# Array-based Locks for Concurrency Control Model

by

**Jad Fawzi Abbass**

B.S., Computer Science, Beirut Arab University, 2003

Project submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Department of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

May 2008

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences - Beirut Campus

# Project Approval Form

Student Name:Jad Fawzi Abbass I.D. #: 200400430

Project Title : Array-based Locks for Concurrency Control Model

Program : Computer Science

Division/Dept : Computer Science and Mathematics

School : School of Arts and Sciences

Approved by: Ramzi A. Haraty

Project Advisor: [redacted]

Member : Abdul Nasser Kassar [redacted]

Date May 20, 2008

## Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Jad Abbass

Signature: 

Date: May 20, 2008

I grant to the LEBANESE AMERCIAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.

*To my little niece Layal & nephew Fawzi*

# Acknowledgment

*First of all, I would like to say a special "Thank you" to the LAU's "three giants of Computer Science"; Prof. Nashat Mansour, Prof. Ramzi Haraty, and Prof. Faisal Abou-Khzam. You have taught me some computer science and much honesty, integrity, seriousness, knowledge, patience, and especially how to be a good instructor; my new career.*

*Thank you for your continuous interest, guidance and support from the moment I enrolled to LAU.*

*Thanks to Prof. Abdul Nasser Kassar for being on my project committee.*

*I would like to express my sincere feelings to my parents and brothers for their infinite patience.*

*My last thanks to my friend and colleagues who have been very kind in all this period.*

# Abstract

Multitasking in both uniprocessor (multithreading) and multiprocessor (multiprocessing) systems have been attracted by many applications. Database systems are somewhat the most important in this regard, especially in centralized and humongous ones. Sometimes thousands, and maybe hundred of thousands of operations are sent to the transaction processing system per second. To handle this bottleneck some queries/ updaters are executed concurrently. However, parallelism in such cases is extremely accurate based on the well-know restriction; locks. In this project, I implemented a lock approach based on a Boolean array (1D and 2D) and on the "or" logical operation to specify which transactions can be executed in parallel.

# Contents


**1. Introduction** **1**

1.1 Overview 1

1.2 Background and Motivation 3

1.3 Concurrency Control 4

1.4 Structure of the Project 4

**2. Literature Review** **6**

2.1 Database Systems 6

2.2 Transaction Processing Systems 7

2.2.1 Basic Operations 8

2.2.2 General Architecture 8

2.2.3 Conflict Operations 10

2.3 Concurrency Control Techniques 11

2.3.1 Concurrency Control Techniques Based on Locking 11

2.3.2 Concurrency Control Techniques Based on Timestamps 12

**3. Array-based Locks** **13**

3.1 Overview 14

3.2 Proposed Techniques 14

3.3 Project's Three Versions 19

3.4 Transactions and Operations 19

3.5 The Three Phases 23

**4. Experimental Results** **25**

4.1 Version 1 25

4.2 Version 2 28

4.3 Version 3 30

4.4 Comparison Among The Three Versions 32

**5. Conclusion and Future Work** **36**

**References** **38**

## List of Figures


| | | |
|---|---|---|
| Figure 2.1 | General Architecture of a TPS | 9 |
| Figure 3.1 | General Overview of the 2D Array Structure | 15 |
| Figure 3.2 | Simple Example of a 2D array of Locks | 16 |
| Figure 3.2.1 | 2D Array for Read Locks | 18 |
| Figure 3.2.2 | 2D Array for Write Locks | 18 |
| Figure 3.3 | Part of the Twenty Transactions | 20 |
| Figure 3.4 | the Button That Generates Randomly the Operations | 21 |
| Figure 3.5 | Part of the List of Operations | 22 |
| Figure 3.6.1 | Phase 1's Run Button | 24 |
| Figure 3.6.2 | Phase 2's Run Button | 24 |
| Figure 3.6.3 | Phase 3's Run Button | 24 |
| Figure 4.1 | Chart Showing the Decreasing of the Number of Sets Among the Three Versions from Phase I to II | 33 |
| Figure 4.2 | Chart Showing the Decreasing of the Number of Sets Among the Three Versions from Phase II to III | 34 |
| Figure 4.3 | Chart Showing the Decreasing of the Number of Sets Among the Three Versions from Phase I to III | 35 |

# List of Tables


Table 2.1 Conflict among Operations 10

Table 4.1 Ten Experiments on Version 1 26

Table 4.2 Conclusion form the Ten Experiments on Version 1 27

Table 4.3 Ten Experiments on Version 2 28

Table 4.4 Conclusion form the Ten Experiments on Version 2 29

Table 4.5 Ten Experiments on Version 3 30

Table 4.6 Conclusion form the Ten Experiments on Version 3 31

Table 4.7 Comparison among the Three Versions 32

# Chapter 1

# Introduction

## 1.1 Overview

The concept of transaction provides a mechanism for describing logical units of database processing. Transaction processing systems are systems with large databases and hundreds of concurrent users that are executing database transactions. Examples of such systems include systems for reservations, banking, credit card processing, stock markets, supermarket checkout, and other similar systems. They require high availability and fast response time for hundreds of concurrent users.

One criterion for classifying a database system is according to the number of users who can use the system concurrently – that is, as the same time. A Database Management System (DBMS) is single-user if at most one user can use the system, and it is multiuser if many users can use the system – and hence access the database – concurrently.

Multiple users can access databases – and use computer systems – simultaneously because of the concept of multiprogramming, which allows the computer to execute multiple programs – or processes – at the same time. I only a single CPU exits – under the same circumstances of this project – it can actually execute some commands from one process. However, multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually interleaved. Interleaving keeps the CPU busy when a process requires an input or output

(I/O) operation, such as reading a block from a disk. If the computer system has multiple hardware processors (CPUs), parallel processing of multiple processes is possible.

This project is based on the former system – interleaved concurrency – used multithreading approach. It seems at first that the response time will not affected by far, when adopting the new lock mechanism which in turn, allows many transactions to be executed in parallel rather than serially, the main purpose of this project is to implement the array-based locks to BUILD the sets of the operations that can be sent to the DBMS at the same time.

Controlling the concurrency is a critical issue in DBMS, building the sets of transactions that can be executed in parallel is not an easy duty to the system. The system should make sure that there is no overlapping in the locks associated with every query/ updater, otherwise several problems can occur. For instance the two well-known problems; The Lost Update Problem and the Dirty Read Problem, both of them produce eventually incorrect results and/ or leave the system in an inconsistent status.

The oldest and maybe the most traditional way to control the access to an item is the mutual exclusion, that is, when a transaction wants to access an item, it would be granted a permission to use this item, and other operations should wait in a queue.

In this project, first I let the transaction execute serially, than building sets of transactions that can be executed in parallel, and execute these sets serially based on one dimensional array for both read and write operations, the last phase is to build again the sets just mentioned, and execute them serially but based on a two dimensional array, one for read and the other for write operations and take into account that read operations do

not conflict, and at the end compare the response time, as well the number of sets in phase II and III.

