MANAGEMENT OF DISTRIBUTED DEADLOCKS IN DISTRIBUTED DATABASES

By

MERCY CHINYUKU

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computer Science

Department of Computer Science

Faculty of Science

University of Zimbabwe

August 2010
ABSTRACT

The developments of database management system (DBMS) technology has run in parallel with the developments in computer networks and distributed computing technologies. Consequently there is the birth of distributed database management systems. These systems cannot be ignored any longer because they have now dominated data management applications. Many DBMS have incorporated some degree of distribution into their products, thus it is very important that the issue of deadlocks in distributed databases has to be solved.

A deadlock is a potential problem in any system having multiple transactions which share resources. Transactions involved in a deadlock will indefinitely wait for the resources to be released, unless the deadlock is detected and recovery measures are taken. In distributed systems, lack of information about the complete system states compounds the deadlock detection problem.

There has been a wide range of research with focus on deadlocks in distributed systems in general, but distributed deadlocks in distributed databases has been neglected to some extend. This thesis describes in detail the distributed databases in terms of the architectures and types distributed databases. There are a number of algorithms that try to resolve and manage distributed deadlocks and some of them have not yet been proved to be correct. In this thesis the researcher analysed some of the algorithms that have been implemented by some researchers. The algorithms analysed used the prevention method which requires that a transaction should have all the required resources before it proceeds which is virtually impossible in a busy distributed environment.

The researcher proposed the ISKE algorithm that shows that it is possible for any requesting transaction to proceed processing without wasting unreasonable amounts of time. The algorithm showed that employing a method of priority as well as queuing (FIFO) can greatly manage the deadlocks effectively well.
I would like to acknowledge the following for their unwavering support throughout the whole research. My supervisor Dr. G. T. Hapanyengwi for his constant encouragement and guidance. He also gave me strength to go on when I felt I wanted to give up. To Mr. T. Rupere for his assistance throughout the research. To the department of Computer Science University of Zimbabwe to afford me another chance to finish the thesis. To my employer Chinhoyi University of Technology for affording me the opportunity to learn. To my student Tichaona Dlamini who taught me that programming is not very frightening. To my mother Dinah, my daughter Isis and son Ike for their support. Lastly but not the least to the God Almighty for seeing me through I could not have done it without him.
# TABLE OF CONTENTS

## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>i</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF ALGORITHMS</td>
<td>viii</td>
</tr>
<tr>
<td>ALGORITHMS</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER ONE: INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 BACKGROUND</td>
<td>2</td>
</tr>
<tr>
<td>1.2 PROBLEM STATEMENT</td>
<td>3</td>
</tr>
<tr>
<td>1.3 JUSTIFICATION</td>
<td>4</td>
</tr>
<tr>
<td>1.4 AIMS AND OBJECTIVES</td>
<td>5</td>
</tr>
<tr>
<td>1.5 WHAT HAVE OTHER RESEARCHERS DONE?</td>
<td>6</td>
</tr>
<tr>
<td>1.6 THE PROBLEM: MY CENTRAL IDEAS</td>
<td>9</td>
</tr>
<tr>
<td>1.7 DELIVERABLES</td>
<td>10</td>
</tr>
<tr>
<td>1.8 DEFINITION OF TERMS</td>
<td>11</td>
</tr>
<tr>
<td>1.9 CONCLUSION</td>
<td>13</td>
</tr>
<tr>
<td>CHAPTER TWO: LITERATURE REVIEW</td>
<td>14</td>
</tr>
<tr>
<td>2.0 INTRODUCTION</td>
<td>14</td>
</tr>
<tr>
<td>2.1 BACKGROUND</td>
<td>15</td>
</tr>
<tr>
<td>2.2 DISTRIBUTED DATABASE ARCHITECTURE</td>
<td>17</td>
</tr>
<tr>
<td>2.3 TYPES OF DISTRIBUTED DATABASES</td>
<td>18</td>
</tr>
<tr>
<td>2.5 DISTRIBUTED DEADLOCK MANAGEMENT IN DISTRIBUTED DATABASES</td>
<td>25</td>
</tr>
<tr>
<td>2.6 CONCLUSION</td>
<td>41</td>
</tr>
<tr>
<td>CHAPTER THREE: METHODOLOGY</td>
<td>43</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

FIGURES

Figure 1: Homogeneous Distributed Database

Figure 2: Resource R is allocated to process P

Figure 3: Resource R is requested by process P

Figure 4: Deadlock Example

Figure 5: Wait For Graph Example

Figure 6: Chandy-Misra-Haas Algorithm

Figure 7: Hierarchical Deadlock detection

Figure 8: Main Screen

Figure 9: Sample Results for the ISKE Algorithm

Figure 10: Sample Results for the Multishot Algorithm

Figure 11: Test 1 – ISKE Algorithm

Figure 12: Test 1- MultiShot Algorithm

Figure 13: Test 1 – Bar Chart

Figure 14: Test 1 Random Requests –Line Chart

Figure 15: Test 1 Random Requests - Bar Chart

Figure 16: Test 2- Line Chart

Figure 17: Test 2 - Bar Chart

Figure 18: Test 2 Random Requests - Line Chart

Figure 19: Test 2 Random Requests - Bar Chart

Figure 20: Test 3 - Line Chart

Figure 21: Test 3 - Bar Chart

Figure 22: Random Requests - Line Chart

Figure 23: Test 3 Random Requests - Bar Chart
LIST OF TABLES

Table 1: Sample Deadlock ........................................................................................................................................... 51
Table 2: Best Case ......................................................................................................................................................... 53
Table 3: Worst Case ....................................................................................................................................................... 57
Table 4: Test 3.............................................................................................................................................................. 61
Table 5: Test 4.............................................................................................................................................................. 65
Table 6: Test 5.............................................................................................................................................................. 69
Table 7: Test 6.............................................................................................................................................................. 71
Table 8: Summary of Results ...................................................................................................................................... 74
LIST OF ALGORITHMS

ALGORITHMS

Algorithm 1: One-Shot ........................................................................................................ 29
Algorithm 2: Repeated One-Shot (MultiShot) ................................................................. 30
Algorithm 3: Hierarchical .................................................................................................. 31
Algorithm 4: Chandy-Misra-Haas ..................................................................................... 37
Algorithm 5: ISKE .............................................................................................................. 47
CHAPTER ONE: INTRODUCTION

1.0 INTRODUCTION

In any application operating in a distributed environment, there are many concerns to deal with. In distributed databases the concerns are mainly concurrency control. Deadlocks affect the performance of the system. If several transactions are using the same records at the same time some transactions would have to wait for use of the records until the records are made available. The waiting time part becomes an issue if the transactions have to wait forever for the release of the records.

This research should benefit ever expanding Universities. Student enrolment is increasing every year so is the need for more storage space because of the increase in the size of the database. Universities are no longer physically located at one site because schools/ faculties/ institutes are now geographically spread all over, hence the introduction of distributed databases.

The success of a distributed database system depends on its ability to arrest the problem of concurrency control. It is more difficult to manage deadlocks in distributed databases than centralised databases. Hence there is need to design more robust and efficient algorithms that detect and resolve these deadlocks.
A distributed database is not stored entirely at a single physical location; instead it is spread across a network of computers that are geographically dispersed and connected via communication links. A distributed database allows faster local queries and can reduce network traffic. With these benefits comes the issue of maintaining database integrity. A key objective for a distributed database is that it looks like a centralised system to the user. The user should no know where a piece of data is stored physically.

