

Investigating Distributed Database Deadlock Based on Attribute Level

البحث في جمود قواعد البيانات الموزعة اعتمادا على مستوى الحقل

Prepared by
Khaled Saleh Salah Maabreh

Supervisor
Prof. Dr. Ala'a Al Hamami

**A Dissertation Submitted in Partial
Fulfillment of the Requirements for the
Degree of Doctor of Philosophy in Computer
Science.**

**Graduate College of Computer Studies
Amman Arab University for Graduate
Studies**

July, 2008

AUTHORIZATION

I, Khaled S. Maabreh, authorize Amman Arab University for Graduate Studies to reproduce this dissertation in whole or part for purposes of research.

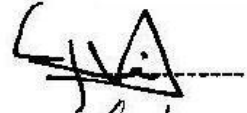
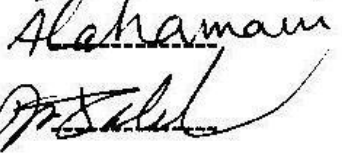

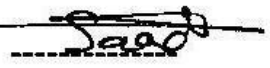
Name: Khaled Saleh Salah Maabreh

Signature: 

Date: 1/9/2008

DISSERTATION COMMITTEE

This dissertation, with the title "Investigating Distributed Database Deadlock Based on Attribute Level", was defended and approved on: 10/9/2008.

<u>Committee Members</u>	<u>Title</u>	<u>Signature</u>
Prof. Dr. Fawaz AlZaghoul	Chair	
Prof. Dr. Alaa Alhammami	Member and Supervisor	
Ass. Prof. Dr. Jalal Atoum	Member	
Ass. Prof. Dr. Sa'ad Al-A'ani	Member	

ACKNOWLEDGMENT

"All praises and thanks to ALLAH"

I would like to thank my supervisor Prof. Dr. Alaa Al-Hamami who provided me with full support, encouragement, and guidance in order to get this dissertation ready. Without his help and support, this work would not have been possible. He was always available any time I needed help.

My sincere thanks go to the Dean of Graduate College of Computer Studies, all of the lecturers, administration, and the staff of Amman Arab University for Graduate Studies.

I also thank Prof. Dr. Mahmoud El-Najar and Dr. Khaled Al-Akhras for their help in refining this dissertation, and thanks to Eng. Abdullah Odat for his consultation to prepare and implement the simulation project.

DEDICATION

I dedicate this work to:

my father, mother and brother

my wife and our daughters Rafeef, Retaj, Jana and Yagoot and

Al Mujahedeen for the sake of ALLAH

.

Table of Contents

AUTHORIZATION	II
DISSERTATION COMMITTEE.....	III
ACKNOWLEDGMENT	IV
DEDICATION	V
.Table of Contents.....	V
List of Tables.....	VIII
List of Figures.....	X
Abbreviations	XV
Abstract.....	XVII
Arabic Summary.....	XIX
CHAPTER ONE INTRODUCTION	1
1.1. Overview.....	1
1.2. Distributed Database	4
1.3.The Problem Statement	5
1.4. Dissertation Questions	7
1.5. Dissertation Methodology.....	8
1.6 .Rationale of the Study.....	8
1.7. Goals of this Dissertation	14
1.8. Dissertation Contribution.....	14
1.9 Dissertation Structure	16
1.10 Conclusion.....	18
CHAPTER TWO LITERATURE REVIEW	19
2.1 Introduction	19
2.2 Handling Concurrency Control in a Database by Locking Techniques	20
2.3 Review of Locking in Different Database Systems	28
2.4 Review of Deadlock Handling in Distributed Database Systems	29
2.5 Handling Deadlocks in Some Well Known Databases.....	34
2.6 Locking Performance in a Database	36
2.7.Conclusion	41
CHAPTER THREE THE PROPOSED METHODOLOGY	43
3.1 Introduction	44
3.2 Hierarchy Tree.....	46

3.3 Database Lock Manager	49
3.4 Proof of Concept by Simulation	51
3.4.1 Building the Simulation	52
3.5 Deadlock Detection Approach.....	56
3.6 The Enhanced Algorithm Description for Locking Attributes.....	58
3.7 Conclusion.....	60
CHAPTER FOUR CENTRALIZED DATABASE RESULTS	61
4.1 Locking Performance	61
4.2 Simulation Runs at Row Level Locking	62
4.2.1 Performance Analysis	71
4.3 Simulation Runs at Field Level Locking.....	78
4.3.1 Performance Analysis	88
4.4 Comparing the Two Alternatives	96
4.5 Conclusion.....	99
CHAPTER FIVE DISTRIBUTED DATABASE RESULTS.....	101
5.1 Distributed Database Population.....	102
5.2 System Behavior at Row Level Locking	106
5.3 System Behavior at Field Level Locking.....	116
5.4 Comparing the Two Alternatives	124
5.5 The Effects of System Parameters.....	127
5.5.1 Operation mode	127
5.5.2 Number of sites.....	129
5.5.3 Degree of replication	131
5.6 Conclusion	131
CHAPTER SIX CONCLUSIONS AND RECOMMENDATIONS	133
6.1 Introduction	133
6.2 Field Level Locking on Centralized Database	134
6.3 Field Level Locking on Distributed Database	135
6.4 Future Works	136
REFERENCES.....	138
Appendices	144

List of Tables

TABLE	Compatibility matrix for granularity hierarchy	9
1.1 :	
TABLE	Gray et al, Compatibility matrix	19
2.1 :	
TABLE	Simulation parameters	41
3.1 :	
TABLE	Results after simulation run at row-level-locking	48
4.1 :	
TABLE	Results of 30 runs of simulation	55
4.2 :	
TABLE	Data Contention workload at Database Size	58
4.3 :	51,152	
TABLE	Results after simulation run at field-level-	60
4.4 :	locking	
TABLE	Building information table	63
4.5 :	
TABLE	Material schedule table	63
4.6 :	
TABLE	Results of 30 runs of simulation at field level	67
4.7 :	locking	
TABLE	Data Contention workload at Database Size	70
4.8 :	248,256	
TABLE	Row level locking versus field level locking	72
4.9 :	performance	
TABLE	Simulation parameters for distributed database	79
5.1 :	
TABLE	Distributing database objects into 15 tables	79
5.2 :	
TABLE	Distributing of 15 tables across three sites	80
5.3 :	
TABLE	Results of 20 runs of simulation at row level	81
5.4 :	locking	
TABLE	Sample results of 170 transactions at row-level-	84
5.5 :	locking	
TABLE	Execution behavior of deadlocked and blocked	86
5.6 :	transactions at row level	

TABLE 5.7 :	Results of 20 runs of simulation at field level locking	87
TABLE 5.8 :	Sample results of 170 transactions at field-level-locking	90
TABLE 5.9 :	Execution behavior of deadlocked and blocked transactions at field level	91
TABLE 5.10:	Row level locking versus field level locking performance	92
TABLE 5.11:	Transaction classified according to the operation mode	95
TABLE 5.12:	Transaction classified according to the number of sites used	96

List of Figures

FIGUR	Graphical representation for deadlock	4
E 1.1 :		
FIGUR	Granularity hierarchy	8
E 1.2 :	
FIGUR	Proposed granularity hierarchy	1
E 1.2 :	2
FIGUR	Overview of the proposed system	3
E 3.1 :	4
FIGUR	Database hierarchies at fields level locking	3
E 3.2 :	6
FIGUR	Data structure of lock manager	3
E 3.3 :	8
FIGUR	Simulation model overview	4
E 3.4 :	0
FIGUR	Execution behavior of transaction number 26 at	5
E 4.1 :	row level locking	1
FIGUR	Execution behavior of transaction number 38 at	5
E 4.2 :	row level locking	2
FIGUR	Execution behavior of transaction number 35 at	5
E 4.3 :	row level locking	2
FIGUR	Execution behavior of transaction number 81 at	5
E 4.4 :	row level locking	3

FIGURE 4.5 :	Execution behavior of transaction number 79 at row level locking	53
FIGURE 4.6 :	System Throughput at row level locking	57
FIGURE 4.7 :	System performance at row level locking	57
FIGURE 4.8 :	System locking overhead at row level locking	59
FIGURE 4.9 :	Execution behavior of transaction number 26 at field level locking	64
FIGURE 4.10 :	Execution behavior of transaction number 38 at field level locking	65
FIGURE 4.11 :	Execution behavior of transaction number 35 at field level locking	65
FIGURE 4.12 :	Execution behavior of transaction number 81 at field level locking	66
FIGURE 4.13 :	Execution behavior of transaction number 79 at field level locking	66

FIGURE 4.14 :	System throughput at field level locking	69
.....		
FIGURE 4.15 :	System performance at field level locking	69
.....		
FIGURE 4.16 :	System locking overhead at field level locking	70
.....		
FIGURE 4.17 :	Throughput for the two alternatives	73
.....		
FIGURE 4.18 :	Mean service time for the two alternatives	73
.....		
FIGURE 4.19 :	Mean waiting time for the two alternatives	73
.....		
FIGURE 4.20 :	Locking overhead for the two alternatives	74
.....		
FIGURE 5.1 :	Distributed database architecture for three sites	78
.....		
FIGURE 5.2 :	System Throughput at row level locking	82
.....		

FIGU	Mean service time at row level locking	8
RE 5.3		2
:	
FIGU	Mean waiting time at row level locking	8
RE 5.4		3
:	
FIGU	System locking overhead at row level locking	8
RE 5.5		3
:	
FIGU	Part of dependency graph for deadlocked and	8
RE 5.6		6
:	blocked transactions at row level	
	
	
FIGU	System throughput at field level locking	8
RE 5.7		8
:	
FIGU	Mean service time at field level locking	8
RE 5.8		8
:	
FIGU	Mean waiting time at field level locking	8
RE 5.9		9
:	
FIGU	System locking overhead at row level locking	8
RE		9
5.10 :	

FIGURE 5.11 :	Part of dependency graph for transactions 2,18,25,33,38 at field level	92
FIGURE 5.12:	Throughput for the two alternatives	93
FIGURE 5.13:	Mean service time for the two alternatives	93
FIGURE 5.14:	Mean waiting time for the two alternatives	94
FIGURE 5.15:	Locking overhead for the two alternatives	94
FIGURE 5.16:	The effects of operation mode	96
FIGURE 5.17:	The effects of the number of sites used	97

Abbreviations

CPU	Central Processing Unit
D	Database size (lockable units)
DB	Database
DBMS	Database Management System
DC	Data Contention
DDBMS	Distributed Data Base Management System
DML	Data Manipulation Language
F	File (or database table)
FCFS	First Come First Served
ID	Identification
IS	Intention Shared
IX	Intention Exclusive
K	Number of locks
LCB	Lock Control Block
Ms	Millisecond
N	Number of transitions (or Number of users)
NL	NiL (No locks)
S	Shared
SIX	Shared with Intent Exclusive

SQL	Structured Query Language
T	Transaction
X	Exclusive

Abstract
(PHD)

**Investigating Distributed Database Deadlock Based on
Attribute Level**

Prepared by
Khaled Saleh Salah Maabreh

Supervisor
Prof. Dr. Ala'a Al Hamami

The use of a database is increasing day by day; it has become the core of most applications. Also the number of users is growing in an unexpected way, and the information must be available in an efficient and reliable way to satisfy user requirements and to cover the increasing needs. Because the database has the ability to work in a multi user environment, there must be a technique to preserve the data contained in the databases. Locking database items are the most popular. Distributed databases may contain huge data, and several hundreds or even several thousands users over the connected sites.

This research suggests a method to increase the availability of data, by reducing the size of lockable entities. This can be done by increasing the granularity hierarchy tree one more level down at the attributes to allow several transactions to access the same row

simultaneously. A simulation program was implemented to show and compare the system behavior at the field level locking approach. The experimental results proved that using attribute level locking will decrease the competition for acquiring the data and increases the concurrency control for the Database (Centralized or Distributed). Also the suggested level increases the database performance and decreases deadlock occurrences.

The discussion presented shows that the system at field level locking behaves much better than at row level locking in both environments (centralized and distributed), because multiple transactions can process the same database row simultaneously, which decreases the mean service time as well as the mean waiting time. At the same time, more transaction executes on field level than row level locking before the systems begins thrashing, which means that the field level locking works better on a heavy work load than systems at row level locking.

Key words: Centralized Database, Distributed Database, Locking, Attribute Level, Database Deadlocks, Concurrency Control, and Performance.

Arabic Summary

البحث في جمود قواعد البيانات الموزعة اعتماداً على مستوى الحقل

إعداد

خالد صالح صلاح معايرة

إشراف الأستاذ الدكتور

علاء الحمامي

الملخص (دكتوراه)

إن استعمال قواعد البيانات يزداد يوماً بعد يوم، حيث أنها أصبحت لبنة الأساس لمعظم الأنظمة، بالإضافة إلى تزايد أعداد المستخدمين وبشكل غير متوقع، مما استدعى إلى توفير المعلومات بشكل مستمر وموثوق لتلبية تلك الحاجات. لأن أهم ما يميز قواعد البيانات هو استعمالها المتاح لأكثر من مستخدم في آن واحد، مما استدعى وضع تقنيات خاصة للحفاظ على محتويات هذه القواعد من التلف أو الضياع نتيجة الاستعمال المتزامن لمحتوياتها، وتعدّ سياسة قفل البيانات من السياسات الأكثر انتشاراً وتعمل هذه السياسة على حمايتها ومنع تعديلها في الوقت نفسه، خصوصاً في بيئة نحتاج فيها إلى قواعد البيانات الموزعة والتي تنتشر وتربط من خلال شبكات المعلومات، وتتميز باحتوائها على بيانات ضخمة متاحة لعدد كبير من المستخدمين.

جاءت هذه الدراسة باقتراح زيادة فاعلية البيانات والعمل على إتاحتها على نحوٍ موثوق من خلال العمل على تخفيض حجم المحجوزة منها، ويتم ذلك بزيادة الشجرة التي تمثل قاعدة البيانات ليكون الحجز أو القفل على مستوى الحقل عوضاً عن السجل الذي يمثل مجموعة من الحقول، وذلك لإتاحة الفرصة لأكثر من معاملة (مستخدم) لاستعمال السجل نفسه في ذات الوقت. فتم بناء برنامج محاكاة لمراقبة ومقارنة سلوك النظام في حالة الحجز على مستوى الحقل، وقد تبين من خلال النتائج التجريبية، أن استعمال القفل على مستوى الحقل سينقّص المنافسة في الحصول على البيانات ويزيد توفرها، كما ويزيد الفرصة لعدد أكبر من المستخدمين لاستعمالها في الوقت نفسه، بالإضافة إلى أنه يعمل على التقليل من احتمالية حدوث الجمود. كما تبين من خلال المناقشة أن النظام الذي يستخدم الحقل في حجز البيانات كان أكثر فاعلية من النظام الذي يستخدم السجل، لأنه يمكن لمعاملات متعددة أن تُعالج السجل نفسه بشكل متزامن.

CHAPTER ONE

INTRODUCTION

1.1. Overview

Database is a collection of data that contains information relevant to an enterprise [34]. The use of a database is increasing day by day; it has become the core of most applications, including web based applications. Also the number of users is growing in an unexpected way due to the need for information in many situations like decision making or even in daily routine applications. Information must be available in an efficient and reliable way to satisfy user requirements and to cover the increasing needs.

Database is distinguished from the file system by the ability to work in a multi user environment. This usage needs specific techniques to protect the consistency and integrity of data contained in the database. The most popular technique used to attain the data protection (serializability) is locking (i.e. each transaction reserves access to the data it uses). Locking is done by following some rules (locking protocol) [21] and according to compatibility function among different lock types (read or write), where a read is not compatible with a write.

Locking protocols are then a set of rules defining allowable sequences of lock and unlock operations which may appear in a transaction [21]. The purpose of these rules is to guarantee that any possible concurrent execution of a set of transactions is the only rule that has an effect on database equivalent to that of some serial execution of the transactions in the set. In such case, a protocol is said to guarantee serializability and any serial execution of a set of such transactions is said to be serializable.

When a transaction sets a lock, it delays other transactions that need to set a conflicting lock, the more transactions that are running concurrently, the more delays will happen (i.e. locking affects performance).

The level of locking is a main factor that affects such delays, by assuming that a database is represented as a multiple granularity hierarchy tree [1, 5, 11, 31, 33], locking could exist at the database level which satisfies the high security and integrity but it will be so slow and similar to the file management system, which means one user uses a database at a time (more delays). Also it is possible to use it at the table level which gives the chance for more users to use the database (less delay), but at the same time there will be a lot of

unused data. It is possible to use locking at the row level (the smallest granule that can be locked), which gives more chance for more users to access the data (increasing the concurrency).

By using the locking techniques to ensure database integrity and consistency, they produce a problem related to locking database items, which is the deadlock problem. Deadlock is defined by [33, 43] as a situation, where two or more competing transactions are waiting for each other to finish, and thus neither ever does. Also defined by [31] as a situation that occurs when two or more transactions wait for each other to unlock data. In a distributed database, deadlock becomes more complex and needs much time to resolve than in centralized database. Figure 1.1 shows graphical representation for deadlock situation [28, 33].

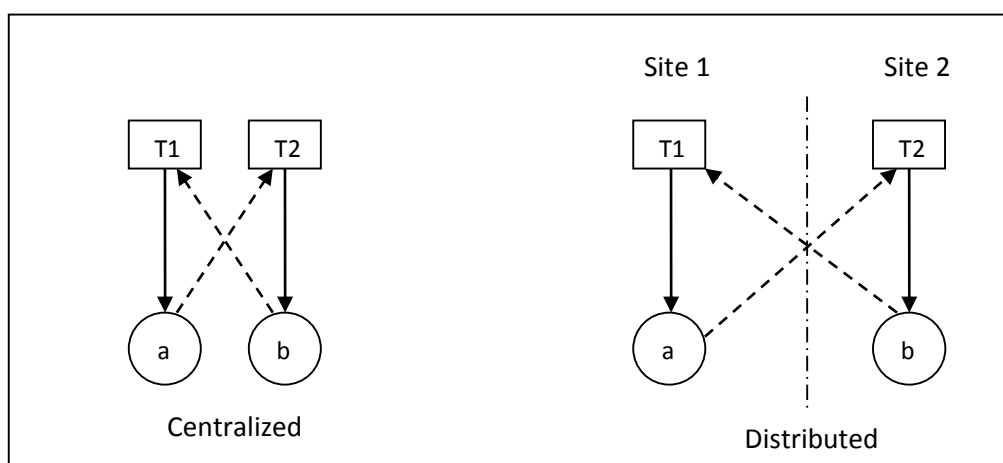


Figure 1.1 (Graphical representation for deadlock)

1.2. Distributed Database

A distributed database system is consisted of a number of sites connected via a computer network [28], which has a high number of resources, besides, the number of users to these items or resources is higher than centralized. A distributed transaction is a set of operations, in which two or more network hosts are involved. Usually, hosts provide transactional resources, while the transaction manager is responsible for creating and managing a global transaction that accomplishes all operations against such resources [44, 45]. Each host or computer has a local transaction manager responsible for interacting with other transaction managers via either a superior or subordinate relationship, in case of a transaction does work at multiple computers [24].

In a distributed database, a transaction consists of several participants or agents to execute over all sites; all participants must guarantee that any change to data will be permanent in order to commit the transaction, if any of the participants fails to make this guarantee, the entire transaction fails and aborts. There are many approaches according to where the lock management is performed, one of them is the centralized locking, where there is one site

responsible for granting locks because it is the only site that has a lock manager, in this case, the central site has a lock table for the entire database. Communications among other sites are performed via transaction manager at the site where the transaction is initiated, the lock manager at the central site, and the data processor at other sites participating to carry out the operations [4, 28].

When a transaction needs to lock a data item, it sends a request to the central site that determines if the lock can be granted, if so, it sends a message to the originating site, else it will wait. In case of read operations, the transaction performs its action from any site that has a copy of the required data item, whereas in a write case, all sites owning a copy must participate in this action [33]. The simplicity in implementation and simplicity in deadlock handling are two factors to be considered in choosing the central locking approach, because we don't have real data and not concerned with designing real distributed database system; rather we are concerned with measuring our approach (attribute level locking) to system performance.

1.3.The Problem Statement

The problem is due to the incremental number of users of the database, and the competition for acquiring a data item becomes very

high. The multi usage of database resources after locking them enables the contention to take place indicating the deadlock problem. Specifically, a Distributed Database Management System (DDBMS) gives higher degree of multi possibilities of using available resources and thus yielding higher degree to deadlock problem to occur. The initial solution to this problem relied mostly on a DBMS (Distributed or Centralized) to choose a victim transaction to abort according to some criteria like login time, priority of transaction or number of resources required.

Solutions to this problem as suggested by various researchers [8, 10, 13, 47], have so far been based almost exclusively on a strategy of dividing the database into units or entities, giving access that may be controlled by a database concurrency control. These database units have variable sizes, it may be the whole database or entire table, and also it could be the database row, locking these units is done according to some rules (locking protocol e.g. two-phase locking protocol) to ensure data consistency and integrity. One of the famous approaches to lock a database item is by representing a database as multiple granularity hierarchy presented by Gray et al 1976 [14], locking is done in top down and releasing locks are done

in bottom up. For example, a transaction must lock all ancestors of a node in intention mode before locking the node itself in exclusive or shared mode; the smallest lockable unit in this approach is the database row.

This study aims to increase the granularity hierarchy tree one more level down to include the attribute level, i.e. locking will be done at the attribute level to allow several transactions to access the same row simultaneously. The suggested attribute level is expected to decrease the user competition for acquiring data items, which is expected to reduce the total delay time, because the transactions may not need to wait for long time to get unlock state, and due to the several transactions may work simultaneously at the same database row. Also the study expects to increase the performance of the database, by the ability to reduce the mean service time. However, this will increase the overhead on the database.

1.4. Dissertation Questions

- 1- Does increasing the granularity hierarchy by one more level decrease the user competition for acquiring data items?
- 2- Does increasing the granularity hierarchy by one more level increase the performance of the database?

1.5. Dissertation Methodology

This dissertation will pass through the following phases:

- Identifying the shortcomings of the current solutions of deadlock problem.
- Enhance the intention locking algorithm, by adding a new feature which is the attribute locking.
- Simulate the enhanced algorithm to provide a proof of concept.
- Analyze the results and compare the measuring units (response time, delay time, overhead and performance) with the already used units. Then, conclusions and comments will be drawn.

1.6 .Rationale of the Study

Several transactions may be executed concurrently and the system must control the interaction among them in order to prevent those transactions from destroying the consistency of database (i.e. ensuring serializability). Several mechanisms may be used to achieve this control: one of them is by obtaining a lock on a data item before the transaction can use it. The others are time stamp-based protocol and time stamp-ordering protocol.

The lock may be obtained on the entire database, entire table, page, or entire row by representing the database as a tree of multiple granularity levels [11, 33]. It can be achieved by defining a hierarchy of data granularities with variable sizes, where the small granularities are nested within larger ones. An example of such hierarchy [33] is represented graphically in Figure 1.2.

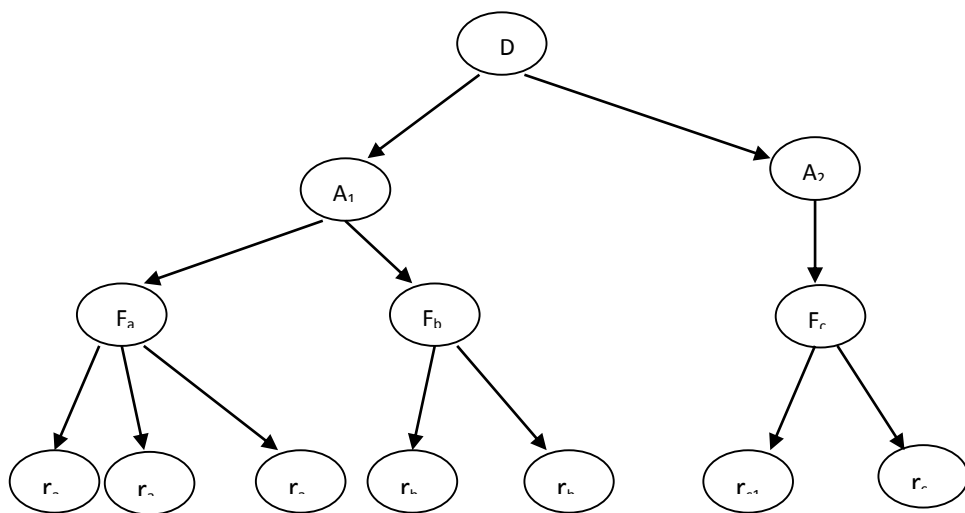


Figure 1.2: granularity hierarchy [33]

The top level (DB) represents the entire database, A_1 in the second level represents node of type area which has a node of type file (F_a , F_b and F_c) in the next level. Finally, each file has node of type record or row ($r_{a1}, r_{a2}, \dots, r_{cm}$), (where r_{a1} represents the record or row number one of the file F_a , r_{a2} is the row number two of file F_a , etc..) , in the lowest level.

To avoid scanning the tree in order to check if the database item is locked in a compatible mode or not, a new class of locking is produced called intention lock mode as shown in table 1.1 [33]. The term intent is used as a flag placed in a database node (in a database hierarchy) to indicate the use of sub tree in a specific mode, it is not considered as a lock mode (i.e. the transaction can't precede with its execution until it acquires the shared or exclusive mode on the database item (node) [20]. Intention locks are acquired at the top level before explicit locking being placed at the lower level. There is an intention mode associated with shared mode called Intention Shared (IS) which protects requested shared locks on some resources lower in the hierarchy (i.e. indicates an intention to read data at lower level), and another with exclusive mode called Intent Exclusive (IX) to protect requested exclusive locks on some resources lower in the hierarchy (i.e. indicates an intention to write data at lower level). The third one is combined from the Shared and Intent Exclusive which is called Shared with Intent Exclusive (SIX) to protect requested shared locks on all resources and intent exclusive locks on some resources of lower levels (i.e. indicates to read all but to write some of resources at lower level). Intent locks serve two purposes [35]:

- To prevent other transactions from modifying the higher-level resource in a way that would invalidate the lock at the lower level.
- To improve the efficiency of DBMS in detecting lock conflicts at higher level of granularity.

The other two modes of locking a resource are the Shared (S) and Exclusive (X) modes [33], the shared (S) is obtained by a transaction in order to read the resource e.g. SELECT statement, which can read but could not write. While exclusive mode (X) is obtained for both read and write to a resource e.g. UPDATE statement.