## 1.2 Background and Motivation

Concurrency control is considered one of the oldest topics in the field of computer science. It has been in research for more than thirty years (Eswaran, Gray, Lorie, traiger 1976), (Schlageter, 1978), and (Bernstein, Goodman, 1980). Many papers had proposed correct techniques and solutions for concurrency control in database systems, yet these solutions showed many drawbacks and disadvantages when there was an increase in the number of parallel operations.

In order to enhance performance in database environment, database applications must be allowed to interleave their execution. However concurrent execution gone unsupervised can lead to erroneous results and inconsistent database state. A database management system prevents such inconsistencies by enforcing a concurrency control strategy, which enhances performance and maintains correctness. In other words, the concurrency control strategy will only produce serializable schedulers. Such schedulers will produce executions which are equivalent to some serial order. A number of methods such as time stamp ordering (TO), two phase locking (2PL), serialization graph testing (SGT), tree locking (TL), optimistic certifiers, and request ordered linked list (ROLL) have been proposed. Many of the existing relational and object-oriented database systems use these methods with varied levels of performance. The focus of this project is to implement a "binary lock" approach and develop a high performance concurrency control algorithm which is both, efficient and correct.

## 1.3 Concurrency Control

The main challenge database systems nowadays is how to conserve consistency and correctness after hundreds or even thousands of queries and/or updaters have been executed on such database. For this reason, too many concurrency control models have been proposed to prove their correctness and that any set of transactions would leave the system in a consistent state.

## 1.4 Structure of the Project

The structure of the report of the project is as follows; the next chapter is a literature review that talks generally about transactions processing systems, database systems, concurrency control, locking mechanisms. Chapter 3, entitled, array-based locks, talks about the heart of the project, its three versions, three phases, its forms, how it is implemented.

Chapter 4 is specialized in experimental results which shows the efficiency of the two proposed approaches using respectively two and three dimensional arrays, which constitutes of Boolean elements.

Chapter 5, as any project or thesis, is the conclusion and future work based on the experimental results conducted form the previous chapter, and finally References.

# Chapter 2

# Literature Review

## 2.1 Database Systems

Database systems are now one of the hearts of the field of computer science/ engineering. Besides algorithms, software engineering, database system has become an extremely essential part of any computer-based information system.

Form the its beginning, database system field has grown dramatically, in terms of types, applications, and studies. For example about the applications, multimedia databases, Geographic Information Systems (GIS), Data warehouses (OLAP), etc.

A database system consists of data and a DBMS (Database Management System) the latter is essence of such systems, it has many benefits and advantages such as, controlling redundancy, restricting unauthorized access, providing multiple user interfaces, representing complex relationships among data, enforcing integrity constraints and providing backup and recovery.

Actually, the main of this project is to develop a part of a DBMS that takes care of controlling concurrent accesses on the database.

## 2.2 Transaction Processing Systems

One of the main types of database systems is the transaction processing system. This latter has spread in the last few years due to the dramatic improvement in the speed of the large computers, known as servers. When the large computers and servers, like banks'

however, all its effects done until the moment the error occurred, should be undone, and of course acknowledge the user that the system is in a state as if the transactions had not started at all.

### 2.2.2 Transaction Processing System General Architecture

The main architecture of a transaction processing system consists of the database itself, and the data manager, which itself contains the cache manger and the recovery manager, then the scheduler and the transaction manager.

The next picture, adopted from Bernstein and Newcomer shows the high level of the architecture of any similar system, however, each part contains multiple modules within it.

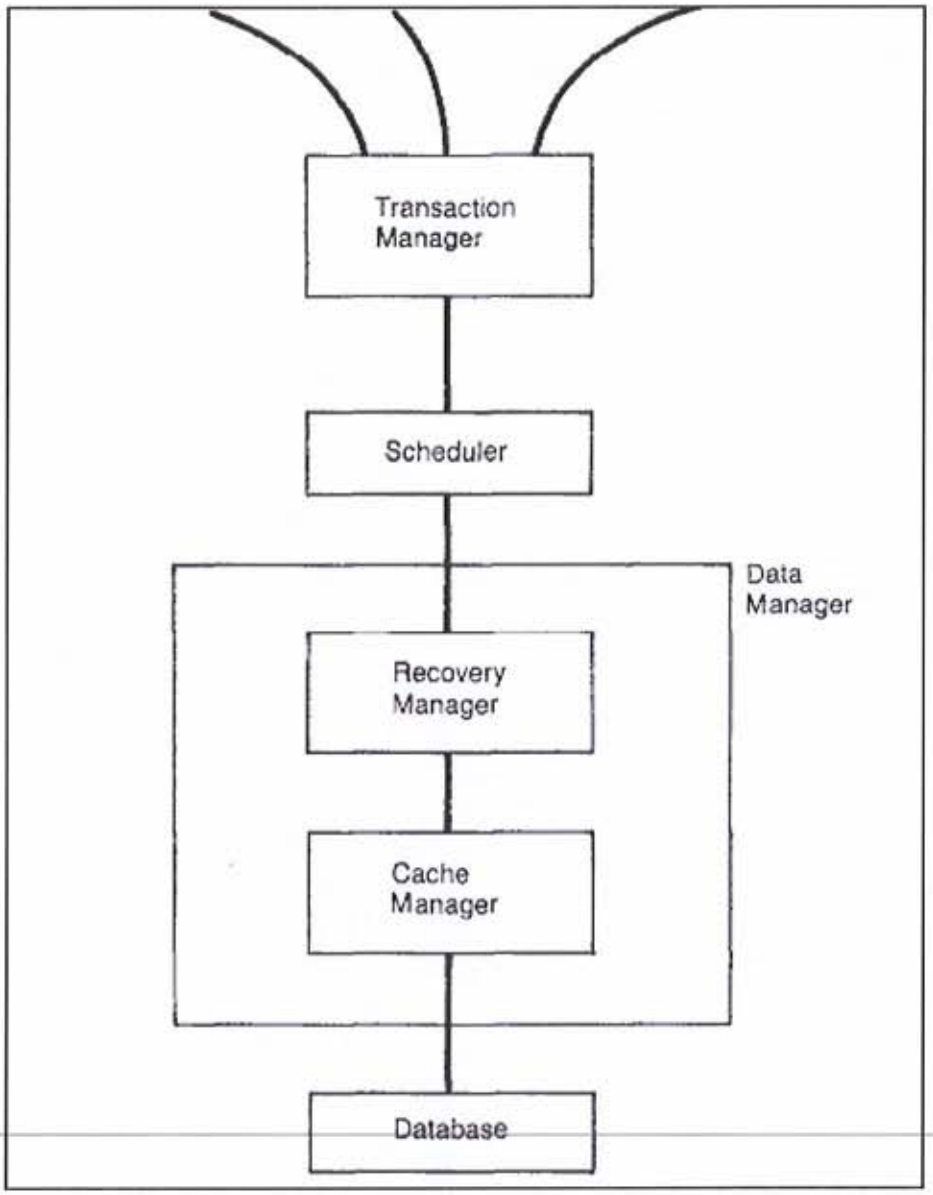


Figure 2.1 General Architecture of a TPS adopted from Bernstein and Newcomer

### 2.2.3 Conflict Operations

When we say two or more transactions interleave their execution, it means that their operations do not execute in a serial way, that is, the system may execute an operation from T1 and then an operation from T2 and so on. For this reason we do need

concurrency control model to handle this problem, because interleaving execution can leave the database in an inconsistent state. However, not all operations conflict with each other for instance the most common known conflict is between read and write. Suppose we have the following simple example: read1(x), write2(x, 2), and read1(x), and suppose that the value of initially is 1. Transaction 1 first reads the value 1, and then it reads the value 2 after few milliseconds! The next table shows which operations can be in conflict.