With the advent of wired and wireless data communication technologies and the Internet, distributed databases have become the most efficient technology for sharing data across distances. Many modern applications and databases are distributed. Distributed deadlocks can occur in banks, airlines and telecommunications and in industries where distributed databases are used.
Currently most applications use distributed databases as their base foundation. As transactions increase concurrency problems also increase, of major concern is deadlock management. When deadlocks occur some systems adopt the "Ostrich" approach of ignoring the problem hoping it will resolve itself. If deadlocks are ignored this is costly to organisations in terms of resources and time. An example of a disruption would be if for example students who want to register online fail to do so because the accounts section are carrying out auditing and are making use of students records. This exercise may take a few hours because there are many students' records locked for updating. The consequence is that the university would be at stand still whilst they wait for the exercise to end.

There is need for an environment in which transactions are carried out without users being aware that there are any deadlocks happening. Thus there is need for a Resource manager that uses an efficient technique to allocate resources (records) to processes (transactions).
The research is important because it will help minimise downtime for companies hence reduce costs incurred when it is down. The research is also important because it explores the different algorithms proposed to solve distributed deadlocks. It is also important because it gives the researcher a chance to explore and analyse these algorithms their limitations and efficiencies. The researcher will also try to apply these algorithms in a new environment like application in a learning institute with vast functional areas. The researcher will get the opportunity to come up with an algorithm that best resolves deadlocks in a distributed environment (like a university with various needs) that can be used or incorporated to improve efficiency of distributed database transactions which occur in real time.
1.4.1 Aim

The aim of the study is to build an appreciation of the functionality of distributed databases.

1.4.2 Objectives

Objectives of the study are;

a) To explore the algorithms used to address distributed deadlocks.

b) To investigate and analyse the current deadlock algorithms being used like the Multi Shot algorithm, the Chandy Misra Haas algorithm and Hierarchical algorithm.

c) To design, implement and assess an alternative algorithm that can be used to Manage distributed deadlocks.
WHAT HAVE OTHER RESEARCHERS DONE?

Several strategies have been proposed to handle distributed deadlocks. The strategies are Ignore (the Ostrich strategy), Detection, Prevention and Avoidance strategies. Avoidance strategies that are based on protocols that further restrict the way transactions may request locks. If transactions obey the protocol, deadlocks are impossible and no additional mechanisms are required to detect and resolve deadlocks. Detect strategies resolve strategies do not restrict the protocol and hence allow deadlocks to occur. These strategies expend effort looking for deadlocks and resolve the deadlock; typically by using undoing and re-executing transactions. Three basic techniques exist to control deadlocks:

1.5.1 Deadlock Avoidance

The transaction must obtain all the locks it needs before it can be executed. The technique avoids rollback of conflicting transactions by requiring that locks be obtained in succession. An advantage of this strategy is that transactions are not undone and re-executed due to deadlocks and hence computing resources are not wasted on these processes. The disadvantage is that they may increase the time it takes a transaction to complete because the protocol forces the transaction to lock more than it needs and to wait simply because giving it access to some other item even if the item is available at the time of request could possibly lead to future deadlocks.
The Banker’s algorithm developed by (Dijkstra Edsger, 1960) is one known algorithm used for deadlock avoidance. It is not discussed in detail in this research. The problem with the algorithm is that resources cannot be allocated to new processes unless old processes have been satisfied.

1.5.2 Deadlock Detection

The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (victim) is aborted (rolled back and restarted) and the other transaction continues. Transactions wait for each other in an uncontrolled manner and are only aborted if a deadlock actually occurs. Deadlocks are detected by explicitly constructing the waits-for graph and searching it for cycles. If a cycle is found, one transaction on the cycle, called the victim, is aborted, thereby breaking the deadlock. To minimize the cost of restarting the victim, victim selection is usually based on the amount of resources used by each transaction on the cycle.

The advantage of this strategy is that it allows transactions to proceed in a less restrictive manner often leading to faster completion; however the strategy forces the transaction manager to devote resources to undoing and re-executing transactions and also the task of deadlock detection.

(Chandy, Misra and Haas, 1983) used distributed deadlock models and presented distributed algorithms for detection of resource and communication deadlocks. The algorithms are analysed in chapter two.
Detect-resolve strategies have the advantage that they allow transactions to proceed in a less restrictive manner often leading to faster completion. However, the strategy forces the transaction manager to devote resources to undoing and re-executing transactions and also the task of deadlock detection.

A transaction requesting a new lock is aborted if there is the possibility that a deadlock may occur. If the transaction is aborted, all the changes made by this transaction are rolled back, and all locks obtained by the transaction are released the transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking. Deadlock prevention is a "cautious" scheme in which a transaction is restarted when the system is "afraid" that deadlock might occur.
Distributed deadlock is a major issue in concurrency control in distributed databases. Deadlocks are costly to organisations hence it is important to manage and allocate resources appropriately so as to minimise downtown. A deadlock is an occurrence where each transaction in a set of transactions circularly waits on locks that are held by other transactions in the set. It is a potential problem in any system having multiple transactions which share resources. Transactions involved in a deadlock will indefinitely wait for the resources to be released, unless the deadlock is detected and recovery measures are taken. In distributed systems, lack of information about the complete system states compounds the deadlock detection problem. The researcher will focus on distributed deadlocks that occur in distributed databases when distributed transactions or concurrency control is used.

The central idea to the research is to aim at maximizing efficiency of distributed databases by minimising distributed deadlocks. The researcher will analyse existing algorithms associated with distributed deadlocks such as the Chandy Misra Haas algorithm, The One-Shot algorithm, the Multi-Shot algorithm and the hierarchical algorithm. The researcher will present an algorithm that avoid, prevent and detect deadlocks in distributed databases.

More than 20 concurrency control algorithms have been proposed for DDBMSs, and several have been, or are being, implemented. Some of these algorithms are complex, hard to understand, and difficult to prove correct, because they are described in different terminologies and make different assumptions about the underlying DDBMS environment, it is difficult to compare the many proposed algorithms, even in qualitative terms.
a) Algorithm(s) that can be used to resolve distributed deadlocks.

b) A Distributed deadlock simulator.

c) Formal thesis document.
a) **Atomicity**: The property of transaction processing whereby either all the operations of a transaction are executed or none of them are (all-or-nothing).

b) **Client/server architecture**: A distributed/parallel DBMS architecture where a set of client machines with limited functionality access a set of servers which manage data.

c) **Concurrency control algorithm**: Algorithms that synchronize the operations of concurrent transactions that execute on a shared database.

d) **Concurrency control**: deals with the problem of coordinating the actions of processes that operate in parallel, access shared data, and therefore potentially interfere with each other.

e) **Concurrency**: every node of a distributed system wants to use shared resources or same variables. An example for shared resource is the network. It has to be possible that all nodes are able to transmit their messages. For real-time system it is also necessary that this can be done in predictable time. The usage of the same variables on different nodes implies the risk that this variable is written by one node and overwritten immediately by another node before it has been replicated.