Table 1.1: Compatibility matrix for granularity hierarchy in Figure 1.2 [33]

	IS	IX	S	SI X	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

A locking compatibility function controls multiple transactions for acquiring locks on same resource at the same time. If a resource is already locked by another transaction, a new lock request can be granted only if the mode of requested lock is compatible with the mode of the existing lock. If not, the transaction requesting a new lock must wait for the existing lock to be released or for the lock timeout interval to expire. For example, no lock modes are compatible with exclusive locks, while an Exclusive (X) lock is held, no other transactions can acquire a lock of any kind (shared or exclusive) on that resource until the Exclusive (X) lock is released. Alternatively, if a Shared (S) lock has been applied to a resource, other transactions can also acquire a shared lock on that item even if the first transaction has not completed. However, other transactions cannot acquire an exclusive lock until the shared lock has been released

The locking on granularity tree can be summarized as follows:
"If a node is locked in an intention mode, it implies that explicit locking is being done at a lower level of tree. For example if a node is locked in intention-shared mode (denoted by IS), this implies that explicit locking is being done at lower level of the tree but only with shared

mode locks. Similarly, if a node is locked by intention-exclusive mode (denoted by IX) then the explicit locking will be used at a lower level of the tree with exclusive mode or shared-mode locks. If a node is locked in shared and intention-exclusive mode (denoted by SIX), this implies that the sub tree rooted by that node is locked explicitly in shared mode and that explicit locking is being done at lower level with exclusive-mode locks" [33]. The transaction can lock a node in top-down order and unlock in bottom-up order by using the following rules [33]:-

1. The lock can be done according to the compatibility matrix Table
2. The transaction must lock the root first in any mode.
3. The node can be locked in S or IS, if the parent of that node is locked in IX or IS mode.
4. The node can be locked in X, SIX or IX, if the parent of that node is locked in IX or SIX modes.
5. The transaction can lock a node if the transaction has not previously unlocked any node.
6. The transaction can unlock any node, if none of the children of that node is locked by that transaction.

Deadlock is particularly troubling because there is no general solution to avoid it. Two common places where a deadlock may occur: between processes in an operating system (distributed or centralized) and between transactions in a database [9, 31, 33]. By reducing the competing parts among transactions, the suggested method is expected to reduce the deadlock occurrences.

1.7. Goals of this Dissertation

The purpose of this study is to develop a new technique to divide the database row among more than one transaction by using the multiple granularity, allowing the transaction to lock the needed attributes instead of locking the whole row and trying to reduce the overhead. This technique will be applied to a single database (centralized) first, and then modifying it to the distributed database environment.

1.8. Dissertation Contribution

As shown in Figure 1.3, the new level is representing the attributes for each row after some modifications. Each node in the hierarchy can be locked individually as in the two-phase locking protocol according to the compatibility function matrix Table 1.1.

The locking can be done on the part of the row including the key or the index abreast with the attributes needed by the transaction. This can be done by ensuring that no qualification conflicts will occur among the competing transactions. This procedure is expected to satisfy the following:

1. Increase the concurrency, because the same row may be manipulated by more than one transaction at the same time.
2. Reduce the deadlock problem occurrences, because the competing parts are reduced into some attributes instead of the whole row.
3. Increase performance and system throughput, by increasing the number of transactions executed in the system.

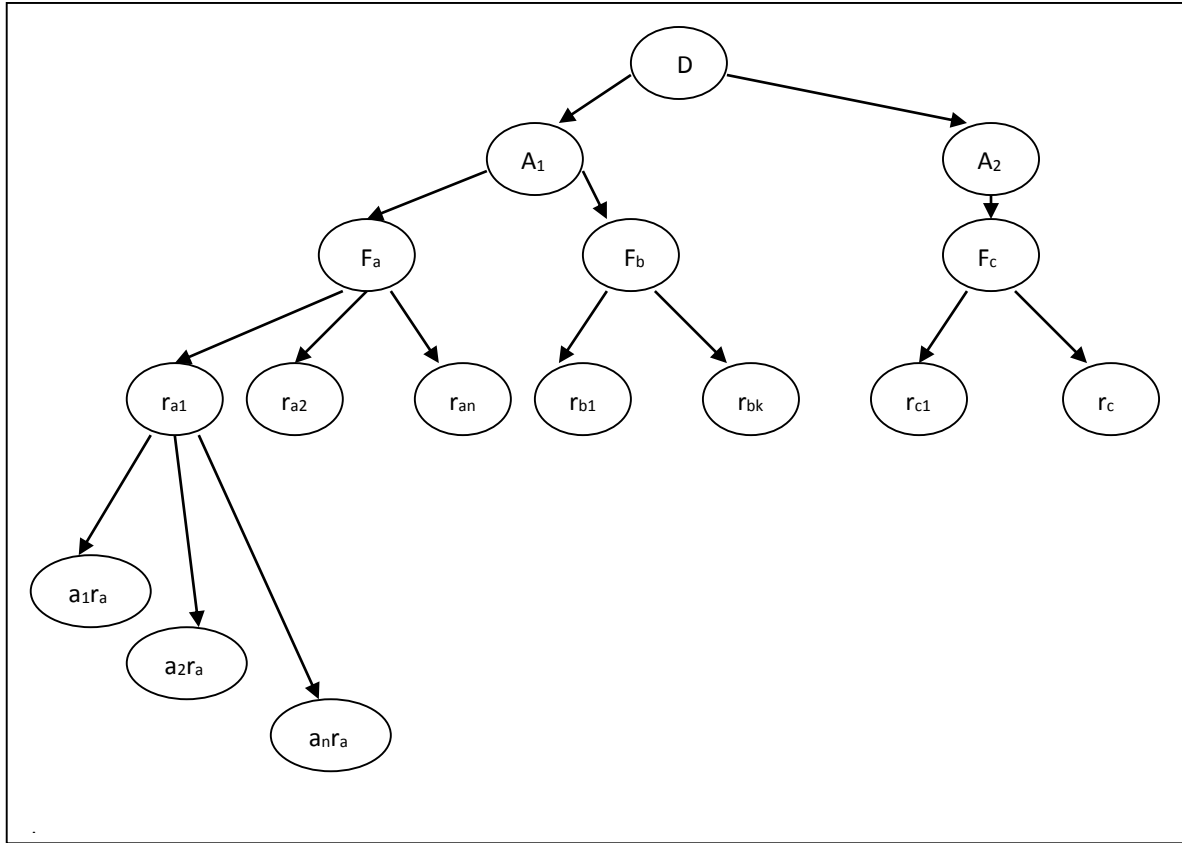


Figure 1.3: Proposed granularity hierarchy

1.9 Dissertation Structure

This dissertation is organized as follows:

- **Chapter One:** Presents introduction to dissertation subject, introduces the problem to be addressed, gives aim, a justification and purpose of this work, and lists the basic structure of the thesis.
- **Chapter Two:** Provides Preliminaries for Current Locking based protocols and Current solutions to deadlock problem in distributed database. This chapter provides an overview of

- currently used mechanisms for locking database items and current solutions to distributed deadlock problem. Also pointing the drawbacks of the previous solutions.
- **Chapter Three:** Presents the enhanced algorithm for locking attributes and how it deals with the distributed deadlock. A simulation model will be used to validate the proposed structure and algorithm.
- **Chapter Four:** Deals with the analysis of the proposed algorithm on centralized database.
- **Chapter Five:** Deals with the analysis of the proposed algorithm on distributed database (analyzes possible benefits or drawbacks). It will investigate to what degree the algorithm facilitates solving the distributed deadlock problem.
- **Chapter Six:** Provides basic conclusions as well as directions for future research.
- **Appendices:** One appendix exists for showing the input and output parameters of a simulation program.

1.10 Conclusion

The important use of databases and the increasing number of users, who use these databases, become dominant factors in increasing the availability and reliability of the data contained in a database. Increasing lockable database items by allowing fields to be locked instead of rows, is the approach presented in this dissertation. This approach is expected to increase concurrency as well as decrease deadlock occurrences.

CHAPTER TWO LITERATURE REVIEW

This chapter is an overview of the lock and deadlock processes. It discusses the existing mechanisms for locking database items, and handling deadlock problems in distributed database systems, and lists the shortcomings of the available solutions. The preliminaries are given on five areas:

- (i) Handling concurrency control in a database by locking techniques.
- (ii) Review of locking techniques in different database systems.
- (iii) Review of deadlock handling in distributed database systems.
- (iv) Handling deadlock in some well known databases.
- (v) Locking performance in a database.

2.1 Introduction

In a database concurrency control, there are several approaches used to preserve the database consistency and integrity; the more popular approach is by locking a database item before using it. The size of the database items that can be locked (lockable size)

is discussed and proposed by many researchers. During the investigation of the solutions produced to choose a suitable database lockable units and to solve the database deadlock problem, it may easily be found that there is a trade off between the produced solutions and the performance of the system. Many researches are presented here to show the different approaches among them. It should be a compromise between choosing the ideal solution and the suitable environment.

2.2 Handling Concurrency Control in a Database by Locking Techniques

Several transactions may be executed concurrently and the system must control the interaction among them in order to prevent those transactions from destroying the consistency of the database (i.e. ensuring serializability). Several mechanisms may be used to achieve this control: one of them is by obtaining a lock on data item before a transaction can use it. The others are time stamp-based protocol and time stamp-ordering protocol. This dissertation will concentrate on locking techniques to explain the proposed method.

The two-phase locking protocol was introduced by Eswaran et al, 1976 [12]; they present some basic definitions for transaction, consistency and properties of locking, by formalizing the concepts via a data model. Their paper demonstrates that a transaction can't

request a new lock when releasing one, showing that a transaction is executed in two phases, the first one when a transaction requesting its locks which is named as growing phase, and the second is when a transaction releasing locks named as shrinking phase. They also state that a transaction needs to lock a logical portion instead of lock physical subset of a database.

Gray et al, 1976 [14], discuss the database lockable size, and its effects on concurrency and overhead, because there is a tradeoff between concurrency and locking overhead. They present a new locking protocol based on two-phase locking to allow requesting of concurrent locking by different transactions on various granularities (variable database unit's size). The proof of this concept is presented by introducing new lock modes (Intention modes) in addition to the original shared and exclusive modes, and they introduced four degrees of consistency named as degree zero for update protect, one to lost update, two to protect reading incorrect data and degree three to protect reading incorrect relationships among database items.

The main factor that could affect the concurrency and locking overhead is the choice of lockable size. So, if the lockable size is chosen to be small like a database row then, concurrency is increased and there is a lot of managing overhead, while the granularity size is

chosen to be large like a database table then, less concurrency and less overhead to be managed. In this case, one transaction can use the table at a time, so choosing a granularity of variable sizes may be more suitable for many environments, and the database management system must choose the appropriate data item size to be locked by a specific transaction according to its need. This can be achieved by assuming that a database is organized as a hierarchy with unique parent to each node. These nodes are lockable, and the intention is used to prevent the ancestor of a specific node to be locked in an incompatible mode and to avoid scanning the hierarchy to determine if a new lock can be granted or not.

Gray et al. [14], introduce six deferent intention modes and give the description to each one as follows:

- NL: Represents that the node is not locked.
- IS: Represents intention share to a node and to allow the transaction to lock descendant nodes in S or IS mode.
- IX: Represents intention exclusive access to a node and allow the transaction to lock descendant in X, S, SIX, IX, or IS modes.
- S: Represents implicitly and explicitly lock to a node and to all descendants.

- SIX: Represents implicitly shared locks to all descendants and allow the transaction to explicitly locks descendants in X, SIX, or IX modes.
- X: Represents exclusive lock to a node and its descendants.

They also present the rules of their protocol and its proof that must follow to grant the requested lock according to the compatibility matrix shown in Table 2.1:

- Transaction must hold the ancestor of the required node in IX or IS before holding the node itself by S or IS mode.
- Transaction must hold the ancestor of the required node in SIX or IX before holding the node itself by X, SIX, or IX mode.
- Locks are requested in top down and released in bottom up order.

Table 2.1 (Gray et al, Compatibility matrix)

	NL	IS	IX	S	SIX	X
NL	Y	Y	Y	Y	Y	Y
IS	Y	Y	Y	Y	Y	N
IX	Y	Y	Y	N	N	N
S	Y	Y	N	Y	N	N
SIX	Y	Y	N	N	N	N
X	Y	N	N	N	N	N

Lomet, 1992 [21] discusses the cost of locking needed in distributed systems and introduces the concept of lock covering which is the way of performing local or private locking at the system component to improve the trade-off between concurrency and overhead. Converting global locks into intention locks for hot data can reduce the conflict among transactions which improves the concurrency. Using this converting for cold data will reduce the locking overhead. Lomet presents two ways to access a distributed data, the first way with the server independence responsible for accessing data portion at a time, and to use messages to coordinate the accessing of distributed portions. The other way is accessing shared data by several servers. He concentrates on explaining that lock management is to be done with more independence, by performing lock management privately on the resources by each server. He also introduces some locking fundamentals by seeking formal definitions used by the lock manager to grant or deny an access to a resource.

Galindo and Rabitti, 1995 [13], discuss the gap between theoretical and practical DBMS situations like the access of redundant data which is the core of indices. Also they present a proposal for

locking problem in practical situations by explaining the difficulties to implement such redundancy of data to achieve the aim of increasing concurrency and decreasing overhead. They implement their aim through discussing the multi granularity locking, locking of logical structure rather than locking of logical rows and locking of redundant data to reduce overhead. In multi granularity locking, database items are arranged in a hierarchy and could be locked by using intention lock to coordinate transactions. The path of a node must be locked in intention mode to prevent other transactions from locking intermediate nodes in that path in an incompatible mode. The hierarchy is static in terms of fixed number of fields in each record, and dynamic in terms of number of records in each file, so the number of records is growing dynamically. To lock a field of a row consisting of N fields in shared or exclusive modes, there are $2N$ type locks needed to perform this and each row needs a compatibility table to determine the conflicts among them.

In locking logical structure when in some situations and sometime, there is a need to acquire a lock to non existence item, they proposed to add virtual field to logical database structure with values zero or one to indicate if the required record or item really exists in a database or not.

Concerning the locking of redundant data, they propose the concept of write implication to force the transaction to execute its write operations according to a given constraint e.g. $A + B = 0$, then when the transaction is going to update x it must also update y . This is can be done by issuing a single lock to cover A and B in order to reduce the locking overhead. For more complicated constraints, the concept of equivalent classes is proposed to copy these constraints into classes and lock all of them before the transaction can perform a write operation to any class, the authors called this procedure a collective lock.

Ries and Stonebraker, 1997 [29], present a study about the more suitable granule size and introduce an example of some well known databases using physical locks on the items like records, pages, or files. These granules are records in CODASYL and System R, pages in IMS and a column in INGRES. Some systems are using a variable granule sizes dynamically, or in terms of using predicate locks, that is, lock is set to specific database portion according to some qualifications. In case of physical locking used, fine granularity increases concurrency as well as increases locking overhead, while coarse granularity decreases both of them. So, the granule size

affects the system performance and throughput. When using predicate locks, they conclude three results during their proof. First, the number of locks to be managed is decreased because the number of locks is related to the number of active transactions in the system not the database size. Second, predicate locking requires more CPU time. Third, there is a reverse relation between transaction size and granule size, instead of pre specified granularity, the transaction determines the database portion to be locked.

Croker 2001 [10], introduces new mechanism to increase concurrency by restricting the lock and unlock steps within the transaction, and he defines a cost function to measure the conflicts among transactions executed in the system as well as to measure the locking time period for transactions, because when this time becomes long, this means that the probability of the conflict time between competing transaction becomes high. The techniques for reducing the conflict time between competing transactions and a way to measure this time presented here is for two phase locking and tree locking protocols is considered as a base for future building transaction compiler to optimally use of locks.

2.3 Review of Locking in Different Database Systems

Sybase, 2003 [37] has an adaptive server that provides locking schemes, All pages locking, which locks data pages and index pages, Data pages locking, which locks only the data pages, Data rows locking, which locks only the data rows. For each locking scheme, Adaptive server can choose to lock the entire table for queries that acquire many pages or row locks, or can lock only the affected pages or rows.

Oracle, 2008 [26, 27] uses locking technique to solve the problems associated with data concurrency, consistency, and integrity. Two types of oracle resources: the user objects (tables and rows), and the system objects (shared memory). The lowest level of lockable database item is the row. With a row-level locking strategy, each row within a table can be locked individually; locked rows can be updated only by the locking process. All other rows in the table are still available for updating by other processes. While, using table-level locking, the entire table is locked as an entity. Once a process has locked a table, only that process can update (or lock) any row in the table. None of the rows in the table are available for updating by any other process. Oracle uses two modes of locking in a multi-user

database, exclusive lock mode and share lock mode, and it does not use lock escalation, because the probability of a deadlock becomes higher.

In SQL server 7.0/2000, 2008 [20, 35], the smallest data item that can be locked is the row; there are three types of modes, Shared locks, Update locks and Exclusive locks. The Shared locks are used for operations that do not change or update data, such as a SELECT statement. While the Update locks are used when SQL Server intends to modify a page, and later promotes the update page lock to an exclusive page lock before actually making the changes. Finally, the Exclusive locks are used for the data modification operations, such as UPDATE, INSERT, or DELETE.

Ingres, 2008 [16] is a relational database that maintains a lock on different levels with the row level locking as a lowest level, Ingres also provides a lock to be done at the whole column (e.g. locking the balance column in an accounting table).

2.4 Review of Deadlock Handling in Distributed Database Systems

Deadlock is a problem related to computer systems when using multiprogramming environment. It appears when the mechanism of

locking system recourses is used to ensure the lost update problem will not occur, and to preserve system resources either in a database or in operating systems. The locking techniques preserve the consistency of a database for example and prevent multiple transactions from modifying the same database item in a conflicting mode, such prevention is obtained by allowing some transactions to perform their tasks and deny others i.e. blocking some transactions, when two or more transactions are blocked, each one waiting for database item held by the others, then a deadlock problem occurs.

Deadlocks not only affect system performance, but they also decrease system throughput as well as increase the cost of recovery to solve the problem. Four conditions must hold in order for the deadlock problem to occur [9], first one is by the ability to assign the resource to only one process or transaction at a time (mutual exclusion). The second is the ability to request another resource while already getting one (hold and wait), third is by releasing the resource only by the process itself (no preemption), and finally is by obtaining circular chain among transactions, that is holding a resource and waiting for others (circular wait). There are three main approaches to deal with a deadlock problem [15], deadlock avoidance, deadlock prevention and deadlock detection.

In a deadlock avoidance approach, deadlocks can be avoided if certain information about transaction requests is available in advance. So the database system only grants request that will lead to safe states, but this is not available in many situations, because most transactions know the resources at the run time. One known algorithm that is used for deadlock avoidance is the Banker's algorithm [3, 42]. Deadlock prevention may be achieved by ensuring that at least one of the above mentioned conditions does not occur, and deadlock detection attempts to find and resolve an actual deadlock by building a wait-for-graph and searching for cycles. If a cycle exists, the action to break the deadlock is necessary by killing and restarting some transactions that caused the deadlock according to some certain criteria like the starting time, number of holding resources or the number of resources required.

In distributed database systems which consist of a number of sites connected via a network, the deadlock problem becomes more complex, because the system must build a global wait-for-graph by the union of local wait-for-graph from each site and searching for a cycle.

Chandy and Misra, 1983 [8], present decentralized deadlock detection algorithm by assuming that; messages are received in the same order which they were sent. The algorithm is suitable for both resources and communication models, and they prove that the algorithm does not report false deadlocks. Also does not affect performance because the process can initiate deadlock when it is idle for a given time which will decrease the computation for deadlock, only boundary transactions can send a message and use the first element of the dependency table in the wait path.

An enhancement to this algorithm is introduced by Yeung et al 1995 [49], it uses the wait-for-graph instead of building dependency table, the source and the destination transactions send messages rather than all transactions which reduces the overhead. They compared their presented algorithm with the Candy algorithm and showed that it was better under high data contention, but it had more overhead because of the frequent checking of the wait-for-graph.

Wu et al 2002 [47], introduce a new deadlock avoidance algorithm based on rank of processes within wait-for-graph, this algorithm does not need prior information about processes and does

not restrict the order of resource requests and it is unnecessary abortion free. This algorithm is built for the AND model where the request for several resources may be issued simultaneously. It uses the partial order of the processes rank to reduce the deadlock detection. They prove that their algorithm is better than the others in terms of reducing the number of deadlock detections by reducing the number of unnecessary abortions, handling gently the overflow and underflow of ranks. It has the ability of adding multiple edges in the wait-for-graph at the same time.

When a deadlock occurs in the systems, the overall system will be degraded until resolving the deadlock. So as long as the deadlock persists in the system, the system performance and throughputs are decreased. (Ling and Chiang, 2006) [19]), discuss these factors in detail and introduce an optimal scheduling for deadlock detection and resolution in order to minimize the average cost time. In addition, they introduce a cost function to measure the effect of deadlock in the system. Whenever a deadlock exists in the system, the process which needs a resource currently deadlocked can't proceed. So the deadlock size (number of deadlocked processes) and the time that the deadlock exists in the system are two main factors that affect and

increase the cost of resolution. The deadlock size is increased whenever the deadlock exists (i.e. the deadlock persistence time increases). The model they built is time dependent that associates the deadlock resolution cost and the deadlock persistence time. They show that the performance of deadlock handling depends on both per-execution of deadlock detection algorithm and deadlock scheduling and formation.

2.5 Handling Deadlocks in Some Well Known Databases

Sybase, 2003 [37] differentiates between server side deadlock and client side deadlock, when tasks deadlock in Adaptive Server, a deadlock detection mechanism rolls back one of the transactions involved in a deadlock situation and sends messages to the user. When a client opens multiple connections each of which is waiting for the other, deadlocks may occur at the application side, but these are not true server-side deadlocks and cannot be detected by Adaptive Server deadlock detection mechanisms. Avoiding deadlocks in Sybase comes from reducing lock contention such as locking fine granularity instead of coarse granularity, as well as acquiring locks in the same order, such as updates to several tables must be performed in the same order.

Oracle Database, 2008 [25] automatically detects deadlocks and resolves them by rolling back one of the transactions involved in the deadlock, and releasing one of the conflicting row locks, then send back an error message to the user. In terms of deadlock avoidance, deadlocks can be avoided, if transactions accessing the same tables by locking those tables in the same order, either through implicit or explicit locks. Oracle has some rules to perform locking order, for example locking of master table must be done before locking of detail table in case of update. If such rules are followed in application, deadlocks occurrences are decreased.

SQL Server, 2008 [36] ends the deadlock when it occurs by automatically choosing one process to be aborted and allowing the other process to continue. The aborted transaction is rolled back and an error message is sent to the user, abortion of the process is done by identifying which of the two processes will use the least amount of resources to rollback. In case of deadlock avoidance, SQL Server provides some tips to avoid deadlocks which are:-

- Database must be well normalized.
- Accessing the database objects in the same order each time.

- User inputs must be collected before the transaction begins.
- Cursors have to be avoided as much as possible.
- Transactions must be as short as possible, and avoid reading the database item many times in the same transaction, by storing it in a temporary variable or in an array to be read from this location not from the server.

Ingres database, 2008 [16] like the other databases, aborts one of the transactions involved in the deadlock situation when detecting, and allowing the other transaction to continue, as well as an error message is sent back to the user. All updates made by the transaction are backed out and the transaction retried in an application program. To avoid deadlock occurrences, careful use of lock escalation and transactions must be as short as possible.

2.6 Locking Performance in a Database

Tay et al 1985, [48] present mathematical model to study the behavior of the database system with dynamic locking. The model is separated into data contention and resource contention and this model can be used to determine the level of data contention that is allowed in the system. Their model is showing that the transactions

must minimize lock requests, because data contention is proportional to the square number of locks requested. So a transaction needs $k+2$ independent requests: one for start, one for termination and the others ($k; 1 \leq i \leq k$) are for data lock with T as an average inter-request time uniformly distributed over $(0, 2T)$. During the analysis process and because of contention for data and resources, it appears that the more transactions executing in the system, the slower the execution for each transaction as well as the number of active transactions decreases. They present the relationship between system workload (k^2N/D where N is the number of transaction and D is the database size) and thrashing, so when the workload is about 1.5, the system goes to thrashing (the value 1.5 of the workload has no theoretical explanation as Tay et al. claimed). The important thing about workload is the database size D as a factor for conflicts, so as D increases, the conflict ratio decreases, the number of waiting transactions becomes less, and the number of active transactions increases, but unfortunately the overhead increases.

Wolfson 1987 [46] proposes three measures for evaluating distributed database locking overhead, and comparing these measures against three locking protocols: the two-phase locking, two-

phase locking with fixed order and tree protocol. The first metric is for measuring parallel execution (maximum number of protocol steps). The second is for measuring the longest sequence of inter-site messages that the transaction must send to follow the protocol. The last one is for measuring the total number of messages between sites. He showed that, the two-phase locking protocol has the minimum overhead among the others as well as, it has the smallest inter site message path, but is not deadlock free.

Thomasian and Ryu 1990 and 1991 [32, 38], produced and developed a mathematical model with dynamic two-phase locking. Transactions are classified according to the number of data items needed and the mode of request. The developed model is affected by blocking and by restart. Analysis of the model is based on the mean number of locks held by a transaction. Their model is composed from two sub models, the database model and the system model. Exclusive mode requests to update a data item and mixed of read and write modes are two sub models of database model. The system is analyzed by hierarchical model to derive parameters in a lower level and use them in a higher one, such models are response time, throughput, resource contention and data contention.

Their model shows that the mean number of lock conflicts is a good measure for lock contention which is expressed by mean number of locks multiplied by the probability of conflict, when the value of this measure approaches 0.75, the system of dynamic locking of fixed size transactions goes to thrash. By analyzing the model, Thomasian and Ryu found that, the system performance is affected by transaction blocking caused by lock conflict rather than restart, in a certain degree of concurrency, the system performance is decreased due to an increase in arrival rate, while, in a systems with variable transaction sizes, the degradation of system performance is proportional to the transaction size.

Thomasian 1998 [39] summarizes the performance metrics of dynamic locking during derivation of mathematical formulas representing the database model. The standard locking model analyzed in his article is useful to understand the factors leading to system degradation. He found that the blocking is the main cause of decreased system performance, while the transaction restart to solve deadlock is considered as secondary reason. In terms of system thrashing, there is no ideal way to control system workload to prevent thrashing. The mean number of active transaction is maximized at the

point of 30% of transactions are blocked, so the system begin thrashes after reaching this point. The probability of lock conflict is proportional to k^2 which is dependant on transaction length (number of operations) rather than number of transactions.