Table 2.1 Conflict among Operations

| | read | write |
|---|---|---|
| read | NC | C |
| write | C | C |

NC: No Conflict

C: Conflict

Accordingly, under these circumstances:

1. working on the same data item
2. either of the operations is write

Interleaving operations two or more transactions needs concurrency control model to manage this problem by avoiding deadlocks, cycles, and starvations.

## 2. 3 Concurrency Control Techniques

In general, concurrency control techniques can be classified into two categories; Locking and Timestamp. The former relies on locking a data item (it depends on the

granularity of this data item, it can be a single attribute value, a record, a block of a disk) while it is in use by a transaction, when this transaction finishes using this data item, it releases the lock to be used by other transactions.

### 2. 3.1 Concurrency Control Techniques based on Locking

Actually, we can see the lock as a variable or a value associated with each data item. The status of this variable/ value determines the status of the data item whether it is busy or not.

The types of locks are:

- Binary Lock: this technique is quite simple and efficient, when the binary value is set to 0, it means that the associated data item is free to use, and vice versa.
- Shared/ Exclusive (or Read/Write) Locks: this approaches is quite similar to the previous one, except that it relies on the fact that read and read operations do not conflict, so it allows two or more transactions to access the same data item for read purpose. This project is based on this technique as we will see in the later chapters.
- Conversion of locks: it allows a transaction to "upgrade" its lock from read to write.
- 2PL (Two Phase Locking): this technique has many versions like, basic, rigorous, strict, conservative…the most common used is the basic and it is the most widely used nowadays.

## 2. 3.2 Concurrency Control Techniques based on Timestamps

Basically, this technique does not use locks at all, so it is free from deadlocks. A timestamp is like a unique identifier associated with each transaction, generated by the DBMS.

A timestamp ordering (TO) is similar to a schedule based on transactions' timestamps in which are serializable and hence have the effects of serials ones. This technique is known as basic TO, besides strict TO, and Thoma's Write rule.

# Chapter 3

# Array-Based Locks

As mentioned before, too many approaches and techniques have been proposed to control concurrency problems in multiuser database systems environment. The oldest and simplest one is based on Boolean variables that is usually corresponds to a data item for a special transaction/ operation. If this bit is assigned to 0 it is free to be used by other transactions/ operations, otherwise no other tasks can grant the access on that data item.

## 3.1 Overview

The closet approach used to the one applied in this project is ROLL (Request Ordered Linked List) published by Hakimzadeh in 92 and developed at North Dakota State University. However, the latter is based on Boolean variables embedded in a linked list. My project is based on the same techniques and the same operation (OR) to know whether the data item is busy or not, but on a different data structure; array of two dimensions first and then of three dimensions. Actually this method is supposed to be more efficient, since static data structures are more efficient than dynamic ones in memory and CPU speed terms, and of terms of implementation.

## 3.2 Proposed Techniques

Basically, the project is built on two parts, the first one uses two dimensional array, and the second one uses three dimensional array. In both cases, the array is composed of Boolean arrays corresponding in essence to a special transaction for a

special data item, the difference lies in merging read and write locks in one Boolean element in that array in the first part, and splitting this element into two elements one for read locks and the other one for write locks in order to improve parallelism among transactions. In other words, to increase the number of transactions that can be executed in parallel or simply interleaved.

The granularity of the data item used in the project is the record. So, whenever a record is locked, all fields within this record are locked too, even if some of them are in use. When a transaction wants to access at least one attribute in a record, the system set the corresponding element in the array to one to prevent any other transaction to access this record. Also, recall the 4 properties of any transaction ACID, atomicity indeed here is valuable. For instance, if the transaction wants to use 248 records and one of them is locked the whole transaction waits until this one is free.

The next figure 3.1 shows the general structure of an array of two dimensions that maintains the information needed to lock or unlock records.

| | R1 | R2 | R3 | R4 | ..... | RN |
|---|---|---|---|---|---|---|
| T1 | 0 | 1 | 1 | 0 | ..... | |
| T2 | 1 | 0 | 1 | 1 | | |
| T3 | 0 | 0 | 0 | 1 | | |
| T4 | 0 | 1 | 1 | 0 | | |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| TM | | | | | | |

Figure 3.1: General Overview of the 2D Array Structure

The above picture shows a general structure of a two dimensional array used in the project of M transactions and N records. It is worth mentioning that a transaction, in general, and especially in this project can contain one or more operation within it. Also, the atomicity property of the transaction plays a critical role in this regard, for instance, the transaction does not start executing unless all locks needed by all operations embedded within this transaction are granted.

The next figure 3.2 shows a simple example of six transactions and six records and their corresponding elements in the array to see how the system can handle concurrency control based on the logical operation: OR.

| | R1 | R2 | R3 | R4 | R5 | R6 |
|---|---|---|---|---|---|---|
| T1 | 0 | 1 | 1 | 0 | 0 | 0 |
| T2 | 1 | 1 | 0 | 0 | 0 | 1 |
| T3 | 0 | 0 | 0 | 1 | 0 | 1 |
| T4 | 1 | 0 | 0 | 0 | 1 | 0 |
| T5 | 0 | 0 | 1 | 0 | 1 | 0 |
| T6 | 1 | 1 | 0 | 1 | 0 | 1 |

Figure 3.2: Simple Example of a 2D array of locks

As figure 3.2 shows, and based on "oring" logical operation, we can conclude the following results that can be used in concurrency control:

Set1: T1, T3, and T4.

Set2: T2 and T5.

Set3: T6

The above three lines say that T1, T3, and T4 can be executed in parallel or simply interleaved. In other words, in a uniprocessor system, as is the case in this project, we can assign for each transaction a thread, and let theses 3 threads execute without any possibility of deadlocks or let any thread waiting for another one to complete to release a data item. In a multiprocessors system, cluster, each transaction can be assigned to a particular processor in parallel with the other two processes freely.

How can the system conclude such sets? If in the columns (records) where T1 is 1, the system applies the OR operation on every remaining set of transaction, we can see that for R2 and R3 the OR (T3 and T5) = 0, accordingly, these three transactions can be executed in parallel. The same rule applies for Set2 and Set 3.

