Transactional Concurrency Control for Resource Constrained Applications

Kamal Solaiman

A thesis submitted for the degree of
Doctor of Philosophy

School of Computing Science, Newcastle University

May 2014

Declaration

I certify that no parts of the material included in this thesis have previously been submitted by me for a degree at Newcastle University or any other university.

Parts of the work presented in this thesis have been published in the following:

1. K Solaiman, M. Brook, G Ushaw, G. Morgan, ‘ Optimistic Concurrency Control for Energy Efficiency in the Wireless Environment’, in the 13th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), pp. 115-128, Springer International Publishing, 2013. (Best paper award)

2. K Solaiman, M. Brook, G. Ushaw, G. Morgan, ‘A Read-Write-Validation Approach to Optimistic Concurrency Control for Energy Efficiency of Resource-Constrained Systems’, in the 9th International Wireless Communication and Mobile Computing Conference (IWCMC), IEEE, pp. 1424-1429, 2013.

3. K. Solaiman and G. Morgan, ‘Later Validation/Earlier Write: Concurrency control for Resource-Constrained Systems with Real-Time Properties’, in 30th Symposium on Reliable Distributed Systems Workshops (SRDS), IEEE, PP. 9-12, Oct. 2011.

4. K. Solaiman and G. Morgan, ‘Later Validation/Earlier Write: Concurrency Control for Resource-Constrained Systems with Real-Time Properties’, Poster Session at Computing Department, Newcastle University, 2012.

Dedication

I dedicate this thesis to my beloved daughter (Hdel); she was my infinite resource of love during my PhD study.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr Graham Morgan for his advice and comments throughout the work on this thesis and for his support and encouragement during the difficult times I faced in previous years.

I would like to thank my colleagues and friends in the School of Computing Science at Newcastle University for their positive discussions and comments, especially Dr. Gary Ushaw, Matthew Brook and Ayad Keshlaf.

Certainly, I would like to thank my beloved father and mother for their continuous support throughout my study. Finally, I would like to thank my devoted wife for her support and patience during my research.

Abstract

Transactions have long been used as a mechanism for ensuring the consistency of databases. Databases, and associated transactional approaches, have always been an active area of research as different application domains and computing architectures have placed ever more elaborate requirements on shared data access. As transactions typically provide consistency at the expense of timeliness (abort/retry) and resource (duplicate shared data and locking), there has been substantial efforts to limit these two aspects of transactions while still satisfying application requirements. In environments where clients are geographically distant from a database the consistency/performance trade-off becomes acute as any retrieval of data over a network is not only expensive, but relatively slow compared to co-located client/database systems. Furthermore, for battery powered clients the increased overhead of transactions can also be viewed as a significant power overhead. However, for all their drawbacks transactions do provide the data consistency that is a requirement for many application types. In this Thesis we explore the solution space related to timely transactional systems for remote clients and centralised databases with a focus on providing a solution, that, when compared to other's work in this domain: (a) maintains consistency; (b) lowers latency; (c) improves throughput. To achieve this we revisit a technique first developed to decrease disk access times via local caching of state (for aborted transactions) to tackle the problems prevalent in real-time databases. We demonstrate that such a technique (rerun) allows a significant change in the typical structure of a transaction (one never before considered, even in rerun systems). Such a change itself brings significant performance success not only in the traditional rerun local database solution space, but also in the distributed solution space. A byproduct of our improvements also, one can argue, brings about a "greener" solution as less time coupled with improved throughput affords improved battery life for mobile devices.