f) **Deadlock**: In database systems, a deadlock occurs when some transactions wait indefinitely on each other for their requests to be satisfied. A deadlock hampers the progress of transactions in a database and lowers the resource availability; therefore, all deadlocks must be promptly detected and eliminated. A set $X$ of processes is said to be involved in a deadlock if there exists cyclic dependency among processes in $X$. If a set $X$ of processes is involved in a deadlock, then no process in $X$ can ever become active without external action being taken to resolve the deadlock.

g) **Dirty read**: this occurs when one transaction writes a database item and then the transaction fails or aborts for some reason. If a second transaction has read the updated
If the new value is committed back to the old value due to the abort of the first traction, the read value is called dirty data because this value is not allowed to exist in a valid database state.

h) Distributed Database Management System (DDBMS): A database management system that manages a database that is distributed across the nodes of a computer network and makes this distribution transparent to the users. OR A distributed database management system is a collection of sites interconnected by a network. Each site is a computer running one or both of the following software modules: a transaction manager (TM) or a data manager (DM). TMs supervise interactions between users and the DDBMS while DMs manage the actual database.

i) Distributed database: (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network.

j) Distributed deadlock: when a process try to access a locked resource which is being used by another process in which in turn is also waiting for locked resources by other processes.

k) Distributed system: is a system of multiple autonomous processing elements, cooperating for a common purpose. It can either be a tightly or loosely coupled system, depending on whether the processing elements have access to a common memory or not.

l) Global state: is information about all other sites in a distributed environment.

m) Heterogeneous database: different sites might use different schemas and software.

n) Homogenous database: all sites have identical software.
It can be concluded that distributed databases are beneficial to organization, institutes and companies, but the issue of distributed deadlocks can be a major problem if they are not well managed. Distributed can be costly to organizations if they are left unchecked. The next chapter will look at existing database architectures as well as the analyses existing algorithms in terms of complexity and efficiency.
CHAPTER TWO: LITERATURE REVIEW

2.0 INTRODUCTION

This chapter focuses at what other researchers have done in the area of concurrency problems in distributed databases in particular the issue of distributed deadlocks and how they have been dealt with. There are several strategies that have been proposed by different researchers to deal with the problems of distributed deadlocks, some of the algorithms are yet to be proved. In this chapter the researcher analysed some of these strategies. The chapter briefly describes the various distributed database architectures as well as the different types of distributed databases.
According to (M. Tamer Özsu and Patrick Valduriez, 1991) they defined a distributed database as a collection of multiple, logically interrelated databases distributed over a computer network. It is a set of databases stored on multiple computers that typically appear to applications as single databases. Consequently an application can simultaneously access and modify the data in several databases in a network. A distributed database management system (DDBMS) is a software system that permits management of a distributed database and makes the distribution transparent to the users. DDBMS is a specially developed for heterogeneous management systems.

A distributed database is not stored entirely at a single location; instead it is spread across a network of computers that are geographically dispersed and connected via communication links. A distributed database allows faster local queries and can reduce network traffic. With these benefits comes the issue of maintaining database integrity. A key objective (Bhavani and Hai-Ping Ko, 2004) for a distributed database is that it looks like a centralised system to the user. The user does not need to know where a piece of data is stored physically.

The concept of distributed databases took off with the advent of wired and wireless data communication facilities. It assumed greater significance when the Internet was born. Now that the Internet has become an extremely important medium for conducting business worldwide, distributed databases have become the most efficient technology for sharing data across distances. The technology behind computer networks and data communication has profound implications on the way computers work. Computer networks make it possible to
connect multiple computers to each other. These computers can be located in the same or different buildings, cities, countries or even continents. This flexibility in terms of the distribution of processing logic and data across these networks unlike earlier when everything was centralised. Many modern applications and databases are distributed. Examples where distributed deadlocks can occur is in banks, airlines and telecommunications industries where distributed database are used.
There are three possible architectures that can be considered when dealing with distributed databases.

2.2.1 Shared Nothing

A Shared Nothing architecture is a parallel DBMS architecture where each processor has exclusive access to its main memory and disk unit(s). (M. Tamer Özsu and Patrick Valduriez, 1997). In this type of architecture, different computers are connected to one another to form a distributed environment. However each computer owns its database. No computer shares its database with any other. Thus although this is distributed environment containing computers connected by a network, the data itself is not shared at all.

2.2.2 Networked Architecture With One Centralised Database

The architecture has one common database which is shared by the entire computers in the distributed environment. Thus all the applications on the distributed computers share a single database. This architecture is also referred to as Multiple-client/Single-server (M. Tamer Özsu, 1991). The database is stored on a single server.

2.2.3 Truly Distributed Database

The truly distributed database architecture, here every computer in the distributed environment has its own database. However, all these databases are shared, unlike in shared nothing architecture (Atul Kahate, 2004).
Distributed database systems can be categorised according to type of software depending on the requirements of software.

2.3.1 Homogeneous Distributed Database

In homogeneous distributed databases all sites have identical software. They are aware of each other and agree to cooperate in processing user requests. Each site surrenders part of autonomy in terms of right to change schema or software. It appears to a user as a single system (Silberschatz Korth and Sudarshan, 2005). Figure 1 shows a homogeneous distributed database.
In heterogeneous distributed databases different sites might use different schemas and software. The difference in schemas is a major problem for query processing while difference in software is a major problem for transaction processing. Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing. Sites may not be aware of each other and may only provide only limited facilities for cooperation in transaction processing (Silberschatz Korth and Sudarshan, 2005).
DISTRIBUTED DEADLOCKS

Krzyzanowski defined a deadlock as a condition where a process cannot proceed because it needs to obtain a resource held by another process and it itself is holding a resource that the other process needs. Krzyzanowski considered two types of deadlocks namely communication deadlock and resource deadlock.

In resource deadlocks, a process needs multiple resources for an activity. Deadlock occurs if each process in a set request resources held by another process in the same set, and it must receive all the requested resources to move further.

In communication deadlocks, processes wait to communicate with other processes in a set. Each process in the set is waiting on another process’s message, and no process in the set initiates a message until it receives a message for which it is waiting. Communication deadlock occurs when for example, let’s say process A is trying to send a message to process B, which is trying to send a message to process C which is trying to send a message to A.

A resource deadlock occurs when processes are trying to get exclusive access to devices, files, locks, servers, or other resources. Since a communication link is a shared resource, hence we can consider a communication deadlock to be also an instance of a resource deadlock.

According to (E.G. Coffman, M.J. Elphick and A Shoshani, 1971), there are four conditions necessary for a deadlock to occur known as Coffman conditions. These are Mutual Exclusion, Hold and Wait, No pre-emption and Circular Wait conditions.
In Mutual Exclusion, the processes claim exclusive control of the resources they require, i.e. a resource can be held by at most one process. Removing the mutual exclusion means that no process may have exclusive access to a resource. This proves impossible for resources that cannot be spooled and even with spooled resources deadlocks could still occur. Algorithms that avoid mutual exclusion are called non-blocking-synchronisation algorithms.

2.4.2 Hold and Wait

Processes that already hold resources allocated to them can wait for additional resources. The "hold-and-wait" condition may be removed by requiring processes to request all resources they will need before starting up (or before embarking on a particular set of operations). This advanced knowledge is frequently difficult to satisfy and in any case an inefficient use of resources. Another way is to require processes to release all their resources before requesting all the resources that need. This too is often impractical.