Following the same approach, but using one bit for read locks and another one for write locks, we get a three dimensional array. In details; instead of having one bit for both read and write locks, I split this element into two elements. One would ask, what is the

benefit of such an action? As mentioned in the introduction read and read operations do not conflict, thus based on this extremely important rule, we can improve parallelism and thus the efficiency of a transaction processing system. The next example and figure taken from the same example mentioned earlier prove that the enhancement made by using a three dimensional array instead of two can improve, in its turn, the overall execution of a huge number of transactions that want to access a database system.

| | R1 | R2 | R3 | R4 | R5 | R6 |
|---|---|---|---|---|---|---|
| T1 | 0 | 1 | 0 | 0 | 0 | 0 |
| T2 | 1 | 1 | 0 | 0 | 0 | 1 |
| T3 | 0 | 0 | 0 | 1 | 0 | 1 |
| T4 | 1 | 0 | 0 | 0 | 1 | 0 |
| T5 | 0 | 0 | 1 | 0 | 1 | 0 |
| T6 | 1 | 1 | 0 | 1 | 0 | 1 |

Figure 3.3 2D array for Read Locks

| | R1 | R2 | R3 | R4 | R5 | R6 |
|---|---|---|---|---|---|---|
| T1 | 0 | 0 | 1 | 0 | 0 | 0 |
| T2 | 0 | 0 | 0 | 0 | 0 | 1 |
| T3 | 0 | 0 | 0 | 1 | 0 | 1 |
| T4 | 0 | 0 | 0 | 0 | 1 | 0 |
| T5 | 0 | 0 | 1 | 0 | 1 | 0 |
| T6 | 1 | 1 | 0 | 1 | 0 | 0 |

Figure 3.4 2D array for Write Locks

From the above two pictures, and based on the facts that the operations write and write, read and write conflict, however, read and read do not. Remark that the above example we are working on is the same as mentioned earlier, but using two bits instead of one, or simply three dimensional array instead of two. Taken into account all the above, we can construct the following sets:

Set1: T1, T3, and T4.

Set2: T2, T5, and T6.

Note that rather than having three sets, we have only two sets, so, we have 2 sets that can be executed serially. Within each sets, all transactions can be executed in parallel, accordingly improving the parallelism and efficiency of the system.

## 3.3 Project's Three Versions

In this section we talk about the three versions of this project. As we know, every transaction can contain one or more operations. In order to generalize the experimental results in the next chapter, I created three versions of the project based on the factor: number of operations per transaction. The first one creates randomly, (in the next section, I will talk in details about the randomization technique used) from 1 to 5 operations per transaction. (Low) The second one creates 6 to 10, (Medium) and the third one creates 11 to 15 operations per transaction (High).

## 3.4 Transactions and Operations

During the implementation of this project, I tried as much as possible, to simplify the interaction between the user and the system. This project, in its three versions, constitutes of twenty transactions, as the next figure shows, that try to access the database and eventually commit successfully.

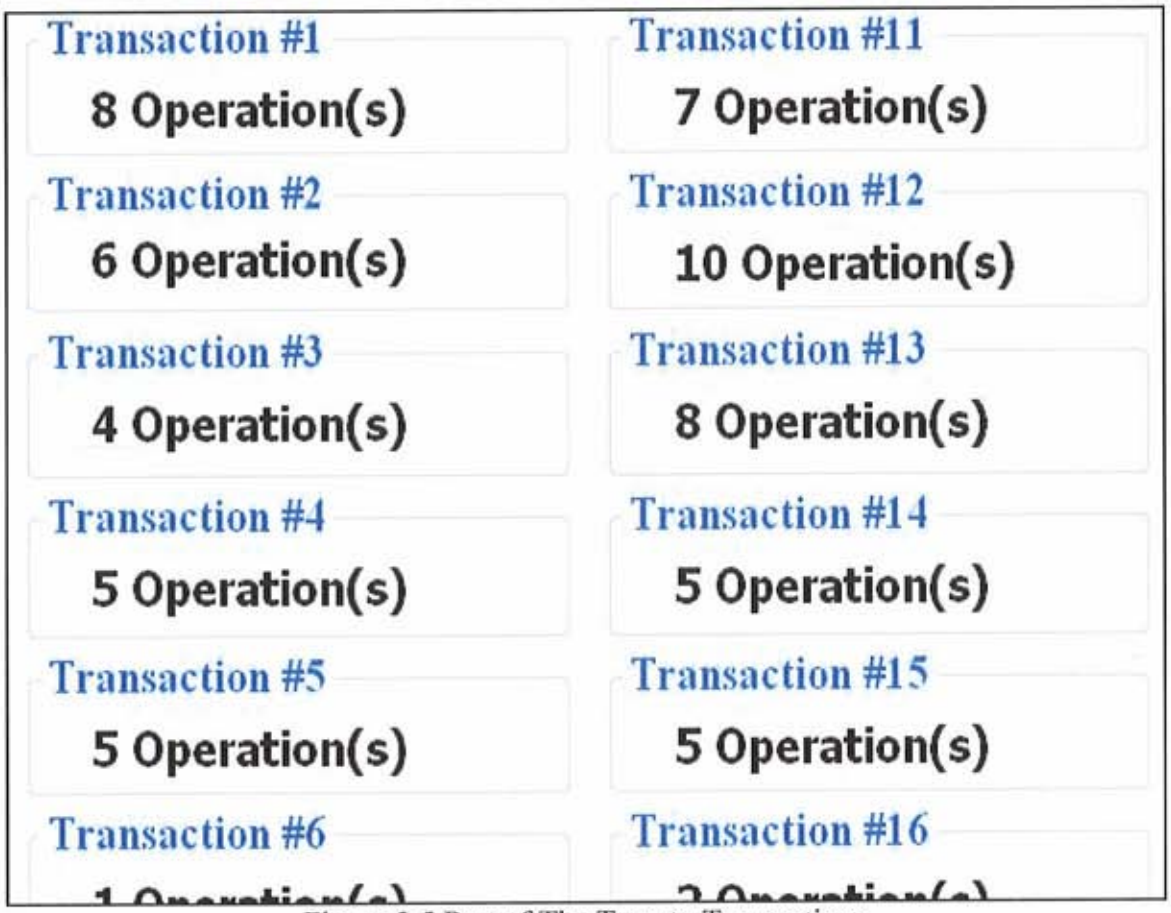


Figure 3.5 Part of The Twenty Transactions

To simplify the action of writing these, relatively big number of operations (queries or updaters), A list of 200 operations has been built but hidden within this system. The user, then, simply clicks on a button, shown in the next picture below, to select randomly these operations for each transaction.


Figure 3.6 The Button that Generates Randomly the Operations

The set of operations from which the system generates a random number of operations from a specified range, depending on the project's version, is composed of 200 queries and updaters. As the statistics shows, 20% of the operations are usually updaters and the remaining are queries, that is, write and read operations respectively. The next figure shows this list that conforms to this ratio.

