implementation. The operational framework is summarized as in Figure 3.2.

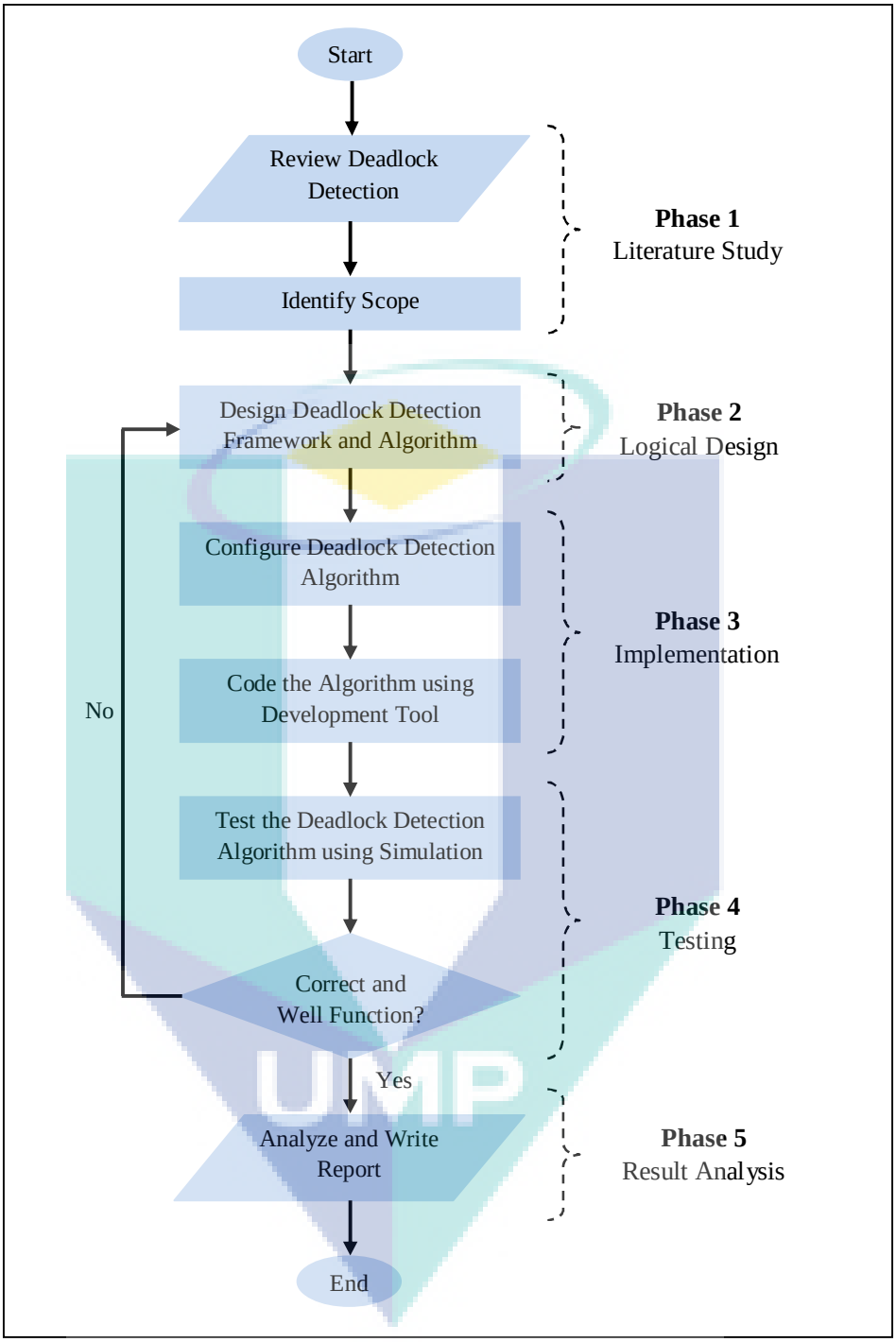


Figure 3.2: Operational framework

3.3 NRGDD MODEL

In the proposed NRGDD model, the deadlock detection algorithm was developed to be implemented in the Neighbour Replication on Grid (NRG) Model. In NRG, all sites are logically organized in the form of two-dimensional grid structure. For example, if NRG consists of twenty-five sites, it will logically organize it in the form of 5 x 5 grids. The detailed explanations on NRG are in Chapter 2.

A site Y is a neighbour to site X, if Y is logically located adjacent to X. A relation replicates to the neighbouring sites from its primary site. Four sites on the corners of the grid have only two adjacent sites, and other sites on the boundaries have only three neighbours. Thus, the number of neighbours of each site is less than or equal to 4. Figure 3.3 shows the NRG model consists of sixteen sites. A site A is a primary site for site B and E. Each site can be primary or neighbour to other sites such as site B become neighbours to site A, but at the same time it becomes primary to its neighbour site A, C and F. A site becomes a primary site will replicate its data to its neighbours. For example, data k from site K replicates to site G, J, L and O.

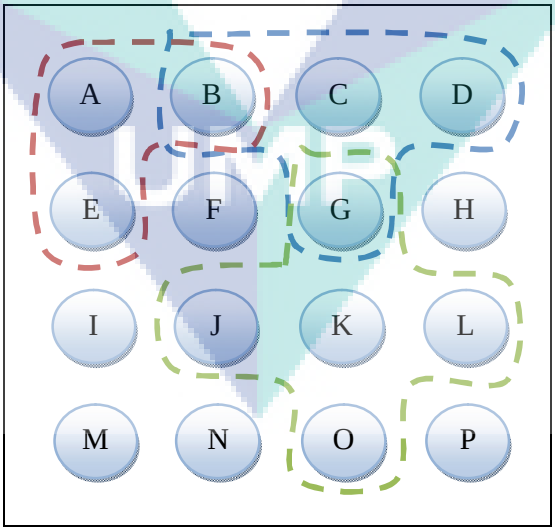


Figure 3.3: Sixteen sites of NRG

The replicated data is requested by different sets of transaction to grant the available resources on the grids. When the transaction is waiting for each other to obtain the same resource at an infinite time, the deadlock may happen. The NRGDD simulation model was developed to handle deadlock on NRG. It can detect more than one cycle of deadlock that happens in NRG.

3.3.1 NRGDD Algorithm Definition

In this section, we defined the following notations:

a) T is a transaction

b) D_x, D is the union of all data object manages by all transaction T of NRG and x represents one data object (or data file) in D to be modified by an element of T_α, T_β, T_γ, T_δ, and T_θ.

c) The element of T_α, T_β, T_γ, T_δ, and T_θ will request the same replicated data object on different sites.

d) $\lambda = \alpha, \beta, \gamma, \delta, \theta$ where it represents a different group for the transaction T (before and until the deadlock is detected and resolved). μ is feedback from other transaction during sending and receiving probe messages and detection and resolution of deadlocks.

e) The *PM* is a probe message. It contains a set of probe messages where *the PM (initID, victimized, ProWait, RouteString)*. See Table 3.1.

f) NRG transaction elements $T_\alpha = \{T_{\alpha_x,PM(initID,victimID,ProWait,RouteString)}\}$, where $T_{\alpha_x,PM}$ is a probe message elements of T_α transaction.

g) NRG transaction elements $T_\beta = \{T_{\beta_x,PM(initID,victimID,ProWait,RouteString)}\}$, where $T_{\beta_x,PM}$ is a probe message element of T_β transaction.

h) NRG transaction elements $T_\gamma = \{T_{\gamma_x,PM(initID,victimID,ProWait,RouteString)}\}$, where $T_{\gamma_x,PM}$ is a probe message element of T_γ transaction.

i) NRG transaction element $T_\delta = \{T_{\delta_x,PM(initID,victimID,ProWait,RouteString)}\}$, where $T_{\delta_x,PM}$ is a probe message element of T_δ transaction.

j) NRG transaction element $T_\theta = \{T_{\theta_x,PM(initID,victimID,ProWait,RouteString)}\}$, where $T_{\theta_x,PM}$ is a probe message element of T_θ transaction.

k) Each node or transaction has a probe message storage structure also known as *ProbeS*, at least, one probe message will be stored in *ProbeS* at a particular time. The history of *ProbeS* is independent; when the deadlock has been detected the probe message is erased from *ProbeS*.

l) Transaction $T_{\lambda_x,PM}$ that detects the deadlock sends a victim message to the transaction found to be victimized for the deadlock resolution. Victim message elements are *initID* and *victimID*. Victim message will be used for deleting probes from respective storage entries.

Table 3.1: Probe message

Probe Message	Descriptions
initID	Contains the identity of the initiator of the algorithm
victimID	A node or transaction that causes the deadlock to occur. This node will be victimized for deadlock resolution.
ProWait	The number of successors representing a node or transaction which is waiting for a resource.
RouteString	The node or transaction IDs visited by another node's (transaction's) probe message in order.

The NRGDD transaction model considers a different set of transactions T_α, T_β, T_γ, T_δ, and T_θ. All elements of the transaction may request data object x simultaneously at any site of *S (B)* either at the same or different sites. Each set of transaction communicate with each other by message passing. Each of them brings the elements of probe message or *PM* where *PM(initID, victimID, ProWait, RouteString)*. At least there will be one probe message being stored in the probe storage, *ProbeS*.

3.3.2 Illustration Of Example

Let's illustrate the working of NRGDD algorithms for detecting deadlock, through an example. Consider the situation shown in Figure 3.4. A different set of transactions T_α, T_β, T_γ, T_δ, and T_θ request a lock from a set of sites where $S(B_x) = \{J, F, I, K, N\}$. Each site contains replicates of data x. If the transaction of $T_{\alpha_x,PM}$ gets a lock from site $i \in S(B_x)$ and on the other transaction will get a lock from other site $j \in S(B_x) \mid j \neq i$. Each site $i \in S(B_x)$ has its own Lock Manager (LM) that processes a request for a lock from the transaction and decide whether the lock can be granted or not. If the lock is free, it is granted immediately; otherwise, the lock manager will send a reject message and insert the requesting transaction or node ID into a waiting list for the lock.

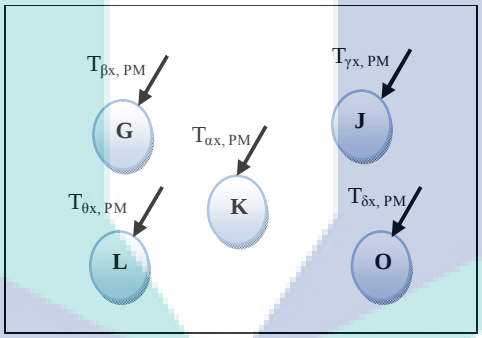


Figure 3.4: Different set of transaction requests a different site

Each of the transaction thus already gets locked from site; they can propagate lock to another site to grant the resources. If the resource is already granted by another transaction, it must wait until the resource is released. While the transaction is waiting to grant the resource at an infinite time, anything happens like it is idle. Figure 3.5 shows how the deadlock occurs and the cycle develops.

The NRGDD algorithm will be implemented by initiating the deadlock algorithm. The algorithm is initiated by any transaction with the waiting time more than the time out. For example, T_α initiates deadlock detection algorithm. Then, T_α creates probe message and sends it to its successor, T_β. When T_β receives probe

message it will compare whether T_β obtains the same replicated data as T_α. If it is the case, then it will check whether the probe storage, *ProbeS* of T_β is empty or not. And it also checks for the number of successors for both transactions. If it is empty and T_β has the highest number of successors, then it will update its *ProWait* and *RouteString*. This step will continue until the transaction that causes the deadlock receives a victim message from the transaction which detects a deadlock. The transaction that becomes a victim for the deadlock to occur has the highest number of successors. Once received the victim message, it sends the message to its successors then it will abort and release the resource that it is holding.

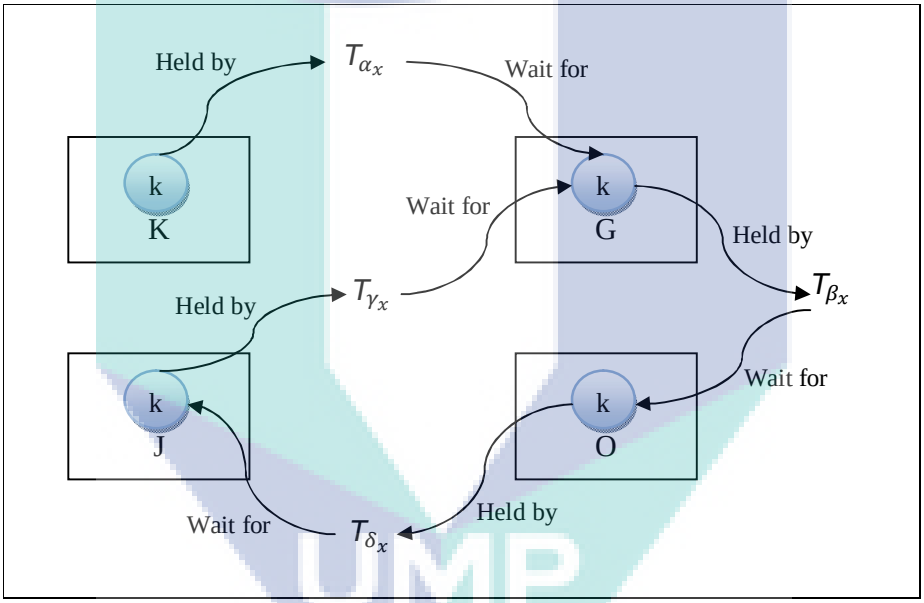


Figure 3.5: Transaction waiting for each other to obtain resources

3.4 NRGDD FRAMEWORK

The process involves in this model is shown in Figure 3.6 that starts with initiating the deadlock algorithm when the time for waiting resource is longer than the time out. Any transaction can initiate a deadlock algorithm. Next, a transaction creates its probe message then sends to its successors. A successor is a transaction that holds a resource which other transaction is waiting for it to be granted. The difference between MC2DR and NRGDD is on the stage of Send and Receive Probe.

In this stage, every transaction that obtains a resource is being compared to ensure it requests for the same replicated data object. This is because the deadlock occurs when a set of different transaction requests to obtain the same resource. Besides that, it is also to ensure that the deadlock really exists. This stage continues to happen until a transaction receives a victim message from a detector which detects another transaction that causes the deadlock to occur. Once the deadlock is detected, a transaction that causes the deadlock will receive a victim message from the transaction which detects it as a victim. After receiving a victim message, it sends the message to its successor(s) and then it will abort or kill itself to resolve the deadlock.