2.4.3 No Pre-emption

Resources cannot be forcibly removed from the tasks holding them until the resources are used to completion, i.e. once a resource is granted to a process it cannot be taken away from a process. A "no pre-emption (lockout)" condition may also be difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time or the processing outcome may be inconsistent or thrashing may occur. However inability to enforce pre-emption may interfere with a priority algorithm. Pre-emption of a "locked out" resources generally implies a roll back and is to be avoided since it is very costly in overhead.
2.4.4 Circular Wait

A cyclic chain of tasks exists, such that each task holds one or more resources that are being requested by the next task in the chain.

The resource allocation can be represented as a graph where: \( P \rightarrow R \) means that a resource \( R \) is currently held by a process \( P \), see Figure 2. \( P \leftarrow R \) means that a process \( P \) wants to gain exclusive access to resource \( R \), see Figure 3. Deadlock exists when a resource allocation graph has a cycle, see Figure 4.

![Figure 2: Resource R is allocated to process P](image1)

![Figure 3: Resource R is requested by process P](image2)
Circular wait method allows processes to wait for resources but ensure that waiting cannot be circular. One approach might be to assign a precedence to each resource and force processes to request resources in order of increasing precedence, i.e. to say that if a process holds some resource and the highest precedence of those resources is $m$, then this process cannot request any resource with precedence smaller than $m$. This forces allocation to follow a particular and non-circular ordering so circular wait cannot occur. Another approach is to allow holding only one resource per process. If a process requests another resource, it must first free the one it is currently holding (that is disallowing hold and wait).

The circular wait condition has algorithms that avoid circular waits and this includes disable interrupts during critical sections and use a hierarchy to determine a practical ordering of resources where no obvious hierarchy exists, even the memory address of resources has been used to determine ordering and Dijkstra’s solution.
If a deadlock occurs, all four conditions must be true. To prove that a deadlock will never occur one has to show that the four conditions never occurs for some algorithm.
According to (Kshemkalyani and M. Singhal, 2009), deadlock handling using the approach of deadlock detection entails addressing two basic issues: First, detection of existing deadlocks and second, resolution of detected deadlocks. Detection of deadlocks involves addressing two issues: Maintenance of the Wait-For-Graph (WFG) and searching of the WFG for the presence of cycles (or knots). For explanation see section 2.5.3.1.

Like almost everything else in a distributed environment, distributed deadlocks are a great design challenge. Dealing with deadlocks even in non-distributed environment is not easy, coupled with the challenge of multiple sites, the task becomes even tougher. There are three strategies for handling deadlocks, viz., deadlock prevention, deadlock avoidance, and deadlock detection. Handling of deadlock becomes highly complicated in distributed systems because no site has accurate knowledge of the current state of the system and because every inter-site communication involves a finite and unpredictable delay.

The researcher considered three strategies related to deadlocks in a distributed environment.

a) Avoidance.
b) Prevention.
c) Detection.
In deadlock avoidance approach to distributed systems, a resource is granted to a process if the resulting global system state is safe (note that a global state includes all the processes and resources of the distributed system). However, due to several problems, deadlock avoidance is impractical in distributed systems.

According to (Atul Kahate, 2004), he proposed that, after a lot of research DDBMS designers have concluded that they cannot avoid a deadlock in a distributed environment. He gave two main reasons for this:

a) In order to avoid deadlocks completely, every site in the distributed system needs to be aware of the global state (that is information about all other sites). This is clearly impossible because the states of the sites would keep changing constantly, even if they communicate these changes to each other, there would be inevitable delays, making the information obsolete. As such this requirement of global state would never be met.

b) Maintaining a global state, if at all, would entail tremendous amount of overheads in terms of network communication, processing and storage overheads.

The thought is contradicted by (Paul Helman, 2004), who proposed that in their purest form, avoidance strategies are based on protocols that further restrict the way transactions may request locks. If transactions obey this protocol, deadlock is impossible and no additional mechanisms are required to detect and resolve deadlocks. These strategies are considered to be pessimistic in the sense that they assume deadlock will occur unless actions are taken in advance that will certainly avoid it from occurring.
The advantage of avoidance strategy is that transactions are not undone and re-executed due to deadlock and hence computing resources are not wasted on these processes. On the other hand, the protocols have the disadvantage that they may increase the time it takes a transaction to complete because the protocol forces the transaction to lock more resources than it needs and to wait simply because giving it access to some resources, even if the resource is available at that time of request could possibly lead to feature deadlocks. In summary it is difficult as it needs global state information in each site (that handles resources)

The Banker’s algorithm is one known algorithm used for deadlock avoidance. It was developed by (Dijkstra Edsger, 1968). Two other algorithms for avoidance are Wait-Die and Wound-Wait (D. Rosenkrantz, R. Stearns and P. Lewis, 1987.) algorithms. These are not discussed in detail in this research.

2.5.2 Deadlock Prevention

Deadlock prevention is commonly achieved either by having a process acquire all the needed resources simultaneously before it begins executing or by pre-empting a process which holds the needed resource. This approach is highly inefficient and impractical in distributed systems.
Paul Helman (2004) does not mention anything on the prevention strategies for distributed deadlocks. However, Atul Kahate (2004) considers the prevention strategy. In this approach, the technique of timestamps is used. Each transaction comes with its own timestamp. In his theory, an older transaction is restarted with its original timestamp value that allows it to retain its older priority which would be higher than most transactions when it restarts. Also, there is immediate access to the request of an older transaction by killing newer transaction. This means that an older transaction never has to wait for a younger transaction.

(Silberschatz, Galvin and Gagne, 1988) proposed their ideas for the distributed deadlock prevention algorithms. They proposed that one protocol requires each process to request and be allocated resources before it begins execution. See Algorithm 1 (One-Shot algorithm) supports what they proposed. The One Shot Algorithm is adapted from (R. A. Finkel, 1988).
Given a request from process P for resources R1, R2, ..., Rn, the resource manager follows these rules:

if process P has ever acquired any resources before, then
    refuse the request
else if any resource R1, ... Rn does not exist then
    refuse the request
else
    { if any resource R1, ... Rn is not free, then
        wait until all resources R1, ... Rn are free
        grant process P exclusive access to resources R1, ... Rn
    end if
} end if

Analysis of One-Shot Algorithm

The above algorithm has the advantage that it tries to prevent a deadlock situation from occurring but it has the disadvantage that if one of the n resources is not available or does not exist, the process P cannot execute since all resources have to be present for P to execute. This causes major overheads because as long as one resource is missing no execution takes place. Also this algorithm might enter into a continuous loop that is an infinite loop where by all the n resources are never available all at once which will be a form of deadlock which would have occurred ultimately. Message complexity for the algorithm is (n+1+m)/2 and time complexity is O (2n^2) a polynomial function.
Silberschatz, Galvin and Gagne, 1988) also proposed another algorithm; an alternative protocol calls a process to request resources only when the process has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated. This is illustrated by the following Algorithm 2.