Bernstein and Newcomer 2004 [6] introduce a mathematical model for locking performance to measure the probability of lock conflict, deadlock occurrences and throughput. The model is based on three variables which are the mean number of lock requests, database items and the number of transactions in the system. The model is built by assuming that each transaction has a mean of K write locks request on average with T time between requests, D of lockable data items (database size) and with N transactions running at a given time, by considering that all database items have the same chance of access, the following formulas can be derived:

- Probability of lock conflict is proportional to $K^2 * N / D$.
- Probability of deadlock occurrences is proportional to $K^4 * N / D^2$.
- Throughput is proportional to $(N / T') * (1 - A * K^2 * N / (2*D))$.

Where T' is transaction execution time (i.e. total transaction time – waiting time for locks to grant) and A is a ratio of transaction waiting time per lock conflict to total transaction time.

2.7.Conclusion

Locking is the most popular technique used in a database to preserve consistency and integrity; locking could be obtained at different levels when the database is represented as a hierarchy tree (which is the reality view of a database, i.e. data item could be block of data, file, record of a file, or field of record) [5]. The level of locking produces a tradeoff between increasing concurrency and locking overhead, so it should be a compromise between choosing the ideal solution according to the suitable environment. Many researchers recommend to fragment the database relation into two or more, in order to reduce the data contention, which may happen among users [6], yielding to other problems such as the need to repeat the key for each fragments which increases redundancy in a data and the need to combine those fragments for reporting or viewing the data, which may need to rewrite some application programs to reflect such fragments, in addition to the extra load needed for updating all fragment.

For deadlock, most researchers [8, 19, 47, 49], concentrate their studies on the most effective criteria to detect deadlock in an optimal time, or on improving an efficient method for deadlock

avoidance or prevention. Some recommendations presented by many researchers like [40] to reduce the level of lock contention, represented by adjusting the locking mechanism of the database system by using fine granularity of locking, this could be done by increasing database size (lockable units). So, by increasing the database size, deadlock occurrences could be minimized [6], and could make the methods of detecting deadlocks more efficient.

CHAPTER THREE

THE PROPOSED METHODOLOGY

In this chapter, the full description of the methodology and procedure is given, discussion of the motivation for enhanced locking techniques to cover the attributes of the database rows, and the enhanced protocol to achieve the aim will be shown with explanation examples. The proof of the enhanced procedure will pass through three stages: the first stage is by building and implementing a hierarchy tree representing the database with new level added to represent the attributes, the second stage is by building a database lock manager responsible for coordinating transactions execution, the final stage is by building full parameterized simulation program to generate transactions randomly. The three stages are combined together in order to measure the system performance, system throughput, and locking overhead. The combined stages or procedures are executed first using the existing situation as the database row is lowest level to be locked and processed, then it has been executed to reflect the new added level (attributes level) after some modification. Comparison of the two results is given as well as comparison of the known and published results.

3.1 Introduction

The locking techniques used in a databases to preserve consistency and integrity, may be applied on different levels, by assuming that the database is organized as a hierarchy tree [11, 33]. When the locking is placed on the entire database, one transaction can use this database at a time, yielding to no concurrency, but higher security of the use of a database will be achieved, as well as achieving high consistency and integrity. As the level of placing lock goes down, the concurrency increases, so the lowest level applicable in databases is the row. The problem that this dissertation is attempting to solve, stems from the increasing need for data to be available at all time, but there are still much data unused most of time because the lock of the whole row. So the expanding of one level down may provide a solution to the problem of increasing concurrency as well as, decrease the occurrences of deadlock, because there is a relation between level of locking (competing parts among transactions) and deadlock occurrences. An enhanced mechanism to increase concurrency and decrease deadlock is presented here through the following phases as shown in figure 3.1:

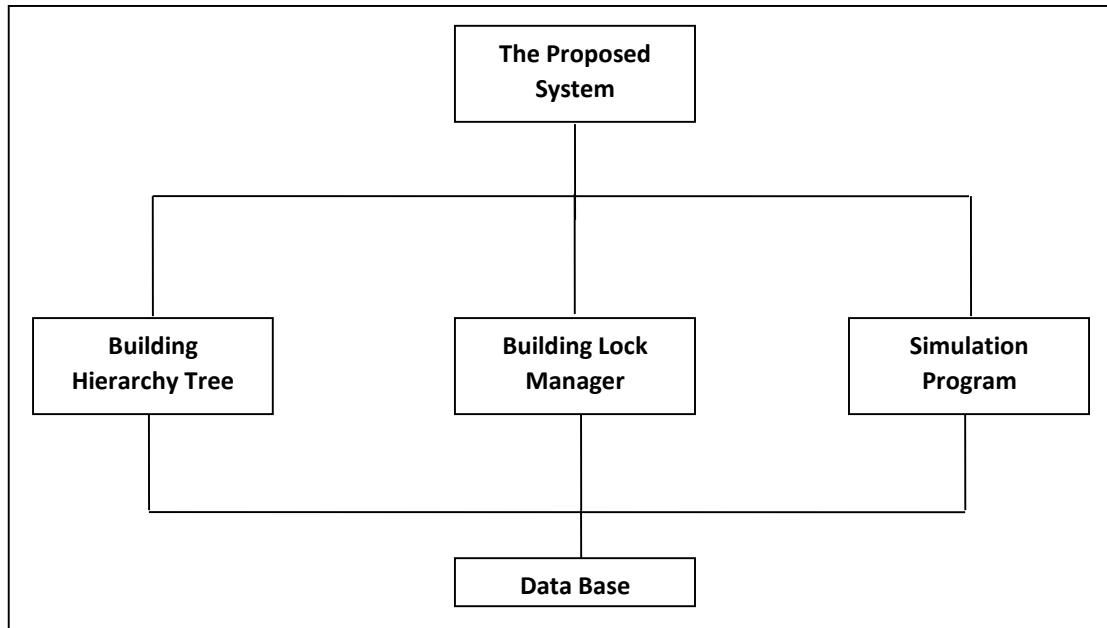


Figure 3.1 (Overview of the proposed system)

- Building the database as a hierarchy tree in two stages: first with the existing solution based on row level locking as the minimum lockable unit, then expanding the tree one level down to reflect the fields.
- Implementing database lock manager, that is responsible for associating locks to database items and ensuring that no conflict may occur among transactions.
- Building simulation to generate random transactions or threads to measure and compare the performance and throughput of the two situations.

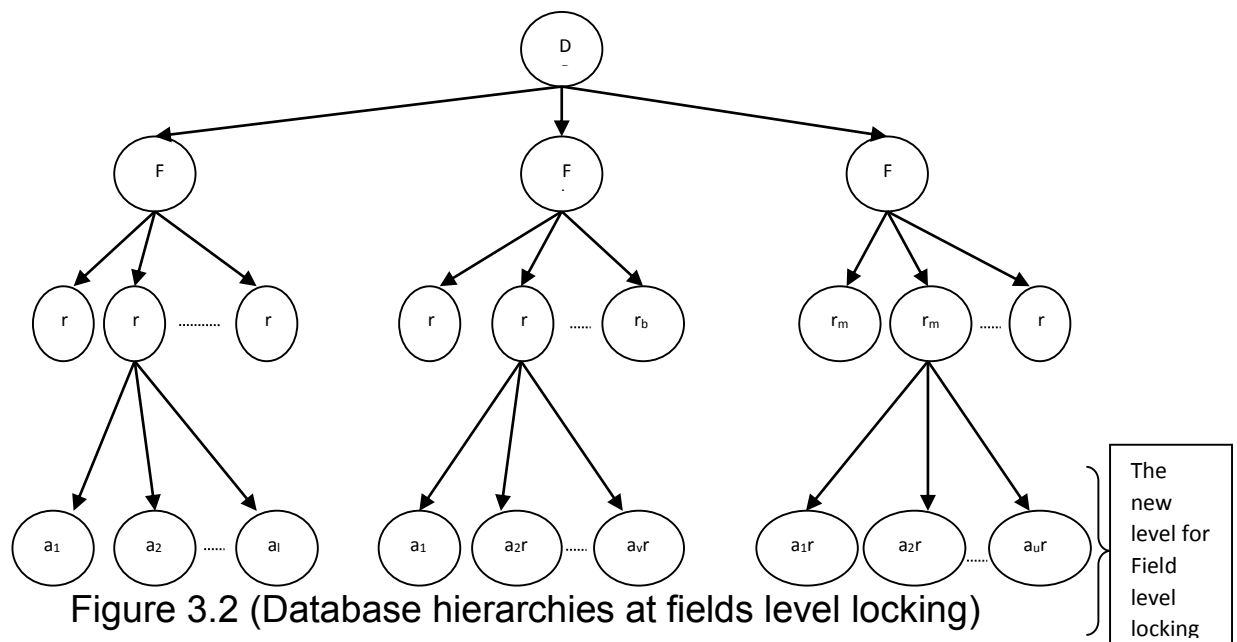
All these stages are built using the java programming technology.

3.2 Hierarchy Tree

There are many situations in which the transaction does not need the whole attributes during its process, so according to the increasing need for data to be available, there must be solutions to make the data available in a reliable way. The proposed method here is to increase the availability of data by reducing the competing parts among users by trying to reduce the data item lockable entities. For example, in the registration system, the material schedule table contains many attributes like material number, section number, academic year, semester number, building number, room number, time from, time to, instructor number, real number, maximum number, section status and many other attributes. During the registration process, the registrar (say transaction A) may want to register a student while other registrar (transaction B) needs to modify the room number, building number, maximum number or the instructor number to the same material simultaneously with the first registrar at the same row. This will not work because transaction A locked the whole row in exclusive mode, because it needs to modify the real number by adding one. In this situation, there are some database items that are not used during the first process (e.g. the maximum number or instructor number);

also they couldn't be used by the other transactions although they are available, indicating that, the concurrency is degrading and the deadlock problem has a higher degree of occurring.

If the table consists of two attributes like (flight number and seat number) then, any transactions that are competing for the same row in this table will have a conflict, because they need to lock the attributes which are two in this situation, and according to database constraints, the transaction must lock the two attributes, so the locking will be back one level up to lock the row. In general if any conflicts occur between transactions, then the locking will be backing one level up to be at the row level, regardless of the number of attributes.



The hierarchy will be built as shown in Figure 3.2, and the locking will be done according to the two-phase locking protocol with

multiple granularity locking. If there is a transaction trying to update a field within a row, for example update the room number or the maximum size, this can be accomplished by locking the database file containing the record and the record itself by IX. Then lock the material number in Shared mode (S) and the room number in Exclusive mode (X), (because the material number is the key). Then proceed with its update, at the same time another transaction could register a student by locking the database, the file containing the record and the record itself by IX. Then lock the material number in Shared mode (S) and the room number in Exclusive mode (X) to accomplish its task (update the room number). So the two transactions in this case as an example, are working simultaneously at the same record. The rules to lock a database item (node in the tree) are mentioned in chapter one, originally presented by (Gray et al 1976) [14], and explained in detail by Silberschatz et al [33], in addition to lock the record in intention mode rather than in shared or exclusive mode when the transaction need not the whole row.

The tree is implemented to represent a central database with the record as smallest lockable data item, and then is expanded to include the attributes (fields) to be locked instead of the whole row.

After measuring some metrics like performance, throughput and overhead, the tree is implemented to represent a distributed database.

3.3 Database Lock Manager

In order to build and implement a database lock manager, there are three requirements that must be taken into consideration for its data structures [41], these requirements are:-

- When a lock is requested, there must be an efficient way to check if a conflict will occur with an existing one.
- When a lock is released, any transaction waiting on the same lock must be resumed.
- When a transaction terminates, all its lock must be released at once.

The first requirement is to build a single key hash table containing locks with resource identifier; entries then used to point to a resource control block for each resource; all resource control blocks with the same value are linked together in a chain to solve the conflict and does not scan the whole table to detect a conflict. A hash table is used for fast content based retrieval [5].

The second requirement is to build a queue of lock control blocks attached to the same resource control block, because multiple locks can be held for the same resource (i.e. shared locks), and multiple lock requests can be waiting for that resource. This queue can be implemented as a linked list ordered by waiting lock requests; the linked list may have zero or more lock control block with shared locks, one or more exclusive locks, and an arbitrary number of shared or exclusive, in this order, because if all requested locks are shared, then there is no need to wait for the others. The waiting locks are managed as first in first out.

The third requirement is to build and implement a transaction control block to determine if all lock control blocks belong to the same transaction, in order to release all locks held by that transaction at once, this can be done by deleting the list of lock control blocks that correspond to the resource control block, then resume the next lock control block that is waiting in the queue for the resource.

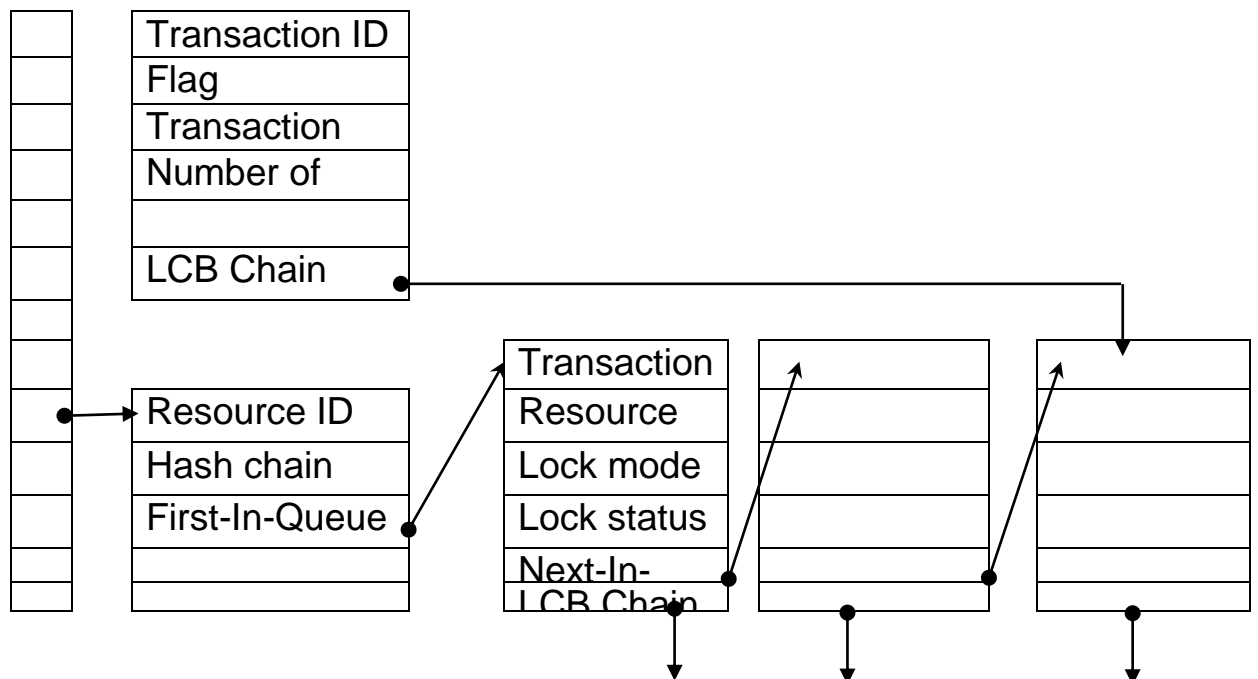


Figure 3.3 (Data structure of lock manager [41])

3.4 Proof of Concept by Simulation

The choice of simulation as a proof of concept comes from the advantages of this method as mentioned in [2] which are:

- Simulation is a popular method and widely used for studying systems.
- Most complex real world systems can't be evaluated mathematically in a correct way.
- Simulation provides an easy way to estimate the performance of existing systems under some operation conditions.
- Simulation provides easy way to compare system alternatives

- to choose the best according to predefined requirements.
- Simulation provides a high degree of control over experimental conditions.
- Systems can be studied by simulation in both long and short time frames.

According to these advantages, and the clarity of our motivation, the proof process will be introduced by using simulation.

3.4.1 Building the Simulation

Building a discrete event and full parameterized simulation program will be done by generating random transactions with different lock modes as well as different sizes (number of database items needed for each transaction), Figure 3.4, shows the suggested model overview, because the original model [29] is deadlock free, the deadlock block has been enhanced. When the simulation begins, transactions are generated, assumed to be arriving one at a time, and start to request some data, the time for these requests is chosen, and placed in the pending queue. Then the transaction is removed from pending queue in FCFS discipline, and requests its lock. Each data granule has a list of transactions holding a lock on it in case of shared locks, and a queue of transactions waiting to lock it. The transaction

requests are scheduled according to the transactions issued time. The transaction needs a data item to be locked, the lock mode and the request time are generated randomly. When a request starts processing (i.e. the lock is granted), the transaction goes to the processing queue, the pending queue is decremented as well as the system clock is incremented, the program select the next request and records its time.

If the request is blocked, the transaction goes to the blocked queue to be recorded by the simulation program. In case of deadlocked transactions, all resources are released at once, and the simulation will also record it in the log file. After the transaction processing is complete, it releases its lock, and then the blocked transaction which was waiting for the completed transaction will go to the front of the pending queue.

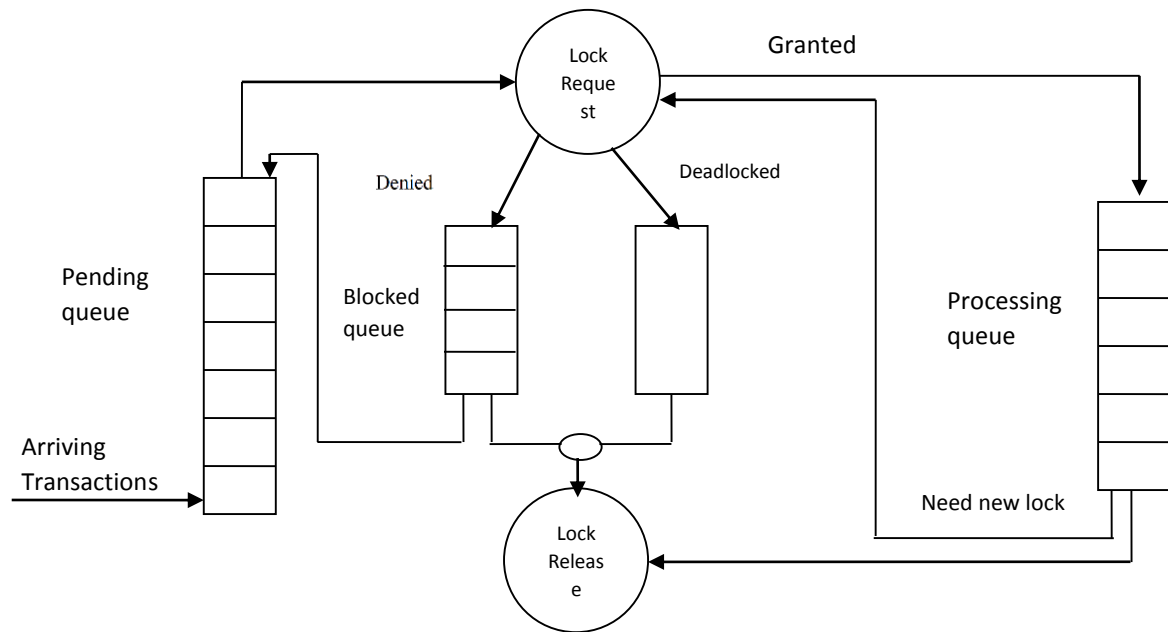


Figure 3.4 (Simulation model overview)

The simulation parameters (Table 3.1) will be used to generate multiple snapshots during progress of a simulation, these parameters are user driven and will vary for each run in order to show the system behavior. The simulation will be run according to some assumptions in three stages:

- The first stage when the database lockable unit is the row as the minimum level.
- The second stage will run after expanding the database hierarchy one level down, to cover the attributes as

- lockable units (both stages will be implemented in a centralized database).
- The third stage will run on a distributed database environment.

The assumptions to be considered during progress are:

- The time needed for checking the availability of lock is assumed to be 1 ms.
- The time needed for setting as well as releasing a lock is assumed to be 1 ms.
- Time needed to complete data processing is randomly selected between 20 to 100 ms.

Table 3.1 (Simulation parameters)

Parameter	Description	Values for Centralized	Values for Distributed
Num-table	Number of tables in a database	10	15
Min-num-tuples	Minimum number of tuples in each table	1	1
Max-num-tuples	Maximum number of tuples in each table	1000,5000	1000,5000
Min-col	Minimum number of columns in each table	1	1
Max-col	Maximum number of columns in each table	10	10
Num-trans	Number of transactions in the system	Up to 1000	Up to 1000

Min-trans-size	Minimum number of operation	1	1
Max-trans-size	Maximum number of operation	20	20
Queue-length	Maximum queue length	10, 20	10,20
num-site	Number of sites	1	3

Many researchers [18, 28, 45] assume such times to lock and release locks, and if these times are changed, both alternatives will change (row and field), so the environment is still comparable. Another assumption is the size of the read and write sets in a transaction, it is assumed to be equal, because of simplifying the analysis and we did not have actual data that could serve as an indication of what would be realistic distribution of the size of the read or write sets. The operations used by transactions are the DML operations [11, 31, 33], which are the Insert, Update, Delete and Select.

3.5 Deadlock Detection Approach

Several approaches are used to detect a deadlock in a database, one of them which is used in this study, is the timeout approach. In this approach, a transaction sets a time out for every lock required, if the lock is not granted within this time, it assumes that the deadlock has occurred. The simplicity and ease of implementation

are two advantageous for this method, in addition it does not cause network traffic when detecting deadlock in distributed database, while the timeout must be tuned carefully in order to not detect false deadlocks or to not allow the deadlock to persist in the system for a long time [18].

In this study, the check for an available resource is assumed to take one millisecond, if the lock is not granted immediately, one millisecond is needed before the next trial, when the lock is granted, a random number between 20 and 100 milliseconds is chosen as a processing time, (because we don't have real data), so 51 trials for acquiring a lock is sufficient in this study to determine if the resource is blocked or deadlocked. Because if a transaction is granted a lock to a resource and needs 100 milliseconds to complete its operation at the resource, then after completion, one millisecond is needed to release a lock, another transaction may try 51 times to get a lock at the same resource with one millisecond between each two successive trials, so it needs 102 millisecond which exceeds the total time for the first transaction by one, so in the case of not granted a lock, deadlock has occurred.

3.6 The Enhanced Algorithm Description for Locking Attributes

The database is assumed to be organized as a hierarchy tree (Figure 3.2). A lock could be obtained on the entire database, entire table, page, row or attribute according to compatibility matrix for granularity hierarchy [33] table 1.1. The transaction can lock a node in top-down order and unlock it in bottom-up order by using the rules mentioned in [33] in addition to:

1. The database row is considered as a node, and can be locked in an intention mode (IS or IX).
2. The key of the row must be locked in a Shared (S) mode, when the transaction does not need the whole row.
3. The locking of attributes as database nodes must be done according to the database constraints.
4. Other attributes can be locked in S or X mode.
5. When a conflict occurs, or when the transaction needs to read or update the whole row, it locks the whole row.

Databases are assumed to be well normalized and have a set of assertions to satisfy its correct state [12, 34], these assertions are integrity constraints which specify the characteristics of a data

item independent of others, and consistency constraint which specifies a relationship among database items. For example, if a database is associated with the following assertions:

- a. $Z=X+Y$.
- b. $A=2B$.
- c. Or the value of an item W (say for example the medical status for a patient), is dependent on the values of other items (for example some results of medical analysis).

So, these items must be locked together when using attribute level locking, this is the responsibility of the database lock manager to accomplish this task, in this example case, the transaction must lock both X and Y when its need to lock Z (constraint (a) in the above example). As a theoretical example, suppose that the employee information table has the following structure:

(Employee ID, Employee Name, Employee Job Title, Marital Status, Number of Dependents, ... etc). So we can't modify Marital Status or Number of dependents, without locking both of them, because it is not realistic that a single employee has a number of dependents greater than zero, and vice versa.

3.7 Conclusion

The proof of concept for enhancing concurrency control and decreasing deadlock occurrences, by allowing locks to be done at field level in a database system, will be presented by building a database lock manager, hierarchy tree, and implementing a simulation program, to show the system behavior, in addition to presenting the enhanced algorithm for locking attributes based on the multi granularity locking, using two phase locking protocol with dynamic locking. Transactions will be generated randomly as well as the DML operations, time out approach will be used to detect deadlock in a system. Database integrity will be preserved by generating random assertions to ensure the validity of the presented approach.

CHAPTER FOUR

CENTRALIZED DATABASE RESULTS

In this chapter, the findings of the simulation run will be drawn according to the parameters mentioned in table 3.1; the simulation will run first as a central database with row level locking as the minimum lockable unit, then the simulation will be re-run to cover the fields by expanding the tree (Figure 3.1) one level down to show the results with field level locking as the lockable unit, and finally results will be drawn to show the distributed environment. Each run will have a comparative analysis by drawing some performance measurements.

4.1 Locking Performance

During the performance analysis for the system alternatives, some fundamental formulas [17, 23] will be used to measure system performance metrics (System throughput, Mean service time, Mean waiting time and, locking overhead)

- Arrival rate (λ) which is the number of jobs divided by the total simulation time (T). $\lambda = \text{number of jobs} / T$
- System throughput (X) is the number of completed jobs divided by total simulation time (T). $X = \text{number of completed jobs} / T$

- Mean service time (S) is the total simulation time served divided by number of completed jobs $S = T / \text{number of completed jobs}$
- Mean waiting time is the sum of waiting times divided by the number of completed jobs $W = \sum \text{waiting time} / \text{number of completed jobs}$

Then the system utilization can also be produced by $U = X * S$.

4.2 Simulation Runs at Row Level Locking

The simulation runs according to the following parameters:

Minimum number of tuples is 1.

Maximum number of tuples is 1000.

Minimum number of columns in a table is 2.

Maximum number of columns in a table is 10.

Number of tables is 20.

Minimum Transaction size is one operation.

Maximum Transaction size is 25 operations.

Maximum queue length is 10.