Moreover, the operations within this list are evenly distributed in terms of the number of records, they access; for instance, 10% of them access 1 record, 10% access 3 records, 10 % access 10 records, and so on…

In general the whole list accesses roughly 70 to 80% of the whole database, which constitutes 2180 records and 12 fields. The latter, in their turn, are distributed evenly among all data types (text, number, auto number, date, time, currency)

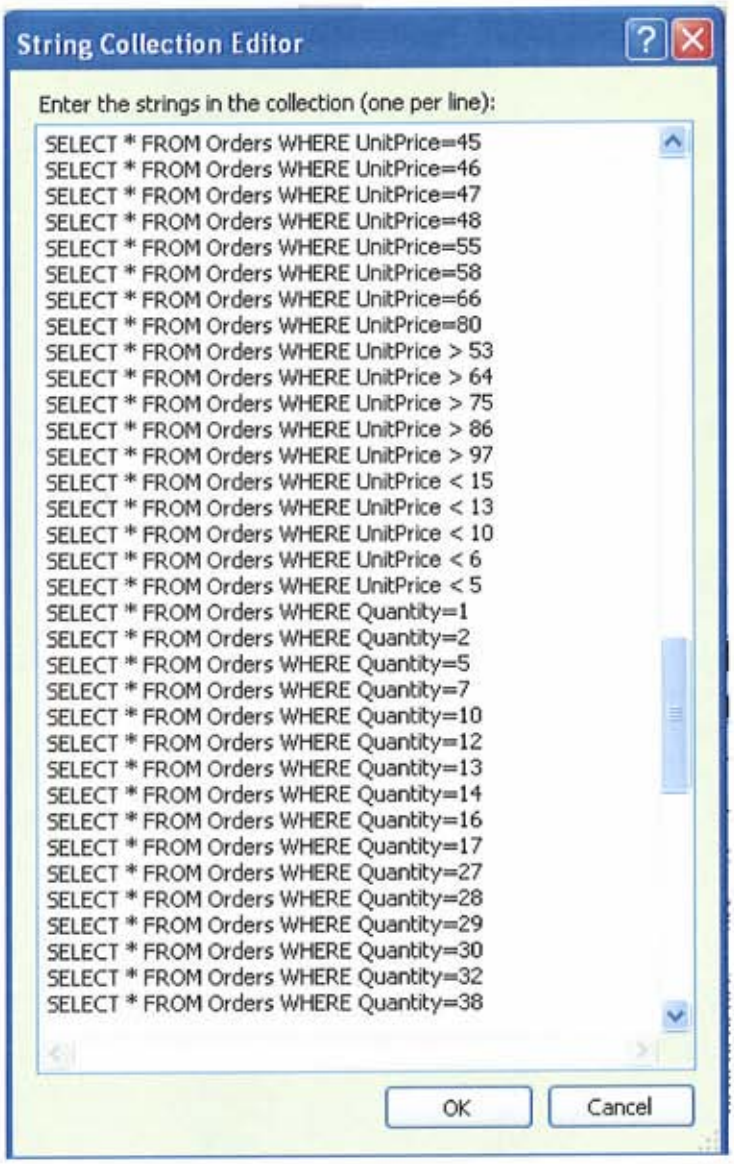


Figure 3.7: Part of The List of Operations

## 3.5 The Three Phases

Excluding the introductory and experimental results forms, the project is composed of three phases (each version). The first one is serial, that is, the 20 transactions are executed serially, one after the other, without any kind of parallelism. The purpose of this execution is to record the running time, and compare it with the next two phases, to show

that, even in a uniprocessor system, multithreading is more efficient. The next phase, is the application mentioned before about using a 2D array, in other words, using one bit for both read and write locks, however the third and last phase uses two bits, one for read and the other for write locks.

Between the phase I and phase II, the purpose is to improve the efficiency of the system by introducing the ability to run more than one transaction at a time, and to see that instead of having 20 sets of transactions that run serially, this set will decrease, such that every set can contain one or more transactions.

Between the phase II and III, the running time is not a critical issue like the previous one; however, the number of sets of transactions that run serially is the moral of the story here. So, improving the parallelism based on the fact that read and read operations do not conflict, to see by how many sets the third phase can decrease.

The next figures show for each phase, the corresponding button to run the sets of transactions.


Figure 3.8.1: Phase 1's Run Button


Figure 3.8.2: Phase 2's Run Button


Figure 3.8.3: Phase 3's Run Button

# Chapter 4

# Experimental Results

This chapter shows in details 30 experiments, that is, 30 executions, distributed evenly among the three versions; 10 for the first, Low, 10 for the second, Medium, and 10 for the Big one. Recall that we divide the project into 3 versions depending on the size of the transaction. For each experiment, we show the results for the three phases and after completing the 10 experiments, we draw some charts showing the important factors and variants during these phases.

## 4.1 Version 1

In version one, the number of transactions does not change, however, the number of operations within each transaction changes. In the first version, named Low, we, randomly, assign for each transaction form 1 to 5 operations. These operations could be queries or updaters, depending on the randomization action.

Table 4.1 10 experiments on Version 1

| | Total Number | | | Average | | | Phase I | | | Phase II | | | Phase III | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Experiment # | Operations | Queries | Updaters | Opers/ Trans | Queries/ Trans | Updaters/ Trans | Number of sets | Transactions/ Set | Running Time | Number of sets | Transactions/ Set | Running Time | Number of sets | Transactions/ Set | Running Time |
| 1 | 50 | 45 | 5 | 2.5 | 2.25 | 0.25 | 20 | 1 | 10429 | 7 | 2.857 | 3637 | 4 | 5 | 4076 |
| 2 | 61 | 55 | 6 | 3.05 | 2.75 | 0.3 | 20 | 1 | 8974 | 4 | 5 | 3310 | 1 | 20 | 2342 |
| 3 | 57 | 50 | 7 | 2.85 | 2.5 | 0.35 | 20 | 1 | 10708 | 4 | 5 | 3154 | 2 | 10 | 2045 |
| 4 | 61 | 50 | 11 | 3.05 | 2.5 | 0.55 | 20 | 1 | 12240 | 9 | 2.222 | 5276 | 4 | 5 | 3388 |
| 5 | 65 | 57 | 8 | 3.25 | 2.85 | 0.4 | 20 | 1 | 9397 | 8 | 2.5 | 5684 | 3 | 6.666 | 2279 |
| 6 | 64 | 58 | 6 | 3.2 | 2.9 | 0.3 | 20 | 1 | 9302 | 9 | 2.222 | 4793 | 4 | 5 | 2309 |
| 7 | 64 | 58 | 6 | 3.2 | 2.9 | 0.3 | 20 | 1 | 9617 | 5 | 4 | 3686 | 4 | 5 | 3607 |
| 8 | 47 | 41 | 6 | 2.35 | 2.05 | 0.3 | 20 | 1 | 8614 | 5 | 4 | 4295 | 5 | 4 | 4295 |
| 9 | 59 | 56 | 6 | 2.95 | 2.65 | 0.3 | 20 | 1 | 9100 | 10 | 2 | 4693 | 6 | 3.333 | 3448 |
| 10 | 63 | 56 | 7 | 3.15 | 2.8 | 0.35 | 20 | 1 | 10723 | 7 | 2.857 | 3526 | 6 | 3.333 | 3713 |
| Average | 59.1 | 52.6 | 6.8 | 2.955 | 2.615 | 0.34 | 20 | 1 | 9910.4 | 6.8 | 3.2658 | 4205 | 3.9 | 6.7332 | 3150.2 |
| Max | 65 | 58 | 11 | 3.25 | 2.9 | 0.55 | 20 | 1 | 12240 | 10 | 5 | 5684 | 6 | 20 | 4295 |
| Min | 47 | 41 | 5 | 2.35 | 2.05 | 0.25 | 20 | 1 | 8614 | 4 | 2 | 3154 | 1 | 3.333 | 2045 |

Let us take the line of "Average" aside, to extract some useful tables and information.