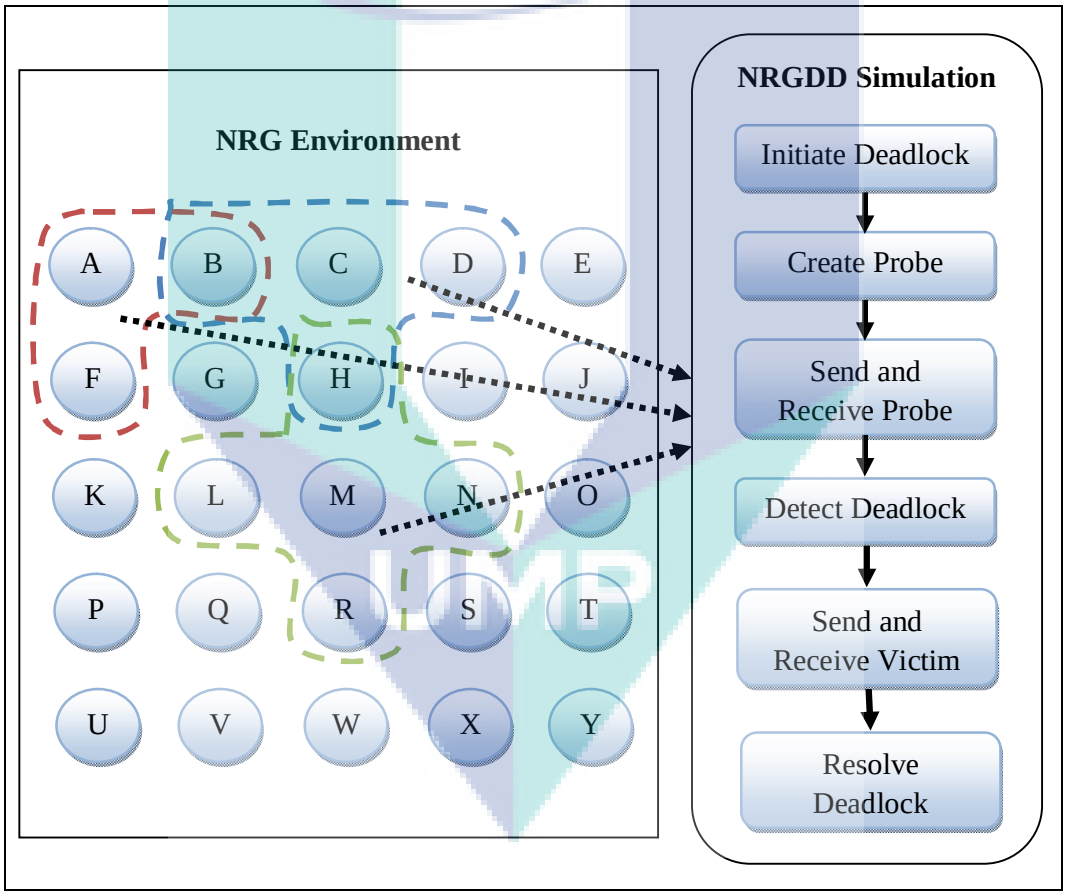


Figure 3.6: Framework of NRGDD model

3.5 COMPLETED FLOWCHART OF NRGDD FRAMEWORK

In Figures 3.7a, 3.7b, and 3.7c, the details of the NRGDD framework are illustrated. The Figure 3.7a shows the transaction that initiates a dead lock algorithm, Figure 3.7b shows the transaction that sends and receives the probe message between them and Figure 3.7c shows the deadlock detection and resolution.

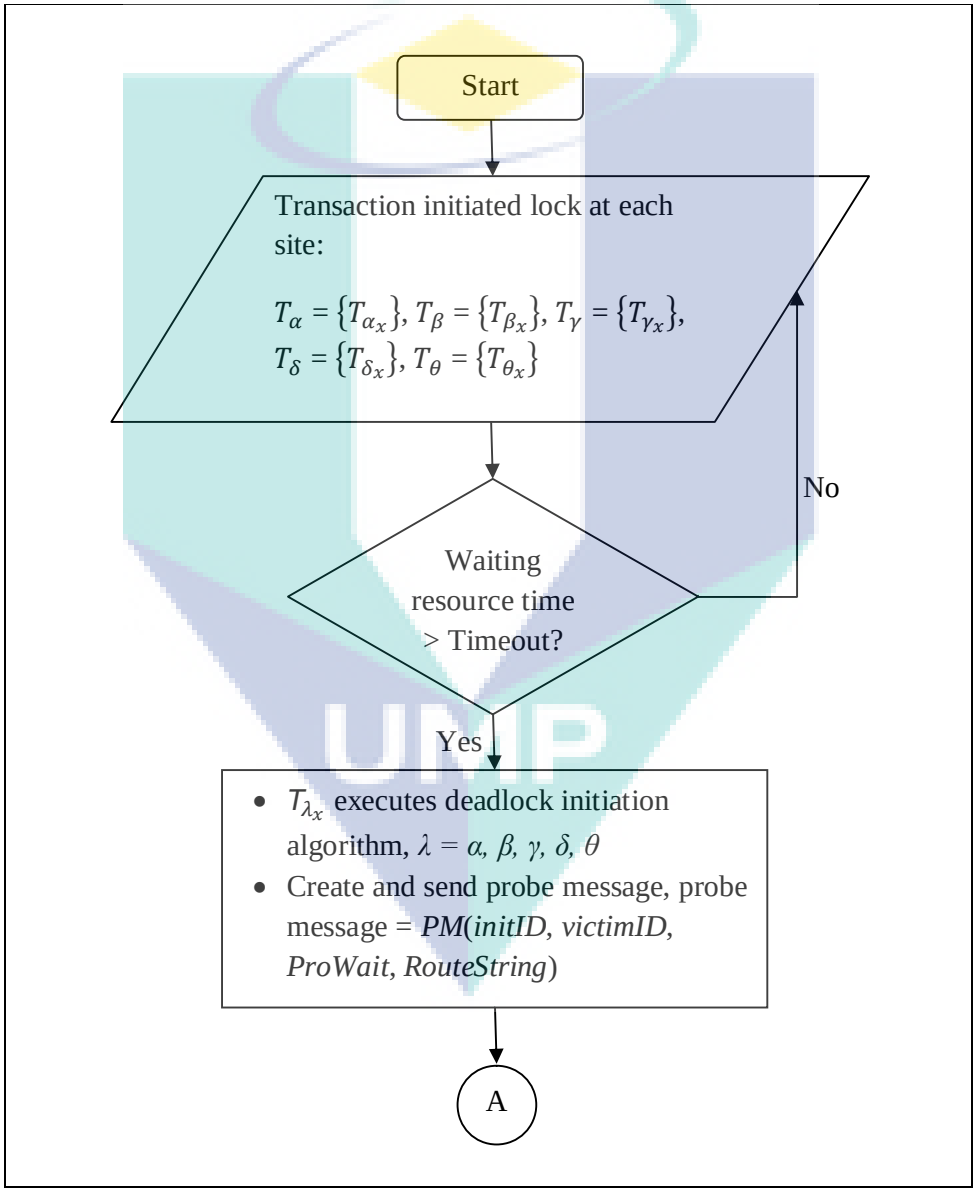


Figure 3.7a: Deadlock initiation

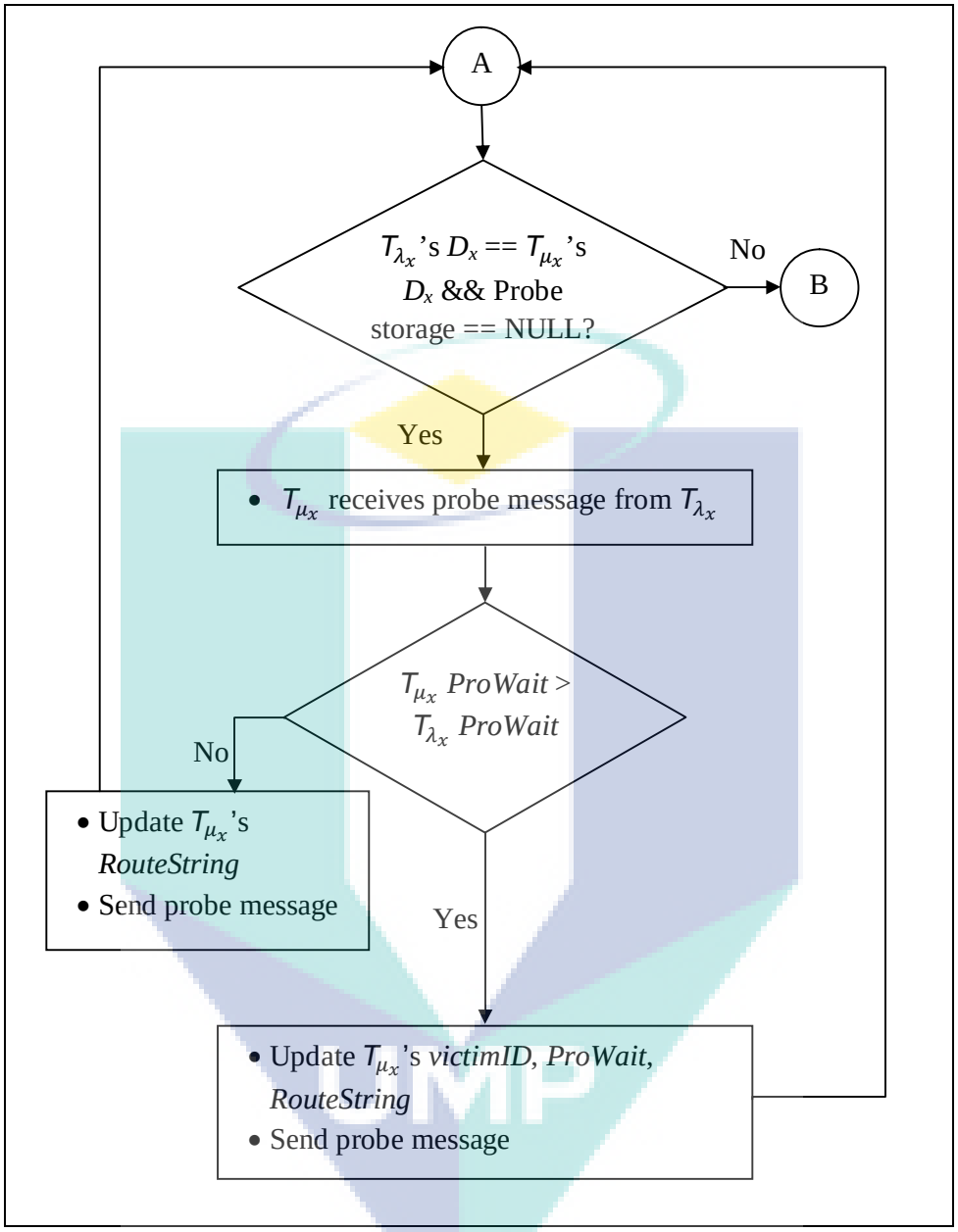


Figure 3.7b: Send and receive probe message

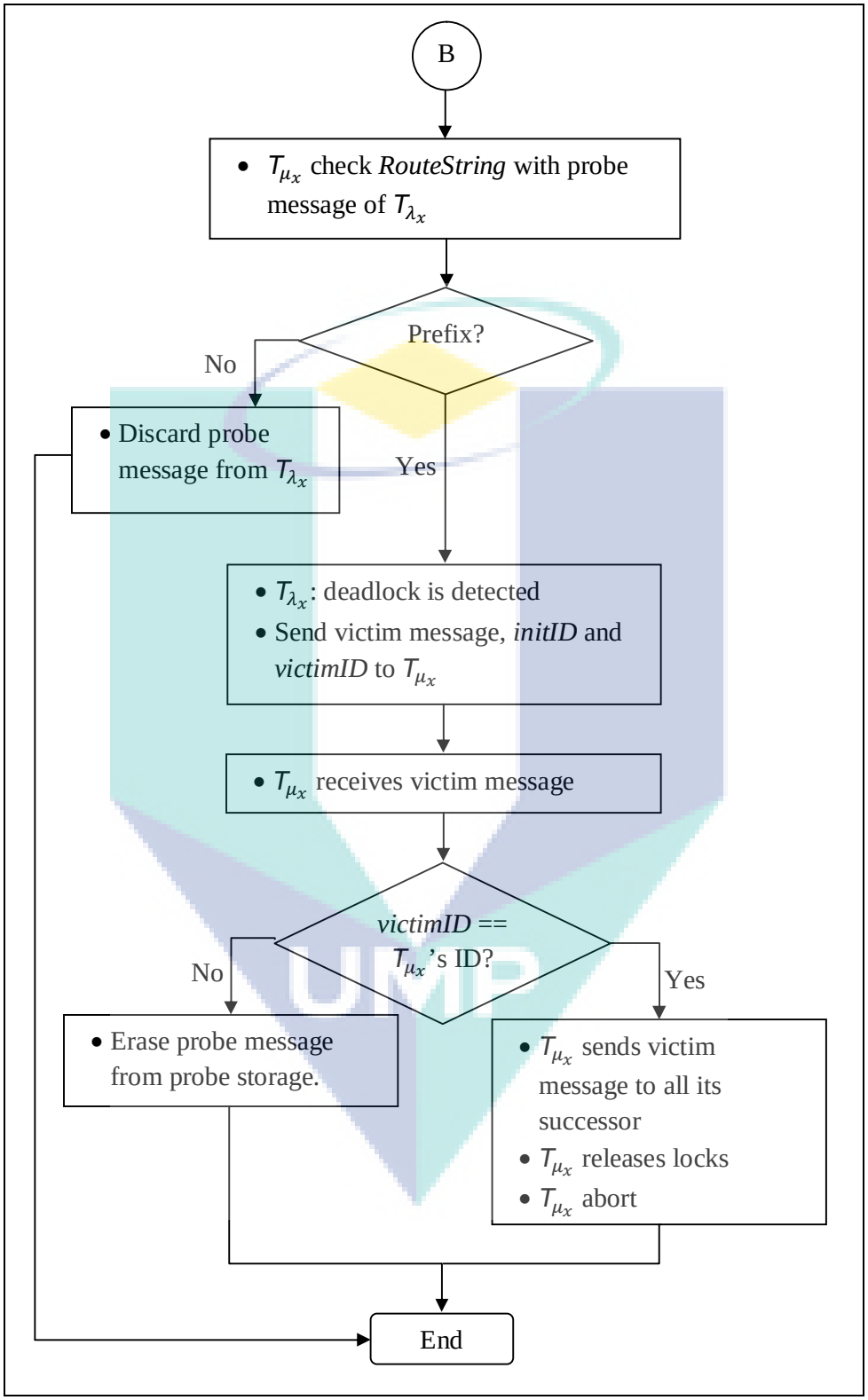


Figure 3.7c: Deadlock detection and resolution

3.6 NRGDD ALGORITHM

To execute the NRGDD framework, a new NRGDD algorithm was proposed by considering the data replication for a data object.