Then, the database size (number of lockable database items)

is the average

number of tuples times the number of tables.

$$\text{DB size} = [(1000+1)/2] * 20.$$

$$= 10,000 \text{ items.}$$

The database size that is used by the simulation is (10,141) lockable units, this is because of counting the actual number of database items after building the tree, according to these parameters, the simulation runs for 100 transactions, in order to show the execution behavior for these transactions against the database, results are produced in Table 4.1:

Table 4.1 (Results after simulation run at row-level-locking)

Transaction ID	Arrival Time	Start Service	End Service	Waiting Time	Execution Time	Number of Locks	Number of Operations	Status
1	0	0	121	0	121	5	2	Done
2	31	31	359	0	328	10	4	Done
3	31	47	219	0	172	8	3	Done
4	31	78	2141	1035	2063	32	18	Done
5	31	94	2031	1009	1937	29	17	Done
6	31	125	1172	87	1047	31	15	Done
7	31	125	2766	1649	2641	37	17	Done

8	47	141	1453	652	1312	22	12	Done
9	47	141	1297	417	1156	18	9	Done
10	47	156	984	0	828	26	10	Done
11	47	172	891	0	719	24	9	Done
12	47	219	453	0	234	8	3	Done
13	47	219	391	0	172	5	2	Done
14	47	234	703	0	469	13	5	Done
15	78	250	578	0	328	10	4	Done
16	78	266	516	0	250	12	4	Done
17	78	281	484	0	203	8	3	Done
18	78	297	703	0	406	16	6	Done
19	94	312	703	0	391	13	5	Done
20	94	312	434	0	122	6	3	Done
21	94	422	1547	74	1125	27	14	Done
22	94	453	859	196	406	6	3	Done
23	94	484	578	51	94	3	2	Done
24	109	547	1328	0	781	23	10	Done
25	109	578	1141	0	563	24	9	Done
26	125	609	6891	4547	6282	17	8	Deadlo cked
27	141	672	3219	1520	2547	27	11	Done
28	172	734	1172	0	438	13	6	Done
29	172	797	1141	0	344	14	5	Done
30	172	828	2641	1085	1813	21	8	Done
31	188	859	2312	769	1453	18	7	Done
32	188	922	2094	265	1172	28	13	Done
33	188	953	2578	860	1625	25	15	Done
34	188	984	3562	1226	2578	35	18	Done
35	188	1047	3016	1481	1969	9	8	Blocke d
36	203	1078	1812	249	734	19	8	Done
37	203	1109	1797	289	688	17	8	Done
38	203	1172	6734	4490	5562	6	8	Deadlo cked
39	219	1203	1812	0	609	24	8	Done
40	219	1234	1391	0	157	5	2	Done
41	219	1297	3375	1728	2078	13	6	Done
42	219	1328	1844	150	516	14	6	Done
43	234	1359	1578	0	219	11	4	Done

44	234	1422	5703	2594	4281	10	3	Done
45	266	1453	5516	2503	4063	9	2	Done
46	266	1484	1602	0	118	7	5	Done
47	266	1547	2953	304	1406	12	4	Done
48	281	1578	2891	225	1313	21	9	Done
49	281	1609	3781	1096	2172	22	8	Done
50	281	1672	1766	0	94	4	1	Done
51	297	1703	1791	0	88	18	10	Done
52	297	1734	4297	1471	2563	14	4	Done
53	297	1844	2703	170	859	15	9	Done
54	313	1922	3844	740	1922	16	7	Done
55	313	1922	3375	612	1453	22	15	Done
56	313	1953	2891	435	938	10	7	Done
57	313	2000	2312	0	312	12	4	Done
58	328	2062	2609	0	547	19	7	Done
59	328	2094	6141	2955	4047	24	18	Done
60	328	2094	3375	1049	1281	8	3	Done
61	328	2219	2292	0	73	9	3	Done
62	344	2250	2484	0	234	9	3	Done
63	344	2281	2484	0	203	6	2	Done
64	344	2344	3094	284	750	18	8	Done
65	344	2375	2703	0	328	13	5	Done
66	344	2406	6641	2964	4235	20	9	Done
67	344	2516	4703	1217	2187	19	16	Done
68	344	2547	2734	0	187	6	2	Done
69	344	2578	2672	0	94	3	1	Done
70	344	2641	7109	3344	4468	18	11	Done
71	344	2672	3656	567	984	9	7	Done
72	344	2703	4297	449	1594	34	15	Done
73	344	2766	7953	3216	5187	18	8	Done
74	344	2797	5359	1007	2562	29	17	Done
75	344	2828	6484	2066	3656	36	19	Done
76	344	2891	3656	93	765	21	9	Done
77	344	2922	6047	1879	3125	21	17	Done
78	344	2953	6078	1600	3125	27	20	Done
79	344	3016	3719	0	703	24	9	Done
80	344	3047	3922	0	875	25	9	Done

81	359	3094	3984	365	890	10	9	Blocked
82	359	3187	4484	240	1297	3	1	Done
83	359	3187	5672	1314	2485	27	18	Done
84	359	3187	6516	2377	3329	12	19	Done
85	359	3250	3469	0	219	9	3	Done
86	359	3281	3750	0	469	16	6	Done
87	359	3312	3687	0	375	14	5	Done
88	359	3406	3481	0	75	11	3	Done
89	359	3437	5016	115	1579	31	14	Done
90	359	3500	3656	0	156	6	2	Done
91	359	3531	3594	0	63	3	1	Done
92	359	3562	6703	1609	3141	28	10	Done
93	359	3625	6578	1539	2953	23	9	Done
94	359	3656	5234	238	1578	28	14	Done
95	359	3687	5641	536	1954	35	12	Done
96	359	3750	3781	0	31	2	1	Done
97	359	3781	3852	0	71	9	3	Done
98	359	3812	4266	102	454	8	4	Done
99	359	3875	4047	0	172	5	2	Done
100	359	3875	9359	3139	5484	30	16	Done

Simulation total time is: 11547 milliseconds.

Average transactions execution time is: 1.436 seconds.

Total number of transactions is: 100.

Number of completed transactions is 96.

Number of blocked transactions is 2.

Number of deadlocked transactions is 2.

According to the results shown in table 4.1, there are two deadlocked and two blocked transactions (waiting for a resource but no cycle exists), viewing the snapshots for these transactions to show the behavior of the system, the following figures present the flow of

```
Transaction 26 runs at time (609):  
Trying to lock [DB (1)-TABLE (8)-ROW (33)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (1)-ROW (10)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (2)-ROW (9)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (8)-ROW (47)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (9)-ROW (3)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (1)-ROW (51)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (9)-ROW (7)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (3)-ROW (15)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (2)-ROW (1)]  
Error encountered: lockmanager.ResourceLockException:  
Error, try to lock [-DB (1)-TABLE (2)-ROW (1)] in [X] mode which was locked in [S] mode by transaction [38]  
Re-Trying to lock [DB (1)-TABLE (2)-ROW (1)]  
Error encountered: lockmanager.ResourceLockException:  
Error, try to lock [DB (1)-TABLE (2) – ROW (1)] in [X] mode which was locked in [S] mode by transaction [38]  
Re-Trying to lock [-DB (1)-TABLE (2)-ROW (1)]  
Error encountered: lockmanager.ResourceLockException:  
Error, try to lock [-DB (1)-TABLE (2)-ROW (1)] in [X] mode which was locked in [X] mode by transaction [38]
```

execution, and when deadlocks or blocks occur.

Figure 4.1 (Execution behavior of transaction number 26 at row level locking)

Transaction 38 runs at time (1172):

Trying to lock [DB (1) - TABLE (2)-ROW (1)] in [S] mode..... DONE

Trying to lock [DB (1) - TABLE (5)-ROW (12)] in [X] mode..... DONE

Trying to lock [DB (1) - TABLE (3)-ROW (15)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [-DB (1)-TABLE (3)-ROW (15)] in [X] mode which was locked in [X] mode by transaction [26]

Re-Trying to lock [-DB (1)-TABLE (3)-ROW (15)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [-DB (1)-TABLE (3)-ROW (15)] in [X] mode which was locked in [X] mode by transaction [26]

Deadlock detected, transaction will release all resources...

Transaction 35 runs at time (1047):

Trying to lock [DB (1) - TABLE (1)-ROW (68)] in [X] mode..... DONE

Trying to lock [DB (1) - TABLE (8)-ROW (97)] in [X] mode..... DONE

Trying to lock [DB (1) - TABLE (6)-ROW (51)] [S] mode..... DONE

Trying to lock [DB (1) - TABLE (6)-ROW (61)] in [S] mode..... DONE

Trying to lock [DB (1) - TABLE (2)-ROW (9)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [DB (1) - TABLE (2) – ROW (9)] in [X] mode which was locked in [X] mode by transaction [26]

Re-Trying to lock [DB (1) - TABLE (2) – ROW (9)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [DB (1) - TABLE (2) - ROW (9)] in [X] mode which was locked in [X] mode by transaction [26]

Re-Trying to lock [DB (1) - TABLE (2) - ROW (9)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [DB (1) - TABLE (2) - ROW (9)] in [X] mode which was locked in [X] mode by transaction [26]

Figure 4.2 (Execution behavior of transaction number 38 at row level)

Transaction 81 runs at time (3094):

Trying to lock [DB (1) - TABLE (3) - ROW (29)] in [S] mode..... DONE

Trying to lock [DB (1) - TABLE (3) - ROW (30)] in [S] mode..... DONE

Trying to lock [DB (1) - TABLE (1)] in [IS] mode..... DONE

Trying to lock [DB (1) - TABLE (2) – ROW (7)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [DB (1) - TABLE (2) – ROW (7)] in [X] mode which was locked in [X] mode by transaction [79]

Re-Trying to lock [DB (1) - TABLE (2) – ROW (7)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [DB (1) - TABLE (2) – ROW (7)] in [X] mode which was locked in [X] mode by transaction [79]

Re-Trying to lock [-DB (1) - TABLE (2) – ROW (7)] in [X] mode..... DONE

Trying to lock [-DB (1) - TABLE (10) – ROW (14)] in [S] mode..... DONE

Trying to lock [-DB (1) - TABLE (2) – ROW (9)]

Error encountered: lockmanager.ResourceLockException:

Error, try to lock [-DB (1) - TABLE (2)-ROW (9)] in [X] mode which was locked in [X] mode by transaction [26]

Figure 4.4 (Execution behavior of transaction number 81 at row level

Transaction 79 runs at time (3172):

Trying to lock [DB (1)-TABLE (2) –ROW (7)] in [X] mode..... DONE

Trying to lock [DB (1)-TABLE (7) –ROW (36)] in [S]..... DONE

...

...

Trying to release [DB (1)-TABLE (2) –ROW (7)]...... DONE

Trying to release [DB (1)-TABLE (7) –ROW (36)]...... DONE

locking)

Figure 4.5 (Execution behavior of transaction number 79 at row level locking)

During the inspection of figures 4.1 through 4.5, for execution behavior of blocked and deadlocked transactions, we can see the following:

- Transaction 26 (T26): holds a lock on Table (3) Row (15) in [X] mode, and tries to lock Table (2) Row (1) in [X] mode, but it was held by Transaction (38) in [S] mode, which in an incompatible mode, so T26 will wait for this lock to be released by T38.
- Transaction 38 (T38): holds a lock on Table (2) Row (1) in [S] mode, and tries to lock Table (3) Row (15) in [X] mode, which was held by T26 in [X] mode, so T38 will wait for this lock to be released by T26. Then they wait for each other, so a deadlock has occurred.
- Transaction 35 (T35): tries to lock Table (2) Row (9) in [X] mode, but it was held by T26 in an incompatible mode, so T35 will wait in the blocked queue for random period of time, and retry to get the lock for several times until the lock is obtained, or maximum queue length reached. So, (T35) is blocked.

- Transaction 81 (T81): tries to lock Table (2) Row (7) which was held by (T79), in an incompatible mode, waits for random period of time and retries again, during this time, (T79) is finished its task so (T81) get its lock, and trying to obtain a lock on Table (2) Row (9), but it was held by (T26) in an incompatible mode, waits for this lock until maximum queue length is reached. So, (T81) is blocked.

We can see that, when a transaction holds a lock on database item and needs to obtain another lock at another database item already held by another transaction in a conflicting mode, the first transaction will be blocked, waiting for a resource (database item) to be released (transactions 35 and 81). In a situation like this, the second transaction waiting for the first data item is held by the first transaction (cycle exist), a deadlock occurs (transactions 26 and 38).

4.2.1 Performance Analysis

The performance measures for the results shown in Table 4.1 can be summarized according to the formulas mentioned in section 4.1 as follows:

- Arrival rate: $\lambda = 100 / 11.547$
= 8.66 transactions / second.

- System throughput $X=96/11.547$
= 8.31 transactions / second.
- Mean waiting time $W= 68.243 / 96$
= 0.7108 second.

In order to show the system behavior on different workloads, the simulator executes 30 times according to the parameters mentioned in section 4.2, after changing the parameter named (maximum number of tuples to be 5000), so the database size becomes 50,000 items.

$$\text{DB size} = [(5000+1)/2] * 20.$$

$$= 50,000 \text{ items.}$$

DB size calculated by the simulation is 51,152 items.

Table 4.2 (Results of 30 runs of simulation)

Total Number of Transactions	Completed Transactions	Simulation Time	Mean Service Time	Mean Waiting Time	Mean Number of Operations	Mean Number of locks	Arrival rate	Throughput
10	10	1.732	0.756	0	7	14	5.77	5.77
20	20	2.387	0.874	0.104	8	12	8.38	8.38
30	30	3.226	0.993	0.258	11	16	9.30	9.30

40	40	3.656	0.965	0.26 1	11	16	10.9 4	10.94
50	50	3.828	0.948	0.27 3	9	17	13.0 6	13.06
60	60	3.952	0.952	0.28 2	11	17	15.1 8	15.18
70	70	4.087	0.967	0.30 4	11	18	17.1 3	17.13
80	80	4.345	0.98	0.37 2	11	17	18.4 1	18.41
90	90	4.678	1.082	0.39 8	12	20	19.2 4	19.24
100	100	4.983	1.112	0.40 7	13	20	20.0 7	20.07
110	110	5.34	1.221	0.41 3	12	21	20.6 0	20.60
120	120	5.735	1.223	0.42 1	11	21	20.9 2	20.92
130	130	6.11	1.24	0.43 8	12	22	21.2 8	21.28
140	140	6.355	1.301	0.45 1	11	21	22.0 3	22.03
150	150	6.74	1.341	0.47 1	12	19	22.2 6	22.26
160	160	6.952	1.384	0.48 3	11	20	23.0 1	23.01
170	170	7.356	1.402	0.54 1	12	20	23.1 1	23.11
180	180	7.641	1.451	0.71 1	11	19	23.5 6	23.56
190	181	9.906	2.456	1.42	14	21	19.1 8	18.27
200	187	10.40 2	2.613	2.02 9	13	21	19.2 3	17.98
210	194	11.20 3	2.669	2.06 1	12	22	18.7 4	17.32
220	197	11.89	3.045	2.62 7	14	21	18.5 0	16.57

230	204	12.43 8	3.242	2.83	12	22	18.4 9	16.40
240	207	14.42 2	3.481	3.01 1	12	24	16.6 4	14.35
250	213	15.80 6	3.422	3.12 1	11	24	15.8 2	13.48
260	219	16.30 4	3.744	3.20 4	15	25	15.9 5	13.43
270	223	17.25 1	3.574	3.38 9	13	26	15.6 5	12.93
280	227	19.42	3.552	3.41 4	15	24	14.4 2	11.69
290	231	20.32 4	3.835	3.65 7	17	25	14.2 7	11.37
300	234	22.50 3	3.609	3.80 2	14	25	13.3 3	10.40

According to the results shown in Table 4.2, we can see that the arrival rate and throughput are equal when the system runs up to 180 transactions at a time unit, i.e. all transactions entering the system are completed successfully. But when the number of transactions becomes 190 or higher, the system starts thrashing due to deadlock, or exceeding the waiting queue length, in case of blocked transactions, this means that the competition among transactions is increased, Figure 4.6, clarify this.

The mean service time increases when the number of transactions as well as the size of these transactions entering the system is increased, the same thing occurs with mean waiting time Figure 4.7. This happens because the system takes more time to coordinate and execute the transactions.

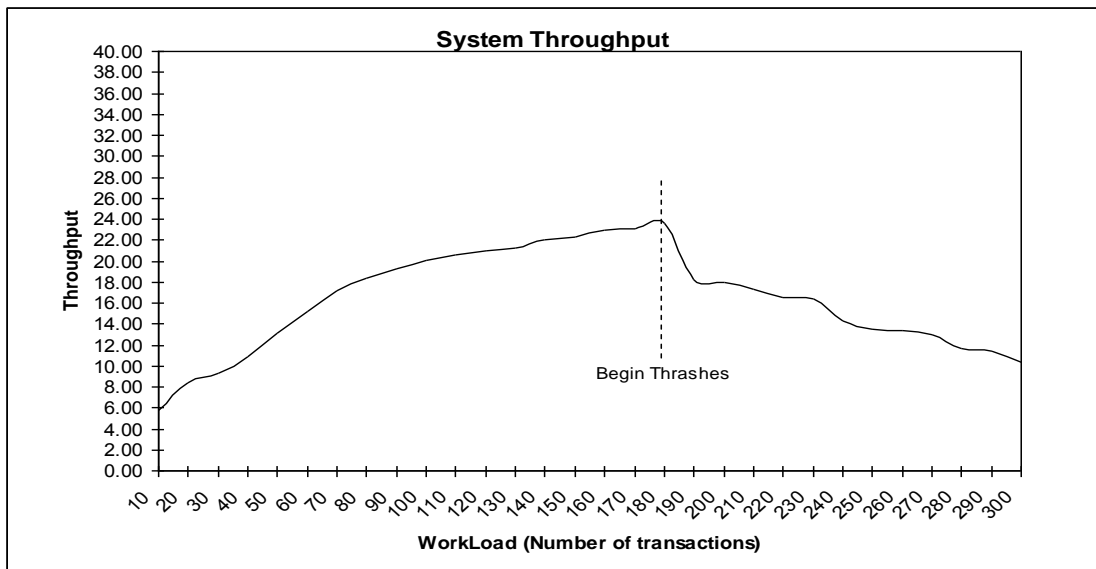


Figure 4.6 (System Throughput at row level locking)

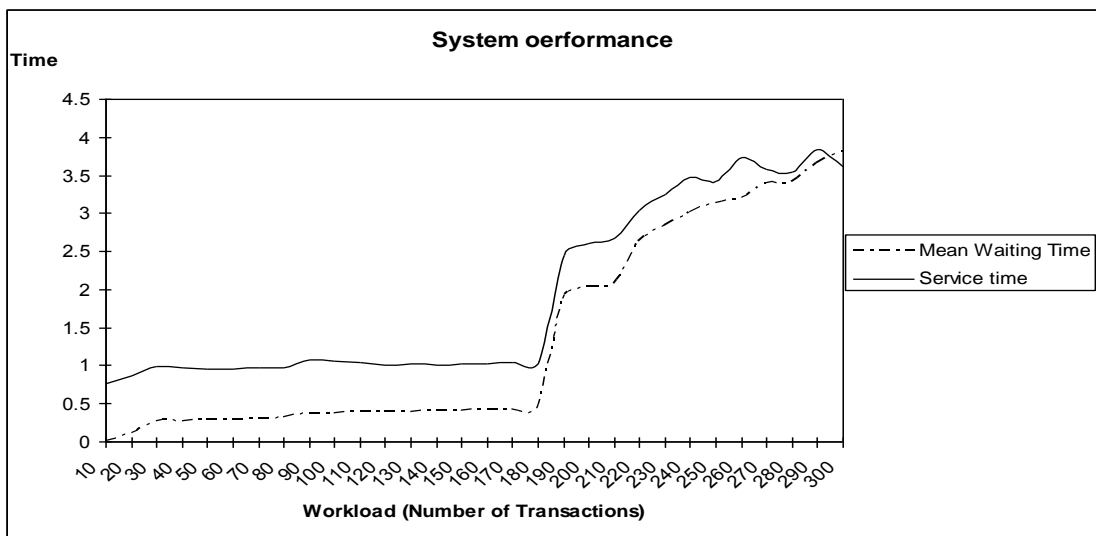


Figure 4.7 (System performance at row level locking)

The system throughput shown in figure 4.6 and the data listed in table 4.3, agree with (Tay et al 1985 [48]), who claimed that, the data contention workload (K^2N/D) should not exceed 1.5, where K is the mean number of locks, N is the number of transactions in the system, and D is the database size. So, when we apply this formula to the data listed in Table 4.2, we can get the results shown in Table 4.3:

Table 4.3 (Data Contention workload at Database Size 51,152)

Number of Transactions in the System	Mean Number of Locks	Data Contention workload
10	14	0.0383
20	12	0.0563
30	16	0.1501
40	16	0.2002
50	17	0.2825
60	17	0.3390
70	18	0.4434
80	17	0.4520
90	20	0.7038
100	20	0.7820
110	21	0.9484
120	21	1.0346
130	22	1.2301
140	21	1.2070
150	19	1.0586
160	20	1.2512
170	20	1.3294
180	19	1.2703
190	21	1.6381
200	21	1.7243
210	22	1.9870
220	21	1.8967

230	22	2.1763
240	24	2.7025
250	24	2.8151
260	25	3.1768
270	26	3.5682
280	24	3.1530
290	25	3.5434
300	25	3.6655

The rule of thumb presented in [48] which is "The DC-workload on a system should not exceed 1.5", is used by several researchers such as Thomasian 93 and 98 [39, 40] who claimed that the rule of thumb used by [42] is acceptable because the maximum number of locks requested per transaction is smaller than the database size, however (Roak et al, 1996) [30] claimed that "it rarely happens in conventional applications because most access is random and most transactions lock only a small amount of the database. It can be mathematically predicted to occur much sooner, if a significant portion of the transactions are accessing a large part of the database sequentially" [30].

Figure 4.8, shows the system locking overhead, which represent the mean number of locks needed by transactions

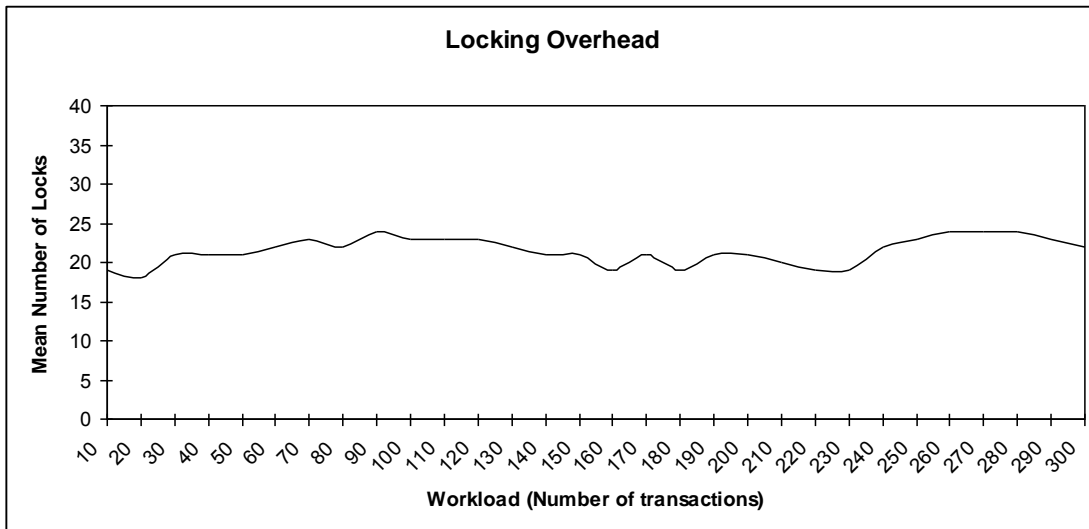


Figure 4.8 (System locking overhead at row level locking)

4.3 Simulation Runs at Field Level Locking

The simulation runs according to parameters listed in Section 4.2, the database size (number of lockable database items) is calculated by multiplying average number of tuples, number of tables, and average number of columns. Because the locking will be done at field level plus the key for the table, so by assuming that each table must have at least one field as a key, then the average number of columns is decreased by one, because the fields can't be locked without locking the key first in a shared mode.

DB size = average number of tuples * average number of columns * number of tables.

= 500 * [(maximum number of columns – 1) + (minimum number of columns -1)]/2 * 20.

$$= 500 * 5 * 20$$

= 50,000 items.