Table 4.2 Conclusions From The 10 Experiments on Version 1

| | From Phase I to II | | From Phase II to III | | From Phase I to III | |
|---|---|---|---|---|---|---|
| | Number Of Sets | Running Time | Number Of Sets | Running Time | Number Of Sets | Running Time |
| Decreasing In Numbers | **20 to 6.8** | **9910.4 to 4205** | **6.8 to 3.9** | **4205 to 3150.2** | **20 to 3.9** | **9910.4 to 3250.2** |
| Decreasing In Percentage | **66%** | **57.569%** | **42%** | **25.084%** | **80.5%** | **67.204%** |

From this small table, we can see that in the project's version1, Low, we have relatively a very high decreasing in both the number of sets of transactions that runs sequentially and running time.

Even form phase II to III, we can conclude that adopting two bits instead of one is quite efficient despite the fact that we have an increasing in the computations, that is, in using the logical operator "OR".

## 4.2 Version 2

In version two, the number of operations within each transaction varies from 6 to 10.

Table 4.3 10 Experiments on Version 2

| | Total Number | | | Average | | | Phase I | | | Phase II | | | Phase III | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Experiment # | Operations | Queries | Updaters | Opers/ Trans | Queries/ Trans | Updaters/ Trans | Number of sets | Transactions/ Set | Running Time | Number of sets | Transactions/ Set | Running Time | Number of sets | Transactions/ Set | Running Time |
| 1 | 171 | 154 | 17 | 8.55 | 7.7 | 0.85 | 20 | 1 | 10648 | 15 | 1.333 | 8150 | 13 | 1.538 | 701 |
| 2 | 161 | 142 | 19 | 8.05 | 7.1 | 0.95 | 20 | 1 | 9239 | 18 | 1.111 | 7693 | 12 | 1.666 | 602 |
| 3 | 154 | 137 | 17 | 7.7 | 6.85 | 0.85 | 20 | 1 | 7618 | 17 | 1.176 | 7887 | 15 | 1.333 | 950 |
| 4 | 157 | 140 | 17 | 7.85 | 7 | 0.85 | 20 | 1 | 9070 | 15 | 1.333 | 8024 | 10 | 2 | 446 |
| 5 | 164 | 148 | 16 | 8.2 | 7.4 | 0.8 | 20 | 1 | 10319 | 18 | 1.111 | 9946 | 12 | 1.666 | 766 |
| 6 | 165 | 140 | 25 | 8.25 | 7 | 1.25 | 20 | 1 | 7710 | 20 | 1 | 7664 | 18 | 1.111 | 649 |
| 7 | 162 | 149 | 13 | 8.1 | 7.45 | 0.65 | 20 | 1 | 9755 | 16 | 1.25 | 9414 | 12 | 1.666 | 649 |
| 8 | 166 | 149 | 17 | 8.3 | 7.45 | 0.85 | 20 | 1 | 8679 | 18 | 1.111 | 9615 | 14 | 1.428 | 996 |
| 9 | 155 | 142 | 13 | 7.75 | 7.1 | 0.65 | 20 | 1 | 7881 | 17 | 1.176 | 10555 | 10 | 2 | 593 |
| 10 | 151 | 139 | 12 | 7.55 | 6.95 | 0.6 | 20 | 1 | 9132 | 20 | 1 | 8150 | 12 | 1.666 | 610 |
| Average | 160.6 | 144 | 16.6 | 8.03 | 7.2 | 0.83 | 20 | 1 | 9005.1 | 17.4 | 1.1601 | 8710 | 12.8 | 1.6074 | 6966. |
| Max | 171 | 154 | 25 | 8.55 | 7.7 | 1.25 | 20 | 1 | 10648 | 20 | 1.333 | 10555 | 18 | 2 | 996 |
| Min | 151 | 137 | 12 | 7.55 | 6.85 | 0.6 | 20 | 1 | 7618 | 15 | 1 | 7664 | 10 | 1.111 | 446 |

Table 4.4 Conclusions From The 10 Experiments on Version 2

| | From Phase I to II | | From Phase II to III | | From Phase I to III | |
|---|---|---|---|---|---|---|
| | Number Of Sets | Running Time | Number Of Sets | Running Time | Number Of Sets | Running Time |
| Decreasing In Numbers | **20 to 17.4** | **9005.1 to 8710** | **17.4 to 12.8** | **8710 to 6966** | **20 to 12.8** | **9005.1 to 6966** |
| Decreasing In Percentage | **13%** | **3.277%** | **26.436%** | **20.022%** | **36%** | **22.643%** |

It is clear that we still have a decreasing in both; the number of sets and the running time. However, comparing with the version 1, there is relatively a high difference in that decreasing. Accordingly, we can say that, when the number of operations increases within the transaction, the efficiency of the adopted concurrency control model decreases.

Anyway, after the results of version 3, we can conclude a more generalized rule in this regard.

## 4.3 Version 3

In version three, the number of operations within each transaction varies from 11 to 15.

Table 4.5 10 Experiments on Version 3

| | Total Number | | | Average | | | Phase I | | | Phase II | | | Phase III | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Experiment # | Operations | Queries | Updaters | Opers/ Trans | Queries/ Trans | Updaters/ Trans | Number of sets | Transactions/ Set | Running Time | Number of sets | Transactions/ Set | Running Time | Number of sets | Transactions/ Set | Running Time |
| 1 | 255 | 225 | 30 | 12.75 | 11.25 | 1.5 | 20 | 1 | 9930 | 20 | 1 | 11290 | 17 | 1.176 | 7494 |
| 2 | 260 | 239 | 21 | 13 | 11.95 | 1.05 | 20 | 1 | 10024 | 20 | 1 | 7868 | 15 | 1.333 | 8432 |
| 3 | 260 | 231 | 29 | 13 | 11.55 | 1.45 | 20 | 1 | 9755 | 20 | 1 | 8477 | 16 | 1.25 | 8709 |
| 4 | 268 | 234 | 34 | 13.4 | 11.7 | 1.7 | 20 | 1 | 8820 | 20 | 1 | 8291 | 18 | 1.111 | 8540 |
| 5 | 263 | 232 | 31 | 13.15 | 11.6 | 1.55 | 20 | 1 | 9929 | 20 | 1 | 8508 | 19 | 1.052 | 8415 |
| 6 | 261 | 230 | 31 | 13.05 | 11.5 | 1.55 | 20 | 1 | 9070 | 20 | 1 | 8991 | 16 | 1.25 | 8774 |
| 7 | 254 | 224 | 30 | 12.7 | 11.2 | 1.5 | 20 | 1 | 10210 | 20 | 1 | 7569 | 17 | 1.176 | 1015 |
| 8 | 260 | 244 | 16 | 13 | 12.2 | 0.8 | 20 | 1 | 11289 | 20 | 1 | 6151 | 12 | 1.666 | 5713 |
| 9 | 263 | 231 | 32 | 13.15 | 11.55 | 1.6 | 20 | 1 | 8164 | 20 | 1 | 9306 | 16 | 1.25 | 8664 |
| 10 | 266 | 229 | 37 | 13.3 | 11.45 | 1.85 | 20 | 1 | 9664 | 20 | 1 | 5803 | 18 | 1.111 | 8805 |
| Average | 261 | 232 | 29.1 | 13.05 | 11.595 | 1.455 | 20 | 1 | 9685.5 | 20 | 1 | 8225 | 16.4 | 1.2375 | 8369.7 |
| Max | 268 | 244 | 37 | 13.4 | 12.2 | 1.85 | 20 | 1 | 11289 | 20 | 1 | 11290 | 19 | 1.666 | 10151 |
| Min | 254 | 224 | 16 | 12.7 | 11.2 | 0.8 | 20 | 1 | 8164 | 20 | 1 | 5803 | 12 | 1.052 | 5713 |