Listing 1: The algorithm of Neighbour Replication on Grid Deadlock Detection

```
1  Start
2  T_λx initiate deadlock
3     If Wt > To Then
4           Execute deadlock initiation
5           Create Probe message
6     End If
7  Create Probe
8     T_λx create probe message, PM(initID,victimID,ProWait,RouteString)
9     Send Probe message
10 Send Probe
11    T_λx,PM(initID,victimID,ProWait,RouteString) or
12    T_μx,PM(initID,victimID,ProWait,RouteString)
13 T_μx receive probe message from T_λx,PM
14    If T_λx's D_x == T_μx's D_x && ProbeS == NULL Then
15          If T_μx ProWait > T_λx,PM ProWait Then
16                T_μx update victimID, ProWait, RouteString
17                Send Probe message
18          Else
19                T_μx update RouteString
20             Send Probe message
21          End If
22    Else
23          T_μx,PM(initID,victimID,ProWait,RouteString) check its RouteString with
            RouteString of T_λx,PM(initID,victimID,ProWait,RouteString)
24          If T_λx's RouteString prefix with T_μx's RouteString Then
25                T_λx detect deadlock
26                Send Victim message to T_μx, waiting transaction where μ = α, β,
                  δ, γ, θ
27          Else
28                Discard probe message from T_λx,PM
29          End If
```

```
30      End If
31  T_μx receive victim message
32      If victimID == T_μx's ID Then
33          T_μx send victim message to its successors
34          T_μx release lock
35          T_μx abort lock
36      Else
37          Erase probe message from ProbeS
38      End If
39  End
```

3.7 NRGDD DEVELOPMENT

This section specifies the hardware and software components. Besides that this section also describes the programming implementation and an explicit explanation of the NRGDD Simulation Model.

3.7.1 Hardware And Software Components

The implementation of NRGDD requires some minimum hardware and software specifications. The hardware specifications as shown in Table 3.2 were used for implementation.

Table 3.2: Hardware component specifications

Hardware	Specifications
Processor	Intel (R) Core ((TM) 2 Duo CPU T6600 @2.20 GHz 2.20 GHz
Memory	3.00 Gigabyte
Hard Disk	300 Gigabyte

The implementation of the NRGDD was carried out by using C# programming language. Table 3.3 shows the system development tool specification

for this implementation. C# was selected because of the object oriented capabilities (Craig Utley, 2002). Even simple data types can be treated as objects means that a data type like *int* has methods associated with it. Besides that, C# attempts to simplify the syntax to be more consistent while also removing some of the more complex features of C++ (Craig Utley, 2002).

Table 3.3: System development tool specifications

System Development Software	Specifications
C#	Microsoft Visual Studio 2010 Express
Windows 7	Home Premium

3.7.2 Programming Implementation

The programming implementation is developed using C# language. Microsoft Visual C# 2010 Express is used to write the source code of deadlock detection. Besides that, the interface is designed by using Windows Form. Visual C# is developed and maintained by Microsoft Corporation. After installing Microsoft Visual C# 2010, it can be accessible to the start menu. Microsoft Visual C# 2010 has its own file format to maintain the source code. Text files are used as the data file. The screen shot and the usability of the experiment tools are shown and described below.

Figure 3.8 shows the project and different solutions view of NRGDD application. This snap shot is shown in Microsoft Visual C# 2010 Express. This window appears when the project is loaded on the Microsoft Visual C# 2010 Express. The Microsoft Visual C# 2010 Express provides various solutions such as “Properties," “References," and “Form”." The source code typically appears by right click on project name “NRGDD_sim” then choose new items to add a class and items for source code. The class file format for C# is “.cs”.

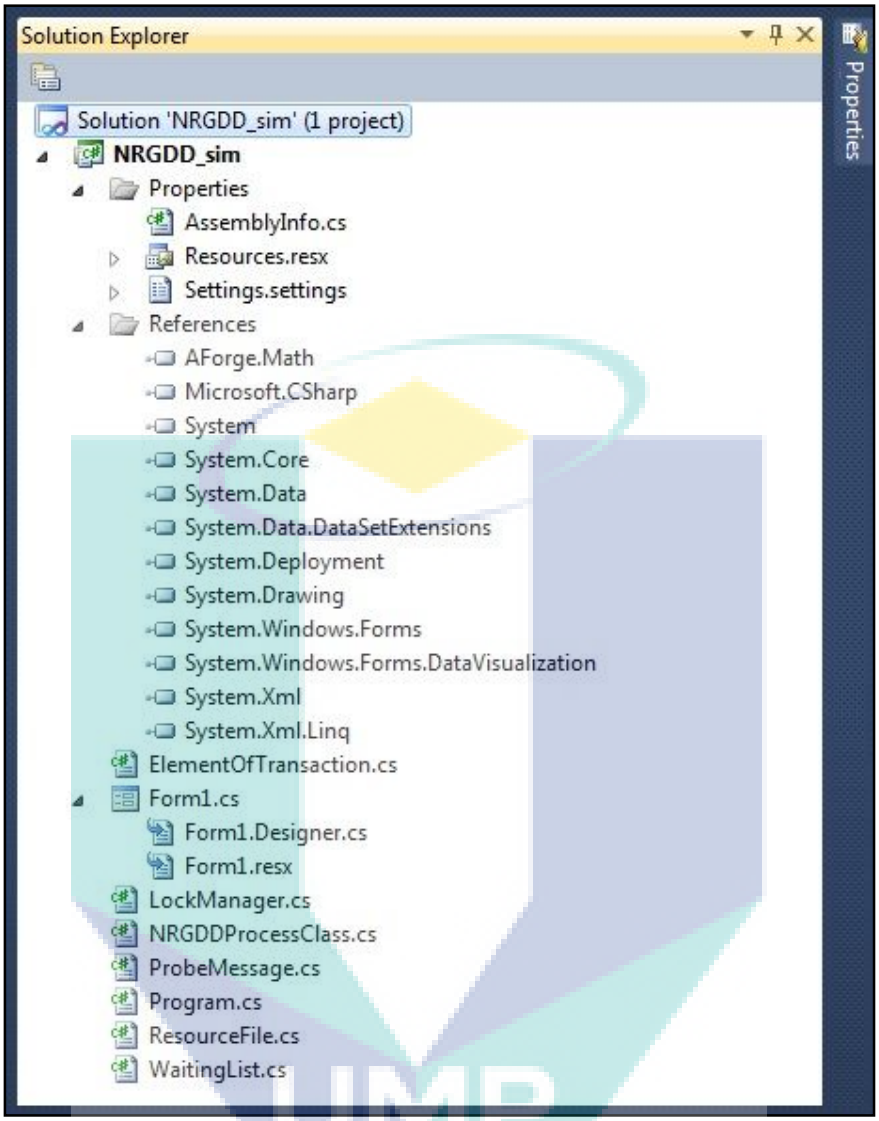


Figure 3.8: Project view of NRGDD model

The Figure 3.9 shows the Microsoft Visual C# 2010 Express toolbox for designing the interface. In designing the interface for the system, the user can drag the "Common Control," "Containers," and others to the Windows Form applications.

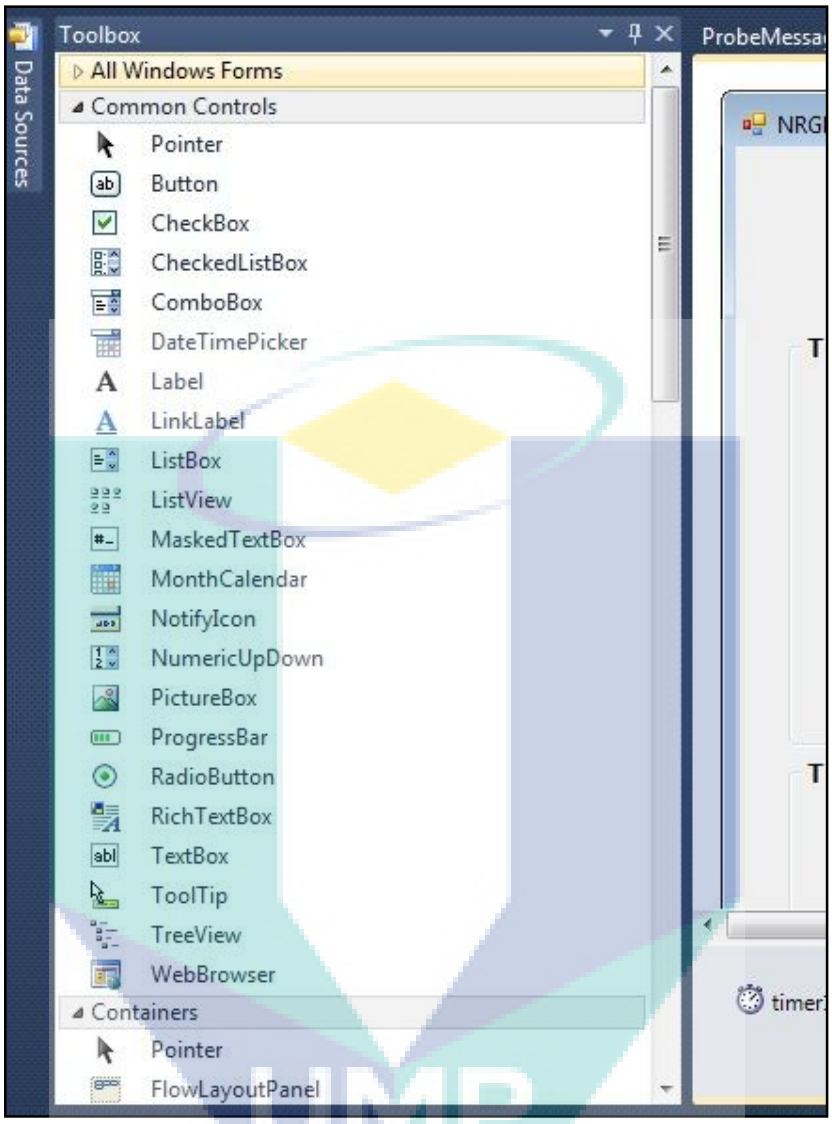


Figure 3.9: Toolbox used for designing interface

Figure 3.10 shows the source code of NRGDD model. The source code is written in Microsoft Visual C # 2010 Express, and the screen shot shows the part of the source code (probe message on NRGDD) which is written in C#. The probe message class detects the deadlock on NRG replication model and determine which one of the transactions is a victim that causes the deadlock to occur when program is running.

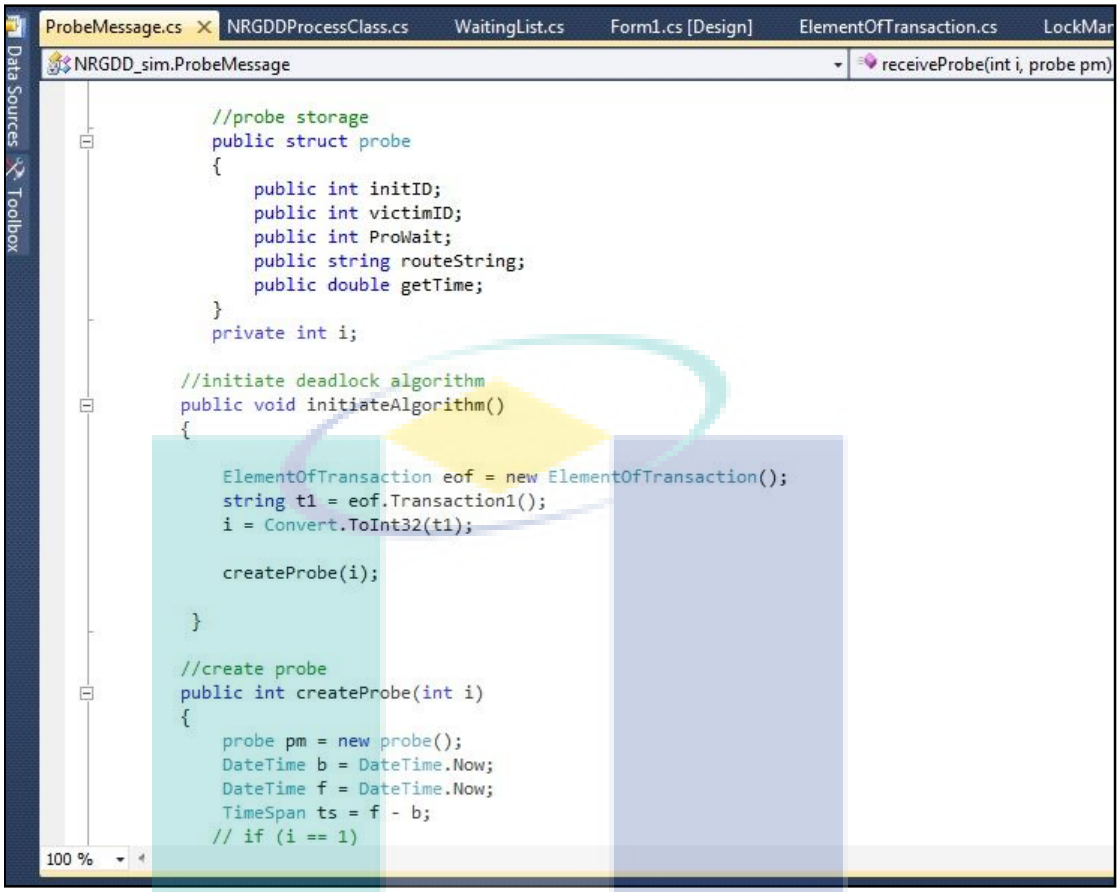


Figure 3.10: Source code view of NRGDD model

3.7.3 NRGDD Simulation Model

The NRGDD model is developed to detect the deadlock in NRG replication environment. For the experiment, the data file is used to represent the server. There are five servers used that contains the identical replicated data, data *x*. The NRGGD has been simulated in the NRG replication model.

Figure 3.11 below shows an interface for NRGDD model. There are different sets of transactions that are represented as Transaction 1 until Transaction 5. Each button A, B, C, D, and E represents the server A, B, C, D and E respectively. Every transaction will lock its own server such as Transaction 1 locks server A, Transaction 2 locks server B, Transaction 3 locks server C, Transaction 4 locks server D and Transaction 5 locks server E.