The database size as the simulations sees it is (49,981) lockable units, the deference between the database calculated above and the actual database size as the simulation sees it, is due to the number of fields that compose the key for each table which is varied from table to table (expected to be one field as a key for each table) and, because of the number of database constraints, according to these constraints, the lock manager can't set a lock on each individual field in order to preserve database consistency. The following results are produced for 100 transactions as shown in Table 4.4:

Table 4.4 (Results after simulation run at field-level-locking)

Transaction ID	Arrival Time	Start Service	End Service	Waiting Time	Execution Time	Number of Locks	Number of Operations	Status
1	0	0	195	0	195	9	2	Done
2	0	46	312	0	266	14	4	Done
3	0	78	218	0	140	8	3	Done
4	1	78	2078	419	2000	34	18	Done
5	1	78	3312	1426	3234	33	17	Done

6	2	93	2796	993	2703	34	15	Done
7	3	93	1625	0	1532	41	17	Done
8	3	109	1031	0	922	32	12	Done
9	4	109	1312	174	1203	27	9	Done
10	4	125	1328	0	1203	26	10	Done
11	6	125	3875	2394	3750	32	9	Done
12	8	140	875	0	735	19	3	Done
13	8	156	875	0	719	12	2	Done
14	8	156	906	0	750	17	5	Done
15	9	171	984	0	813	19	4	Done
16	9	187	2546	1148	2359	13	4	Done
17	9	203	1250	0	1047	13	3	Done
18	11	203	1484	0	1281	23	6	Done
19	11	218	1656	0	1438	21	5	Done
20	11	234	1296	0	1062	11	3	Done
21	12	234	953	0	719	31	14	Done
22	12	250	1359	0	1109	14	3	Done
23	14	265	1281	0	1016	9	2	Done
24	14	265	2828	1734	2563	26	10	Done
25	14	281	671	0	390	24	9	Done
26	14	296	953	0	657	23	8	Done
27	14	312	921	0	609	31	11	Done
28	14	312	609	0	297	14	6	Done
29	14	328	546	0	218	19	5	Done
30	16	343	671	0	328	24	8	Done
31	16	359	812	0	453	21	7	Done
32	16	375	1859	409	1484	42	13	Done
33	16	375	1281	0	906	41	15	Done
34	16	375	1484	0	1109	42	18	Done
35	16	390	1406	0	1016	14	8	Done
36	16	406	921	0	515	31	8	Done
37	16	421	1406	561	985	29	8	Done
38	16	421	3140	1568	2719	11	8	Done
39	16	437	1312	0	875	36	8	Done
40	16	437	734	0	297	20	2	Done
41	18	453	859	0	406	19	6	Done
42	18	468	1046	0	578	21	6	Done
43	18	484	1078	0	594	19	4	Done
44	21	484	656	0	172	11	3	Done
45	21	500	640	0	140	11	2	Done

46	21	515	765	0	250	12	5	Done
47	21	515	734	0	219	17	4	Done
48	22	531	937	0	406	27	9	Done
49	22	546	1593	454	1047	27	8	Done
50	25	562	968	0	406	7	1	Done

51	25	562	1906	574	1344	31	10	Done
52	25	578	812	0	234	19	4	Done
53	25	578	734	0	156	17	9	Done
54	25	593	1078	0	485	21	7	Done
55	27	609	875	0	266	25	15	Done
56	27	625	731	0	106	11	7	Done
57	27	625	3406	1000	2781	18	4	Done
58	27	640	703	0	63	23	7	Done
59	27	640	734	0	94	31	18	Done
60	27	656	2265	217	1609	17	3	Done
61	27	671	2218	93	1547	14	3	Done
62	27	671	3421	1032	2750	12	3	Done
63	28	687	2140	0	1453	9	2	Done
64	28	703	1046	0	343	23	8	Done
65	28	718	953	0	235	13	5	Done
66	28	718	1203	0	485	28	9	Done
67	28	734	1281	0	547	26	16	Done
68	28	734	828	0	94	10	2	Done
69	28	750	812	0	62	5	1	Done
70	28	765	1031	0	266	19	11	Done
71	28	781	937	0	156	14	7	Done
72	29	796	2390	0	1594	38	15	Done
73	29	812	4312	1842	3500	25	8	Done
74	29	828	3015	589	2187	34	17	Done
75	29	828	2875	222	2047	38	19	Done
76	29	843	2921	743	2078	26	9	Done
77	29	859	1937	0	1078	27	17	Done
78	29	875	2453	138	1578	31	20	Done
79	29	875	2062	0	1187	29	9	Done
80	30	890	2546	0	1656	31	9	Done
81	30	906	2687	273	1781	13	9	Done
82	30	921	953	0	32	3	1	Done
83	30	921	1200	0	279	34	18	Done
84	30	937	2296	0	1359	18	19	Done
85	30	953	3093	706	2140	17	3	Done
86	30	968	2890	332	1922	21	6	Done
87	30	968	2781	522	1813	19	5	Done
88	31	984	2015	0	1031	17	3	Done
89	31	1000	2156	92	1156	33	14	Done

90	31	1015	2359	0	1344	11	2	Done
91	31	1015	2093	0	1078	9	1	Done
92	31	1031	5078	2953	4047	38	10	Done
93	31	1046	2906	1008	1860	27	9	Done
94	45	1062	1953	0	891	31	14	Done
95	45	1062	1937	0	875	43	12	Done
96	45	1078	2609	0	1531	8	1	Done
97	62	1093	6281	3445	5188	14	3	Done
98	62	1093	2578	0	1485	12	4	Done
99	62	1109	3343	367	2234	11	2	Done
100	62	1140	3390	916	2250	38	16	Done

Simulation total time is: 6360 milliseconds.

Average transactions execution time is: 1.155 seconds.

Total number of transactions is: 100.

Number of completed transactions is 100.

Number of blocked transactions is 0.

Number of deadlocked transactions is 0.

According to the results shown in Table 4.4, the two transactions (26 and 38) were deadlocked when running the application at row level locking, but were not deadlocked when running the simulation at field level locking. Figures 4.9 through 4.13 show the system behavior while executing transactions 26 and 38; they were deadlocked and transactions 35 and 81 were blocked. Transaction 26 tries to lock DB (1)-TABLE (3)-ROW (15)-FIELD (3)] in Exclusive mode, which was done by locking the fields 1, 2 and 3,

because fields 1 and 2 are the key for table 3. At the same time transaction 38 tries to lock [DB (1)-TABLE (3)-ROW (15)-FIELD (8)], which was done after locking fields 1 and 2 also. Both transactions execute their task simultaneously against the same row, without affecting the database consistency. In such cases, the lock manager obtains the lock for each transaction on different fields by locking the key in Shared mode and locking the required fields in Exclusive or Shared mode.

In such cases, and according to our example of registration system (Chapter 3), building information table is shown in Table 4.5, and the material schedule table is shown in Table 4.6 in database (1) Figure 4.9, which has the following scheme:

Table 4.5: Building information table

Building Number	Room Number	Room Name	Room Capacity
1	1	AAU - Room	80
...
...
...

Table 5.6: Material schedule table

<u>Material Number</u>	<u>Section Number</u>	Building Number	Room Number	Time From	Time To	Instructor Number	Real Number	Maximum Number	Section Status
...
...
...
100	1	Hall 1	Room 1	10	11	300	20	35	Open
...
...

Where (Building Number and Room Number) are the key for Table 4.5, and (Material Number and Section Number), are the key for Table 4.6. Then transaction (26) can update field (8) in table (4.6), real number in this scheme, in order to register a student, simultaneously with transaction (38), whose update field (4) in table (4.5), the room capacity. Transaction (26) need to read the room name, which is obtained by locking the key with room name in table (4.5), whereas transaction (38) may require to update the Maximum

Number, which is obtained by locking the key with the Maximum Number in table (4.6). Both transactions can proceed with their tasks together, which is not allowed when row level locking is the minimum lockable unit, figures 4.9 and 4.10 show this process.

```
Transaction 26 runs at time (296):  
Trying to lock [DB (1)-TABLE (8)-ROW (33)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (1)-ROW (10)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (2)-ROW (9)-FIELD (4)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (8)-ROW (47)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (9)-ROW (3)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (1)-ROW (51)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (9)-ROW (7)] in [S] mode..... DONE  
Trying to lock [DB (1)-TABLE (3)-ROW (15) - FIELD (8)] in [X] mode..... DONE  
Trying to lock [DB (1)-TABLE (2)-ROW (1) – FIELD (3)] in [S] mode ..... DONE  
  
*****  
Locking is done at Field 1, 2 and 3 because Fields 1 and 2 is the key.  
*****  
Trying to release [DB (1)-TABLE (8)-ROW (33)]..... DONE  
Trying to release [DB (1)-TABLE (1)-ROW (10)]..... DONE
```

Figure 4.9 (Execution behavior of transaction number 26 at field level locking)

Transaction 38 runs at time (421):

Trying to lock [DB (1) - TABLE (2)-ROW (1)] FIELD (4) in [X] mode..... DONE

Trying to lock [DB (1) - TABLE (5)-ROW (12)] in [X] mode..... DONE

Trying to lock [DB (1) - TABLE (3)-ROW (15)- FIELD (7)] DONE

Locking is done at Field 1, 2 and 7 because Fields 1 and 2 is the key.

Trying to release [-DB (1)-TABLE (2)-ROW (1)]-FIELD (4)]..... DONE

Trvng to release [-DB (1)-TABLE (5)-ROW (12)]..... DONE

Figure 4.10(Execution behavior of transaction 38 at field level locking)

Transaction 35 runs at time (390):

Trying to lock [DB (1) - TABLE (1)-ROW (68)] in [X] mode..... DONE

Trying to lock [DB (1) - TABLE (8)-ROW (97)] in [X] mode..... DONE

Trying to lock [DB (1) - TABLE (6)-ROW (51)] [S] mode..... DONE

Trying to lock [DB (1) - TABLE (6)-ROW (61)] in [S] mode..... DONE

Trying to lock [DB (1) - TABLE (2)-ROW (9)-FIELD (6)] in [X] mode... DONE

Trying to release [DB (1) - TABLE (1)-ROW (68)]..... DONE

Trying to release [DB (1) - TABLE (8)-ROW (97)]..... DONE

Figure 4.11((Execution behavior of transaction 35 at field level locking)

Transaction 81 runs at time (906):

```
Trying to lock [DB (1) - TABLE (3) - ROW (29)] in [S] mode..... DONE
Trying to lock [DB (1) - TABLE (3) - ROW (30)] in [S] mode..... DONE
Trying to lock [DB (1) - TABLE (1)] in [IS] mode..... DONE
Trying to lock [DB (1) - TABLE (2) – ROW (7) – FIELD (3)] in [X] mode .... DONE

*****

Locking is done at Field 1 and 3 because Field 1 is the key.

*****

Trying to release [DB (1) - TABLE (3) - ROW (29)]..... DONE
Trying to release [DB (1) - TABLE (3) - ROW (30)]..... DONE
```

Figure 4.12 (Execution behavior of transaction 81 at field level

Transaction 79 runs at time (875):

```
Trying to lock [DB (1)-TABLE (2)-ROW (7)-FIELD (4, 5, and 8)] in [X] mode..... DONE
Trying to lock [DB (1)-TABLE (7) –ROW (36)] in [S]..... DONE
...
...
Trying to release [DB (1)-TABLE (2) –ROW (7)-FIELD (4, 5, and 8)]..... DONE
Trying to release [DB (1)-TABLE (7) –ROW (36)]..... DONE
```

locking)

Figure 4.13 (Execution behavior of transaction 79 at field level

locking)

4.3.1 Performance Analysis

The performance measures for the results shown in Table 4.4 can be summarized according to the formulas mentioned in section 4.1 as follows:

- Arrival rate: $\lambda=100/6.360$
= 15.72 transactions / second.
- System throughput $X=100/6.360$
= 15.72 transactions / second.
- Mean waiting time $W= 28.344 / 100$
= 0.28344 second.

According to the results shown in Table 4.4 above, we notice that, the 100 transactions are completed successfully without blocks or deadlocks. The important thing is that, the mean transaction service time is decreased when running the application using field level locking; it was (1.436 seconds) in row level locking as the minimum lockable unit, then, decreased to (1.155 seconds) at field level locking as the minimum lockable unit. The same thing occur with the mean waiting time, which was (0.7108 seconds), decreased to (0.28344 seconds), while the throughput is increased. This occurred, because the competing parts among transactions are tightening and the transactions have a higher chance to execute concurrently at the same row.

After changing the maximum number of tuples in a table to 5000 instead of 1000, the database size becomes 250,000 items; (248,256 items as simulation sees it); then, the simulation runs 30 times in order to show the system behavior, results are shown in Table 4.7.

Table 4.7 (Results of 30 runs of simulation at field level locking)

Total Number of Transactions	Completed Transactions	Simulation Time	Mean Service Time	Mean Waiting Time	Mean Number of Operations	Mean Number of locks	Arrival rate	Throughput
10	10	2.36	0.501	0	7	18	4.24	4.24
20	20	2.375	0.604	0.021	8	19	8.42	8.42
30	30	3	0.671	0.091	11	22	10.00	10.00
40	40	3.64	0.682	0.121	11	21	10.99	10.99
50	50	3.821	0.721	0.162	9	23	13.09	13.09
60	60	4.185	0.763	0.206	11	24	14.34	14.34
70	70	4.282	0.804	0.211	11	26	16.35	16.35
80	80	4.5	0.851	0.238	11	24	17.78	17.78
90	90	4.656	0.942	0.241	12	24	19.33	19.33
100	100	4.94	0.953	0.253	13	26	20.24	20.24
110	110	5.247	0.961	0.274	12	27	20.96	20.96

120	120	5.431	0.98 4	0.27 9	11	27	22.1 0	22.10
130	130	5.661	0.99 7	0.28 7	12	28	22.9 6	22.96
140	140	5.738	1.15 4	0.32 4	11	28	24.4 0	24.40
150	150	5.91	1.11 7	0.33 5	12	29	25.3 8	25.38
160	160	6.2	1.13 3	0.34 4	11	29	25.8 1	25.81
170	170	6.341	1.19 3	0.35 1	12	30	26.8 1	26.81
180	180	6.171	1.15 8	0.37 9	11	29	29.1 7	29.17
190	190	6.203	1.21 3	0.42 2	14	32	30.6 3	30.63
200	200	6.421	1.28 8	0.46 9	13	31	31.1 5	31.15
210	210	6.622	1.32 1	0.47 3	12	30	31.7 1	31.71
220	220	6.594	1.40 4	0.47 9	14	31	33.3 6	33.36
230	230	6.722	1.43 1	0.48 1	12	36	34.2 2	34.22
240	240	6.969	1.48 7	0.50 4	12	36	34.4 4	34.44
250	245	8.24	1.66	0.91 4	11	39	30.3 4	29.73
260	248	9.406	1.96 2	1.21 2	15	38	27.6 4	26.37

270	252	10.40 1	2.36 1	1.53 2	13	38	25. 96	24.23
280	257	12.40 6	2.43 7	1.61 2	15	37	22. 57	20.72
290	258	13.24	2.90 4	2.61 8	17	37	21. 90	19.49
300	254	15.56 3	3.11 7	2.80 4	14	37	19. 28	16.32

We can see that, the system runs up to 240 transactions, as the sample of run shows, without problems; it starts thrashing, when the workload becomes 250 transactions at the run time interval or higher (Figure 4.14 and the results are presented in tables 4.7 and 4.8). The system is subjected to thrashes when the data contention workload exceeds the value (1.5) the shaded area in the table 4.8, showing this. Figure 4.15, shows the decreasing behavior for the mean service time and the mean waiting time, because the transaction does not need to wait while it can proceed with another one at the same row (average service time became less as well as mean waiting time).

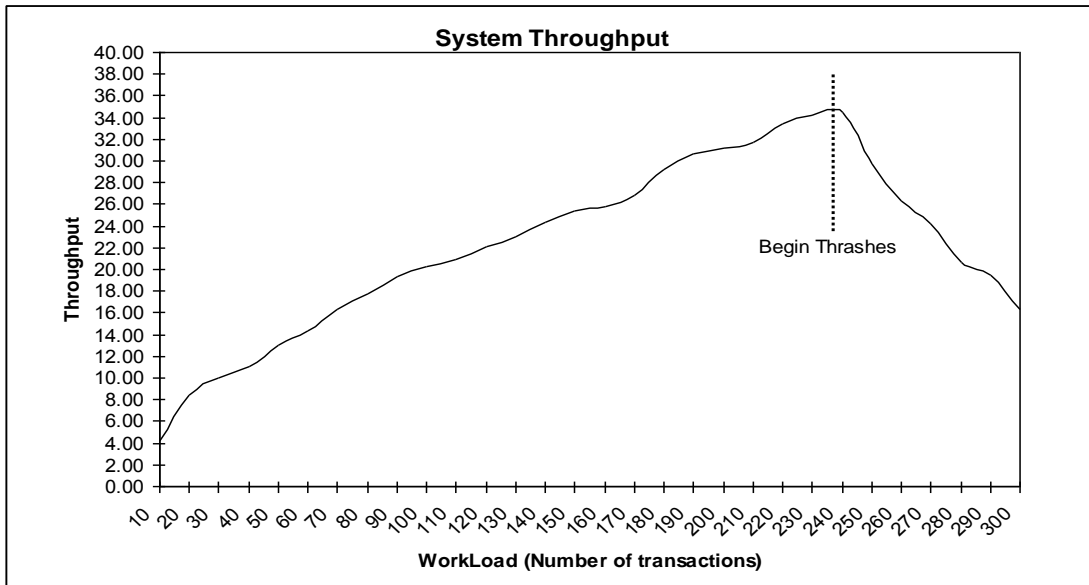


Figure 4.14 (System throughput at field level locking)

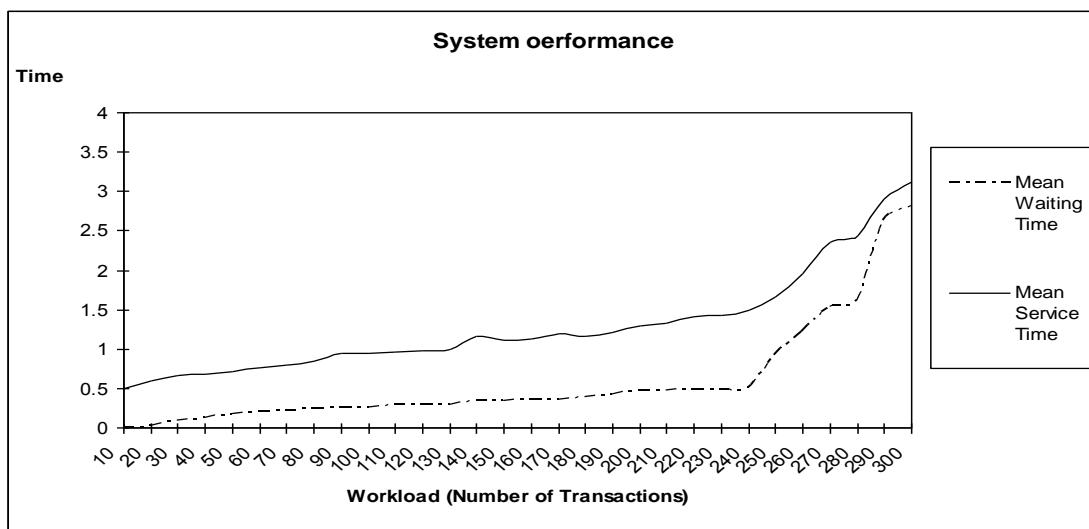


Figure 4.15 (System performance at field level locking)

But unfortunately, the locking overhead is higher, because the lock manager needs to manage extra locks (the fields or attributes) for transactions to execute their jobs figure (4.16).

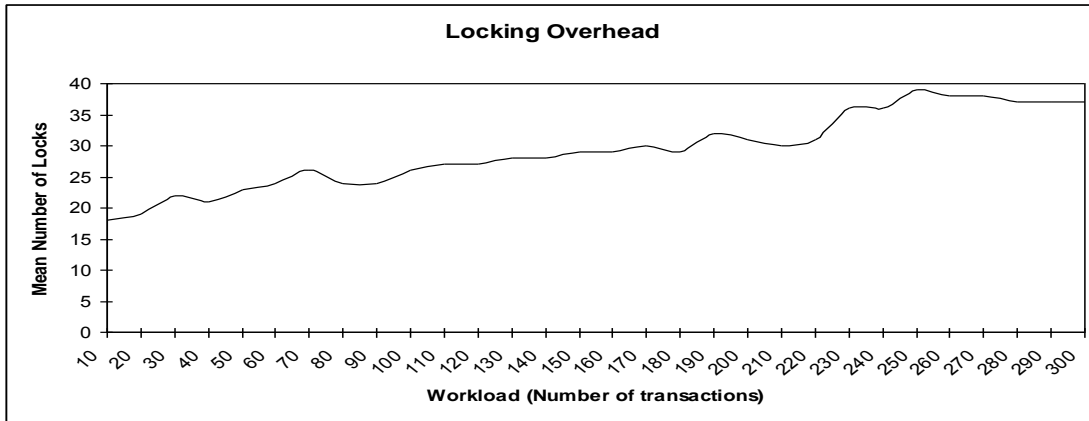


Figure 4.16 (System locking overhead at field level locking)

Table 4.8 (Data Contention workload at Database Size 248,256)

Number of Transactions in the System	Mean Number of Locks	Data Contention workload
10	18	0.013
20	19	0.029
30	22	0.058
40	21	0.071
50	23	0.107
60	24	0.139
70	26	0.191
80	24	0.186
90	24	0.209
100	26	0.272
110	27	0.323
120	27	0.352
130	28	0.411
140	28	0.442
150	29	0.508
160	29	0.542
170	30	0.616
180	29	0.610
190	32	0.784
200	31	0.774
210	30	0.761
220	31	0.852

230	36	1.201
240	36	1.253
250	39	1.532
260	38	1.512
270	38	1.570
280	37	1.544
290	37	1.599
300	37	1.654

The factors that affect performance when using fine granularity, mentioned in [5] can be overcome by the following:

- The overhead of locking can be reduced by allowing transactions to lock a database granule appropriate to their need.
- The increase of conflicts or data contention because of fine granularity, can be overcome by increasing database size when using fields as lockable unit, so the database size is increased many times more than the increase of locks needed, figure 4.20 shows this increase, while sections 4.2.1 and 4.3.1 clarify the large increasing behavior of database when using field level locking.
- The resource contention factor caused by releasing too many transactions from lock queues to spending more time in resource queues, can be overcome by the increasing developments in the hardware.

4.4 Comparing the Two Alternatives

The mean service time, mean waiting time, throughput and locking overhead for row level locking as well as for field level locking, are listed in table 4.9, in order to compare the two alternatives. The throughput for field level locking is higher than row level locking as shown in figure 4.17, because the competitions among transactions become less due to the increase of the database size, this agrees with Bernstein and Newcomer 2004, [6], who claimed that the probability of lock conflict is proportional to (K^2N/D) . So, the database size (D) is increased when the locking becomes at fields (because multiple transactions can be processed at the same row simultaneously). Alternative one (row level locking) thrashes when the number of transactions becomes 190 or higher, while alternative two (field level locking thrashes when the number of transactions becomes 250 or higher). At the same time, the mean service time and mean waiting time (figures 4.18 and figure 4.19), becomes less in general, because transactions can proceed immediately when no conflicts occur, and do not need to wait for a long time to get their locks, for the same reason. The mean service time as well as the mean waiting time for both alternatives becomes high when the system goes to thrashes, because the system needs more time to recover form deadlock.

Table 4.9 (Row level locking versus field level locking performance)

Number of Transactions	Row level locking				Field level locking			
	Mean Ser	Mean Wait	Througput	Mean Num	Mean Ser	Mean Wait	Througput	Mean Num
10	0.75	0	5.77	14	0.50	0	4.24	18
20	0.87	0.10	8.38	12	0.60	0.02	8.42	19
30	0.99	0.25	9.30	16	0.67	0.09	10.00	22
40	0.96	0.26	10.94	16	0.68	0.12	10.99	21
50	0.94	0.27	13.06	17	0.72	0.16	13.09	23
60	0.95	0.28	15.18	17	0.76	0.20	14.34	24
70	0.96	0.30	17.13	18	0.80	0.21	16.35	26
80	0.98	0.37	18.41	17	0.85	0.23	17.78	24
90	1.08	0.39	19.24	20	0.94	0.24	19.33	24
100	1.11	0.40	20.07	20	0.95	0.25	20.24	26
110	1.22	0.41	20.60	21	0.96	0.27	20.96	27
120	1.22	0.42	20.92	21	0.98	0.27	22.10	27
130	1.24	0.43	21.28	22	0.99	0.28	22.96	28
140	1.30	0.45	22.03	21	1.15	0.32	24.40	28
150	1.34	0.47	22.26	19	1.11	0.33	25.38	29
160	1.38	0.48	23.01	20	1.13	0.34	25.81	29
170	1.40	0.54	23.11	20	1.19	0.35	26.81	30
180	1.45	0.71	23.56	19	1.15	0.37	29.17	29
190	2.45	1.42	18.27	21	1.21	0.42	30.63	32
200	2.61	2.02	17.98	21	1.28	0.46	31.15	31
210	2.66	2.06	17.32	22	1.32	0.47	31.71	30
220	3.04	2.62	16.57	21	1.40	0.47	33.36	31
230	3.24	2.83	16.40	22	1.43	0.48	34.22	36
240	3.48	3.01	14.35	24	1.48	0.50	34.44	36
250	3.42	3.12	13.48	24	1.66	0.91	29.73	39
260	3.74	3.20	13.43	25	1.96	1.21	26.37	38
270	3.57	3.38	12.93	26	2.36	1.53	24.23	38
280	3.55	3.41	11.69	24	2.43	1.61	20.72	37
290	3.83	3.65	11.37	25	2.90	2.61	19.49	37
300	3.60	3.80	10.40	25	3.11	2.80	16.32	37

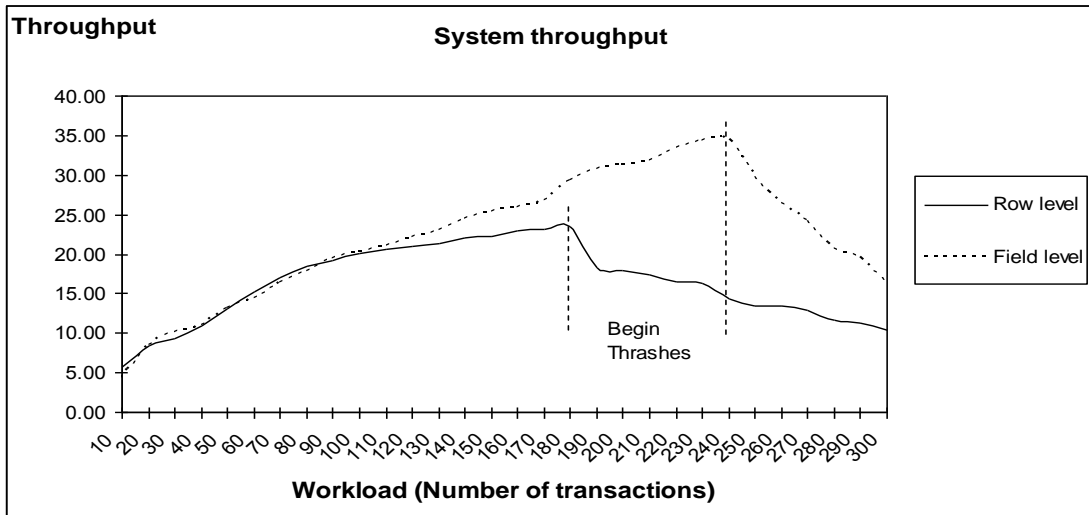


Figure 4.17 (Throughput for the two alternatives)