Contents

List of Tables ix

List of Figures x

List of Algorithms xiii

Glossary xiv

1 Introduction **1**

1.1 Introduction 1

1.2 The Concept of Transaction 1

1.3 Concurrency Control Approaches 2

1.3.1 Pessimistic Concurrency Control 2

1.3.2 Optimistic Concurrency Control (OCC) 3

1.3.3 Pessimistic vs Optimistic 4

1.4 Cost of Aborted Transactions 4

1.5 Research Contributions.. 5

1.6 Publications 7

1.7 Thesis Structure 8

2 Background and Related Work **9**

2.1 Database Consistency 9

2.1.1 Serial Schedule 12

2.1.2 Serializable Schedule 12

2.1.3 non-serializable schedule 13

2.2 Database Architecture 13

2.2.1 Centralized Database 13

2.2.2 Distributed Database 14

2.2.3 Mobile Database 15

2.3 Real-time Database ………………………………………………………… 16
2.3.1 Timelines Requirements ………………………………….. 17
2.3.2 Handling Late Transactions ……………………………….. 18
2.4 General Enhancement ……………………………………………….. 18
2.4.1 Caching ………………………………………………….... 18
2.4.2 Virtual Execution ………………………………………….. 21
2.5 Related Work 22
2.5.1 Optimistic Concurrency Control Techniques ……................. 22
2.5.2 Aspects of Optimistic Concurrency Control …..…………….. 26
2.5.3 Data Delivery Choice ……………………………...………… 38
2.5.4 Broadcast Datacycle Approach …………………….…..….... 40
2.5.5 Indexing on the Air …………………………………………... 41
2.5.6 Broadcast Disks ……………………………………………..... 43
2.5.7 Fault-tolerant in Broadcast Disks ……………………….……. 45
2.5.8 Forward and Backward OCC (FBOCC) ……………...……… 46
2.5.9 Discussion ……………………………………………………. 49
2.6 Summary ……………….………………………………………….…. 51
3 The Read-Write-Validate **52**
3.1 Introduction ……………………………………………..…………… 52
3.2 Read-Write-Validate Approach ……………………………………… 54
3.3 Justifying Read-Write-Validate Approach ………………………….…….. 55
3.4 Advantages of Read-Write-Validate Approach ……………………...... 55
3.4.1 Transaction Lifespan Minimized ……………............................. 56
3.4.2 Blocking of Concurrent Transactions Eliminated ….…………… 57
3.4.3 Newly Starting Transactions Never Blocked …………………..... 59
3.4.4 Earlier Visible Updates …………….………………………….. 62