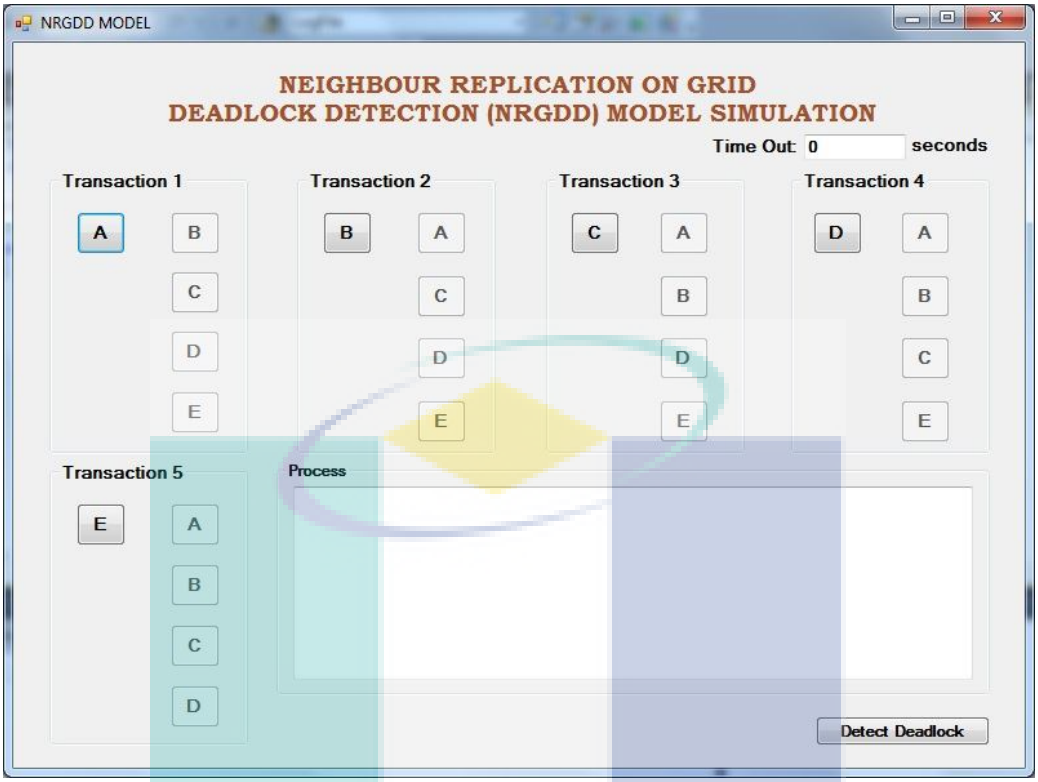


Figure 3.11: Interface for NRGDD simulation model

Figure 3.12 shows a set of transaction that locked its own server. After it has locked its own server, the transaction can request for other server. If the lock is free at that server, it can grant the request; otherwise it will wait until the lock is released. For the every process that occurs during simulation, it will be displayed on the text box called "Process". The time required to lock the server in milliseconds is also displayed in "Process".

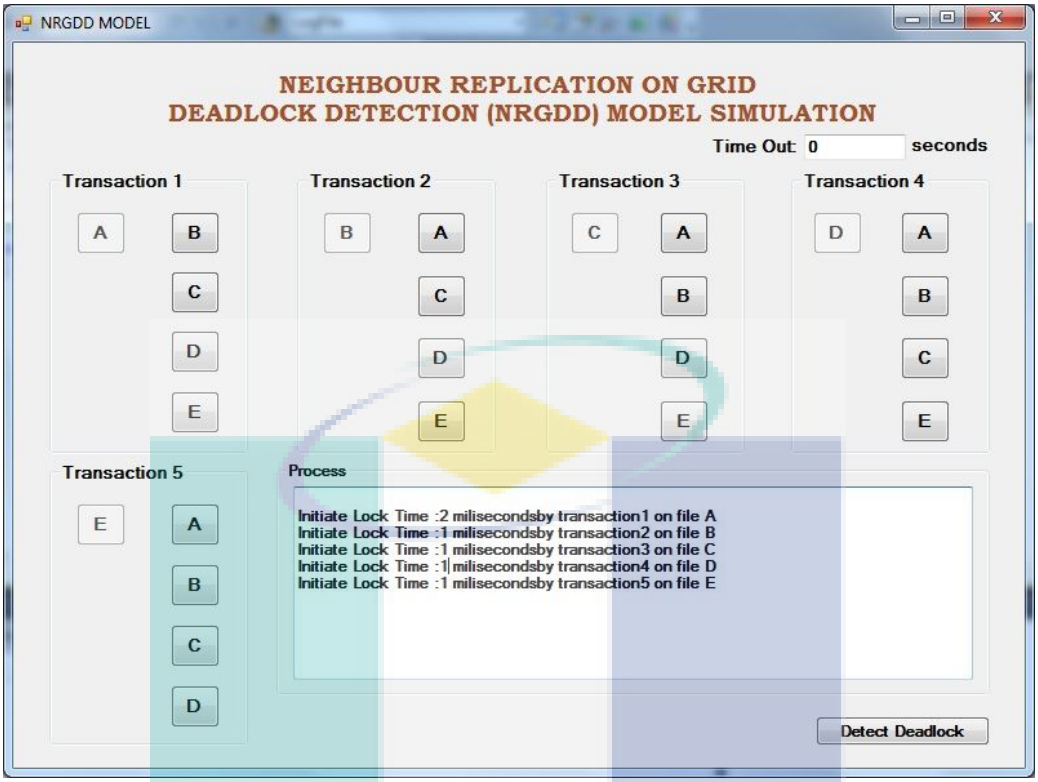


Figure 3.12: A different set of transactions locked its server

3.8 EXAMPLE

Two case studies are presented in this section to test the correctness of algorithm. The first case study detects only one existing deadlock cycle on NRG using four sites. While the second case study detects two cycles of existing deadlock on NRG using five sites. The uses of four and five sites for both case studies are because in the NRG replication technique, it only uses 3 until 5 sites to replicate the same database or data object.

A set of different transactions, T_α, T_β, T_γ, T_δ and T_θ can either come concurrently or otherwise. All the T_{λ_x} of $\lambda = \alpha, \beta, \gamma, \delta$ and θ get lock respectively. To illustrate this, let's say all elements T_{α_x} , T_{β_x} , T_{γ_x}, T_{δ_x} and T_{θ_x} come to modify data object x at site A, B, C, D and E respectively. T_{α_x} gets to lock data object x at site A, then T_{β_x} gets to lock data object x at site B, T_{γ_x} gets to lock object x at site C, T_{δ_x}

gets to lock object *x* at site D and T_{θ_x} gets to lock object *x* at site E. Figure 3.13 shows the different elements of T_{λ_x} get locked at the different sites.

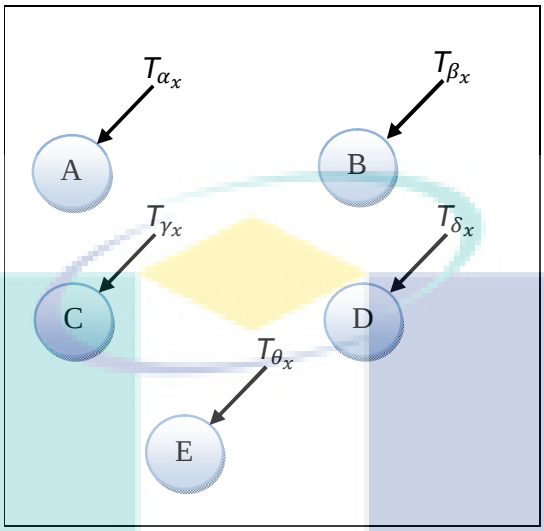


Figure 3.13: Different set of transaction, T_{λ_x} requests to update data object *x* at different sites, $i \in S(B_x)$

3.8.1 Case #1: detects only one existing deadlock cycle on NRG using four sites

T_α, T_β, T_γ, and T_δ can either come concurrently or otherwise. All the T_{λ_x} of $\lambda = \alpha, \beta, \gamma, \delta$ get locked respectively. To illustrate this, let's say all elements T_{α_x} , T_{β_x} , T_{γ_x} and T_{δ_x} come to modify data object *x* at site A, B, C, and D respectively. T_{α_x} gets to lock data object *x* at site A, then T_{β_x} gets to lock data object *x* at site B, T_{γ_x} gets to lock object *x* at site C whereas T_{δ_x} gets to lock object *x* at site D. The Figure 3.14 shows the different elements of T_{λ_x} get locked at different sites.

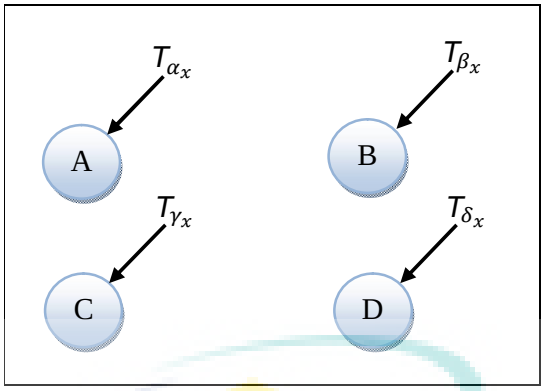


Figure 3.14: Different set of transaction, T_{λ_x} requests to update data object *x* at different sites, *i Є S*(B_x)

Then, after all the T_{λ_x} of $\lambda = \alpha, \beta, \gamma, \delta$ are locked, this element may request data object *x* at the other site that is held by other elements of T_{λ_x}. The Figure 3.15a shows the element of T_{λ_x} requests for other site that is currently held by other elements of T_{λ_x}.

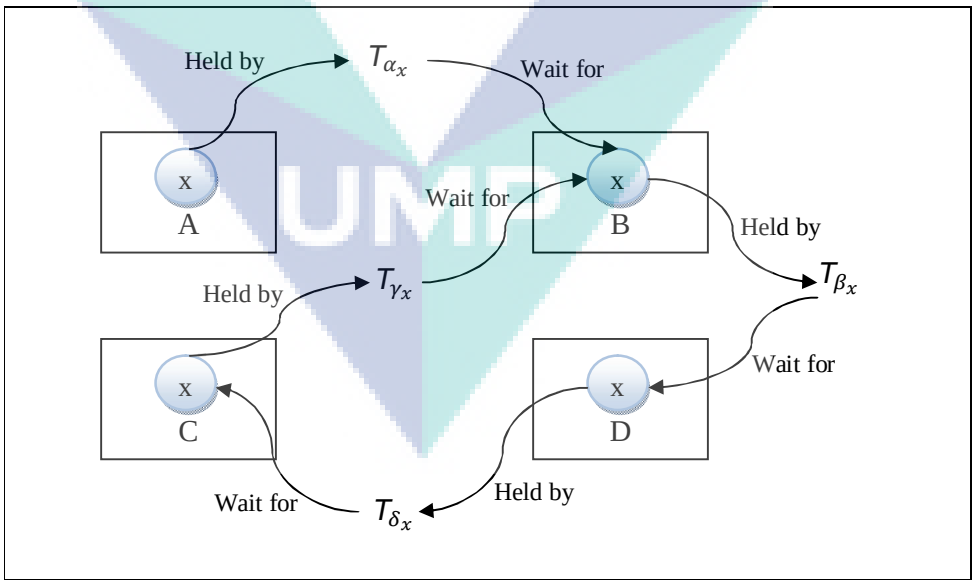


Figure 3.15a: Elements of T_{λ_x} request for other site that is held by other elements of T_{λ_x}

As any transaction can only wait for one object at a time, objects can be left out of 'wait-for graph' as in Figure 3.15b. It shows only one cycle of deadlock. The NRGDD algorithm will be used to detect the existing cycle of deadlock by sending a probe message.

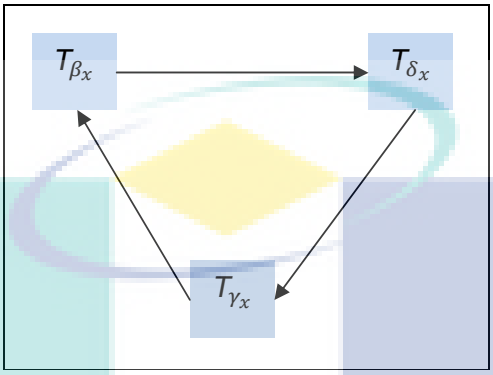


Figure 3.15b: Wait-for graph

3.8.2 Case #2: detects two-cycles of existing deadlock on NRG using five sites

Then, after all the T_{λ_x} of $\lambda = \alpha, \beta, \gamma, \delta, \theta$ are locked, this element may request data object *x* at the other site that is held by other elements of T_{λ_x}. The Figure 3.16a shows the element of T_{λ_x} request for other site that is currently held by other element of T_{λ_x}.

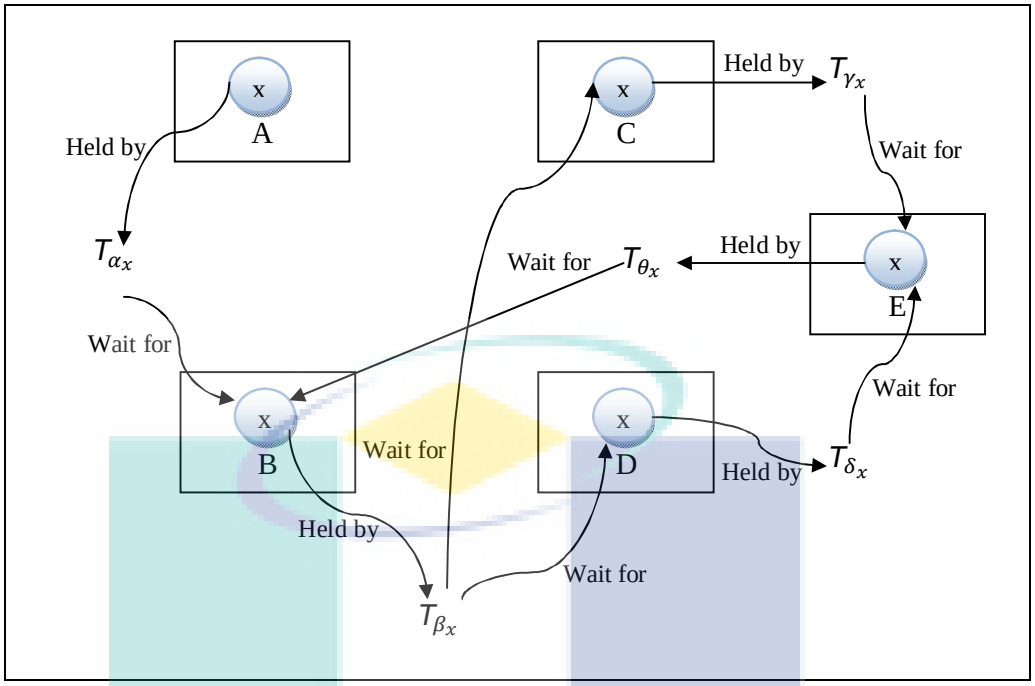


Figure 3.16a: Elements of T_{λ_x} request for other site that is held by other element of T_{λ_x}

As any transaction can only wait for one object at a time, objects can be left out of 'wait-for graph' in as in Figure 3.16b. The wait-for graph below shows the two cycles of deadlock exist on the NRG replication model. The NRGDD algorithm will detect the two cycles of deadlock and solve it by sending a probe message to communicate with other transaction, T_{λ_x}.