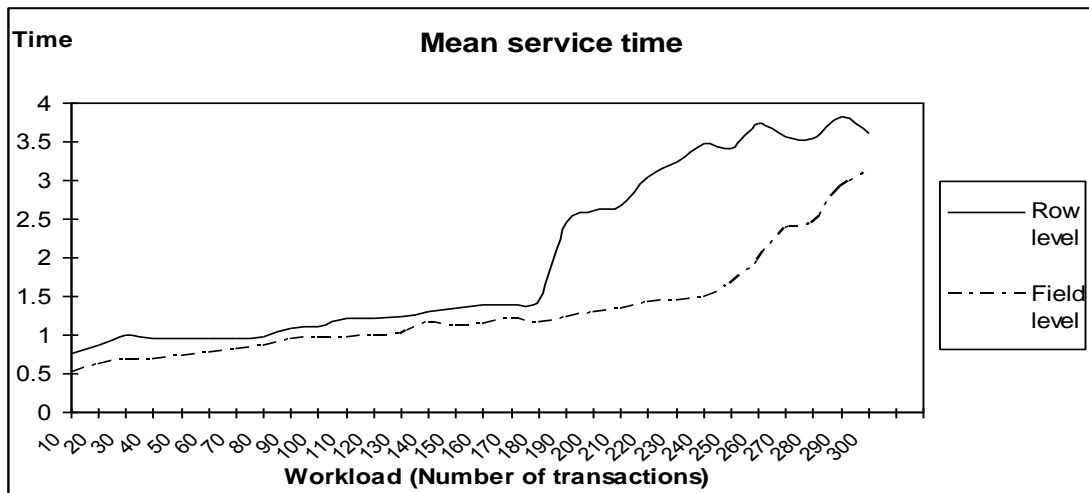


Figure 4.18 (Mean service time for the two alternatives)

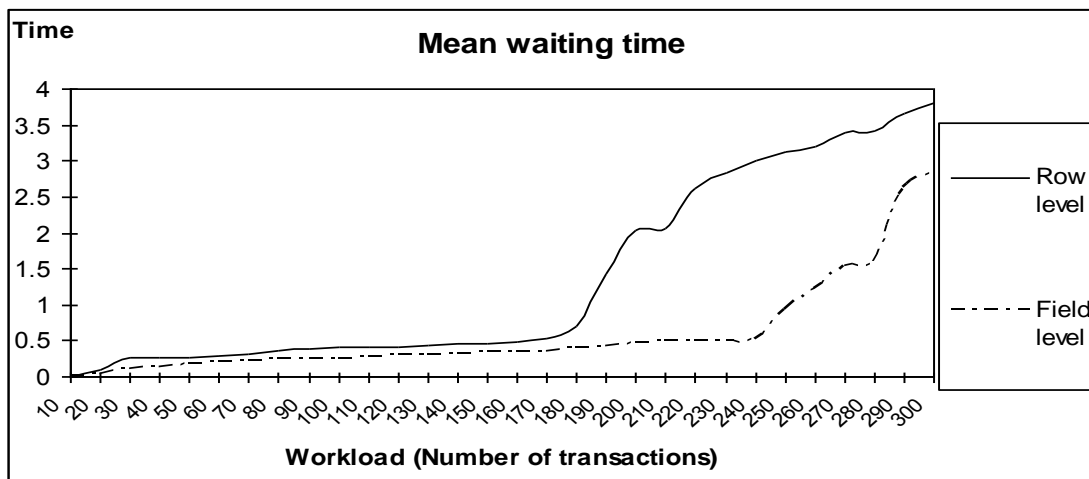


Figure 4.19 (Mean waiting time for the two alternatives)

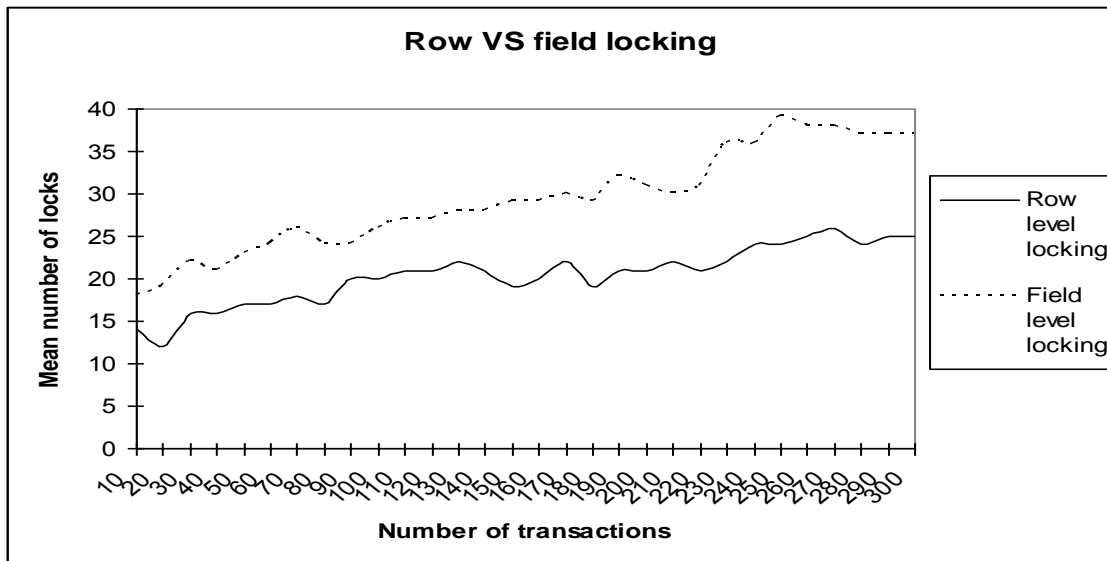


Figure 4.20 (Locking overhead for the two alternatives)

Figure 4.20, shows the locking overhead for the two alternatives, in field level locking, the locking overhead becomes higher because of the extra management of locking needed by the lock manager, but this is much less than what was expected before running the simulation.

4.5 Conclusion

Simulation was implemented to prove that obtaining a lock at field or attribute level in a database as a lockable is much better than obtaining a lock at the entire row, this comes from the increasing need to databases, and due to the availability of data as major requirements to satisfy the user needs. The discussion presented in sections 4.2 through 4.4, shows that the system at field level locking behave much

better than at row level locking, because multiple transactions can process the same database row simultaneously, which decreases the mean service time as well as the mean waiting time, because transactions do not need to wait for a long time to get their locks, which increases the availability of data. At the same time, field level locking can execute more transactions than row level locking before thrashing occurs, it works better on a heavy work load. Our results agree with [48] for data contention work load and also agree with [6] for probability of conflicts and deadlocks, because the database size as a dominator in their model decreases the ratio of conflicts and deadlocks. So in our approach, the database size is increased, because of using the fields as lockable units. Increasing the locking overhead, can be managed by choosing the appropriate data granule size for each transaction [5], for example, if a transaction needs too many fields of a database row, it locks the row, instead of locking each individual field.

CHAPTER FIVE DISTRIBUTED DATABASE RESULTS

In this chapter, the findings of the simulation run will be drawn on distributed database environment, the database is assumed to be partially replicated over sites. This study uses the single lock manager approach [33], where the lock manager resides in a single site, and all lock and unlock requests are made at that site. When a transaction needs to lock a data item, it sends a request to the site where the lock manager resides, and then the lock manager determines if the lock can be granted or not, if yes, the lock manager sends a message to the initiated site, else the lock request will be delayed. In case of update operations, and in order to preserve data consistency and integrity, the lock manager will be responsible for locking all the copies in all sites which are having a copy. In case of read; the transaction can read any copy from the sites at which a replica of the data item resides.

The studied database is assumed to be homogeneous distributed database, i.e. all sites have identical database management system software, in this case, the sites agree to cooperate in processing transactions, and also they are capable of exchanging information about transactions, to facilitate transaction

processing across multiple sites [33]. Transactions generated randomly with different sizes (number of operations in each transaction), and also with different modes of DML operations (Read, Write, Delete and Insert) are considered according to the nature of transactions generated by the system.

5.1 Distributed Database Population

The distributed database in this study, is composed of three sites, logically correlated as shown in Figure 5.1, each site consists of one database. According to the system parameters listed in Table 5.1, there are 15 tables partially replicated over these sites (even in structure), because it is our concern to measure the performance of the system by implementing global transactions (i.e. to make the most of transactions generated by the simulator global). In the sample run for distributed database, the tables distributed over three sites as one dimensional partial replication (some objects to all sites) [22]. The simulation program fills randomly the 15 tables with 5000 database objects (rows), and then it also randomly distributes the tables across the three sites. The parameter named, the degree of replication is considered to replicate the database objects over sites; in this sample, there are 3 out of 15 ($0.2 * 15$) tables are replicated as shown in Tables 5.2 and 5.3.

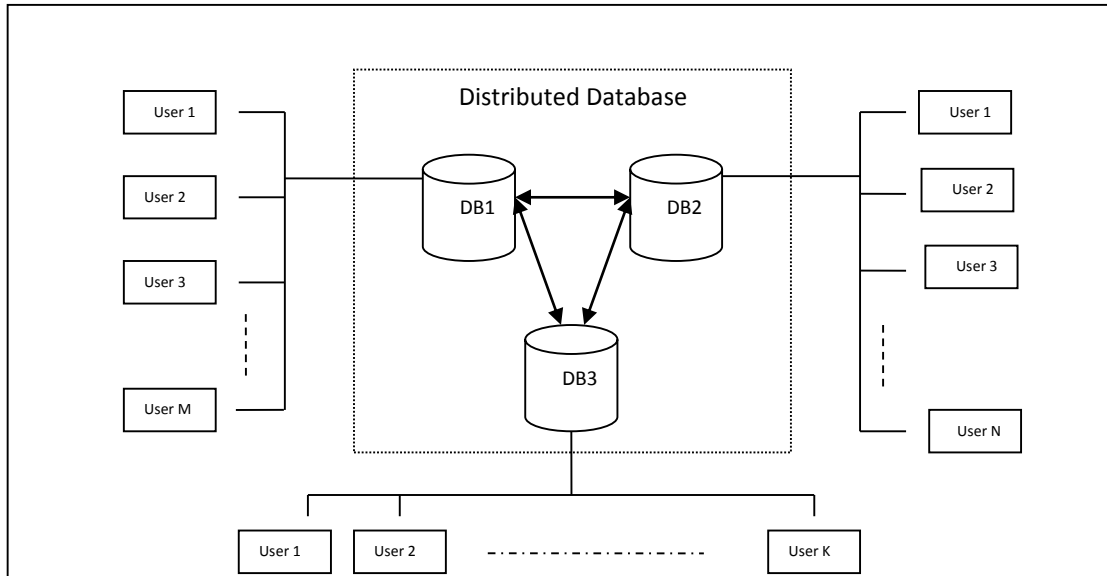


Figure 5.1 (Distributed database architecture for three sites)

Table 5.1 (Simulation parameters for distributed database)

Parameter	Description	Values
Num-site	Number of sites	3
DB-num	Number of databases in each site	1
DB-obj	Number of database objects for each site	5000
Rep_deg	Degree of replication	0.2 *
Num-table	Number of tables in a database	15
Num-trans	Number of transactions in the system	Up to 500
Min-trans-size	Minimum number of operation	1
Max-trans-size	Maximum number of operation	20
Op-mod	Operation mode	R, RW, W **
Queue-length	Maximum queue length	20
Time_check	Mean time to check a lock	1 ms
Time_set	Mean time to set a lock	1 ms

Time_rel	Mean time to release a lock	1 ms
Time_acc	Mean time to access a data object	20 – 100 ms

* The degree of replication (0.2) is expressed for replication 20% of logical data items over sites [22].

** R, RW and W are shorts for, all the operations of a transaction are Read, mixed of Read and Write or Write, respectively.

Table 5.2 (Distributing database objects into 15 tables)

Table ID	Number of Database Objects
1	500
2	300
3	350
4	420
5	280
6	690
7	280
8	340
9	420
10	220
11	235
12	130
13	275
14	305
15	255

Table 5.3 (Distributing of 15 tables across three sites)

Site 1	Site 2	Site 3
Table 1	Table 1	Table 1
Table 4	Table 2	Table 4
Table 6	Table 3	Table 5
Table 8	Table 4	Table 7
Table 9	Table 10	Table 12
Table 13	Table 13	Table 13
Table 14	Table 11	Table 15

We can notice that, the tables (1, 4 and 13) are replicated to all sites (three sites in this study). This distribution of database objects will be used for both alternatives (row level and field level locking). The simulation will run first at row level locking as the minimum lockable unit, and then the simulation will be re-run to cover the fields of rows. Each run will have a comparative analysis by drawing some performance measurements. The effects of the system parameters will be studied in section 5.5

Transactions will be generated randomly with different mode of DML operations, if the transactions have all the operations with read mode, they will be read only transactions, if they are mixed of read

and write, they will be mixed, and if they have write mode to all operations, they will be write transaction. This will be done, in order to show the system behavior under different modes of operation.

5.2 System Behavior at Row Level Locking

In order not to repeat the processing discussed in chapter four, the results shown in table 5.4 are presented to show the behavior of the system during 20 runs, all times are measured in seconds, 20 runs are sufficient to show the system behavior. We can see that, the system begins thrashing when the number of transactions entering the system becomes 170 or higher, this is clear by inspecting the two columns named arrival rate and throughput, the two metrics are equal up to running 160 transactions at a time unit, after this, arrival rate becomes greater than throughput, which means the system does not complete all transactions entering the system, (i.e. the competition among transactions as well as the probability of conflict becomes high), Figure 5.2, shows such thrashing. By tracing Table 4.2 and Table 5.4, we can see that the row level locking in centralized environment executes 160 transactions within 6.952 seconds, while in distributed environment, 160 transactions are finished within

7.8354 seconds, and this is because of the increase in number of users within a time unit and due to communication delays needed to execute transactions among different sites.

Table 5.4 (Results of 20 runs of simulation at row level locking)

Total Number of Transactions	Completed Transactions	Simulation Time	Mean Service Time	Mean Waiting Time	Mean Number of Operations	Mean Number of locks	Arrival rate	Throughput
10	10	1.4267	0.7689	0	7	22	7.01	7.01
20	20	2.3584	0.9801	0.13431	8	26	8.48	8.48
30	30	3.025	1.0857	0.32791	7	23	9.92	9.92
40	40	3.2516	1.1242	0.42636	11	24	12.30	12.30
50	50	3.5409	1.4397	0.46673	11	24	14.12	14.12
60	60	3.8335	1.5859	0.50028	8	26	15.65	15.65
70	70	4.169	1.7332	0.51722	9	25	16.79	16.79
80	80	4.4759	1.7574	0.56111	8	25	17.87	17.87
90	90	4.9159	1.8322	0.57541	9	26	18.31	18.31
100	100	5.1458	1.8949	0.64581	12	29	19.43	19.43
110	110	5.4032	1.8631	0.68244	11	28	20.36	20.36
120	120	5.7717	1.8841	0.80289	9	30	20.79	20.79
130	130	6.0841	1.9201	0.94886	10	32	21.37	21.37

140	140	6.523 2	1.998 7	1.016 51	9	31	21.4 6	21.46
150	150	7.022 2	2.456 3	1.275 64	8	29	21.3 6	21.36
160	160	7.835 4	3.060 1	2.005 4	8	31	20.4 2	20.42
170	165	17.62 33	4.879 1	4.323 1	7	28	9.65	9.31
180	170	21.46 02	5.528 7	4.733 4	7	27	8.39	7.92
190	178	24.41 59	6.802 4	5.066 7	7	32	7.78	7.29
200	181	27.81 93	8.840 2	6.358 2	7	31	7.19	6.51

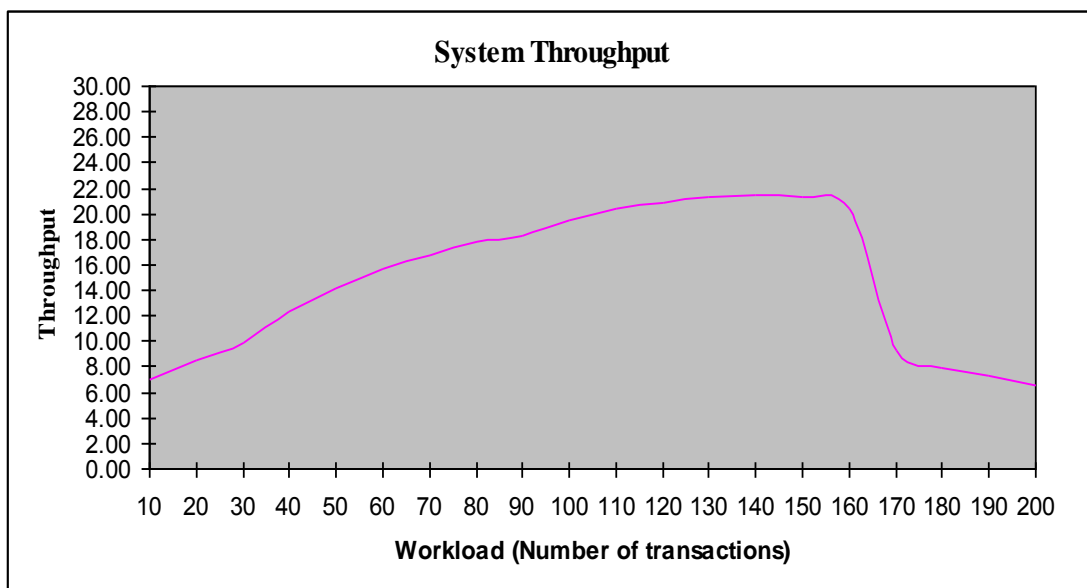


Figure 5.2 (System Throughput at row level locking)

Mean service time and mean waiting time increase when the number of transactions entering the system increases, Figure 5.3 and 5.4 shows this increase. Figure 5.5 shows the mean number of locks needed by transactions at row level locking.

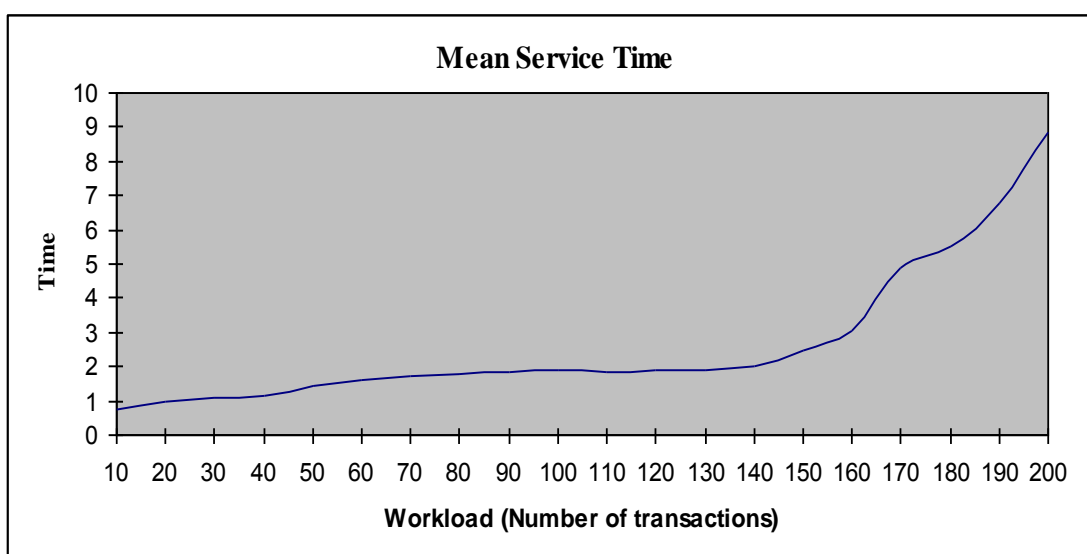


Figure 5.3(Mean service time at row level locking)

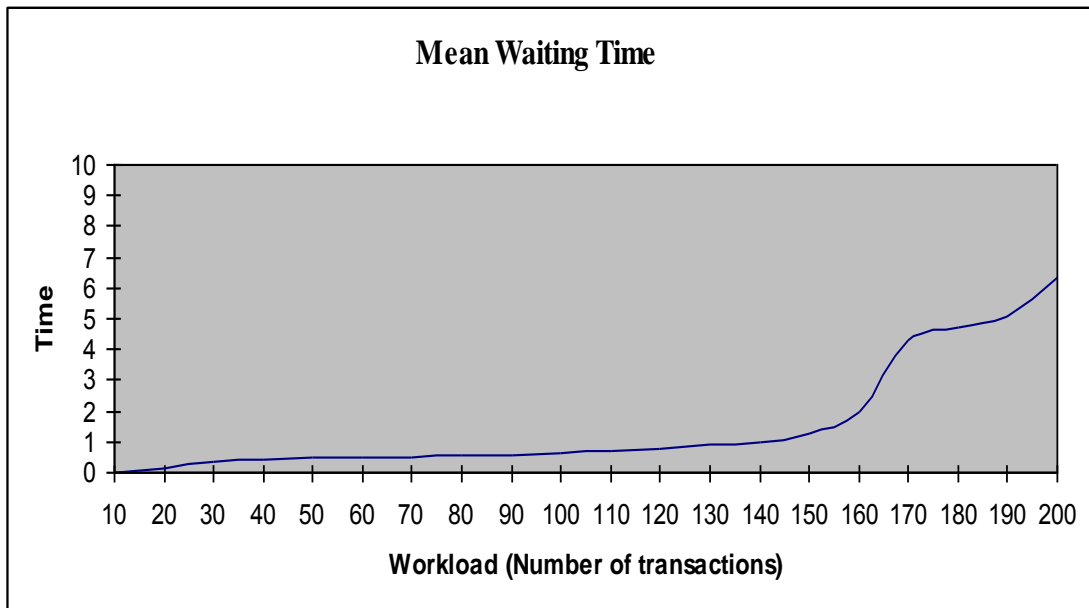


Figure 5.4(Mean waiting time at row level locking)

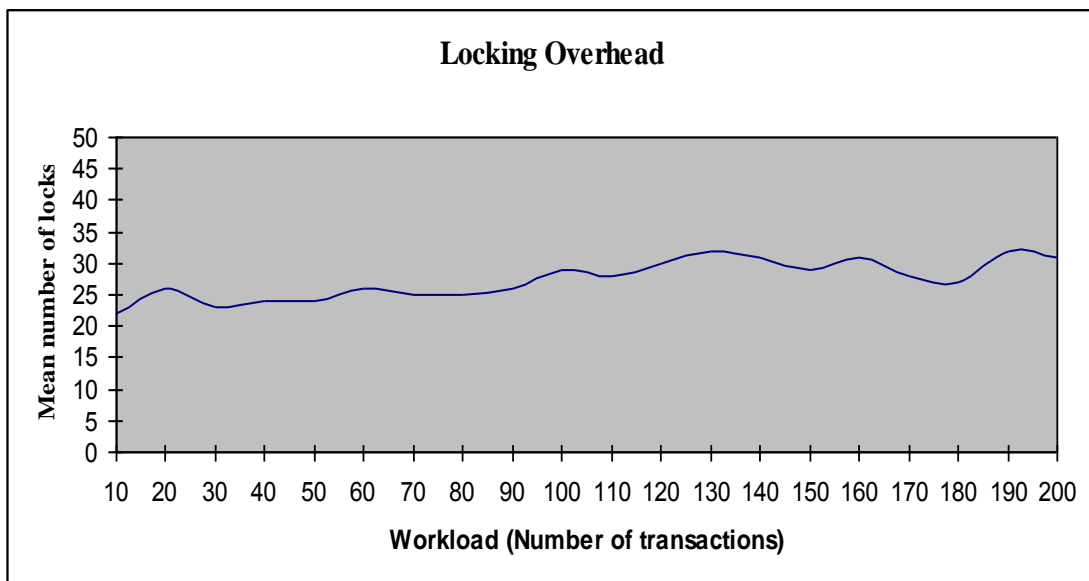


Figure 5.5 (System locking overhead at row level locking)

Table 5.4 shows that, the system executes successfully 165 transactions out of 170 due to deadlock, Table 5.5 show a sample execution behavior for the 170 transactions, the time here is measured in milliseconds.

Table 5.5 (Sample results of 170 transactions at row-level-locking)

Transaction ID	Arrival Time	Start Service	End Service	Waiting Time	Execution Time	Number of Locks	Number of Operations	Sites Used	Operation Mode	Status
1	0	0	306	0	306	16	6	2	R	Done
2	0	17	2786	1334	2769	27	10	2	RW	Deadlocked
3	0	17	630	0	613	25	10	3	RW	Done
4	5	41	848	0	807	31	12	3	RW	Done
5	5	56	161	123	105	7	2	1	R	Done
6	8	56	1395	214	1339	33	15	2	W	Done
7	8	56	239	0	183	10	4	2	R	Done
8	8	72	708	0	636	26	9	2	RW	Done
9	11	72	1286	0	1214	39	16	2	W	Done
10	11	88	661	0	573	29	12	3	RW	Done
11	11	88	1442	0	1354	31	17	3	W	Done
12	11	103	208	0	105	11	4	1	R	Done
13	13	114	989	0	875	26	11	3	RW	Done
14	13	118	551	0	433	16	4	3	RW	Done
15	14	134	1083	0	949	32	15	3	RW	Done
16	16	149	1130	97	981	31	15	2	RW	Done
17	16	165	413	0	248	11	3	2	R	Done
18	16	180	2194	774	2014	29	14	2	RW	Deadlocked
19	16	180	538	0	358	18	4	2	RW	Done
20	16	196	991	0	795	34	9	3	RW	Done

21	16	196	3694	179 3	349 8	39	17	1	W	Done
22	16	212	1241	0	102 9	27	11	3	W	Done
23	22	227	257	0	30	6	2	3	R	Done
24	22	227	663	0	436	18	5	2	R	Done
25	22	243	1788	613	154 5	31	14	3	RW	Blocked
26	31	259	725	0	466	22	8	2	RW	Done
27	31	273	1850	312	157 7	34	14	2	W	Done
28	31	273	350	0	77	6	1	1	R	Done
29	44	289	991	47	702	37	11	1	RW	Done
30	44	304	428	0	124	11	4	2	R	Done
31	44	320	1174	88	854	34	11	1	RW	Done
32	44	336	393	0	57	7	2	1	R	Done
33	44	336	1909	767	157 3	33	9	2	RW	Blocked
34	51	351	518	0	167	12	3	1	R	Done
35	51	351	3284	188 1	293 3	34	11	1	W	Done
36	51	367	1565	0	119 8	38	16	2	RW	Done
37	51	383	1440	594	1057	22	7	3	RW	Done
38	66	383	4003	2203	3620	37	17	3	RW	Blocked
39	66	398	924	0	526	21	8	3	R	Done
40	66	398	1221	0	823	32	1	2	RW	Done
..
16 8	94	1598	5264	1914	366 6	35	5	3	RW	Done
16 7	94	1614	2546	302	932	20	7	2	RW	Done
16 8	94	1614	2889	374	127 5	32	1	2	RW	Done
16 9	94	1630	3358	0	172 8	39	7	3	RW	Done

170	94	1677	3389	421	1712	30	11	11	W	Done
-----	----	------	------	-----	------	----	----	----	---	------

Simulation total time is: 14314 milliseconds.