Table 4.6 Conclusions From The 10 Experiments on Version 3

| | From Phase I to II | | From Phase II to III | | From Phase I to III | |
|---|---|---|---|---|---|---|
| | Number Of Sets | Running Time | Number Of Sets | Running Time | Number Of Sets | Running Time |
| Decreasing In Numbers | **20 to 20** | **9685.5 to 8225** | **20 to 16.4** | **8225 to 8369.7** | **20 to 16.4** | **9685.5 to 8369.7** |
| Decreasing In Percentage | **0%** | **15.079%** | **18%** | **-1.759%** | **18%** | **13.585%** |

It is obvious now that the restriction that is implemented within this project, the atomicity of the transaction; one thread per transaction, has led to the "bad" results when a transaction contains between 11 and 15 operations, and if one record required per one operation conflicts with another operation from a different transaction, these two transactions cannot work in parallel.

It is worth mentioning also, that without phase III, version 3 has almost no efficiency at all in terms of parallelism.

## 4.4 Comparison among the results of the three versions of the project

In this section, we will conduct a study that compares the results among the three versions of the projects especially in terms of number of sets that run sequentially, and try to see the de facto causes that have affected theses results.

Table 4.7 Comparison among the Three Versions

| | | From Phase I to II | | From Phase II to III | | From Phase I to III | |
|---|---|---|---|---|---|---|---|
| | | Number Of Sets | Running Time | Number Of Sets | Running Time | Number Of Sets | Running Time |
| Version I | Decreasing In Percentage | 66% | 57.569% | 42% | 25.084% | 80.5% | 67.204% |
| Version II | | 13% | 3.277% | 26.436% | 20.022% | 36% | 22.643% |
| Version III | | 0% | 15.079% | 18% | -1.759% | 18% | 13.585% |

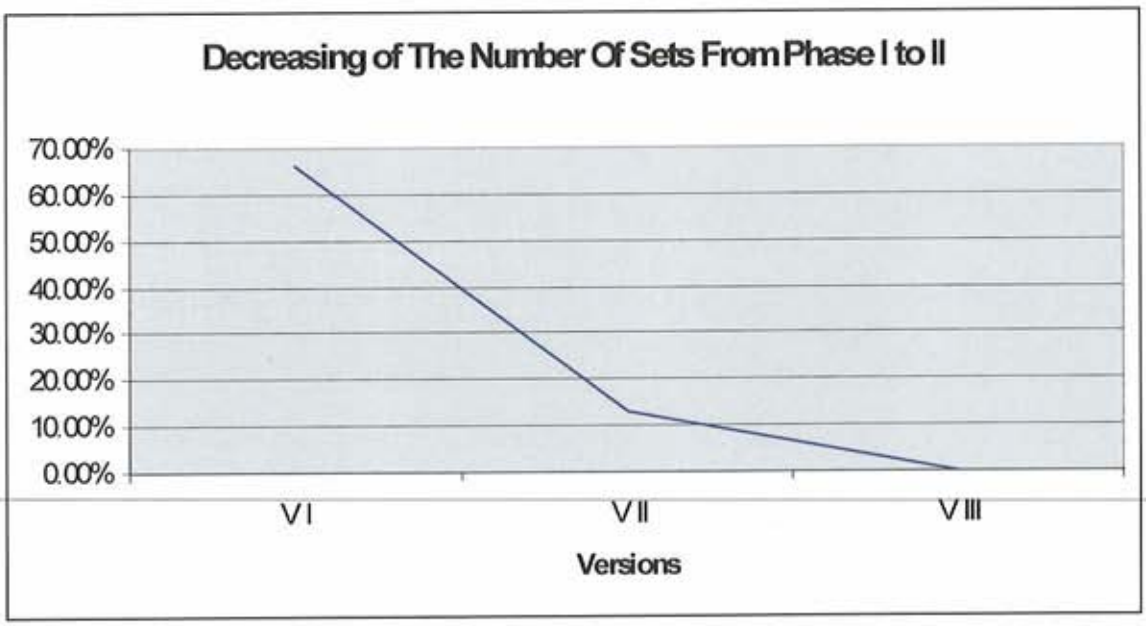


Figure 4.1 Chart Showing The Decreasing Of The Number Of Sets Among the Three Versions From Phases I to II