**Algorithm 2: Repeated One-Shot (MultiShot)**

Given a request from process P for resources R1, R2, ..., Rn, the resource manager follows a similar rule to that for one-shot.

```plaintext
if process P currently has any resources, then
    refuse the request
else if any resource R1, ... Rn, does not exist, then
    refuse the request
else
{
    if any resource R1, ... Rn is not free, then
        wait until all resources R1, ... Rn are free
    end if
    grant process P exclusive access to resources R1, ... Rn
}
end if
```

If a process P wants to request resources while holding resources, they follow these steps:

- P frees all resources being held
- P requests all resources previously held plus the new resources it wants to acquire
Although the algorithm tries to address the issue of resource allocation in a way to prevent deadlocks, the algorithm tries to improve the one-shot algorithm but does not do it completely. It has the drawback that once the process P releases the already acquired resources, when trying to re-acquire them, they might have been already requested by other processes resulting in P having to even wait for longer periods before it can have all the necessary resources. Message complexity \((2n+m)/2\) and time complexity \(O(n^2+n)\) better performance than one shot algorithm.

(Silberschatz, Galvin and Gagne, 1988) proposed the hierarchical algorithm. Given a request from process P for the resource R the resource manager follows these rules:

**Algorithm 3: Hierarchical**

Assume the resources have unique priorities (i.e., all priorities are different). Given a request from process P for resource R, the resource manager follows these rules:

- If process P currently has any resources with equal or higher priority than resources R, then refuse the request.
- Else if resource R1 does not exist, then refuse the request.
- Else
  - If the resource R is not free, then wait until resource R is free.
  - Grant process P exclusive access to resources R.

Analysis of Hierarchical Algorithm

The hierarchical algorithm makes use of priority to allocate resources. It has that advantage that a process with higher priority will never wait for a process with a lower priority. The disadvantage of this algorithm is that processes with very low priorities might never execute since priority is always given preference to a process with a higher authority. Message complexity is $m(n-1)/2$ and time complexity $O(2^n-2)$.

2.5.3 Deadlock Detection

Deadlock detection requires examination of the status of process-resource interactions for presence of cyclic wait. Deadlock detection in distributed systems seems to be the best approach to handle deadlocks in distributed systems. Deadlock Detection Focuses on finding cycles. Deadlock detection techniques must address two basic issues: First no detection of existing and secondly resolution of existing deadlocks in finite time. The principal problem is building a correct deadlock detection algorithm that searches the WFG for the presence of cycles and break the existing cycles.

According to (Atul Kahate, 2004), in the process of detecting deadlock, transactions are allowed to obtain access to shared resources. If a deadlock occurs because of this, it is detected. In such a situation one of the transactions must release its share of resources. The trouble in implementing this scheme in the case of a DDBMS is that every site has knowledge of the local transaction only. It cannot know about other sites and their transactions that is the global state. Therefore, it cannot help in deadlock detection. Consequently some sort of
Detect deadlocks in a DDBMS. Three solutions are suggested to handle this situation namely Centralised control, Distributed control and Hierarchical control.

2.5.3.1 Centralised Control

In this approach one site is designated as the control site, and it decides how to detect and come out of the deadlocks. A control site constructs wait-for graphs (WFGs) and checks for directed cycles. WFG can be maintained continuously (or) built on-demand by requesting WFGs from individual sites.

Wait-For-Graph (WFG) Explained

According to (Ajay Kshemkalyani and Mukesh Singhal, 2009). A process can be in two states: running or blocked. In the running state (also called active state), a process has all the needed resources and is either executing or is ready for execution. In the blocked state, a process is waiting to acquire some resource. The two further described how a deadlock will occur using graphs as follows:

a) The state of the system can be modelled by directed graph, called a wait for graph (WFG).
   In a WFG, nodes are processes and there is a directed edge from node P1 to mode P2 if P1 is blocked and is waiting for P2 to release some resource.

b) A system is deadlocked if and only if there exists a directed cycle or knot in the WFG.
   Figure 5 shows a WFG, where process P11 of site 1 has an edge to process P21 of site 1 and P32 of site 2 is waiting for a resource which is currently held by process P21.

c) At the same time process P32 is waiting on process P33 to release a resource.
If P21 is waiting on process P11, then processes P11, P32 and P21 form a cycle and all the processes are involved in a deadlock depending upon the request model.

![Wait For Graph Example](image)

**Figure 5: Wait For Graph Example**

2.5.3.2 **Distributed control**

This is a democratic approach, where all sites are treated as equal. All the sites need to cooperate with each other. There is a big amount of information exchange, resulting in substantial overheads. WFG is spread over different sites. Any site can initiate the deadlock detection process.
Distributed algorithms can be categorized into the following classes:

a) Path-pushing: resource dependency information disseminated through designated paths (in the graph).

b) Edge-chasing: special messages or probes circulated along edges of WFG. Deadlock exists if the probe is received back by the initiator.

c) Diffusion computation: queries on status sent to process in WFG.

d) Global state detection: get a snapshot of the distributed system. Not discussed further in class.

In (Chandy-Misra-_Haas Algorithm, 1983), processes are allowed to request multiple resources at once this speeds up the growing phase. A consequence of this change in a process may wait on two or more resources at the same time. When a process has to wait for some resources a probe message is generated and sent to the process holding the resources. The message consists of three numbers:

- The process being blocked
- The process sending the message
- The process receiving the message

When the message arrived, the recipient checks to see if it itself is waiting for any process. If so, the message is updated and the second and third fields are replaced by the corresponding process number. The message is then send to the process holding the needed resource. If the message goes all the way round and comes back to the original sender, that is, the process that initiated the probe, a cycle exists and the system is deadlocked as illustrated by Figure 6.
The algorithm uses a special message called probe, which is a triplet (i, j, k), denoting that it belongs to a deadlock detection initiated for process Pi and it is being sent by the home site of process Pj to the home site of processPk. A probe message (i, j, k) travels along the edges of the global Wait-For-Graph (WFG), and a deadlock is detected when a probe message returns to the process that initiated it. A process Pj is said to be dependent on another process Pk if there exists a sequence of processes Pj, Pi1, Pi2, ..., Pim, Pk such that each process except Pk in the sequence is blocked and each process, except the Pj, holds a resource for which the previous process in the sequence is waiting. Process Pj is said to be locally dependent upon process Pk if Pj is dependent upon Pk and both the processes are on the same site. Each process Pi maintains a Boolean array, dependent(i), where dependent(i,j) is true only if Pi knows that Pj is dependent on it. Initially, dependent(i,j) is false for all i and j.
The following algorithm determines if a blocked process is deadlocked:

**Sending the probe:**

if $P_i$ is locally dependent on itself then declare a deadlock
else for all $P_j$ and $P_k$ send a probe $(i, j, k)$ to the site $P_k$
if

a) $P_i$ is locally dependent upon $P_j$, and
b) $P_j$ is waiting on $P_k$, and
c) $P_j$ and $P_k$ are on different site

**Receiving the probe:**

On the receipt of a probe $(i, j, k)$, the site $(P_k)$ takes the following actions:
if

a) $P_k$ is blocked, and
b) $\text{Dependent}(k,i)$ is false, and

then $P_k$ has not replied to all requests $P_j$

Then begin

Set $\text{dependent}(k,i) = \text{true}$;
if $k=i$
then declare that $P_i$ is deadlocked
else
send a probe $(i,m,n)$ to the site of $P_n$
if for all $P_m$ and $P_n$ such that

a) $P_k$ is locally dependent upon $P_m$, and
b) $P_m$ is waiting on $P_n$, and

c) $P_m$ and $P_n$ are on different site

end.
A probe message is continuously circulated along the edges of the global WFG graph and a deadlock is detected when a probe message returns to its initiating process.