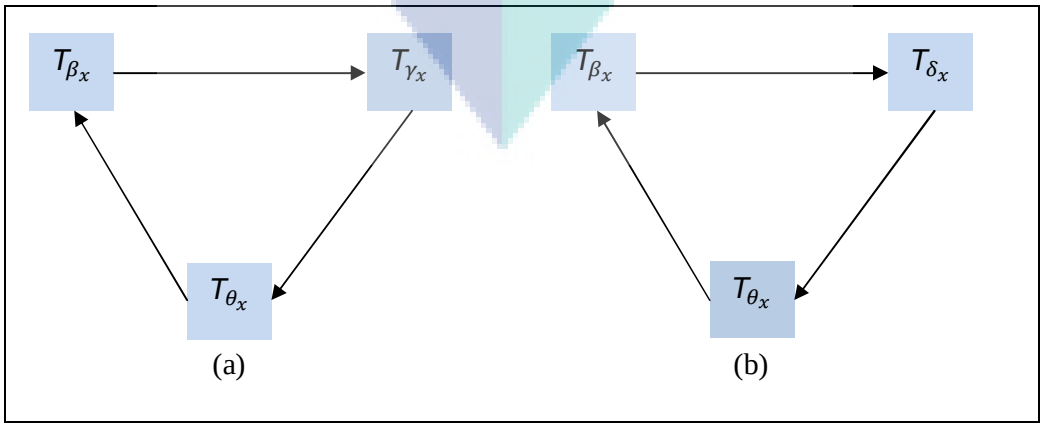


Figure 3.16b: Wait-for graph for two-cycles (a) and (b)

3.9 COMPARISON BETWEEN NRGDD AND MC2DR

There are two differences between NRGDD and MC2DR.The first difference is in the time used for both algorithms to detect whether two cycles of deadlock have occurred on sites. And the second is in the different number of transactions used to get the average time taken by both models to detect a deadlock.

3.9.1 Detect Two Cycles of Deadlock

The uses of two cycles of deadlock are because both the NRGDD and MC2DR are considered as detectors of multi cycles of deadlock in a distributed system. In MC2DR, it shows the steps to detect multi cycles of deadlock on sites that are meant for write operation. On the other hand, NRGDD detects multi cycles of deadlock through replication technique, NRG, which is also considered for write operation. The difference between both models is in the uses of logical data. The NRGDD uses data replication while MC2DR does not consider the data replication.

3.9.2 Average Deadlock Detection by Using Different Number of Transactions

The different number of transaction is used to get the average time of deadlock detection for both models, NRGDD and MC2DR. The 3, 4 and 5 number of transactions will be used. This is because in NRG when a different transaction requests to be locked at one site, only one transaction is able to be locked at one particular site. Therefore, the uses of 3, 4 and 5 transactions are after each of the transaction gets their lock at the sites. Then, the transactions will be waiting for each other to get the same resources.

3.10 CORRECTNESS

Assertion 1: If the transaction waits indefinitely for each other for their requests to be satisfied at an infinite time, the deadlock algorithm will be initiated by one of the transactions until the cycles are formed. Then the deadlock detection has been executed successfully.

Proof: The transaction, T_{λ_x} that is waiting for resources held by another transaction at an infinite time, whereby the time for waiting has increased and becoming longer than the time for requesting resources. Then, the algorithm will execute to initiate the deadlock detection algorithm. T_{λ_x} creates a probe message, *PM* where *PM(initID, victimID, ProWait, RouteString).*

The transaction, $T_{\lambda_x,PM(initID,victimID,ProWait,RouteString)}$ sends its probe message to transaction, T_{μ_x} its waiting resource that is held by this transaction. On receiving probe message, T_{μ_x} checks its probe storage whether it is empty or full. If the probe storage is empty, the T_{μ_x} compares its *ProWait* with T_{λ_x}'s *ProWait*. Since, T_{μ_x} has a *ProWait* greater than T_{λ_x}, and then T_{μ_x} will update its probe message by changing its *victimID*, *ProWait* and *RouteString*. The $T_{\mu_x,PM(initID,victimID,ProWait,RouteString)}$ sends its probe message to other transaction that is holding the resource that it is waiting for. The steps above are repeated until the probe message is sent to the transaction which its probe storage is not empty. When this situation happens, the $T_{\mu_x,PM(initID,victimID,ProWait,RouteString)}$ compares its *RouteString* with $T_{\lambda_x,PM(initID,victimID,ProWait,RouteString)}$'s *RouteString*. If the *RouteString* is prefix then $T_{\mu_x,PM(initID,victimID,ProWait,RouteString)}$ detects the cycle of deadlock. The $T_{\mu_x,PM}$ sends the victim message that contains *victimID* and *ProWait* to $T_{\lambda_x,PM}$. Then, $T_{\lambda_x,PM}$ checks whether its *victimID* is equal to *victimID* which it receives from $T_{\mu_x,PM}$. If the victimID is equal, then the $T_{\lambda_x,PM}$ has detected an occurrence of a victim of deadlock. Therefore, the deadlock has been detected successfully.

Assertion 2: If probe storage of transaction contains probe message, then only one transaction can detect a single deadlock cycle at one time.

Proof: In NRGDD, only one probe message, *PM(initID, victimID, ProWait, RouteString)* will be stored on probe storage for each transaction, T_{λ_x}. Then, the algorithm will execute at Line 23 for this situation. Therefore, there is no possibility for two or more of transactions, T_{λ_x} to detect a single deadlock cycle.

3.11 SUMMARY

In this chapter, a new technique to handle deadlock in replication data namely *Neighbour Replication on Grid Deadlock Detection* is discussed. By using this technique, the multi cycles of deadlock will be detected during the occurrence of transaction in NRG. Besides that, it can detect deadlock during write operation in order to ensure the consistency of replication data on each site. After a different set of transaction is successful to get locked at a particular site, they can request for other lock. If the lock is granted by other transaction, it will wait until they unlock that site. If every transaction is waiting for each other at an infinite time, a form of deadlock is built. The cycle exists in the form of Wait-for Graph (WFG). Then, one of the transactions will initiate the deadlock algorithm. A probe message is created to be sent to the successor(s). This step will continue until one of the transactions detects a transaction that causes deadlock to occur. After that, it will send the victim message to the transactions that become a victim of the deadlock. Finally, the victim will send a victim message to its successor(s) where it will abort and kill itself to release lock at the site that it has granted.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 INTRODUCTION

This chapter describes the experimental results for Neighbour Replication on Grid Deadlock Detection (NRGDD) in handling deadlock on NRG replication model. Besides, this chapter also proves that NRGDD can manage deadlock happened on replication data on grid. Finally, the results are compared with other deadlock detection model.

4.2 NRGDD EXPERIMENTAL RESULTS

4.2.1 Experiment 1

For the first experiment, the case that has been tested is by detecting one deadlock cycle on NRG in four sites of replicated data. There are four different sets of transactions, T_{α_x} , T_{β_x} , T_{γ_x} and T_{δ_x} which are requested to modify data file *x* at site A, B, C, and D respectively. Next, each transactions requests other server hold by other transaction to update data file *x*.

Figure 4.1 shows the processes which occur to detect deadlock in four sites on NRG replication technique. Transaction 1 which represents T_{α_x} locked server A. Then Transaction 2 or T_{β_x} locked server B, Transaction 3 or T_{γ_x} locked server C and transaction 4 or T_{δ_x} locked server D. When each transaction locked its own server, transaction 1 requests for server B that is held by transaction 2. Next, transaction 1 initiates a deadlock algorithm when the waiting time is higher than times out. Transaction 1 creates its probe message, (1, 1, 1, '1'), and send it to its waiting process or successor(s), transaction 2. Transaction 2 updates its probe message and stores it to probe storage as (1, 1, 1, and '12'). Transaction 2 also sends its probe to its waiting process, transaction 3. Transaction 3 will then update its probe storage to (1,1,1, '123') followed by, sending this probe to its waiting process, transaction 4. Next, transaction 4 will update its probe to (1,1,1, '1234') and send its probe to its waiting process, transaction 2. However, transaction 2 has already store probe on its storage. The route string in probe carried by transaction 4 will be compared with the route string in transaction 1's probe. If it is prefix, transaction 4 becomes a detector of deadlock and then send victim message to transaction 2. Besides, transaction 2 sends victim message to its waiting process before transaction 2 kills itself or abort to release lock and delete probe message from the probe storage. For this experiment, Table 4.3 simplifies the results of how NRGDD handles deadlock problems occur in transaction T_{α_x} , T_{β_x}, T_{γ_x} and T_{δ_x} at all sites by detecting only one cycle of deadlock.

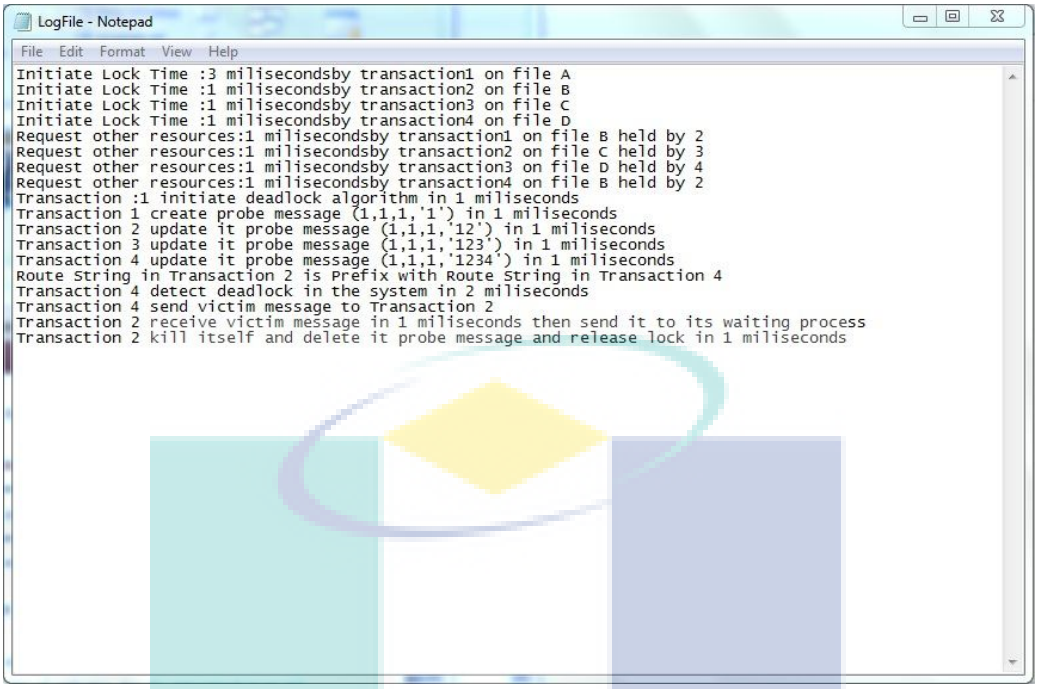


Figure 4.1: Process for detecting deadlock in four sites

From the result of Table 4.1, at time which is equal to 1(*t1*), the instant *x* at all servers are unlock. At *t2*, the transactions begin. At *t3*, there is transactionT_{β_x}, T_{γ_x} and T_{θ_x}in which locked site B, C and D respectively. At *t5*, T_{β_x} propagates lock to server C that is held by T_{γ_x}, and then T_{γ_x} propagates lock to T_{θ_x} while at the same time T_{θ_x} propagates lock to T_{β_x}. Each of the transaction is waiting for each other to grant the resource. Based on Figure 4.1, transaction 1, T_{α_x} initiates deadlock algorithm. After that, it creates a probe message and sends it to T_{β_x}(transaction 2). At *t6*, T_{β_x} updates its probe message to $T_{\beta_x,PM(1,1,1,'12')}$ and simultaneously it sends probe message to T_{γ_x}. At t7, T_{γ_x} receives probe message fromT_{β_x}, and updates its probe to $T_{\gamma_x,PM(1,1,1,'123')}$ before send it to T_{θ_x}. Subsequently, at *t7* T_{θ_x} updates its probe message into $T_{\theta_x,PM(1,1,1,'1234')}$ and sends probe message to its successor, T_{β_x}simultaneously. However, T_{β_x} is already received probe message from T_{α_x} and then T_{θ_x} compares its *RouteString* with T_{β_x}'s *RouteString* at *t9*. At the same time the route strings for both are prefixed and the deadlock is detected. T_{θ_x} will sends a victim message that contains *initID* and *victimID* to T_{β_x}. At *t10*, T_{β_x} receives a

victim message and then sends it to its successor, T_{γ_x}.T_{γ_x} receives a victim message at *t11*. Next, at *t11* the lock is released and probe message is deletes from its probe storage (*ProbeS*). In the interim, T_{γ_x} and T_{θ_x} also delete their probe message from their *ProbeS*. Finally, at *t12* T_{β_x} unlocks site B.

Table 4.1: The experiment results for detecting one cycle of deadlock

Replica Time	B	C	D
t1	unlock(x)	unlock(x)	unlock(x)
t2	begin_transaction	begin_transaction	begin_transaction
t3	write lock(x)	write lock(x)	write lock(x)
t4	wait	wait	Wait
t5	T_{β_x} Propagate lock: C	T_{γ_x}Propagate lock: D	T_{θ_x} Propagate lock: B
t6	Update probe message: $T_{\beta_x,PM(1,1,1,'12')}$. Send probe to T_{γ_x}	wait	Wait
t7		Receive probe: update probe $T_{\gamma_x,PM(1,1,1,'123')}$. Send to T_{θ_x}	
t8			Receive probe: update probe $T_{\theta_x,PM(1,1,1,'1234')}$. Send to T_{β_x}
t9			Detect deadlock: Route string prefix with $T_{\beta_x,PM(1,1,1,'12')}$'s route string. Send victim to T_{β_x}
t10	Receive victim message. Send to its waiting process, T_{γ_x}.		
t11		Receive victim message	

t12	Kill: released lock & delete probe message	Delete probe message	Delete probe message
t13	Unlock(x)		

Figure 4.2 shows transaction 1 which has initiated deadlock algorithm by sending probe message to transaction 2. Then, transaction 2 sends its probe message to transaction 3 and transaction 3 sends its probe message to transaction 4. Transaction 2 received probe message from transaction 4. As described, transaction 2 has already received probe message from transaction 1. Therefore, it compares its route string with route string of transaction 4.Transaction 4 has detected that transaction 2 is a victim of a deadlock that occur.

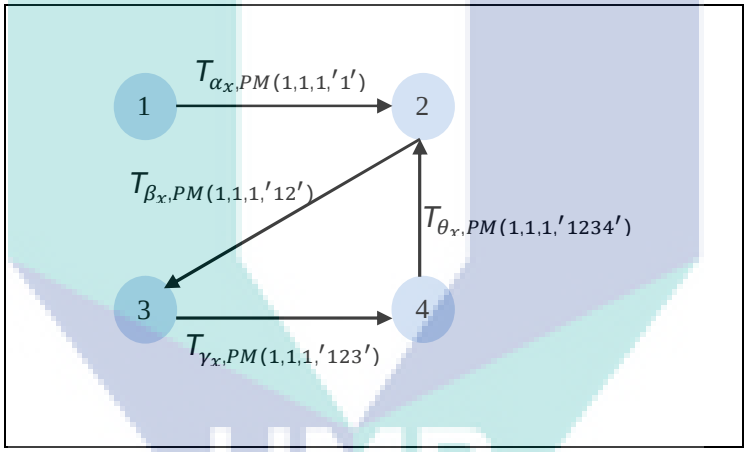


Figure 4.2: Sending of probe message

Figure 4.3 shows a single deadlock cycle which has been detected on NRG which is $T_{\beta_x,PM(1,1,1,'12')}$ waiting for $T_{\gamma_x,PM(1,1,1,'123')}$, for the moment it also waits for$T_{\theta_x,PM(1,1,1,'1234')}$. Furthermore, transaction $T_{\theta_x,PM(1,1,1,'1234')}$ is also waiting for$T_{\beta_x,PM(1,1,1,'12')}$.