3.5 Disadvantages of Read-Write-Validate Approach ……………………. 64

3.5.1 Longer Wasted Execution ……………………………………… 64

3.5.2 Critical Section Constraint ……………………………………… 66

3.6 Read-Write-Validation Enhancement ……………………………….. 66

3.6.1 Energy Efficiency Improvement ……………………………….. 66

3.6.2 Reduction of Conflict Risk ……………………………………… 67

3.6.3 Wasted Execution Elimination ………………………………… 67

3.7 Coping with System Failures …………………………………………….. 68

3.8 Read-Write-Validate Protocol ………………………………………. 68

3.8.1 Protocol Description ……………………………………………… 69

3.8.2 Pseudo-code …………………………………………………………. 70

3.9 Distributed Read-Write-Validate Protocol ……………………………. 71

3.9.1 Validation Stage at Client ……………………………………… 72

3.9.2 Validation Stage at the Server ………………………………... 73

3.10 Summary ………………………………………………………………… 76

4 Evaluation **77**

4.1 Read-Write-Validate Protocol ………………………………………….. 77

4.1.1 Simulation Tool ……………………………………………………. 77

4.1.2 Simulation Model and Setting ……………………………………. 78

4.1.3 Simulation Results ………………………………………………….. 80

4.2 Distributed Read-Write-Validation Protocol …………………………... 85

4.2.1 Simulation Model and Setting ……………………………………. 85

4.2.2 Simulation Results ………………………………………………... 87

4.3 Summary ………………………………………………….………………….. 92

5 Conclusion and Future Work **93**

5.1 Introduction …………………………………………………………… 93

5.2 Contributions of the Thesis …………………………………………… 94

5.2.1 Advantages Gained by the present research …………..…….. 94

5.3 Future Work ………………………………….…………………….... 96

References **98**

List of Tables

1 Simulation Parameters of Read-Write-Validation Protocol **80**

2 Simulation Parameters of Distributed Read-Write-Validation Protocol .. **87**

List of Figures

1.1	OCC algorithms phases …………………………………………………...	4
2.1	Lost update anomaly …………………………………………………..	10
2.2	Inconsistent retrievals ……...…………………………………………	11
2.3	Model of centralized database architecture in a network ……………..….	14
2.4	Model of distributed database architecture ……………………………...	15
2.5 (a)	First run in virtual executions environments …………………………….	22
2.5(b)	Rerun in virtual execution environments ………………………………...	22
2.6	Illustrates serious and non-serious conflicts …..………………………….	28
2.7 (a)	Frequently rolled back scenario ……..…………………………………...	30
2.7 (b)	Transactions backed off scenario …….…………………………………..	30
2.8	Transactions management under an OCC broadcast commit scheme ...….	34
2.9	Schedule with an undeveloped possible conflict ...……………………….	34
2.10	Schedule with a developed conflict ……………………………………….	35
2.11	Pull-based delivery...……………………………………………………	39
2.12	Push-based delivery ………………………..………………………	39
2.13	Broadcast datacycle with no indexing …………………………………	41
2.14	Broadcast datacycle in tune_opt strategy ……………………………...	42
2.15	Broadcast datacycle in (1,m) strategy …………………………………...	43
2.16	Flat broadcast approach ………………………………………………….	44
2.17	Three different broadcast programs ……………………………………...	44
2.18	Broadcast program generation ……………………………………………..	45
2.19	Transaction execution schedule …………………………………………	49
3.1	Read-Write-Validate approach phases ………………..………………..	55
3.2	Transaction lifespan in conventional OCC ………………………………	57
3.3	Transaction lifespan in Read-Write-Validate OCC ……………………...	57

3.4	The occurrence of blocking in the conventional OCC approach...………	59
3.5	Blocking eliminated in Read-Write-Validate approach …..…………….	60
3.6	Blocking in newly starting transactions in the conventional OCC approach ………………………………………………………………	62
3.7	New transactions are never blocked in Read-Write-Validate approach …	63
3.8	Late visible updates in the conventional OCC approach ……………..…	64
3.9	Earlier visible updates in the Read-Write-Validate approach ……….….	65
3.10	Short period of wasted execution in the OCC conventional approach ….	66
3.11	Longer period of wasted execution in the Read-Write-Validate approach	66
3.12	Wasted executions no longer exist in the virtual execution environment.	69
4.1	Throughput with 50% of updates transactions …………………….…….	83
4.2	Average response times with 50% of update transactions ……………...	83
4.3	Late transactions with 50% of update transactions ………………….…..	83
4.4	Throughput with 75% of updates transactions …………………………..	85
4.5	Average response times with 75% of update transactions …………….....	85
4.6	Late transactions with 75% of update transactions ………………….…..	85
4.7	Throughput at the server ………………………………………….……....	89
4.8	Response time at the server ……………………………………………...	89
4.9	Miss Rate at the server ……………………………………………….…....	89
4.10	Throughput of update transactions at clients …………………………….	91
4.11	Late update transactions at clients ……………………………………..…	91
4.12	Throughput of read only transactions at clients ……………………….....	92
4.13	Late read-only transactions at clients ………………………………….....	92

List of Algorithms

1	Backward oriented optimistic concurrency control ……………………	23
2	Forward oriented optimistic concurrency control ……………………..	24
3	Partial backward validation (FBOCC) ……………………………………	46
4	Forward validation (FBOCC) ………………………………………..	47
5	Final validation (FBOCC) ………………………………………………	48
6	Validation phase (Read-Write-Validate) …………………………………	72
7	Partial backward validation (distributed Read-Write-Validate) ………...	74
8	Final backward validation (distributed Read-Write-Validate) …………...	75
74	Read-Write-Validation validation (distributed Read-Write-Validate)…….	76