**Analysis of Chandy-Misra-Haas Algorithm**

- One probe message (per deadlock detection initiation) is sent on every edge of the WFG. Thus, the algorithm exchanges at most \( m (n - 1)/2 \) messages to detect a deadlock that involves \( m \) processes and that spans over \( n \) sites.
- The size of messages is fixed and is very small (only 3 integer words).
- Delay in detecting a deadlock is \( O(n) \).

**Terms Used**

A probe \((i,j,k)\) is used by a deadlock detection process \( P_i \). This probe is sent by the home site of \( P_j \) to \( P_k \). This probe message is circulated via the edges of the graph. Probe returning to \( P_i \) implies deadlock detection. \( P_j \) is dependent on \( P_k \) if a sequence of \( P_j, P_i, .., P_m, P_k \) exist. \( P_j \) is locally dependent on \( P_k \) if above condition plus \( P_j, P_k \) on same site. Each process maintains an array \( \text{dependent}(i): \text{dependent}(i,j) \) is true. If \( P_i \) know that \( P_j \) is dependent on it (initially set to false for all \( i \) and \( j \)).
Hierarchical Control

This creates a tree-like structure. Here the site designated as parent detects deadlocks of its subordinate sites and takes appropriate decision. Figure 7 shows the hierarchical deadlock detection diagram. Sites are arranged in a hierarchy. A site checks for cycles only in descendents.

![Hierarchical Deadlock detection diagram](image)

**Figure 7: Hierarchical Deadlock detection**
Accordingly, the decision of choosing one of them is left to the actual problem specifications, which differ from one situation to another.

The advantages of this algorithm over distributed deadlock detection algorithms include

- Transmission of only potential multisite deadlock cycle information.
- Transmission in only a single direction along the path of the potential deadlock cycle.
- A given site processes only its local wait-for graphs and the specific nodes and edges it receives from other sites.
- It is not dependent on some specialised sites that must be in communication with all other sites, and is therefore less prone to failure.
There are several approaches to break deadlocks. Some of the approaches wait for a deadlock to happen then resolve the deadlock later. Whilst this approach is reasonable, however this might be a problem in a really busy environment because most of the resources like computing time and power would be mostly used in resolving these deadlocks. An efficient approach seeks to break deadlock(s) in a finite reasonable time using minimum resources.

The three algorithms (One-Shot, Multi-Shot and Hierarchical) are prevention algorithms. The major benefit with them is that they are easy to implement and that they can easily prevent deadlocks in non complex distributed environments although they are best suited for centralised systems.

However the three algorithms can enter into an infinite loop because the algorithms require the processes to acquire all resources before proceeding of which the problem has a much greater impact in One-Shot algorithm. In the Multi-Shot algorithm some processes will release their resources to other requesting processes but they will have problems in reacquiring them, thus this may lead to processes having to wait long time. For hierarchical processes with higher priorities will never wait for resources meaning that lower priority processes will be starved of resources and probably never executed.
The major benefit of the Chandy algorithm is that it can completely detect deadlocks and its ability to handle multiple requests from processes at a time. However, the method is laborious to handle large graphs. This means that the algorithm has been found that for some cases when the degree of data contention is high, some deadlocks may not be detected. (G.P. de Souza and G.H. Pfitscher, 2002) found out that the algorithm needs a high message complexity to reach its objectives.

In the next chapter a proposed method uses a combination of priority ordering as well as First Come First Served basis to processes.
CHAPTER THREE: METHODOLOGY

3.0 INTRODUCTION

This chapter describes and explains the methodology deployed in this study and at the research methods reading which informed my choice of methods. The researcher conducted a comprehensive investigation into the common methodologies that are being used to handle distributed deadlocks. The researcher used simulation and algorithm analysis as research tools to consider an alternative approach that can be used to handle deadlocks in distributed databases.

Chapter one introduced the subject of this dissertation, i.e. management of distributed databases. I am interested in finding an efficient resource allocation algorithm within the time, space and resources available.
Simulation is the design and running of experiments so as to provide insights for decision making and selection of appropriate course of action using a model. The model is an idealised representation of an integral part of an everyday life for example in this study distributed databases.

Simulation was used in this study because of some obvious reasons namely:

a) Simulation helps in making use of alternative and cheaper methods. To implement the algorithm on a live distributed database require additional hardware and specialised software which is costly.

b) What IF analysis was used and different scenarios were considered thus varying variables accordingly

c) Real databases were not used to run the actual experiments hence there were no disruptions affected any organisation.

d) Results were produced faster because time was compressed.

e) Distributed deadlocks are complex; simulation simplified the complexity of the problem.

However, it is a complex task to simulate a real world environment because some variables affecting distributed databases may not have been addressed. Simulation deals with assumptions, hence the overall result may not be exact.
Why Simulation for This Research?

Simulation was used because there were limited resources. Live databases proved costly because a lot of hardware and software would be required to setup the distributed databases. Monitoring the deadlocks require more time since these might occur after a long time. Simulation also reduces risks associated with running experiments on real distributed databases with real data. Using simulation gave room for the researcher to vary variables accordingly depending on the scenarios.
The simulator designed illustrates how deadlocks can be addressed using the ISKE algorithm. The algorithm simulates a resource manager. The manager allocates resources to processes according to their requests. The algorithm uses the prevention strategy to address deadlocks.

It considers the resources available and allocates them to the available processes. If there is a process requesting a resource which is already held by another process then the algorithm puts the process in a queue. The algorithm then considers the difference between resources needed and resources allocated to the process. The difference is then used as a priority. If the difference is the greatest it is allocated all the resources it needs. This is done because a process holding more resources is most likely to lead into deadlock if it keeps holding resources. It is therefore important that it releases the resources to minimise the chances of causing a deadlock. A process with a lower priority can be assigned resources ahead of one with higher priority if the difference between Resources Allocated (CPR) and Resources Needed (RPN) is greater (CPR-RPN). After the process with a higher count of resources has released the other processes can safely continue with computation. Below is a pseudo code for the algorithm.
If Process sent request for resources then
   For resources R1 to Rn
      If R_i is needed by a single P then
         If R_i available then
            Assign resource
         End if
      /*check for the number of resources needed by each process
      /*lock resource required by more than two processes and
      /*assign remaining resources to requesting processes
      Else
         If P_{(CPR-RPN)} > any P_{(CPR-RPN)} then
            /*check the resources that each process now needs and
            /*subtract the number of resources it has been given (CPR) from the number of resources it needs (RPN)
               • Assign available resource to P_{(CPR-RPN)} /*greatest difference
               • Assign P greatest priority /*P assigned resources
               • Release resource if P finishes
               • Give P new priority 0 \rightarrow priority = priority + 1 for all other processes (queue)
                  /*reassign priority to new requesting processes
         End if
      End if
   Next R
Endif
3.2.1 Analysis of the ISKE Algorithm

Suppose D1, D2, D3 and D4 are databases each having a processes that are requesting any of the resources R1, R2, R3 and R4. If D1 is currently holding three of the resources (R1, R2 and R3) and is requesting resource R4 from D3. D3 is also requesting R1, R2 and R4 resources held by D1. The algorithm has to avoid a deadlock by weighing resource count (CPR-RPN). D1 only wants 1 resource thus D3 must release R4 to D1 so that D1 completes its work and releases resources. If D1 has many resources held by it that may mean there is a higher probability of it, causing a deadlock as other processes are bound to sooner or later request one of the resources held by D1.