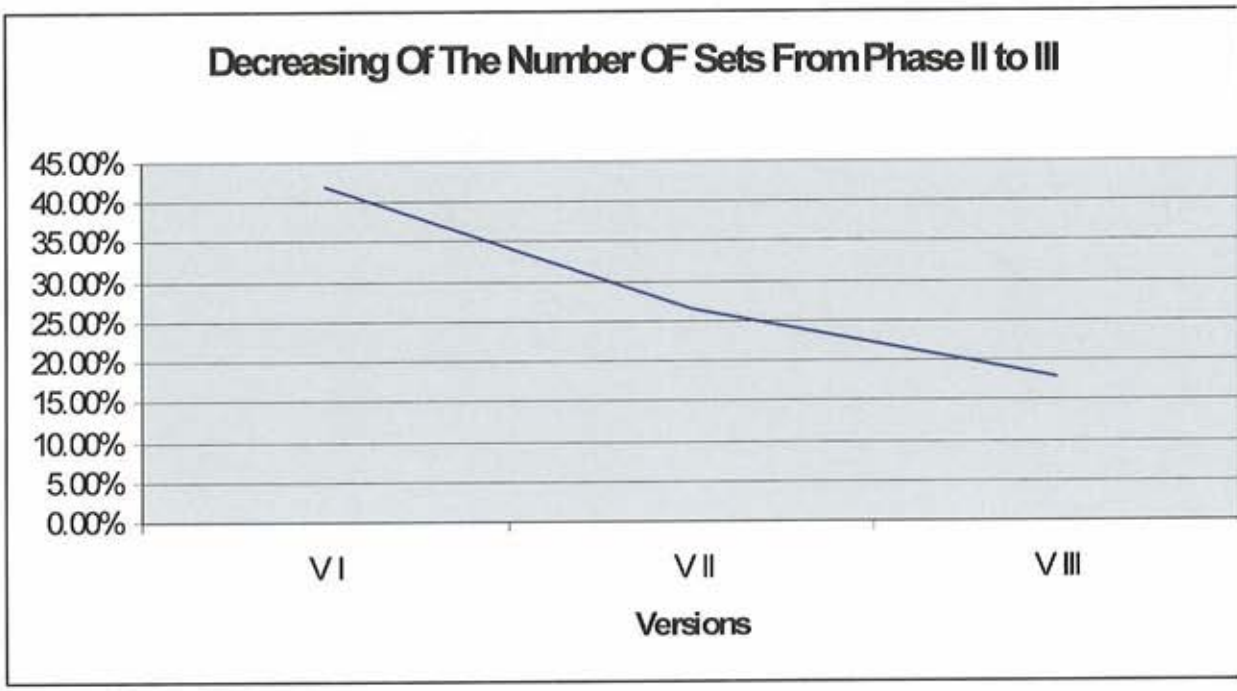


Figure 4.2 Chart Showing The Decreasing Of The Number Of Sets Among the Three Versions From Phases II to III

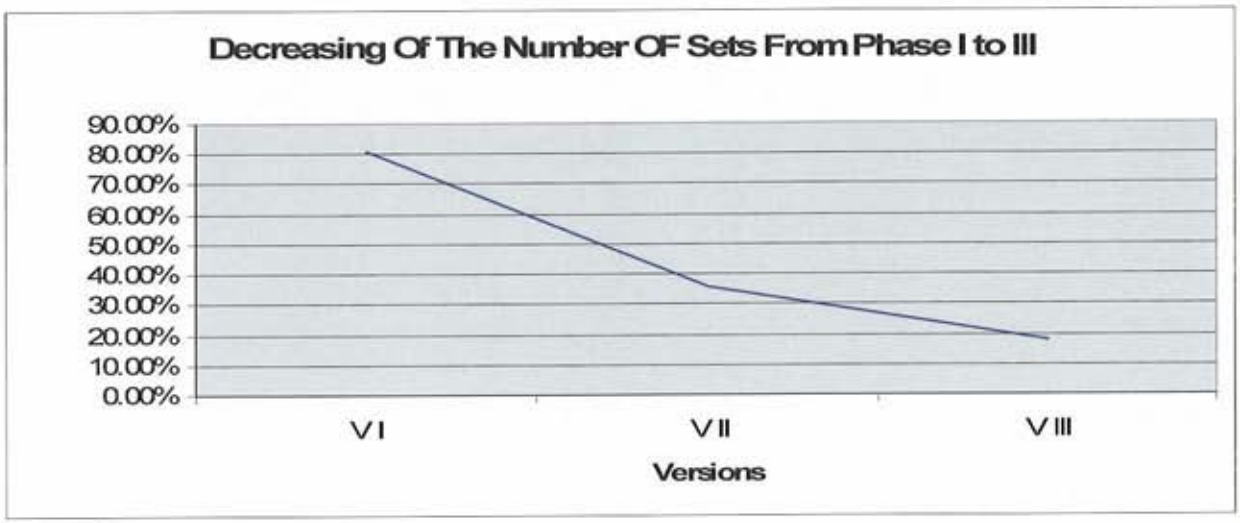


Figure 4.3 Chart Showing The Decreasing Of The Number Of Sets Among the Three Versions From Phases I to III

# Chapter 5

# Conclusion and Future Work

In this project, we have presented relatively, a new techniques for concurrency control model for the real-time database systems known as, transaction processing systems, in order to solve the locking mechanism by using an array (2D and 3D) of Boolean elements.

To improve efficiency of such systems (TPS), we have tried to let more than 1 transactions to run concurrently or simply, interleaved with others, as long as, no common data items, records in this project, are in use mutually.

Experimental results conducted on the three version of the project, depending on the number of operations per transaction, has proved, that despite the additional computational operations, the proposed approaches are efficient in both terms, decreasing the number of sets of transactions that run sequentially, and the running time needed to complete the whole sets of transactions.

Experimental results have shown also, that using two bits instead of one for locking mechanisms, is much more efficient in all versions, especially the last one, when we have from 11 to 15 operations per transaction. In the latter case the phase II, using one bit for both locks; read and write, has no effects at all on the parallelism.

In conclusion both the fact of atomicity of transactions and assigning a thread for each transaction and not for each operation within transaction, can simply explain the results.

Future work includes, as just mentioned, assigning a thread for each operation within a transaction, with the respect of serializability, can improve or even "weaken" the restriction of the atomicity of the transaction. Also future work can be conducted on a larger database, and make more than on every version of the project. However, this time not depending on the number of operation, on the percentage of accessing the database, for instance 30%, 50%, and 70% and so on.

# References


Bernstein P.A. and Goodman N. (1984). An algorithm for concurrency control and recovery in replicated distributed databases. ACM Transactions on Database Systems, 9(4):596–615.

Bernstein P., Brodie M., Ceri S., DeWitt D., Franklin M., Garcia-Molina H., J. Gray, J. Held, J. Hellerstein, H. V Jagadish, M. Lesk,D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. (1998). The Asilomar report on database research. Technical Report MSRTR-98-57, Microsoft Research, One MicrosoftWay, Redmond,WA 98052.

Elmasri R. and Navathe S. (2003). Fundamentals of Databse Systems. Addison Wesley.

Gançarski S., Naacke H., Pacitti E., Valduriez P. (2002). Load Balancing of Autonomous Applications and Databases in a Cluster System, Parallel Processing with Autonomous Databases in a Cluster System, Int. Conf. of Cooperative Information Systems (CoopIS).

Gray J. and Reuter A. (1993). Transaction Processing: Concepts and Techniques. Morgan Kaufmann.

Hakimzadeh, H. (1992), "ROLL Concurrency Control", Computer Science Department, technical report Number: NDSU-CSOR-TR-1992-20). North Dakota State University, Fargo, ND.

Herlihy, M.P. And Wing, J.M. (1990). "Linearizability: A correctness condition for concurrent objects." ACM Trans Program. Lang. Syst. 12, 3, 463-492.

M Tamer Ozsu and Patrick Valduriez. (1999). *Principles of Distributed Database Systems*. Prentice Hall.

P.A. Bernstein and E. Newcomer. Principles of Transaction Processing: For the Systems Professional. Morgan Kaufmann.

Perrizo W., Hakimzadeh H., Haraty R., and Panda B. (1992). A Concurrency Control Model For Multilevel Secure Object-Oriented Databases.

Stonebraker M. (1979). Concurrency control and consistency of multiple copies of data in distributed Ingres. IEEE Transactions on Software Engineering, 5(3):188–194.

Thomas R. H. (1979). A majority consensus approach to concurrency control for multiple copy databases. ACM Transactions on Database Systems, 4(2):180–209.