Average transactions execution time is: 2.159 seconds.

Total number of transactions is: 170.

Number of completed transactions is 165.

Number of deadlocked transactions is 2.

Number of blocked transactions is 3.

Transactions 2 and 18 are deadlocked, while transactions 25, 33 and 38 are blocked. Table 5.6 shows the snapshots for these transactions.

- Transaction 2 (T2): holds a lock on Table (2) Row (211) at site 2, in [S] mode, and trying to lock Table (6) Row (97) at site 1, in [X] mode, but it was held by Transaction (18) in [X] mode, which is incompatible mode, so T2 waits for this lock to be released by T18.

- Transaction 18 (T18): holds a lock on Table (6) Row (97) at site 1, in [S] mode, and trying to lock Table (2) Row (211) at site 2, in [X] mode, which was held by T2 in [S] mode, so T18 waits for this lock to be released by T2. Then the two transactions are waiting for each other, so a deadlock occurs.
- Transaction 25 (T25): blocked, because it is waiting for transaction 18 to release a lock placed on row (66) table (13) at site 1.
- Transaction 33 is waiting for transaction 2 and transaction 38 is waiting for transaction 33 to release a lock, respectively.

Figure 5.6, shows a part of dependency graph for those transactions, the graph has a cycle between transactions 2 and 18, which indicates the deadlock occurrences.

Table 5.6 (Execution behavior of deadlocked and blocked transactions at row level locking)

T2	T18	T25	T33	T38
Write items on row (46) table (13) at site 1		Read items from row(3) table (8) at site 1		
Read items from row (26) table (11) at site 2	Write items on table (14) at site 1		Read items from row(206) table(10) at site 2	Read items from row(121) table (12) at site 3
	Read items from row(97) table(6) at site 1		Write items on row(26) table(11) at site 2	Write items on row(401) table(4) at site 3
	Write items on row(112) table (6) at site 1			Read items from table(5) at site 3
	Read items from row(66) table(13) at site 1	Write items on row(66) table(13) at site 1		Write items on row(206) table(10) at site 2
Read items from row(211) table (2) at site 2				
Write items on row(97) table(6) at site 1	Write items on row(211) table(2) at site 2			

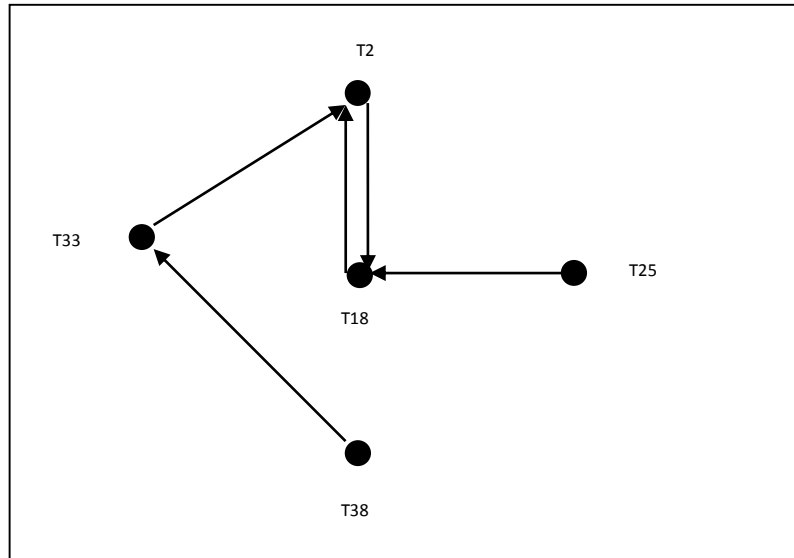


Figure 5.6 (Part of dependency graph for deadlocked and blocked transactions)

5.3 System Behavior at Field Level Locking

After modifying the hierarchy tree by adding the attributes level to be locked, simulation is executed 20 times on different workloads to show the system behavior, the results are presented in table 5.7. The new system (alternative two) executes up to 180 transactions successfully without deadlock, when the number of transactions becomes 190 or higher, the system begins thrashing as shown in figure 5.7. The important thing is that, 170 transactions are completed successfully on alternative two (at field level locking), while two transactions (2 and 18) are deadlocked, when using the row as minimum lockable unit.

Table 5.7 (Results of 20 runs of simulation at field level locking)

Total Number of Transactions	Completed Transactions	Simulation Time	Mean Service Time	Mean Waiting Time	Mean Number of Operations	Mean Number of locks	Arrival rate	Throughput
10	10	1.817	0.592	0	9	37	5.516	5.516
20	20	2.163	0.578	0.0019	6	36	9.238	9.238
30	30	2.691	0.643	0.0231	6	35	11.157	11.157
40	40	3.032	0.734	0.0985	7	43	13.184	13.184
50	50	3.491	0.806	0.1823	8	42	14.318	14.318
60	60	3.539	0.896	0.1972	7	40	16.964	16.964
70	70	3.917	0.913	0.2106	6	39	17.885	17.885
80	80	4.294	0.971	0.3701	9	41	18.644	18.644
90	90	4.563	1.012	0.3592	6	40	19.733	19.733
100	100	4.696	1.115	0.3909	6	42	21.286	21.286
110	110	4.933	1.42	0.4312	7	42	22.290	22.290
120	120	5.378	1.685	0.4899	6	46	22.317	22.317
130	130	5.731	1.681	0.5741	7	45	22.692	22.692
140	140	5.876	1.736	0.5921	6	44	23.818	23.818
150	150	6.216	1.812	0.7123	4	43	24.143	24.143

160	160	6.491	1.88 4	0.774 4	6	44	24.66 1	24.66 1
170	170	6.623	1.90 1	0.801 4	6	42	25.67 2	25.67 2
180	180	6.837	2.13 2	0.881 6	8	39	26.33 9	26.33 9
190	188	9.39	3.02 5	1.294 4	8	49	20.23 2	20.01 9
200	193	12.94 1	3.70 4	1.982 2	10	51	15.46 1	14.92 0

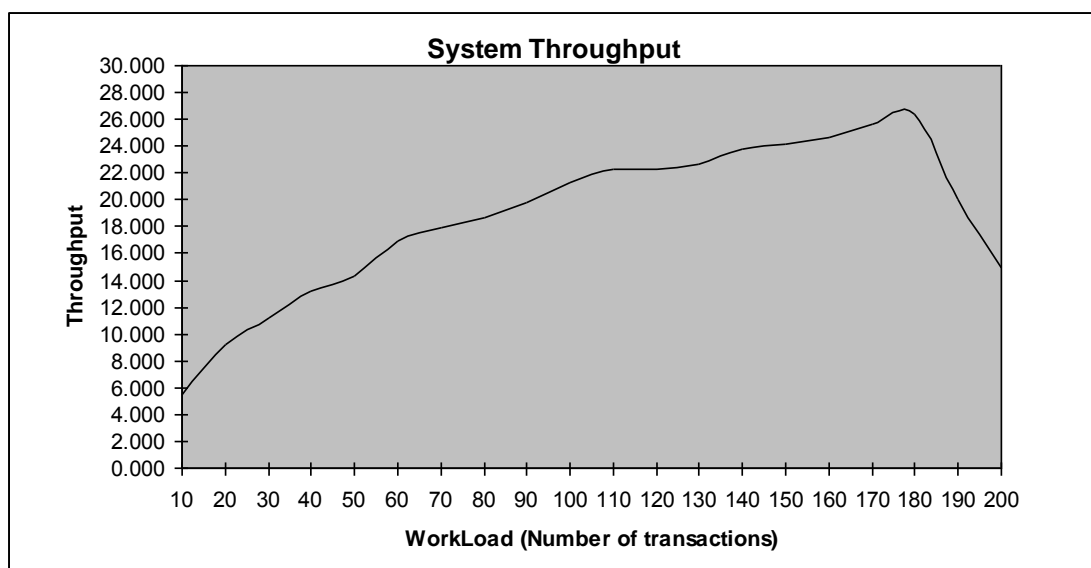


Figure 5.7 (System throughput at field level locking)

Mean service time and mean waiting time for alternative two, becomes less than those produced when using alternative one, figures 5.8 and 5.9 shows this behavior, because the transaction does not need to wait for long time to get its lock. But unfortunately, the mean number of locks is increased, as shown in Figure 5.10.

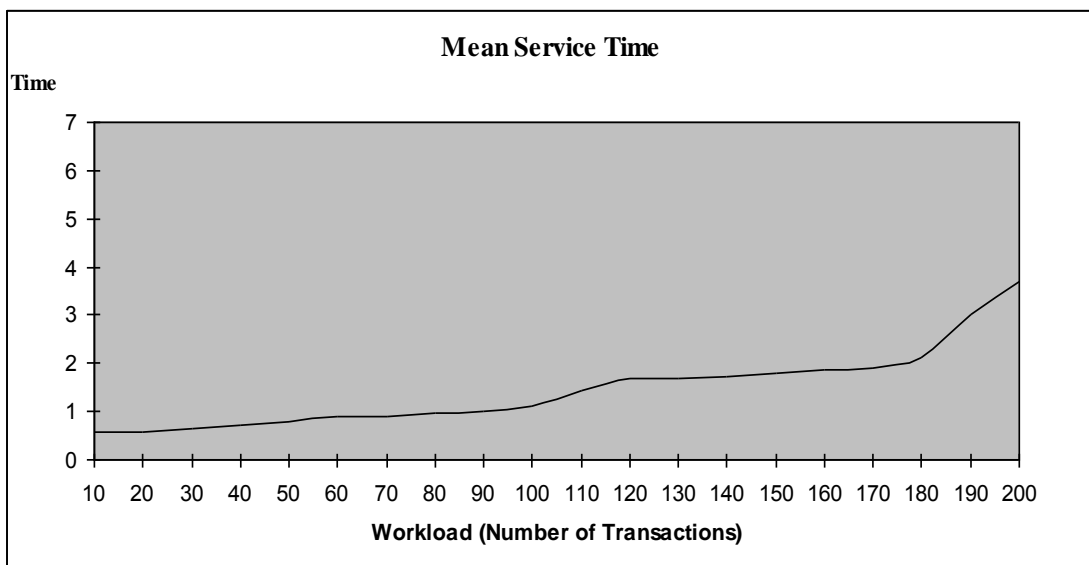


Figure 5.8 (Mean service time at field level locking)

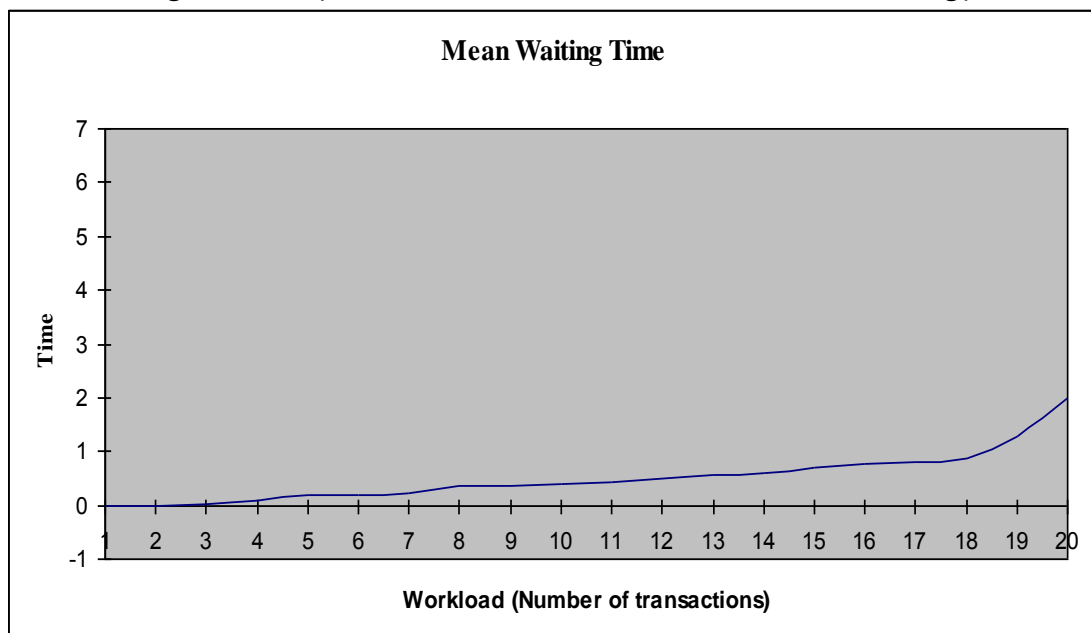


Figure 5.9 (Mean waiting time at field level locking)

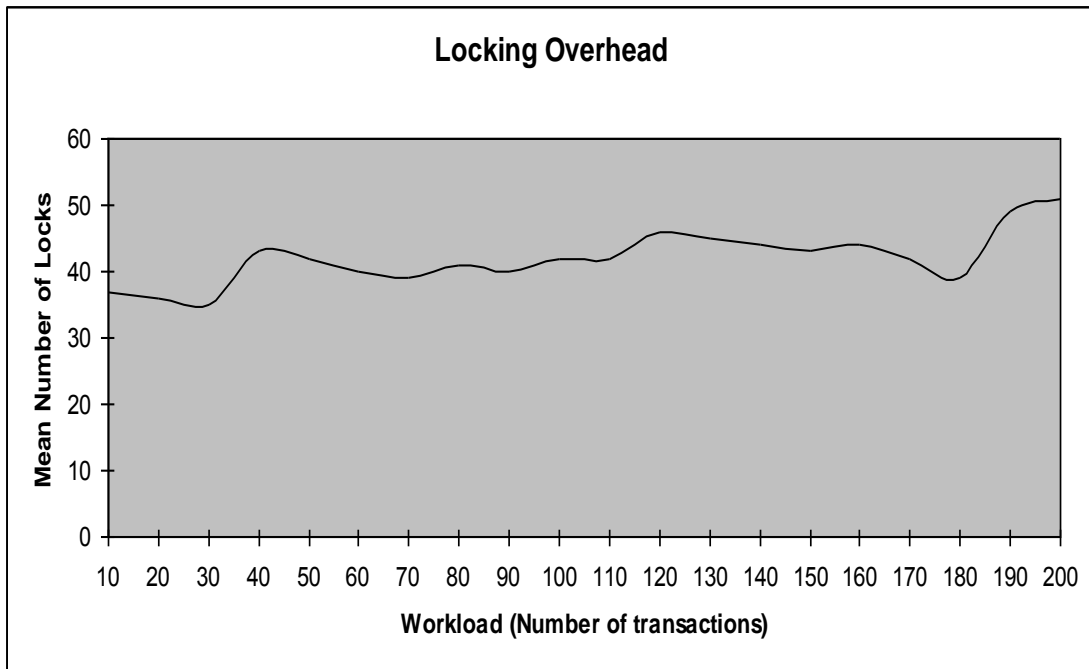


Figure 5.10 (System locking overhead at field level locking)

The simulation reruns at 170 transactions as workload by using alternative two (field level locking), to show the behavior of the system especially for transactions 2 and 18, which are deadlocked, Table 5.8 shows the results.

Table 5.8 (Sample results of 170 transactions at field-level-locking)

Transaction ID	Arrival Time	Start Service	End Service	Waiting Time	Execution Time	Number of Locks	Number of Operations	Sites used	Operation Mode	Status
1	16	0	47	0	47	27	5	2	R	Done
2	32	78	266	0	188	36	9	2	RW	Done
3	32	110	282	0	172	38	8	3	RW	Done

4	32	141	860	0	719	38	11	3	RW	Done
5	32	157	766	0	609	9	2	1	R	Done
6	32	157	1563	359	1406	39	14	2	W	Done
7	32	172	1078	0	906	21	3	2	R	Done
8	32	172	594	0	422	37	8	2	RW	Done
9	32	172	610	0	438	52	17	2	W	Done
10	32	188	750	0	562	44	10	3	RW	Done
11	32	203	782	0	579	39	19	3	W	Done
12	32	203	407	0	204	18	3	1	R	Done
13	32	219	422	0	203	31	12	3	RW	Done
14	32	235	532	0	297	27	5	3	RW	Done
15	32	235	453	0	218	41	14	3	RW	Done
16	32	250	547	0	297	39	14	2	RW	Done
17	32	266	594	0	328	18	3	2	R	Done
18	32	282	703	0	421	38	16	2	RW	Done
19	32	297	547	0	250	22	5	2	RW	Done
20	32	297	297	0	0	44	10	3	RW	Done
..
169	125	1042	3212	0	2170	44	19	3	RW	Done
170	131	1231	3221	449	1990	37	13	1	W	Done

Simulation total time is: 11036 milliseconds.

Average transactions execution time is: 1.898 seconds.

Total number of transactions is: 170.

Number of completed transactions is 170.

Number of deadlocked or blocked transactions is 0.

Simulation total time and average transaction execution time becomes less, and transactions 2 and 18 are completed successfully, Table 5.9, show the behavior of running these transactions.

Transaction 2 tries to lock [TABLE (2)-ROW (211)-FIELDS (7) at SITE 2] in Shared mode, which has been done by locking the fields 1 and 7, because field 1 is the key for table 2. At the same time transaction 18 tries to lock [TABLE (2)-ROW (211) - FIELD (3) at SITE 2], which has been done after locking fields 1 and 3 also, the same thing occurs when transactions 2 and 18 try to lock [SITE (1) – TABLE (6) – ROW (97)] fields 4, 6 and 6 respectively, each of them holds a lock at the fields they need. Both transactions execute their task simultaneously against the same row, without affecting the database consistency, In such cases, the lock manager obtains the lock for each transaction on different fields by locking the key in Shared mode and locking the required fields in Exclusive or Shared mode. Figure 5.11, shows that, the cycle which exists at row level locking, is removed from the dependency graph for the deadlocked and blocked transactions, when executing the field level locking system.

Table 5.9 (Execution behavior of transactions 2,18,25,33,38 at field level locking)

T2	T18	T25	T33	T38
Write items on field(5) row (46) table (13) at site 1	Write items on table (14) at site 1			
Read items from field(5) row (26) table (11) at site 2	Read items from field (5,6) row(97) table(6) at site 1	Read items from row(3) table (8) at site 1	Read items from fields(6,7,8) row(206) table(10) at site 2	Read items from row(121) table (12) at site 3
	Write items on row(112) table (6) at site 1	Write items on field(8) row(66) table(13) at site 1	Write items on field(8) row(26) table(11) at site 2	Write items on row(401) table(4) at site 3
Read items from field (7) row(211) table (2) at site 2	Read items from field(4) row(66) table(13) at site 1			Read items from table(5) at site 3
Write items on field (4) row(97)	Write items on field(3) row(211)			

table(6) at site 1	table(2) at site 2			Write items on field(3) row(206) table(10) at site 2
--------------------	--------------------	--	--	--

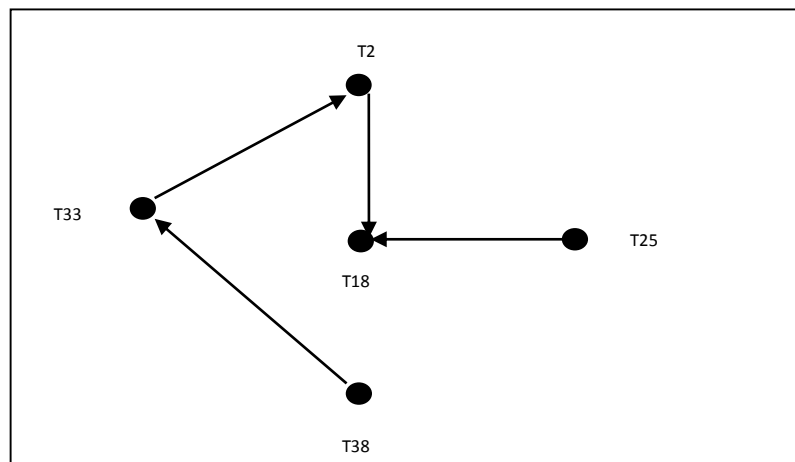


Figure 5.11 (Part of dependency graph for transactions 2,18,25,33,38 at field level locking)

5.4 Comparing the Two Alternatives

Table 5.10 shows the mean service time, mean waiting time, throughput and the mean number of locks for the two alternatives, in order to compare between them.

Table 5.10 (Row level locking versus field level locking performance)

Number of Transactions	Row level locking				Field level locking			
	Mean Service	Mean Waiting	Throughput	Mean Num	Mean Ser	Mean Wait	Throughput	Mean Num
10	0.768	0	7.01	22	0.5	0	5.516	37
20	0.980	0.134	8.48	26	0.5	0.00	9.238	36
30	1.085	0.327	9.92	23	0.6	0.02	11.157	35
40	1.124	0.426	12.30	24	0.7	0.09	13.184	43
50	1.439	0.466	14.12	24	0.8	0.18	14.318	42
60	1.585	0.500	15.65	26	0.8	0.19	16.964	40
70	1.733	0.517	16.79	25	0.9	0.21	17.885	39
80	1.757	0.561	17.87	25	0.9	0.37	18.644	41
90	1.8	0.57	18	2	1.012	0.3592	19.733	40
100	1.8	0.64	19	2	1.115	0.3909	21.286	42
110	1.8	0.68	20	2	1.42	0.4312	22.290	42
120	1.8	0.80	20	3	1.685	0.4899	22.317	46
130	1.9	0.94	21	3	1.681	0.5741	22.692	45
140	1.9	1.01	21	3	1.736	0.5921	23.818	44
150	2.4	1.27	21	2	1.812	0.7123	24.143	43
160	3.0	2.00	20	3	1.884	0.7744	24.661	44
170	4.8	4.32	9.	2	1.901	0.8014	25.672	42
180	5.5	4.73	7.	2	2.132	0.8816	26.339	39
190	6.8	5.06	7.	3	3.025	1.2944	20.019	49
200	8.8	6.35	6.	3	3.704	1.9822	14.920	51

During the inspection of figures 5.12 through 5.14, we can notice that, the system at field level locking in a distributed database behaves much better than the system at row level locking, this is due to the ability to access the same database row (object) by multiple transactions at the same time. Figure 5.15 shows the overhead of locking by the two systems, which clearly shows the extra load done by the field level locking alternative to manage extra locking.

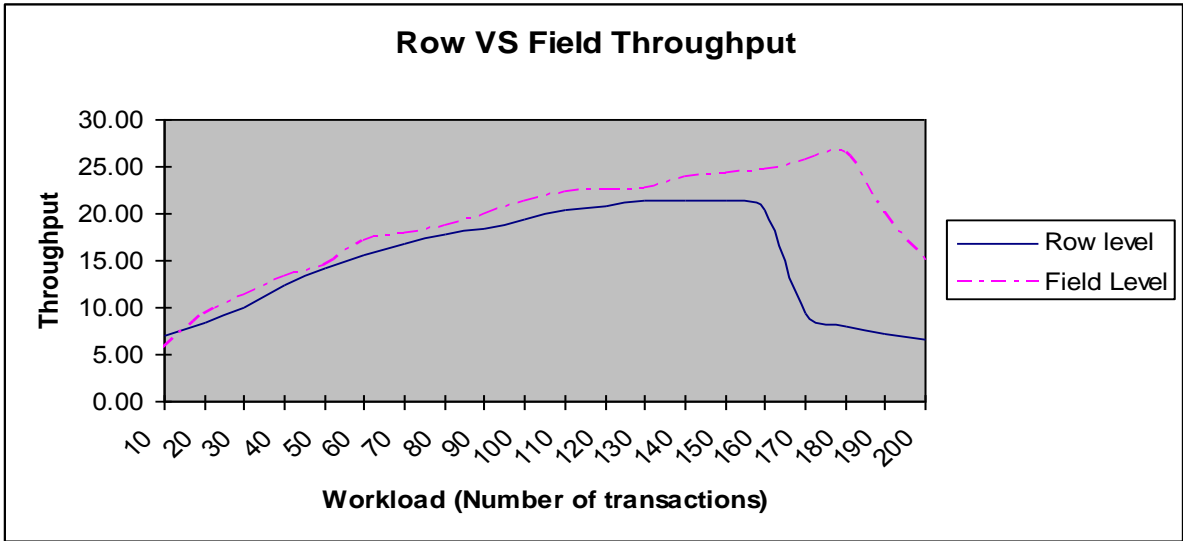


Figure 5.12 (Throughput for the two alternatives)

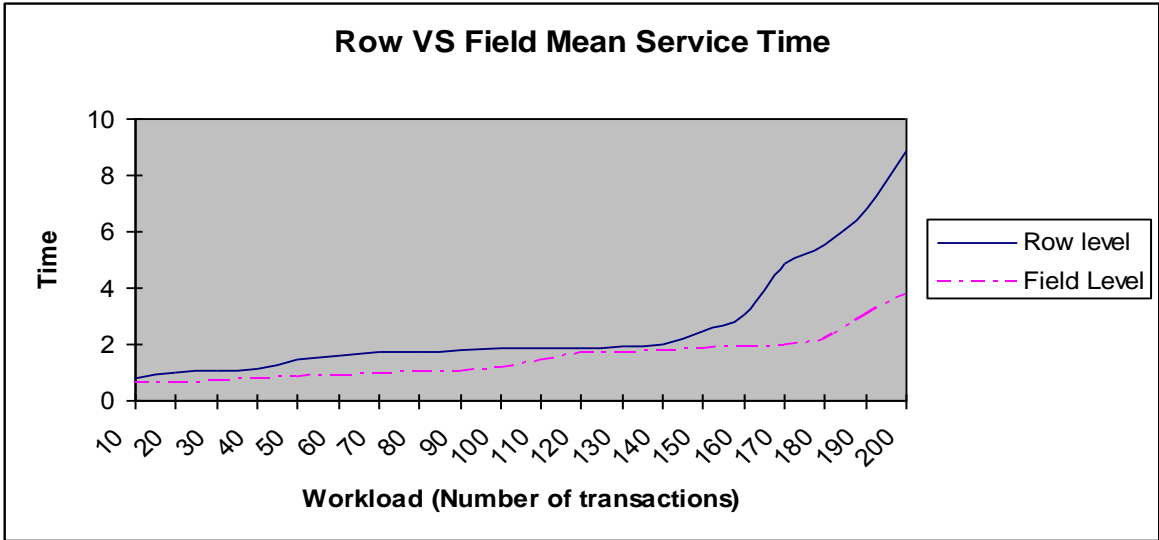


Figure 5.13 (Mean service time for the two alternatives)