A message complexity of O (n) and time complexity of O (n) can be computed from the algorithm. Therefore theoretically this algorithm can executes faster than the One-Shot or the Multi-Shot algorithms. If there are several processes with great differences (CPR-RPN) this might mean that these processes might starve resources to other processes with less priority.

3.3 CONCLUSION

The ISKE algorithm was implemented using Visual Basic 6. Results from the implementation were stored in an Excel sheet. This chapter carried a description of how the study was carried out. The study will enable future researchers who might want to carry out the research in a different context, focus or orientation to do so. It also makes it possible for other researchers to verify the results of the research project. The next chapter will describe in detail the findings of this research.
4.0 INTRODUCTION

The chapter highlights the results that were obtained from the algorithm that was proposed in chapter 3. The researcher compared the results from the ISKE algorithm with the MULTI shot algorithm.

4.1 FINDINGS

A simulator was designed to illustrate deadlocks in a distributed database environment. The simulator addresses the issue of distributed deadlocks using the proposed ISKE algorithm designed by the researcher. The implementation of the algorithm was done using Visual Basic 6.0. The results were stored in a Microsoft Excel worksheet whose results were illustrated using line graphs and column graphs. The algorithm prevents deadlocks in processes that are assumed to be occurring in a distributed database environment. For the purposes of simplicity, it is assumed that one resource manager (global resource manager) manages the resources for all sites.
Figure 8 shows the menu screen that gives menu options

**Figure 8: Main Screen**

The user input all the resources available in the system, allocates processes to requesting transactions (processes) and then simulates the algorithm of choice on the menu. Each algorithm is given by default a maximum of 20 units of time to allocate resources to requesting processes. To store the results in an Excel sheet the option **Plot** will store the results.
Suppose there are 3 databases D1, D2, and D3 distributed over different sites. The requests are shown in Table 1.

<table>
<thead>
<tr>
<th>Database Number</th>
<th>Process Number</th>
<th>Requested Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>P1</td>
<td>R1, R2</td>
</tr>
<tr>
<td>D2</td>
<td>P2</td>
<td>R1, R2</td>
</tr>
<tr>
<td>D3</td>
<td>P3</td>
<td>R1</td>
</tr>
</tbody>
</table>

Table 1: Sample Deadlock

Figure 9: Sample Results for the ISKE Algorithm
it can be indicated from Figure 9 that all processes have been
waiting time of 4 units see Figure 9.

![Figure 10: Sample Results for the MultiShot Algorithm](image)

After 19 units of time the Multi-Shot algorithm could not release R1 and R2 to process P1 because it was being used by P2 and P3. See Figure 10.
A number of simulation test runs were carried out to compare the efficiency of the ISKE algorithm against the MultiShot algorithm. The test runs considered Best case, Average case and Worst Case situations. Requests are allocated in 5 units of time for all general requests. Random requests from processes were run within 20 units of time. The results were tabulated as shown by the following charts and tables.

### 4.3.1 Test Run 1

In a best case complexity each process requests a resource(s) that are not requested by any other process. The processes requested resources as shown in Table 2.

<table>
<thead>
<tr>
<th>Process Number</th>
<th>Requested Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R1</td>
</tr>
<tr>
<td>P2</td>
<td>R2</td>
</tr>
<tr>
<td>P3</td>
<td>R3</td>
</tr>
<tr>
<td>P4</td>
<td>R4</td>
</tr>
<tr>
<td>P5</td>
<td>R5</td>
</tr>
</tbody>
</table>

*Table 2: Best Case*
Figure 11: Test 1 – ISKE Algorithm

Figure 11 shows that the allocation of resources by the resource manager to requesting processes according to priority of FCFS. There is no waiting time for processes since no two processes are requesting the same resources.

Figure 12: Test 1 - MultiShot Algorithm
Figure 12 shows the test run for the MultiShot algorithm. The algorithm behaves the same as the ISKE algorithm. Figure 11 and Figure 12 confirm that using the two algorithms no deadlocks will occur.

![Bar Chart](image)

**Figure 13: Test 1 – Bar Chart**

Figure 13 shows that in their best cases the two algorithms’ performance is the same for the 5 units of time.
Figure 14: Test 1 Random Requests – Line Chart

Figure 15: Test 1 Random Requests - Bar Chart
requests from processes is random not pre ordered. Figure 14 and Figure 15 locating resources to random requests from processes. The ISKE managed to respond to these requests and clear deadlocks. The gaps in the diagrams show that the Multi-Shot algorithm only managed to allocate resources to few processes in the 20 units of time allocated. For the 20 units of time allocated the ISKE algorithm was able to allocate all requested resources to processes.

4.3.2 Test Run 2

Test 2 will test a scenario in a worst case environment where all resources are requested by all known processes. To test for the worst case complexity each process requested all the resources. The requests are shown in Table 3.

<table>
<thead>
<tr>
<th>Process Number</th>
<th>Requested Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R1, R2, R3, R4, R5</td>
</tr>
<tr>
<td>P2</td>
<td>R1, R2, R3, R4, R5</td>
</tr>
<tr>
<td>P3</td>
<td>R1, R2, R3, R4, R5</td>
</tr>
<tr>
<td>P4</td>
<td>R1, R2, R3, R4, R5</td>
</tr>
<tr>
<td>P5</td>
<td>R1, R2, R3, R4, R5</td>
</tr>
</tbody>
</table>

Table 3: Worst Case
Figure 16: Test 2 - Line Chart

Figure 17: Test 2 - Bar Chart
Since all processes are requesting the same resources. The allocation of resources is simply by priority FCFS. The ISKE algorithm therefore manages deadlocks very well. The MultiShot algorithm managed to allocate resources to some requesting processes but failed to allocate resources to process P2. In the 5 units allocated time the ISKE managed to allocate resources to all requests but the Multi-Shot could not allocate resources to P2. See Figures 16 and 17.

![Test 2 Random Requests - Line Chart](image)

**Figure 18: Test 2 Random Requests - Line Chart**
ISKE algorithm managed to prevent deadlocks from requesting processes. The linear complexity can be easily shown from Figure 18 and Figure 19. The absence of red line or red column does not indicate that the MultiShot algorithm was not tested, instead test runs were carried out on the MultiShot algorithm but due to the complexity of the requests my conclusion is that the algorithm was unable to allocate resources to processes.

**Figure 19: Test 2 Random Requests - Bar Chart**
Test Run 3 tested another complex scenario where every process requested at least one resource and at least one process requested all the given resources and the requests are shown in Table 4.