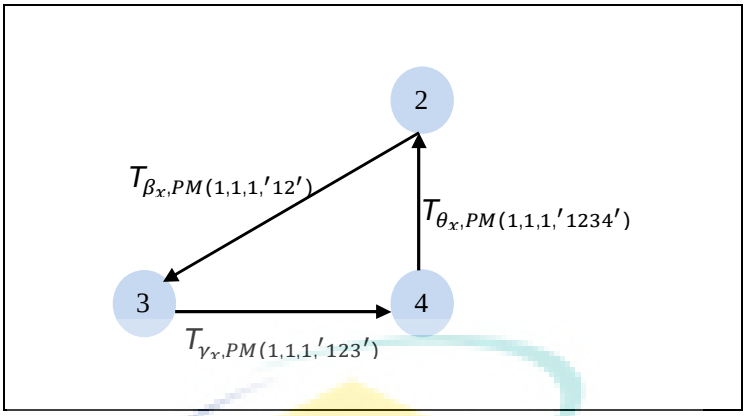


Figure 4.3: A single deadlock cycle

Table 4.2 shows the average time taken until the cycle is detected and resolved for one cycle of deadlock {2,3,4,2).

Table 4.2: Average time taken to detect deadlock cycle for cycle {2,3,4,2}

	Time(s)		
Lock	**B**	**C**	**D**
Initiate Lock	0.001	0.001	0.001
Propagate Lock	0.001	0.001	0.001
Detect Deadlock	0.001	0.001	0.002
Receive Victim Message	0.001	0.002	0.002
Release Lock	0.001	0.001	0.001

4.2.2 Experiment 2

For second experiment, the case that has been test is detecting two deadlock cycles on NRG in five sites of replicated data. There are five different sets of transactions, T_α, T_β, T_γ, T_δ and T_θ which request to modify data file *x* at site A, B, C, D and E respectively. To facilate that, each of the transactions requests other server hold by other transaction to update data file *x*.

Figure 4.4 shows the process to detect two cycles of deadlock existed in NRG for five sites. Transaction 1 which represents T_{α_x} locked server A. Subsequently, transaction 2 or T_{β_x} locked server B while transaction 3 or T_{γ_x} locked server C, transaction 4 or T_{δ_x} locked server D and transaction 5 or T_{θ_x} locked server E. All of these transactions request other servers that are held by other transaction as stated in figure above. When the transaction is waiting for another transaction to release lock at infinite time, transaction 1 initiate a deadlock algorithm and then creates a probe message, (1, 1, 1, '1'). It is then send to the waiting process or successor, transaction 2. Once a probe is received, transaction 2 updates the probe message and stores to its probe storage as (1, 2, 2, '12'). This transaction changes its *victimID* and *proWait* because it is waiting to more than one process. Therefore, it changes *victimID* to its transaction ID equal to 2 and *proWait* which is depend on the sum of waiting process it is waiting for. Transaction 2 also sends its probe to its waiting process, transaction 3 and transaction 4. Then, transaction 3 updates its probe storage to (1,2,2, '123'). And transaction 4 updates its probe to (1,2,2, '124'). Next, transaction 3 sends its probe to its waiting process, transaction 5. Then, transaction 5 updates its probe to (1,2,2, '1235').

Figure 4.4: Detect two cycles of deadlock in NRG for five sites

Transaction 4 also sends its probe to transaction 5. Figure 4.5 shows the way of probe message that is sent among transactions. However, when transaction 5 is already received probe from transaction 3, consequently probe message from transaction 4 will be discarded. Route string in probe carried by transaction 5 will be compared to route string in transaction 2's probe. If it is prefix, then transaction 5 becomes a detector of deadlock and then sends victim message to transaction 2. In addition, transaction 2 sends victim message to its waiting process before it kills itself to release lock and delete probe message from the probe storage.

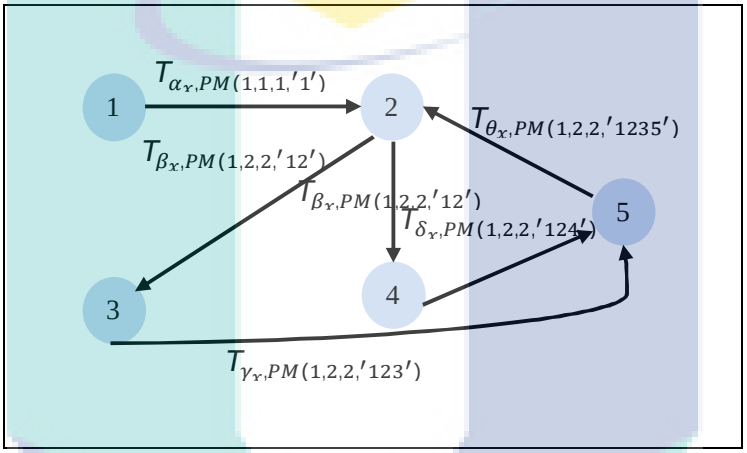


Figure 4.5: Sending of probe message by five transactions

Figure 4.6 shows two cycles of deadlock which are detected on NRG.

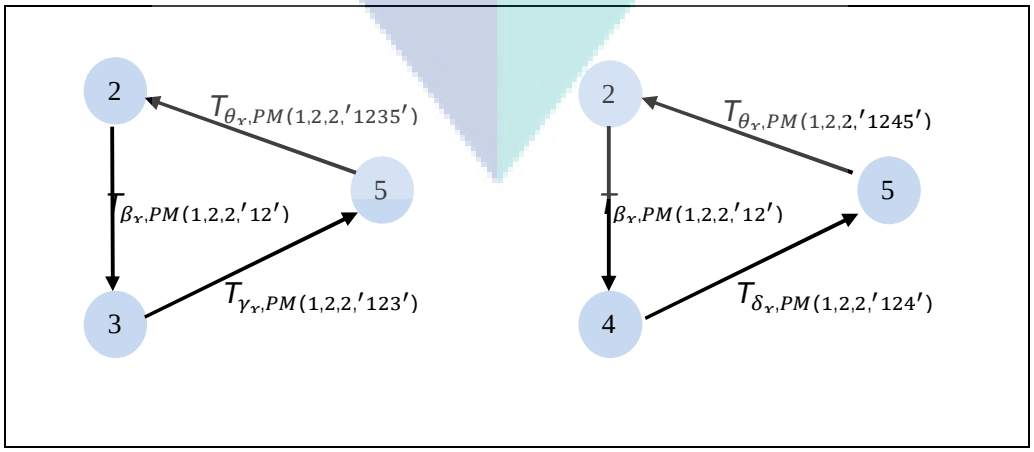


Figure 4.6: Two cycles of deadlock

From the result from Table 4.3, at time equal to 1(*t1*), instant *x* at all servers are unlock. At *t2*, the transactions begin. At *t3*, there is transactionT_{β_x}, T_{γ_x}, T_{δ_x} and T_{θ_x}which locked site B, C, D and E respectively. At *t5*, T_{β_x} propagates lock to server C and D that are held by T_{γ_x} and T_{δ_x} and T_{γ_x} propagates lock to server E that is held by T_{θ_x}. At the same time, T_{δ_x} also propagates lock to server E that is held by T_{θ_x}whereas T_{θ_x} propagates lock to server B that is held byT_{β_x}. Each of the transaction is waiting for each other to grant the resource. Based on Figure 4.5, transaction 1, T_{α_x} initiates the deadlock algorithm. After that, it creates a probe message and sends it to T_{β_x}(transaction 2). At *t6*, T_{β_x} updates its probe message to $T_{\beta_x,PM(1,2,2,'12')}$ and sends probe message to T_{γ_x} and T_{δ_x} simultaneously. At *t7*, T_{γ_x} receives probe message from $T_{\beta_x}$and then updates its probe to $T_{\gamma_x,PM(1,2,2,'123')}$ and then send it to T_{θ_x}. At *t8*, T_{δ_x} receives probe message from T_{β_x}, and updates its probe to $T_{\delta_x,PM(1,2,2,'124')}$ before sends it to T_{θ_x}. At *t9*, T_{θ_x} receives probe message from T_{γ_x}, then updates its probe to $T_{\theta_x,PM(1,2,2,'1235')}$. This is followed by, T_{θ_x} which will send probe message to T_{β_x}. At *t10*, T_{θ_x} receives probe message from $T_{\delta_x,PM(1,2,2,'124')}$. However, this probe will be discarded because T_{θ_x} already have probe message in its *ProbeS*. At *t11*, the deadlock is detected because *RouteString* of $T_{\theta_x}$is prefix with *RouteString* of T_{β_x}. Then, T_{θ_x} sends victim message to T_{β_x}. At *t12*, T_{β_x} receives victim message from T_{θ_x} and then sends it to its waiting process, T_{γ_x} and T_{δ_x}. At *t13*, both of the transactions receive victim message from T_{β_x}. Then, at *t14* T_{β_x} releases its lock and delete probe message from its *ProbeS*. Meanwhile, T_{γ_x} and T_{δ_x} also delete its probe message from their *ProbeS*. Finally, at *t15*, T_{β_x} unlocks server B.

Table 4.3: The experiment results to detect two cycle of deadlock in five sites

Replica Time	**B**	**C**	**D**	**E**
t1	unlock(x)	unlock(x)	unlock(x)	unlock(x)
t2	begin_transaction	begin_transaction	begin_transaction	begin_transaction
t3	write lock(x)	write lock(x)	write lock(x)	write lock(x)
t4	wait	wait	wait	Wait
t5	T_{β_x} Propagate lock: C, D	T_{γ_x} Propagate lock: E	T_{δ_x} Propagate lock: E	T_{θ_x} Propagate lock: B
t6	Update probe message: $T_{\beta_x,PM(1,2,2,'12')}$. Send probe to $T_{\gamma_x}, T_{\delta_x}$	wait	wait	Wait
t7		Receive probe: update probe $T_{\gamma_x,PM(1,2,2,'123')}$. Send to T_{θ_x}		
t8			Receive probe: update probe $T_{\delta_x,PM(1,2,2,'124')}$. Send to T_{θ_x}	
t9				Receive probe: update probe $T_{\theta_x,PM(1,2,2,'1235')}$ Send to T_{β_x}
t10				Receive probe: Discard probe from $T_{\delta_x,PM(1,2,2,'124')}$
t11				Detect deadlock: Route string

				prefix with $T_{\beta_x, PM(1,2,2,'12')}$'s route string. Send victim to T_{β_x}
t12	Receive victim message from T_{θ_x}. Send to its waiting process, T_{γ_x}, T_{δ_x}.			
t13		Receive victim message	Receive victim message	
t14	Kill: released lock & delete probe message	Delete probe message	Delete probe message	Delete probe message
t15	Unlock(x)			

For this experiment, Table 4.4 and Table 4.5 show the average time required for the NRGDD algorithm to detect two cycles of deadlock that existed on NRG.

Table 4.4: Average time taken to detect first deadlock cycle for cycle {2,3,5,2}

Lock	Time(s)		
	B	**C**	**E**
Initiate Lock	0.002	0.001	0.001
Propagate Lock	0.001	0.002	0.002
Detect Deadlock	0.002	0.002	0.002
Receive Victim Message	0.002	0.002	0.002
Release Lock	0.001	0.001	0.001

Table 4.5: Average time taken to detect second deadlock cycle for cycle {2,4,5,2}

Lock	Time(s)		
	B	**D**	**E**
Initiate Lock	0.002	0.001	0.001
Propagate Lock	0.001	0.001	0.001
Detect Deadlock	0.001	0.001	0.002
Receive Victim Message	0.002	0.002	0.002
Release Lock	0.001	0.001	0.001

4.3 COMPARISON BETWEEN NRGDD AND MC2DR

The proposed Neighbour Replication on Grid Deadlock Detection (NRGDD) in replication technique, Neighbour Replication on Grid (NRG) has been compared with deadlock detection algorithm, Multi-cycle Deadlock Detection and Recovery (MC2DR) algorithm (Razzaque *et al.*, 2007) in terms of executing time taken to detect two cycles of deadlock and average time of deadlock detection using different number of transaction.

The NRGDD only compared to MC2DR not to other algorithm because MC2DR simulates their algorithm during write operation happened on sites. Besides , MC2DR is suitable to be contrasted with NRGDD because NRGDD is also done during write operation. The different between NRGDD with MC2DR is NRGDD is implemented on replication data on grid. Furthermore, in other algorithm such as Decentralized Algorithm for Detection Generalized Deadlock in Distributed Systems (Selvaraj *et al.*, 2011) and Deadlock Detection Views of Distributed Database (Alom *et al.*, 2009) elucidate their algorithm in general situations (explained in Chapter 2).

4.3.1 Detect Two-Cycles of Deadlock

The experiment has been done to compare time required for detecting two-cycles of deadlock detection between NRGDD and MC2DR. Table 4.6 shows the results of the time engaged for NRGDD and MC2DR to detect two-cycles of deadlock.

Table 4.6: Required time to detect two-cycles of deadlock in NRGDD and MC2DR

Deadlock Detection Model	Detect two-cycles of Deadlock (seconds)
NRGDD (proposed)	0.005
MC2DR	0.28

Figure 4.7 shows that NRGDD occupied 0.005 seconds to detect two-cycle of deadlock compared to MC2DR that occupied 0.28 seconds to detect two-cycle of deadlock. From Table 4.5, it was found that the NRGDD took less time which is about 27.5% less than MC2DR in detecting two cycles of deadlock.

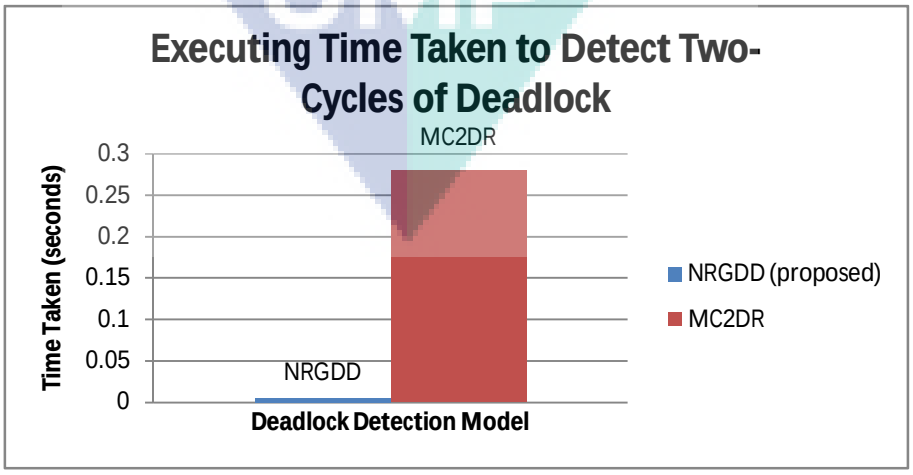


Figure 4.7: Time (in seconds) taken to detect two-cycles of deadlock

4.3.2 Average Deadlock Detection by Using Different Number of Transactions

The comparison is also made by using three to five number of transactions. This is because in NRG when a different transaction requests to get lock at one sites, only one transaction can get lock at one site. Therefore, the uses of 3, 4 and 5 transaction are after each of the transaction get their lock at the sites. Then, the transactions are waiting for each other to get the same resources.