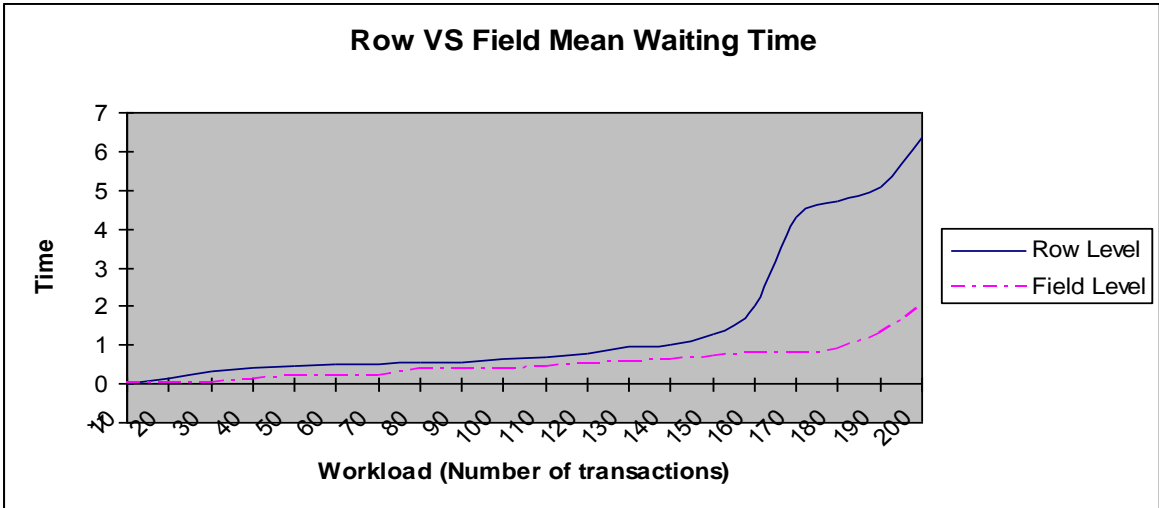


Figure 5.14 (Mean waiting time for the two alternatives)

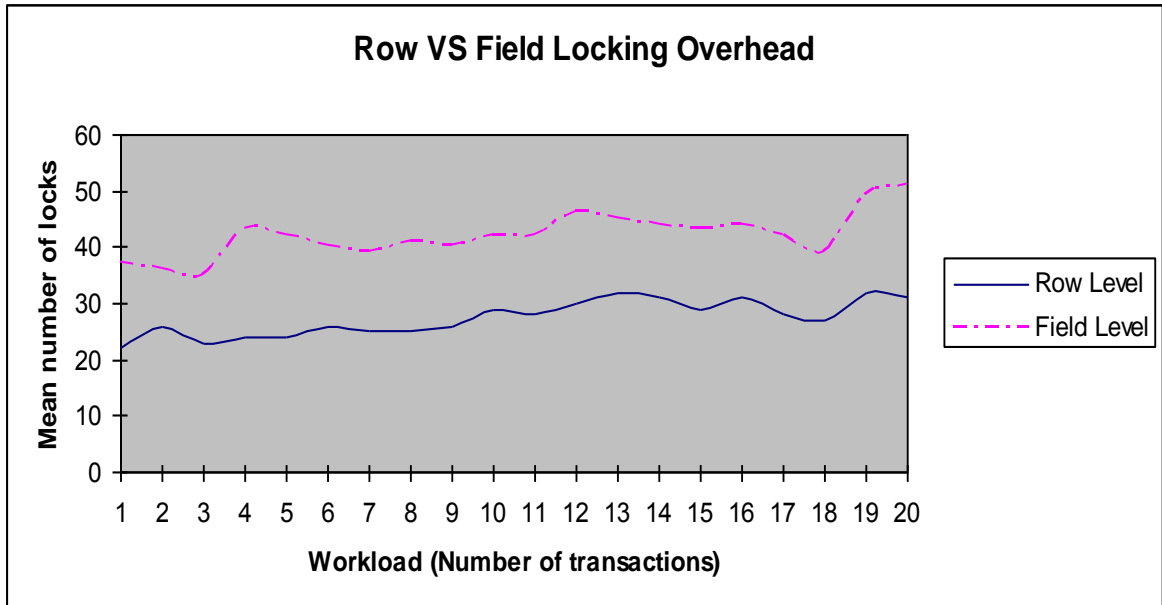


Figure 5.15 (Locking overhead for the two alternatives)

5.5 The Effects of System Parameters

5.5.1 Operation mode

Table 5.11, summarizes the data presented in table 5.4 according to the operation mode required by the transactions, according to this data table, and by the inspection of Figures 5.16 through 5.17 , we can notice that, the mode of operations affects the system performance, because the lock manager when deals with a type of transactions that have a write operation, it needs to lock all the copies in all sites, the same thing is happens when the mode of transaction operation is mixed of read and write, but with less effect, because the read operation is executed from the nearest site or

locally, if the data item needed exists locally. In dealing with read operation, we can notice that, the system performance is better, because of local processing.

Table 5.11 (Transactions classified according to the operation mode)

Transaction ID	Mean Service Time for R-mode	Transaction ID	Mean Service Time for RW-mode	Transaction ID	Mean Service Time for W-mode
1	328	2	2766	6	1344
5	110	3	610	9	1219
7	188	4	812	11	1359
12	110	8	641	21	3500
17	250	10	578	22	1031
23	32	13	875	25	1547
24	438	14	437	27	1578
28	78	15	953		
30	125	16	985		
32	62	18	2016		
34	172	19	360		
39	531	20	760		
		26	468		
		29	703		
		31	859		
		33	1578		
		35	2938		
		38	3625		
		36	1203		
		37	1062		
		40	828		

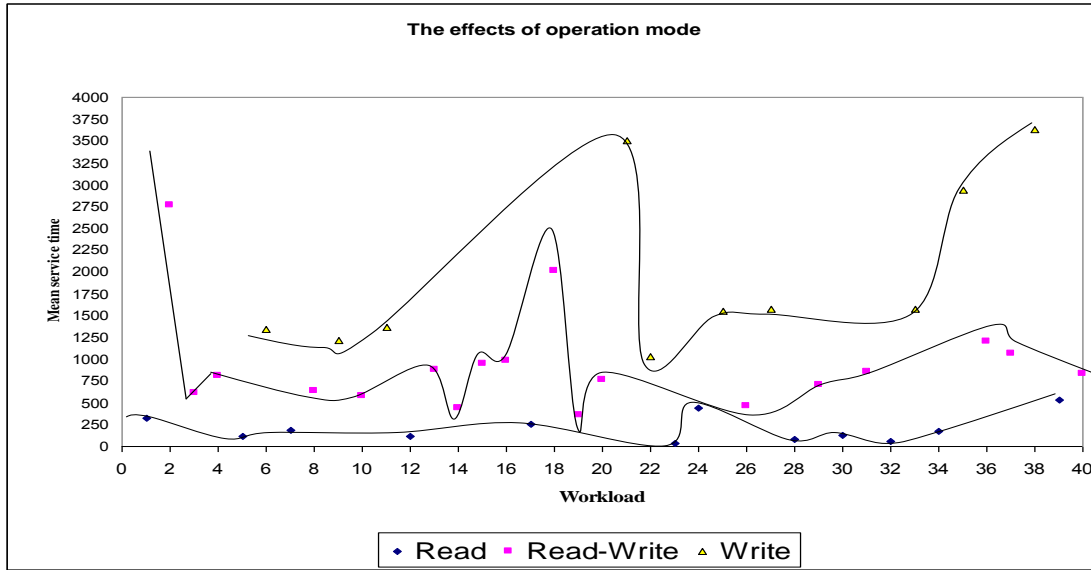


Figure 5.16 (The effects of operation mode)

5.5.2 Number of sites

The number of sites also affects the system performance, for the same reason of distributing database objects Table 5.12 and Figure 5.18 and 5.19. But the mode of operation has greater effect than the number of sites.

Table 5.12 (Transactions classified according to the number of sites used)

Transaction ID	One Site (Mean service time)	Transaction ID	Two Sites (Mean service time)	Transaction ID	Three Sites (Mean service time)
5	110	1	328	3	610
12	110	2	2766	4	812
21	3500	6	1344	10	578
28	78	7	188	11	1359
29	703	8	641	13	875
31	859	9	1219	14	437
32	62	16	985	15	953
34	172	17	250	20	797

35	2938	18	2016	22	1031
		19	360	23	32
		24	438	25	1547
		26	468	37	1062
		27	1578	38	3625
		30	125	39	531
		33	1578		
Mean service time		36	1203		
		40	828		

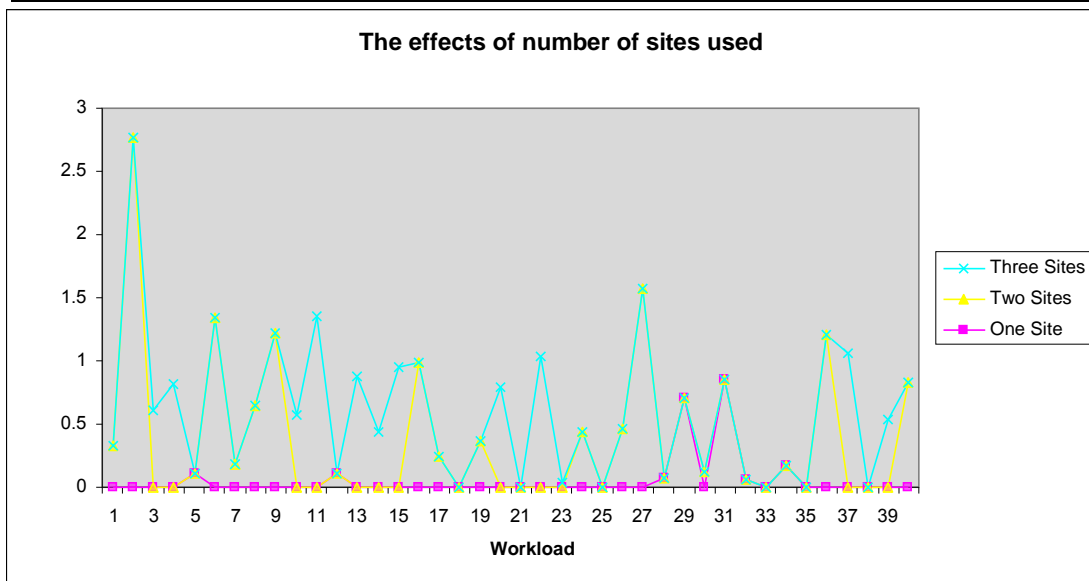


Figure 5.17 (The effects of number of sites used)

In the sample data used in this chapter, the transactions are mixed in terms of operation mode, and in terms of the number of sites used. So, we eliminate the extreme values where the transaction is deadlocked or blocked. The graph in Figure 5.17, shows roughly the effects of sites used, so when the transaction is locally executed, the system behaves much better. When the transaction becomes globally executed, the system performance decreases, due to message passing among the sites.

5.5.3 Degree of replication

The degree of replication parameter has an effect on the performance, especially on read only transaction, i.e. the performance of the system becomes better, because the data may be locally available most of the time. Burger et al [7, 22], study such parameter in distributed two phase locking in distributed database systems, and provide a full description of the parameters that affect the system performance.

5.6 Conclusion

As in centralized database approach discussed in chapter four, simulation is implemented to prove that obtaining a lock at attributes level on a distributed database will improve the performance. The discussion presented in sections 5.1 through 5.4, shows that the system at field level locking behave much better than at row level locking. Due to the multiple transactions process at the same database row will simultaneously decrease the mean service time as well as the mean waiting time. Because transactions do not need to wait for a long time to get their locks, the availability of data will be increased. Also alternative two executes more transactions than alternative one per a time unit before thrashing occurs.

Although the database size is increased in alternative two because of using the fields as lockable units, a transaction needs greater time than in centralized database, because the number of users in distributed database is much greater than in centralized. But in the two cases, the field level locking system behaves better than the row level locking.

CHAPTER SIX

CONCLUSIONS AND RECOMMENDATIONS

This chapter provides basic conclusions as well as directions for future research.

6.1 Introduction

Databases are becoming the core of most applications, and the number of users is incremented in an unexpected way due to the need for information in many situations like decision making or even in daily routine applications. Information must be available in an efficient and reliable way to satisfy user requirements and to cover the increasing needs. This usage needs specific techniques to protect the consistency and integrity of data contained in the database. The most popular technique used to attain the data protection is the locking of database items before using it, two phase locking is the most popular mechanism used in most commercial databases, which coordinates execution among transactions to preserve consistency as well as integrity.

Locking can be obtained at different levels of a database with the row as a minimum lockable unit, in this dissertation, a new approach is introduced to increase concurrency and to decrease

deadlock occurrences, which is implemented by allowing the attributes to be locked individually. The approach is implemented first on centralized, then on distributed database.

6.2 Field Level Locking on Centralized Database

The discussion presented in chapter four, shows that the system at field level locking behaves much better than at row level locking, because multiple transactions can process the same database row simultaneously, which decreases the mean service time as well as the mean waiting time, because transactions do not need to wait for a long time to get their locks, which increases the availability of data. At the same time, more transactions are executed on field level than row level locking before the system begins thrashing, which means alternative two works better on a heavy work load. Our results agree with [48] for data contention work load and also agree with [6] for probability of conflicts and deadlocks, because the database size as a dominator in their model decreases the ratio of conflicts and deadlocks. So, in our approach, the database size is increased by using the fields as lockable units. Increasing locking overhead, can be managed by choosing the appropriate data granule

size for each transaction [5], for example, if a transaction needs too many fields of a database row, it locks the row, instead of locking each individual field.

6.3 Field Level Locking on Distributed Database

As in centralized database, the new approach behaves much better than the existing one, which is the row level locking, because multiple transactions can process at the same database row simultaneously, which decreases the mean service time as well as the mean waiting time. Transactions do not need to wait for a long time to get their locks, which also increases the availability of data that is a major requirements to satisfy user needs

Although the database size is increased in alternative two by using the fields as lockable units, a transaction needs greater time than in centralized database, because the number of users in a distributed database in general is greater than in centralized. Table 5.3 and table 4.5 show that, the system at field level locking executes 200 transactions within 9.736 seconds in the distributed database, while the same number of transactions needs 6.421 seconds in a centralized database. But in the two cases, the field level locking system behaves better than the row level locking.

Alternative two (field level locking), for both environments (centralized and distributed) will be more suitable than row level locking, since most transactions entering the system are of update mode, because the contention becomes higher when exclusive mode is used.

6.4 Future Works

Even though, the researcher in this dissertation proved that the proposed approach (field level locking as a lockable unit) improves the efficiency of the existing one (row level locking as a minimum lockable unit), it opens different areas for future research to improve or support the presented solution such as:

1. The assumption considered in the proposed approach, by assuming the database is represented as a hierarchy tree, so future research may apply to a directed acyclic graph database, which has more than one path to a database item.
2. Developing a tool to monitor and tune the performance parameters.
3. Repeat the experiments provided in this dissertation with actual trace from real-world applications.
4. Developing a tool to reduce the locking overhead, when using the field level locking.

5. We believe that the field level locking approach can provide additional benefits other than facilitating the solution of increasing concurrency and decreasing deadlock occurrences, such as Database Security. Another research in this area would reveal more results.

REFERENCES

1. Al-Hamami A. and Al-Annie S., "**Concepts and Applications of Databases Technology**", Ithraa Publishing and Distribution, Jordan, 2008.
2. Averill M. L. and Kelton W. D., "**Simulation Modeling and Analysis**", McGraw-Hill, Singapore, 2000.
3. Baum T. and Andrew S., "**Modern operating systems**", Pearson Education Asia, New Delhi, 2002.
4. Bell D. and Grimson J., "**Distributed database systems**", Addison-Wesley, Wokingham, 1994.
5. Bernstein P. A., Hadzilacos V., and Goodman N. "**Concurrency Control and Recovery in Database Systems**", Addison-Wesley, 1987.
6. Bernstein P. and Newcomer E., "**Principles of Transaction Processing for the Systems Professional**", Bentham Press, 2004.
7. Burger A., Kumar V. and Hines M., "Performance of Multiversion and Distributed Two-Phase locking Concurrency Control Mechanisms in Distributed Databases", **Information Science**, Vol. 96, Issue 12, P129, 24P, 1997.
8. Chandy K. M., Misra J. and Hass L.M., "Distributed Deadlock detection", **ACM Transactions on Computer Systems**, Vol. 1, No. 2, 1983.
9. Coffman, E.G., Elphick M. J., and Shoshani A., "System Deadlocks", **ACM Computing Surveys**, Vol. 3, No. 2, PP:67–78, 1971.

10. Croker, A. "Improvements in Database Concurrency Control with Locking", **Journal of Management Information Systems**; Vol. 4 Issue 2, 2001.
11. Elmasri R. and Navathe S.B. "**Fundamentals of Database Systems**", 5th edition, Pearson Addison Wesley (Boston), 2007.
12. Eswaran K. P., Gray J. N., Lorie R. A. and Traiger I. L., "The Notions of Consistency and Predicate Locks in a Database System", **ACM archive**, New York, Volume 19 , Issue 11, 1976.
13. Galindo-Legaria C. and Rabitti F., "Extending Locking Techniques to Improve Concurrent Database Access", **CNR**, April 1995.
14. Gray J. N., Lorie R. A., Putzolu G. R. and Traiger I. L., "Granularity of Locks and Degrees of Consistency in a Shared Data Base" In **IFIP Working Conference on Modeling in Database Management Systems**, pages 365--394, 1976.
15. Holliday J. L. and El-abbadi A., "Distributed Deadlock Detection", **Encyclopedia of Distributed Computing**, Kluwer Academic Publishers, 1987. Available at: http://www.cse.scu.edu/~jholliday/dd_9_16.htm. Accessed on 21/6/2008.
16. Ingres:
<http://docs.ingres.com/dba/UnderstandingtheLockingSystem>
Accessed on 5/5/2008.
17. Jain R., "**The Art of Computer Systems Performance Analysis**", John Wiley, New York, 1991.

18. Krivokapi N., Kemper A. and Gudes E., "Deadlock detection in distributed database systems: a new algorithm and a comparative performance analysis", **The VLDB Journal** Volume 8, Issue 2, New York 1999.
19. Ling Y., Chen S. and Chiang C.J., "On Optimal Deadlock Detection Scheduling", **IEEE Transactions on Database**, Vol. 55, Issue: 9, PP: 1178 – 1187, 2006.
20. Locking in Microsoft SQL Server:
http://www.mssqlcity.com/Articles/Adm/SQL70Locks.htm#part_1. Accessed on 5/5/2008.
21. Lomet D. "Private Lock management", **Digital Equipment Corporation**, Cambridge Research Lab, 1992.
22. Matthias N. and Matthias J., "Performance Modeling of Distributed and Replicated Databases", **IEEE transactions on knowledge data engineering**, v. 12 n4, p 645-672, July 2000.
23. Menasce D. A., Almeida V. A. F. and Dowdy L. W., "**Performance by Design, Computer Capacity Planning**", Prentice Hall, New Jersey, 2004.
24. MSDN (Microsoft Developer Network):
[http://msdn.microsoft.com/en-us/library/ms681205\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms681205(VS.85).aspx)
Accessed on 19/5/2008.
25. Oracle, Database Concepts:
http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/consist.htm#i5337. Accessed on 20/5/2008.

26. Oracle, Data Concurrency and Consistency:
http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10743/consist.htm
 Accessed on 5/5/2008.
27. Oracle Locking Survival Guide:
http://www.akadia.com/services/ora_locks_survival_guide.html
 . Accessed on 17/1/2007
28. Ozsu M. T. and Valduriez P., "**Principles of distributed database systems**", 2nd edition, Prentice-Hall, Inc, New jersey1991.
29. Ries R. D. and Stonebraker M.," Effects of locking granularity in a database management system", **ACM Press**, New York, Volume 2, Issue 3, PP:233 -246, 1977.
30. Roark M. B., Bohler M. and Eldridge B. L. "Embedded Real-Time and Database: How Do They Fit Together", 1996.
 Ada:
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.8652>
 .
 Accessed on 23/6/2008.
31. Rob P. and Coronel C. "**Database systems design, implementation, and management**", 7th edition, Thomson, Canada, 2007.
32. Ryu K. I. and Thomasian A., "Analysis of database performance with dynamic locking", **Source Journal of the ACM**, Volume 37 , Issue 3, 1990.

33. Silberschatz A., Korth H.F. and Sudarshan S. "**Database System Concepts**", 5th edition, McGraw-Hill, New York, 2006.
34. Sinha M. K. " Constraints: consistency and integrity", **ACM SIGMOD**, Volume 13, Issue 2, New York, 1983.
35. SQL Server Books Online:
<http://msdn.microsoft.com/en-us/library/ms130214.aspx>.
Accessed on 5/5/2008.
36. SQL Server, Reducing SQL Server Deadlocks, Brad McGehee, 2006, Accessed on 1/5/2008.
37. Sybase, Performance and Tuning: Locking:
Document ID: DC20021-01-1251-01, Sybase, Inc 2003
38. Thomasian A., and Ryu K., "Performance Analysis of Two-Phase Locking," **IEEE Transactions on Software Engineering**, vol. 17, no. 5, pp. 386-402, May, 1991.
39. Thomasian A., " Concurrency control: methods, performance, and analysis", **Source ACM Computing Surveys**, Volume 30, Issue 1, 1998.
40. Thomasian A., "Two-phase locking performance and its thrashing behavior", **Source ACM Computing Surveys**, Volume 18 , Issue 4, 1993
41. Weikum G. and Vossen G., "**Transactional Information Systems**, Theory, Algorithms and the Practice of Concurrency Control and recovery", Morgan Kaufman Publishers, 2002.
42. Wikipedia Encyclopedia, [http:// en.wikipedia.org/wiki/ Banker's algorithm](http://en.wikipedia.org/wiki/Banker's_algorithm). Accessed on 5/2/2007.

43. Wikipedia Encyclopedia,
<http://en.wikipedia.org/wiki/Deadlock>. Accessed on 28/6/2007.
44. Wikipedia Encyclopedia:
http://en.wikipedia.org/wiki/Distributed_transaction_processing
Accessed on 19/5/2008.
45. Wikipedia, Encyclopedia:
http://en.wikipedia.org/wiki/Edge_chasing. Accessed on
13/7/2007.
46. Wolfson O. "The overhead of locking (and commit) protocols in distributed databases", Source ACM Transactions on Database Systems, Volume 12 , Issue 3, New York ,1987.
47. Wu H., Chin W. and Jaffar J., "An efficient distributed deadlock avoidance algorithm for the AND model", **IEEE Transactions**, Volume 28, Issue 1, PP: 18 – 29, Jan 2002.
48. Y. C. Tay, Goodman N. and Suri R., "Locking performance in centralized databases", **ACM Transactions on Database Systems**, Volume 10, Issue 4, 1985.
49. Yeung C., Hung S. and Lam K., "Performance evaluation of a new distributed deadlock detection algorithm", **ACM SIGMOD**, Volume 23, Issue 3, 1994.

Appendices

APPENDIX A

Input and Output Parameters of Simulation Program

In this appendix, the simulation input parameters are listed in Table A.1 for centralized, and Table A.2 for distributed. The simulation output parameters are listed in Table A.3.

Table A.1: Simulation input parameters for centralized database

Parameter	Description	Value
Num-table	Number of tables in a database	20
Min-num-tuples	Minimum number of tuples in each table	1
Max-num-tuples	Maximum number of tuples in each table	1000,5000
Min-col	Minimum number of columns in each table	1
Max-col	Maximum number of columns in each table	10
Num-trans	Number of transactions in the system	Up to 1000
Min-trans-size	Minimum number of operation	1
Max-trans-size	Maximum number of operation	20
Queue-length	Maximum queue length	10,20

Table A.2: Simulation input parameters for distributed database

Parameter	Description	Values
Num-site	Number of sites	3
DB-num	Number of databases in each site	1
DB-obj	Number of database objects for each site	5000
Rep_deg	Degree of replication	0.2
Num-table	Number of tables in a database	15
Num-trans	Number of transactions in the system	Up to 500
Min-trans-size	Minimum number of operation	1
Max-trans-size	Maximum number of operation	20
Op-mod	Operation mode	R, RW, W
Queue-length	Maximum queue length	20
Time_check	Mean time to check a lock	1 ms
Time_set	Mean time to set a lock	1 ms
Time_rel	Mean time to release a lock	1 ms
Time_acc	Mean time to access a data object	20 – 100 ms

The simulation program was implemented in Java programming technology by using NetBeans IDE Version 6.0.1 (www.netbeans.org). When the program starts, the source packages opened, then the input parameters with default values is ready to use after opening the general settings menu. In addition to the parameters listed in tables A.1 and A.2, the following parameters are also used:

SMULATION_TIME, which used to specify the simulation time in milliseconds.

TRANSACTION_LOCK_MAX_TRY_COUNT, used for specifying how many trials the transaction may try before it aborts, incase of deadlock.

After the simulation finished, the output parameters listed in A.3 are produced.

Table A.3: Simulation output parameters

Parameter	Description
Simulation total time	The time needed to complete the number of transactions specified
Average transaction execution time	The average execution time calculated for the transactions executed within a time period
Total number of transactions	How many transactions the simulation executes
Number of completed transactions	The number of succeeded transactions
Number of blocked transactions	The number of transactions failed due to blocks
Number of deadlocked transactions	The number of failed transactions due to deadlock
Transaction ID	The sequence number generated to each transaction
Arrival Time	The system clock time recorded when the transaction arrives
Start Service	The system clock time recorded when the transaction start service
End Service	The system clock time recorded when the transaction finished
Waiting Time	The time that a transactions spent in a waiting queue
Execution Time	The time that the transaction spent in execution
Number of Locks	How many locks needed for a transaction to proceed

Number of Operations	The number of operations needed for each transaction
Status	The status of transaction (Done, Blocked, or deadlocked)
Mean number of Locks	Mean number of locks needed for group of transactions to accomplish their task
Mean number of Operations	Mean number of operations needed for a group of transactions
Mean waiting time	Mean or average waiting time that a group of transactions spent in a waiting queue
Throughput	Number of completed transactions within a time interval
Database size	Number of database items generated by the simulation programs according to the input parameters
Sites used	The number of sites used by transaction
Operation mode	The operation mode for each transaction

After the simulation runs, two files are produced, the first one is for each transaction to show its behavior such as the transaction shows in figure 4.1. The second file is for the transactions running within a time period, to show the transaction ID, Arrival time, Start service, End service, Execution time, Waiting time, Number of operations, Number of locks needed, and the Status.