<table>
<thead>
<tr>
<th>Process Number</th>
<th>Requested Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R1</td>
</tr>
<tr>
<td>P2</td>
<td>R1, R2</td>
</tr>
<tr>
<td>P3</td>
<td>R1, R2, R3</td>
</tr>
<tr>
<td>P4</td>
<td>R1, R2, R3, R4</td>
</tr>
<tr>
<td>P5</td>
<td>R1, R2, R3, R4, R5</td>
</tr>
</tbody>
</table>

Table 4: Test 3

![Figure 20: Test 3 - Line Chart](image)
The two Figures (20 and 21) show that although the ISKE algorithm took a bit of time in responding to requests it managed to allocate the resources. The ISKE used the difference method in allocating the resources. The MultiShot algorithm could not manage the deadlock hence it only allocated resources to P5.
Figure 22: Random Requests - Line Chart
Figure 23: Test 3 Random Requests - Bar Chart

For Random requests the MultiShot could not manage any of the requests. The ISKE managed to allocate resources to all random requests. See Figures 22 and 23.
Another extreme case is when all processes are sharing only one resource. The requests are shown in Table 5.

<table>
<thead>
<tr>
<th>Process Number</th>
<th>Requested Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R1</td>
</tr>
<tr>
<td>P2</td>
<td>R1</td>
</tr>
<tr>
<td>P3</td>
<td>R1</td>
</tr>
<tr>
<td>P4</td>
<td>R1</td>
</tr>
<tr>
<td>P5</td>
<td>R1</td>
</tr>
</tbody>
</table>

Table 5: Test 4

![Figure 24: Test 4 - Line Chart](image_url)
The ISKE algorithm allocated resources according to priority as shown in Figure 24 and Figure 25. The MultiShot however could not handle the deadlock it only managed to allocate resources to other processes except for process P2.
Figure 26: Test 4 Random Requests - Line Chart
The ISKE algorithm allocated resources according to priority as shown in Figure 26 and Figure 27. The MultiShot however could not handle the deadlock it only managed to allocate resources to other processes except for process P2.

Figure 27: Test 4 Random Requests - Bar Chart
To test for average complexity, half of the resources are requested by the processes. The requests are shown in Table 6.

<table>
<thead>
<tr>
<th>Process Number</th>
<th>Requested Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R1</td>
</tr>
<tr>
<td>P2</td>
<td>R2</td>
</tr>
<tr>
<td>P3</td>
<td>R3</td>
</tr>
<tr>
<td>P4</td>
<td>R1</td>
</tr>
<tr>
<td>P5</td>
<td>R2</td>
</tr>
</tbody>
</table>

Table 6: Test 5

![Figure 28: Test 5 - Line Chart](image-url)
Figure 29: Test 5 - Bar Chart

P3 (ISKE) is allocated resources first because it is not sharing any resources. The ISKE has maintained a linear approach. The MultiShot can handle the deadlocks as well as the ISKE. See Figures 28 and 29.

Figure 30: Test 5 Random Requests - Line Chart
When the requests from processes have been randomised the ISKE allocates resources as requested by processes in 20 units of time. The Multi-Shot handled a few requests and cannot handle random requests effectively. See Figures 30 and 31.

4.3.6 Test Run 6

Also considering an average case complexity. Some processes have requested resources from all the resources available. The requests are allocated in Table 7.

<table>
<thead>
<tr>
<th>Process Number</th>
<th>Requested Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R1, R4</td>
</tr>
<tr>
<td>P2</td>
<td>R2, R5</td>
</tr>
<tr>
<td>P3</td>
<td>R3</td>
</tr>
</tbody>
</table>

Table 7: Test 6
The two algorithms behave the same (linear approach) because resources are plenty. See Figures 32 and 33.
All requesting processes have been allocated resources in reasonably time for the ISKE algorithm. MultiShot cannot manage random requests effectively. The MultiShot managed to
The ISKE handled the random requests well. See Figures 34 and 35.

### 4.4 SUMMARY OF RESULTS

I will summarise only random requests from processes because that is the real world phenomenon. From the six tests that were conducted in 20 units of time the summary is as follows shown in Table 8.

<table>
<thead>
<tr>
<th>Test Run Number</th>
<th>ISKE</th>
<th>MultiShot</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8: Summary of Results
4.4 CONCLUSION- ANALYSIS OF RESULTS

From the test runs it can be established that the ISKE algorithm is effective in preventing deadlocks from occurring (see Table 8 and Figure 36). The algorithm also managed deadlocks from any angle demonstrated by the different types of test runs. The ISKE algorithm is more effective than the MultiShot because it can handle more complex requests from processes. The MultiShot took long to respond or when it responded it could not fulfil all the requests. The ISKE algorithm could handle random requests from processes as is the case in the real world. From the given 20 units of time the ISKE algorithm managed to allocate resources to requesting processes as they come. The ISKE algorithm allocated resources at every unit time.
5.0 INTRODUCTION

Chapter five ends the study. A summary of the major findings from chapter four and recommendations are made. The Findings and results can be compared with objectives at stated in chapter one. This chapter also outlines areas of future work related to this research.

5.1 CONCLUSION

The research’s main outcome was to design and assess an effective algorithm that will resolve distributed deadlocks in reasonable time and using minimum resources. The research paper explored and analysed various distributed algorithms that have been designed to manage deadlocks.

Chapter one presented the problem, the research aim and objectives and other issues in chapter one. Chapter two highlighted the problem and explored what other researchers in the same field have done. Chapter three presented the methodology used which was mainly simulation and algorithm analysis. Finally chapter four presented the research findings and analysed them.
The major research conclusions were:

a) The ISKE algorithm is effective in preventing deadlocks from occurring it can be used with large institutions like the University of Zimbabwe for their database systems.

b) The ISKE algorithm was assessed against the MultiShot algorithm in chapter four. (see Table 8) The ISKE proved to be more efficient as it managed to break deadlocks in reasonable time.

c) The ISKE algorithm managed to handle all the random requests for resources from processes (transactions). From the test runs it was observed that the ISKE algorithm cleared deadlocks from random requests. Most times the MultiShot could not clear or satisfy random requests from processes and took a long time to respond to these requests.

d) The ISKE algorithm is presented in chapter four.
Based on the conclusions and findings from chapter four it can be recommended that:

a) The University of Zimbabwe and other similar institutions change their centralised database system and adopt a distributed approach to reduce problems that are associated with centralised approach.

b) Distributed systems have problems with concurrency control of which managing distributed deadlocks is part of them, however, the ISKE approach can be customised and improve the performance of distributed database systems.

c) Since the experiments were done using a simulator, it is recommended they be implemented in a real live distributed environment so as to validate the results from simulation, and also validate the complexities of the ISKE algorithm.

d) It is also recommended that more tests runs be carried out against other algorithms such as the Chandy–Misra-Haas, Hierarchical.

e) It is recommended that more investigations should be carried out to support the research as future work which would involve weighing the option of considering the different conditions that could be set as variables.
The recommendations pointed out that more work is needed to assess the ISKE algorithm that is it needs to test thoroughly against other major distributed deadlock algorithms. Future work should also focus on testing on the accuracy and completeness of the ISKE algorithm.

Mobile databases are an extension of distributed database systems. The mobile stations move in different speeds. Mobile agent systems have unique properties and characteristics and represent a new application development paradigm. Existing solutions to distributed computing problems such as deadlock avoidance algorithms are not suited to environments where both clients and servers move freely through the network. For future work, the researcher can look into the area of mobile agent systems in the area of Distributed systems and Grid Computing as further research.
6.0 REFERENCES


15. K. Mani Chandy, Jayadev Misra and Laura M. Haas. 1983 Distributed Deadlock Detection. ACM 0734-2071/83/0500-0144