Glossary

Transaction	A sequence of operations (reads and writes) executed to perform a single logical task.
ACID	Properties of transaction, including Atomicity, Consistency, Isolation and Durability.
Database	A database is an organized collection of data items; each could be in the form of a record, page, data structure, picture, text, etc.
Real-Time database	A database which maintains traditional database requirements (logical consistency); and also satisfies time-constraints (temporal consistency).
Ubiquitous Database	A small database existing in mobile devices.
Distributed Database	A database physically stored across multiple computers in multiple locations which are connected to each other via a network, yet operate logically as a single database.
DBMS	Database Management System - a special application designed to interact with users.
Serializability	A well-known correctness criteria which means that there is at least one serial schedule which leads to the same final state of the database.
CC	Concurrency Control, a mechanism for coordinating Simultaneous access to shared data.

OCC	Optimistic Concurrency Control, provide a mechanism whereby simultaneously executing transactions validate with one another to determine whether a conflict has occurred, It is a well-known method due to the properties of non-blocking and deadlock-free execution.
FOCC	Forward Optimistic Concurrency Control, OCC based on checking the intersection between the write set of a validating transaction and the read sets of currently executing transactions.
BOCC	Backward Optimistic Concurrency Control, OCC based on checking the intersection between the read set of a validating transaction and the write sets of currently executing transactions.
FBOCC	Forward and Backward Optimistic Concurrency Control, an OCC algorithm suitable for mobile transactions in wireless broadcast environments. It consists of two validation stages, one involving backward validation at the client, and the other forward validation at the server.
2PL	Two-phase locking protocol, CC technique based on locks, which are divided into growing and shrinking phases in each transaction. In the growing phase, a transaction can request locks, but in the shrinking phase a transaction should unlock all locks that have been made in the first phase.

Chapter 1

Introduction

1.1 Introduction

This thesis is concerned with improving performance in shared client access to database systems. In particular, a measure of performance is quantified in terms of client request throughput. If client requests simultaneously update the same data, then erroneous behaviour in the overall system may result. The basic method to overcome this would be via the use of transactions. Therefore, the transactional style of access is used as the basic construct for modelling client requests.

1.2 The Concept of the Transaction

Transactions are a sequence of read and write operations executed in performing a single logical task. Transactions have four properties: atomicity, consistency, isolation and durability (ACID) [1][2]. These properties are described below.

1. Atomicity: this means that all operations involved in a transaction should be seen as one single operation. If one action belonging to a transaction fails, then the entire transaction fails.

2. Consistency: this is a general term used to signify that data must meet all of the validation rules that applications expect.

3. Isolation: this means that any concurrently running transactions do not affect each other at the time of execution. As an example, if T1, T2 and T3 are transactions running concurrently, they should have some equivalent serial order.

4. Durability: this refers to a guarantee that, if a transaction completes, then its effects persist in the database and it is never lost, even if the system crashes. Nevertheless, durability does not imply a permanent state of the database; other transactions may overwrite changes made in current transactions without undermining durability.

Executing transactions in the presence of concurrency requires a concurrency control mechanism to coordinate access to shared data. In such a setting, the main goal of a concurrency control algorithm is the creation of an ordering of read/write access that ensures database consistency.

1.3 Concurrency Control Approaches

Concurrency control has been extensively studied in the literature, resulting in various ways of implementing transactions to maintain database consistency. The two main categories are pessimistic and optimistic approaches [2][3].

1.3.1 Pessimistic Concurrency Control

A straightforward solution to coordinate access to shared data is to simply lock data while it is being accessed by one client, preventing any possible conflict from other clients occurring [4]. Locks are controlled by the concurrency control manager in order to ensure that:

1. Every transaction cannot read or write any element unless it previously requested a lock on that element and has not yet released it.
2. If a transaction locks an element then it must release it later.
3. No more than one transaction can lock the same element at any time.