Table 4.6 and Figure 4.8 show the results of the duration for both models, NRGDD and MC2DR spend to detect deadlock with different number of transactions. It is concluded that NRGDD provides less time average of deadlock detection rather than MC2DR for every number of transactions. The NRGDD performs 0.002 seconds besides MC2DR performs 0.22 seconds to detect deadlock for 3 transactions. By using 4 transactions, NRGDD still applies less time to detect deadlock about 0.004 rather than MC2DR which is 0.26 to detect deadlock. Furthermore, even for 5 transactions the NRGDD still need less time about 0.005 than MC2DR, 0.28 to detect deadlock.

Table 4.7: Average deadlock detection by using different numbers of transactions

Transactions No.	Average Deadlock Detection (Time)	
	NRGDD	MC2DR
3	0.002	0.22
4	0.004	0.26
5	0.005	0.28

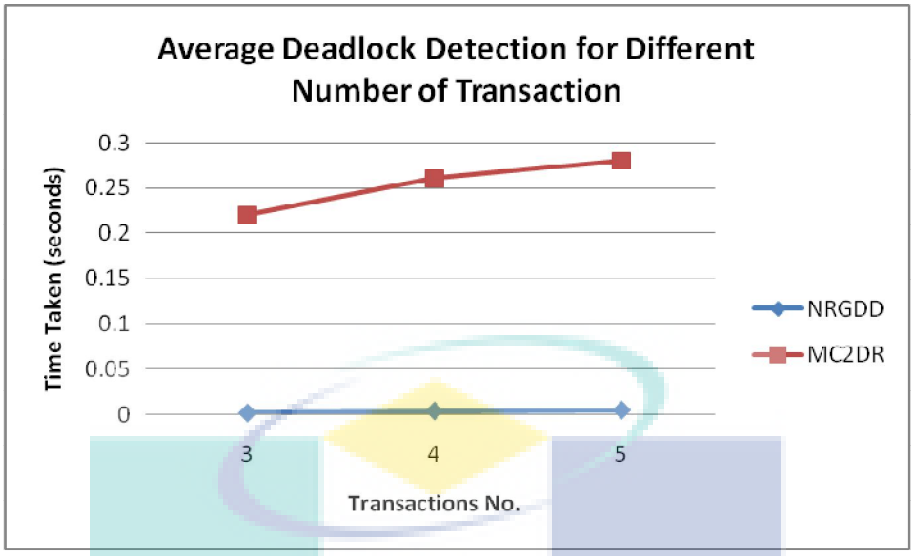


Figure 4.8: Average deadlock detection for different numbers of transactions

4.4 SUMMARY

This chapter presents the detailed process of how this model detects the cycle of deadlock in NRG as what has been described. From the result section, it is clearly shown that handling deadlock in replication data through proposed NRGDD is able to detect the present of deadlock and at the same maximize the availability of resource. The NRGDD spends less time to detect cycles of deadlock in NRG replication model than MC2DR. Besides, even though by using different number of transactions, NRGDD still employs less time to detect deadlock in NRG compared to MC2DR.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 INTRODUCTION

This research has been addressed using Neighbour Replication on Grid Deadlock Detection (NRGDD) model to handle deadlock problems during the execution of transaction on replication technique, in NRG environment. This chapter summarizes the important findings from the work carried out this research. It also includes some suggestions for future work in each of the areas covered during this research.

5.2 RESEARCH OBJECTIVES ACHIEVEMENT

In this research, the technique which is based on the previous work by other researchers has been discussed in Chapter 2. In particular, a new algorithm called Neighbour Replication on Grid Deadlock Detection (NRGDD) is proposed in order to manage deadlock problem in Neighbour Replication on Grid environment (as inChapter 3). Only replication that has the same data copy is considered for deadlock detection especially on write operation. Furthermore, NRGDD only detects the presents of deadlock.

This algorithm has been expanding from the existing MC2DR model. Compared to the existing model which manage on sites without consider any replication technique; this algorithm is used to handle deadlock on NRG replication technique. Hence, the capability of the new algorithm to handle deadlock is better than the MC2DR. This is because in handling deadlock on NRG its only replicated the data to 3, 4 and 5 sites even on NRG used more than 16 sites. Therefore, NRGDD algorithm can detect deadlock faster than MC2DR. This algorithm has been tested through the NRGDD simulation model (Chapter 3).

Experiments have been conducted in order to prove this technique to be able to handle deadlock problem and to increase the throughput by maximizing the available resources through resolution as in Chapter 4. Next, an analysis of NRGDD techniques is presented in terms of executing time taken to detect deadlock in NRG environment. After comparing NRGDD with MC2DR, it proves that NRGDD requires short time to detect existing deadlock on NRG.

Thus, this research is it contributes to a new framework and algorithm, Neighbour Replication on Grid Deadlock Detection (NRGDD) which was successfully developed to manage deadlock problems during transaction execution through Neighbour Replication on Grid (NRG) model (Chapter 3).

The proposed algorithm, NRGDD is compared with Multi-cycle Deadlock Detection and Recovery Algorithm for Distributed System (MC2DR) (Razzaque *et al.*, 2007) in terms of executing time taken to detect two-cycles of deadlock and average time taken for deadlock detection using different number of transactions which are discussed earlier (Chapter 4).

Besides, two experiments have been conducted in order to ensure the correctness of NRGDD algorithm. For the first experiment, the deadlock is detected by using four sites to detect one cycle of deadlock. For the second experiment, five sites are used to detect two-cycles of deadlock on NRG. The results for both experiments are successfully reported in Chapter 4. The NRGDD requires the

shortest time taken than MC2DR to detect deadlock. The NRGDD achieved 27.5% improvement from MC2DR.

5.3 FUTURE WORK

NRGDD can be improved in many different ways. Currently, NRGDD is simulated by the use of NRGDD simulator. In the future, NRGDD can be implemented on real time in distributed database systems by using Local Area Network (LAN) or Wireless Area Network (WAN).

As we know, breakdown can occur at anytime during transactions. Currently, NRGDD does not support handling deadlock by considering failure cases. In the future, NRGDD will take this challenge to handle deadlock in failure cases and fault tolerance in distributed database system in real time environment.

In future this study is able to create a significant improvement for commercial usage. NRGDD can enlarge in Cloud Computing in order to avoid the minimizing of available resource when deadlock occur during transaction in the system.

REFERENCES

Abd El-Gwad, A.O., Saleh, A.I., and Abd-ElRazik, M.M. 2009.A Novel Scheduling Strategy for an Efficient Deadlock Detection. *Proceedings of International Conference on Computer Engineering & Systems*.ICCES 2009, pp.579-583.

Abdi, S., and Mohamadi, S. 2010. Two Level Job Scheduling and Data Replication in Data Grid. *Proceedings of International Journal of Grid Computing & Applications (IJGCA)*, 1 (1): 23 – 27.

Ainul, A.C. F., Noraziah, A., Noriyani, M.Z., Beg, A.H., Nawsher, K., and Elrasheed I.S. 2011. Handling Fragmented Database Replication through Binary Vote Assignment Grid Quorum. *Journal of Computer Science*, 7 (9): 1338 - 1342.

Alkhatib, G., and Labban, R.S. 1995. Transaction Management in Distributed Database Systems: The Case of Oracle's Two-Phase Commit. *Journal of Information Systems Education*, 13 (2): 95-103.

Allcock, W., Bresnahan, J., Bunnb, J., Hegded, S., Insley, J., Kettimuthu, R., Newmanb, H., Ravotb, S., Rimovsky, T., Steenberg, C., and Winkler, L. 2003. Grid-Enabled Particle Physics Event Analysis: Experiences Using a 10 Gb, High-Latency Network for a High-Energy Physics Application. *Future Generation Computer Systems*, pp. 983-997.

AL-Mistarihi, H.H.E., and Yong, C.H. 2009. On Fairness, Optimizing Replica Selection in Data Grids. *IEEE Transactions on Parallel and Distributed Systems*, 20 (8): 1102-1111.

Alom, B.M.M., Henskens, F., and Hannaford, M. 2010. Optimization of Detected Deadlock Views of Distributed Database. *Proceedings of International Conference on Data Storage and Data Engineering*, pp. 44-48.

Alom, B.M.M., Henskens, F.A., and Hannaford, M.R. 2009. Deadlock Detection Views of Distributed Database. *Proceedings of Sixth International Conference on Information Technology: New Generations*, 2009. ITNG'09, pp. 730-737.

Amjad, T., Sher, M., and Daud, A. 2012. A Survey of Dynamic Replication Strategies for Improving Data Availability in Data Grids. *Future Generation Computer Systems*, 28 (2): 337 – 349.

Atreya, R., Mittal, N., Kshemkalyani, A.D., Garg, V.K., and Singhal, M. 2007. Efficient Detection of a Locally Stable Predicate in a Distributed System. *Proceedings of Journal Parallel Distributed. Computing*, 67 (4):369-385.

Bagchi, S. 2011. A Distributed Algorithm for Ordered, Atomic and Simultaneous Group Communication. *Future Generation Computer Systems*, 27 (5) 466-475.

Bai, T., Liu, Y.S., and Hu, Y. 2008. Timestamp Vector Based Optimistic Concurrency Control Protocol for Real-Time Databases. *Proceedings of 4th International Conference on Wireless Communications, Networking and Mobile Computing*.WiCOM '08, pp.1-4.

Barney, H.T., Low, G. C. 2008. Object Allocation with Replication in Distributed Systems. *Proceedings of International Journal of Information Technology*, 4 (1): 28-36.

Beg, A.H., Ahmad, N., AbdAlla, A.N., and Khan, N. 2010. Framework of Persistence Layer for Synchronous Data Replication (PSR). *Proceedings of Australian Journal of Basic and Applied Sciences*, 4(10), pp. 5394-5400.

Bhushan, S., Patel, R.B., and Dave, M. 2007. A Secure Time-Stamp Based Concurrency Control Protocol for Distributed Databases. *Journal of Computer Science*, 3 (7): 561-565.

Ben Charrada, F., Ounelli, H. and Chettaoui, H. 2010b. An Efficient Replication Strategy for Dynamic Data Grids. *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp.50-54.

Ben Charrada, F., Ounelli, H., and Chettaoui, H. 2010a. Dynamic Period vs Static Period in Data Grid Replication. *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp.565-568.

Bhushan, S., Patel, R.B., and Dave, M. 2007. A Secure Time-Stamp Based Concurrency Control Protocol for Distributed Databases. *Journal of Computer Science* 3 (7): 561-565.

Bost, Charron, B., Pedone, F. and Schiper, A. 2009.Replication Theory and Practice. Berlin Heidelberg NewYork, Springer, ISBN-10 3-642-11293-5 Springer, ch. 2.

Cheung, E., Chen, X., Hsieh, H., Davare, A., Sangiovanni-Vincentelli, A., and Watanabe, Y. 2009. Runtime Deadlock Analysis for System Level Design. *Proceedings of Design Automation for Embedded Systems*, 13 (4): 287-310.

Clauss, P.N., and Gustedt, J. 2010. Iterative Computations with Ordered Read–Write Locks. *Journal of Parallel and Distributed Computing*, 70 (5): 496-504.

Craig Utley. 2002. Why You Should Move to C#. http://www.techrepublic.com/article/why-you-should-move-to-c/1050356 (2 July 2012).

Du, Z., Hu, J., Chen, Y., Cheng, Z., and Wang, X. 2011. Optimized QoS-Aware Replica Placement Heuristics and Applications in Astronomy Data Grid. *Journal of Systems and Software*, 84 (7): 1224-1232.

Enokido, T., and Takizawa, M. 2008. A Purpose-Based Synchronization Protocol of Multiple Transactions. *Proceedings of 14th IEEE International Conference on Parallel and Distributed Systems*. ICPADS '08, pp.145-152.

Eya, B. A., AhlemN., FaïezG., 2011, Performance of Short-Commit in Extreme Database Environment. *Proceedings of International Journal of Database Management Systems(IJDMS)*, 3 (2): 1 – 41.

Fan, C., and Xu, M. 2010. A Solution to Deadlock in Implementing Synchronizing Merges of Workflow. *Proceedings of 6th International Conference on Advanced Information Management and Service (IMS)*, Nov. 30 – Dec. 2 2010.pp. 498. (05713501).

Fard, A.M, Kamyar, H., and Naghibzadeh, M. 2008. Multi-Expert Disease Diagnosis System Over Symptom Data Grids on the Internet. *Proceedings of World Applied Sciences Journal*, 3(2): 244-253.

Foster, I., Kesselman, C., and Tuecke, S. 2001. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Proceedings of International Journal of High Performance Computing Application* 15(3): 200-222.

Foster, I., Kesselman, C., Nick, J., and Tuecke, S. 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Globus Project*, pp. 1-30.

Foster, I., Zhao, Y., Raicu, I. and Lu, S. 2008. Cloud Computing and Grid Computing 360-Degree Compared. *Grid Computing Environments Workshop*, GCE '08 , pp.1-10.

Fox, G., Ko, S.-H., Pierce, M., Balsoy, O., Kim, J., Lee, S., Kim, K., Oh, S., Rao, X., Varank, M., Bulut, H., Gunduz, G., Qiu, X., Pallickara, S., Uyar, A., and Youn, C. 2002. Grid Services for Earthquake Science. *Proceedings of Concurrency and Computation: Practice and Experience* 14 (6-7) 371-393.

Garcia-Munoz, L.H., Armendariz-Inigo, J.E., Decker, H., and Munoz-Escoi, F.D. 2007. Recovery Protocols for Replicated Databases-A Survey. *Proceedings of 21st International Conference on Advanced Information Networking and Applications Workshops.* AINAW '07, 1: 220-227.

Gomez, E., and Schubert, K. 2010. Algebra of Synchronization with Application to Deadlock and Semaphores. *Proceedings of IEEE First International Conference on Networking and Computing*, pp. 202-208.

He, X., Ou, L., Engelmann, C., Chen, X., and Scott, S.L. 2009. Symmetric Active/Active Metadata Service for High Availability Parallel File Systems. *Journal of Parallel and Distributed Computing*, Volume 69, Issue 12, December 2009, pp. 961-973.

Hu, H., and Li, Z. 2009. Local and Global Deadlock Prevention Policies for Resource Allocation Systems Using Partially Generated Reachability Graphs. *Comput. Ind. Eng.* 57(4): 1168-1181.

Hu, H., Li, Z., and Zhou, M. 2008. Two Generalized-Petri-net-based Strategies for Deadlock Prevention in Resource Allocation Systems. *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*.pp.1948-1953.

Jiang, B., Deprettere, E., and Kienhuis, B. 2008. Hierarchical Run Time Deadlock Detection in Process Networks. *Proceedings of IEEE Workshop on Signal Processing Systems*, pp. 239-244.

Khachana, R.T., James, A., and Iqbal R. 2011. Relaxation of ACID Properties in AuTrA, The Adaptive User-Defined Transaction Relaxing Approach. *Future Generation Computer Systems*, 27 (1): 58-66.

Khanli, L.M., Isazadeh, A., and Shishavan, T.N. 2011. PHFS: A Dynamic Replication Method, to Decrease Access Latency in the Multi-Tier Data Grid. *Future Generation Computer Systems*, 27 (3): 233-244.

Latip, R., Ibrahim, H., Othman, M., Sulaiman, M.N, and Abdullah, A. 2008. Quorum Based Data Replication in Grid Environment. RSKT'08 *Proceedings of the 3rd International Conference on Rough Sets and Knowledge Technology*, pp. 379-386.

Lee, S., and Joo, K.H.2005. Efficient Detection and Resolution of OR Deadlocks in Distributed Systems. *Journal of Parallel and Distributed Computing*, 65 (9): 985-993.

Li, P., Agrawal, K., Buhler, J., Chamberlain, R.D., and Lancaster, J.M. 2010. Deadlock-Avoidance for Streaming Applications with Split-Join Structure: Two Case Studies. *Proceedings of 21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP)*, pp. 333-336.

Li, T., Yan, B., and Luan, Z. 2010. Fast Large File Distribution in Data Grid. *Proceedings of International Conference on Intelligent Computing and Integrated Systems (ICISS)*, pp.577-582.

Ling, Y., Chen, S., and Chiang, C.J. 2006. On Optimal Deadlock Detection Scheduling. *Proceedings of IEEE Transactions on Computers*, 55 (9):1178-1187.

Mayer, S., and Furmans, K. 2010. Deadlock Prevention in a Completely Decentralized Controlled Materials Flow Systems. *Logistic Research, Springer,* pp. 1-12.

Mitchell, D.P., and Merritt, M.J. 1984. A Distributed Algorithm for Deadlock Detection and Resolution. *In Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pp. 282-284.

Mohammed, T.S. 2007. Performance Improvement and Deadlock Prevention for a Distributed Fault Diagnosis Algorithm. *Journal of Computer Science* 3(2): 107-112.

Naseera, S., and Murthy, K.V.M. 2009. Agent Based Replica Placement in a Data Grid Environement. *First International Conference on Computational Intelligence, Communication Systems and Networks*. CICSYN '09, pp.426-430.

Noraziah A., Deris, M.M., Ahmed, N.A., Saman, M.Y.M, Norhayati, R., and Alfawaer, Z.M. 2007. Preserving Data Consistency through Neighbor Replication on Grid Daemon. *Proceedings of American Journal of Applied Sciences* 4 (10): 751-758.

Noraziah, A. 2009c. Neighbour Replica Failure Semantic Using NRTM in Distributed Environment. *Proceedings of International Conference on Software Engineering & Computer Science, Malaysia*, pp. 530-533.

Noraziah, A., Abdalla, A.N., and Sidek, R.M. 2010d. Data Replication Using Read-One-Write-All Monitoring Synchronization Transaction System in Distributed Environment. *Journal Computer Science* 6: 1033-1036.

Noraziah, A., DerisM.Mat, Ahmed N.A., Norhayati R., Saman, M.Y., Norhaslinda D.C., 2009a. Neighbour Replication Transactions Processing in Distributed System. *Advances in Systems Science and Applications*, (ASSA), *Proceedings of International Journal of IIGSS*, 9 (2).

Noraziah, A., Deris, M.M., Rosli, N., Saman, Md.Y.M.,Mamat, R., and Wan NikShuhadah, W.N. 2006. Managing Neighbour Replication Transactions in Distributed Systems. *Proceedings of International Symposium on Distributed Computing & Applications Business Engineering and Science*, 1: 95-101.

Noraziah, A., Deris, M.M., Saman, M.Y.M., Norhayati, R., Rabiei, M., and Shuhadah, W.N.W. 2009b. Managing Transactions on Grid-Neighbour Replication in Distributed Systems. *Proceedings of International Journal of Computer Mathematics* Vol. 86, Iss. 9, pp. 1-10.

Noraziah, A., AinulAzila, C.F, Roslina, M.S., Noriyani, M.Z. and Beg, A.H. 2010. Lowest Data Replication Storage of Binary Vote Assignment Data Grid. *Proceedings of NDT (2), the Second International Conference on Networked Digital Technologies*, pp. 466 – 473.

Noraziah, A., Klaib, M.F.J., and Sidek, R.M. 2010c.Failure Semantic of Neighbour Replication Grid Transaction Model. *Proceedings of 10th IEEE International Conference on Computer and Information Technology(CIT 2010)*, pp.668-673.

Noraziah, A., Zin, N.M., Sidek, R.M., Klaib, M.F.J., and Wahab, M.H.A. 2010b. Neighbour Replica Transaction Failure Framework in Data Grid. *Proceedings,*

Part II. Communications in Computer and Information Science. *NDT 2010*, Part II, CCIS 88, Zavoral F. et al. (Eds.), Springer-Verlag Berlin Heidelberg, pp. 488–495.

Nyo, T.T. 2009.Maximizing the Degree of Concurrency and Consistency in Distributed Parallel Processing Environment. *Proceedings of International Conference on Future Networks*, pp.258-262.

Olson, A.G., and Evans, B.L. 2005. Deadlock Detection for Distributed Process Networks. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,* 5: 73-76.

Razzaque, M.A., and Hong, C.S. 2008.Multi-Token Distributed Mutual Exclusion Algorithm. *Proceedings of 22nd International Conference on Advanced Information Networking and Applications*, pp.963-970.

Razzaque, M.A., Mamun-Or-Rashid, M., and Hong, C.S. 2007. MC2DR: Multi Cycle Deadlock Detection and Recovery Algorithm for Distributed Systems. *Proceedings of the High Performance Computing and Communications*, pp. 554–565

Roszkowska, E. 2004. Supervisory Control for Deadlock Avoidance in Compound Processes. *Proceedings of IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans,* 34 (1): 52- 64.

Sanchez, C., Sipma, H.B., and Manna, Z. 2007. Generating Efficient Distributed Deadlock Avoidance Controllers. *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, pp.1-8.

Sanzo, P.D., Ciciani, B., Quaglia, F., and Romano, P. 2008. A Performance Model of Multi-Version Concurrency Control. *Proceedings of IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pp.1-10.

Sanzo, P.D., Palmieri, R., Ciciani, B., Quaglia, F., and Romano, P. 2010. Analytical Modeling of Lock-Based Concurrency Control with Arbitrary Transaction Data Access Patterns. *In Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, pp. 69-78.

Sashi, K., and Thanamani, A.S. 2011. Dynamic Replication in a Data Grid Using a Modified BHR Region Based Algorithm. *Future Generation Computer Systems*, 27(2): 202-210.

Sathya, S.S., and Babu, K.S. 2010. Survey of Fault Tolerant Techniques for Grid. *Computer Science Review (2010)*. 4 (2): 101-120.

Shahzad, M.Y., and Rizwan, M. 2011. Performance of Short-Commit in Extreme Database Environment. *International Journal of Database Management Systems (IJDMS)*, 3 (2).

Singh, Y.J., and Mehrotra, S.C. 2009. An Analysis of Real-Time Distributed System under Different Priority Policies. *Proceedings of World Academy of Science, Engineering and Technology*, 56: 166-171.

Srinivasan, S., and Rajaran, R. 2011.A Decentralized Deadlock Detection and Resolution Algorithm for Generalized Model in Distributed Systems. *Proceedings of Distributed Parallel Databases*,29: 261-276.

Sun, X., Zheng, J., Liu, Q., and Liu, Y. 2009. Dynamic Data Replication Based on Access Cost in Distributed Systems. *Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology*, pp. 829-834.

Thiare, O. 2009. A Solution to Improve Algorithm for Distributed Mutual Exclusion by Restricting Message Exchange in Quorums. *Proceedings of Second International Conference on the Applications of Digital Information and Web Technologies*,pp.38-42.

Wang, Y., and Sijun, L. 2006. Research and Performance Evaluation of Data Replication Technology in Distributed Storage Systems. *Proceedings of Computers & Mathematics with Applications*, 51: 1625-1632.

Wong, L., Arora, N.S., Gao, L., Hoang, T., and Wu, J. 2009. Oracle Streams: A High Performance Implementation for Near Real Time Asynchronous Replication. *Proceedings of IEEE 25th International Conference on Data Engineering*, pp.1363-1374.

Wu, H., Chin, W.-N., and Jaffar, J. 2002. An Efficient Distributed Deadlock Avoidance Algorithm for the AND Model. *Proceedings of IEEE Transactions on Software Engineering*, 28(1): 18-29.

Xiao, X., and Lee, J.J. 2010. A True O(1) Parallel Deadlock Detection Algorithm for Single-Unit Resource Systems and Its Hardware Implemention. *Proceedings of IEEE Transactions on Parallel and Distributed Systems*, 21(1): 4-19.

Xiao, X., and Lee, J.J. 2011. A Parallel Multi-Unit Resource Deadlock Detection Algorithm with O (log2 (min (m,n))) Overall Run-Time Complexity. *Journal Parallel Distributed Computing*, 71(7): 938-954.

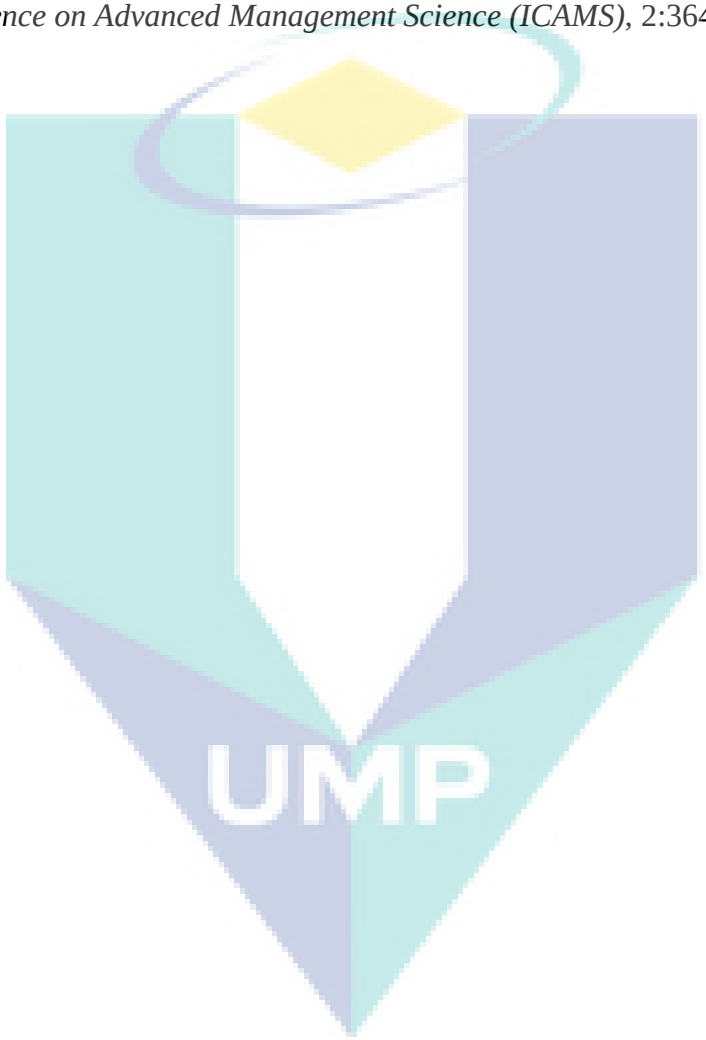
Xiao, Y., Zhang, H., and Wang, F. 2007. Maintaining Temporal Consistency in Real-Time Database Systems. *Proceedings of International Conference on Convergence Information Technology*, pp.1627-1633.

Zhang, Z., Wu, W., and Shekhar, S. 2009. Optimal Placements of Replicas in a Ring Network with Majority Voting Protocol. *Journal of Parallel and Distributed Computing*, 69(5): 461-469.

Zhao, W., Xu, X., Xiong, N., and Wang, Z. 2008. A Weight-Based Dynamic Replica Replacement Strategy in Data Grids. *IEEE Asia-Pacific Services Computing Conference*. APSCC '08, pp.1544-1549.

Zhao, W., Xu, X., Wang, Z., Zhang, Y. and He, H. 2010. A Dynamic Optimal Replication Strategy in Data Grid Environment. *International Conference on Internet Technology and Applications*, pp.1-4.

Zheng, Q., and Bi, X. 2010.An Improved Concurrency Control Algorithm for Distributed Real-Time Database. *Proceedings of 2010 IEEE International Conference on Advanced Management Science (ICAMS)*, 2:364-367.



BIODATA OF THE AUTHOR

The author was born in 1985 in Kelantan, Malaysia. She obtained diploma in Computer Science in 2003 and bachelor degree in Computer Science (Software Engineering) in 2006 from University Malaysia Pahang, Malaysia. Currently she is undergoing her M.Sc program at the Faculty of Computer Systems and Software Engineering, University Malaysia Pahang.

Her current research interests include distributed systems, database systems, database replication and deadlock handling in database. She has published 4 articles in journals and proceedings (international). For this work, she has published one scientific journal and three proceedings.

LIST OF PUBLICATIONS

1. **Noriyani Mohd Zin** , A.Noraziah , Ahmed N. Abdalla, Ainul Azila Che Fauzi, "Solving Two Deadlock Cycles through Neighbor Replication on Grid Deadlock Detection Model", Journal of Computer Science, 8(2): 265-271, 2012.**Index: ISI.**
2. **Noriyani Mohd Zin**, A.Noraziah, Ainul Azila Che Fauzi, "Neighbour Replication on Grid Deadlock Detection Framework", Ezendu et al. (Eds): Communications in Computer and Information Science (CCIS) vol. 194, Springer-Verlag Berlin Heidelberg, pp. 350–357, 2011.**Index: ISI Proceedings and Scopus.**
3. **Noriyani Mohd Zin**, A.Noraziah, A.H.Beg, Ainul Azila Che Fauzi, "Deadlock Detection and Resolution in Neighbour Replication on Grid", ICCCM2011, pp. 426-430, 2011, **ISBN-13:**978-981-08-8636-3.
4. **Noriyani Mohd Zin**, A.Noraziah, Ainul Azila Che Fauzi, Tutut Herawan, "Replication Techniques in Data Grid Environments", Proceedings ACIIDS'12 Proceedings of the 4th Asian conference on Intelligent Information and Database Systems - Volume Part II pp. 549-559.