A two-phase locking protocol (2PL) is a pessimistic approach proposed by Eswaran *et al.* [5]. In 2PL, locks are divided into growing and shrinking phases in each transaction. In the growing phase, a transaction can request locks, but in the shrinking phase a transaction should unlock all locks that have been made in the first phase. Therefore, for each transaction, all lock requests must precede all unlock requests. Although the 2PL protocol grants serializability, it is considered to be too constrained. The general weaknesses of locking approaches can be summarised as following [6][7][8].

1. It is required to always use locking to ensure consistency, even if most of the transactions do not overlap. However, locking is only actually needed in certain cases.

2. Lock maintenance adds an unnecessary overhead to read-only transactions even though these do not affect the consistency of the database and constitute the majority of system transactions [23].
3. When a large part of a database resides in a secondary storage, locking frequently accessed data items significantly decreases concurrency due to the waiting time needed for secondary storage access.
4. Keeping locks in place until the end of the execution of transactions in order to avoid cascading aborts causes a further decline in concurrency.
5. Deadlock problems make 2PL inappropriate in distributed database systems, since current deadlock detection techniques for distributed systems are complex and ineffective.
6. The significantly increased numbers of transactions occurring in distributed database systems increases locks' overhead and the probability of lock conflict. Furthermore, communication delays lead to a worsening of the situation due to increasing lock-hold duration, which makes the probability of lock conflicts even higher. This results in a substantial decline in performance in distributed database systems.

1.3.2 Optimistic Concurrency Control (OCC)

Kung and Robinson proposed the use of optimistic approach methods via the execution of transactions in three phases as shown in Figure 1.1 in order to avoid the problems pointed out in the previous section [8]. During the read phase, transactions access data without restrictions and make their own private copies of such data. All computation carried out by a transaction occurs on a private copy. When a write is requested, it is enacted on the private copy. During the validation phase, resolution policy is enacted where, in principle, other executing transactions are considered to determine whether or not the write requests can be satisfied without invalidating the correctness of the overall read/write schedule. If the writes are valid, the write phase is enacted which commits the changes to persistent storage. Alternatively, the transaction may abort if a valid schedule is not possible, and a renewed attempt is made later. If a transaction has no write operation, then the write phase is not required, with commitment being enacted to bring the transaction to a logical end.

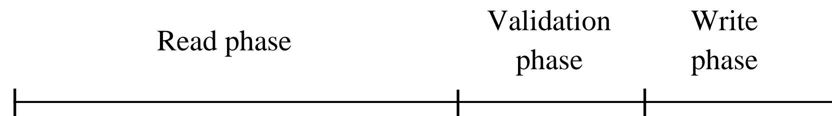


Figure 1.1 OCC phases.

The optimistic approach can overcomes the weaknesses of pessimistic approaches, and works well in low contention environments especially when read operations outnumber writes. However, rollback is a considerable drawback in optimistic approaches when conflict rates are high.

1.3.3 Pessimistic vs Optimistic

In conventional databases, a pessimistic approach is better than an optimistic approach in high contention environments, particularly when physical resources are limited. In such environments, an optimistic approach results in considerable numbers of transactional aborts, which leads to substantial waste of resources. However, the optimistic approach is more convenient in low contention environments, particularly when the amounts of wasted resources involved are tolerable, and the optimistic approach provides a higher degree of concurrent executions [2][9][10]. In contrast, the optimistic approach works better than the pessimistic approach in real-time databases even given high data contention over a wide range of resource availability levels. This is because, in the optimistic approach, conflict resolution is delayed until the times at which transactions commit, which helps in making better conflict decisions. So, optimistic algorithms ensure that no transactions which are likely to miss their deadlines prevent other transacting execution in the system [2][11][12][13][14].

1.4 Cost of Aborted Transactions

Aborted transactions run again, and this requires them to retrieve data again from the database. The state of the database may by then have changed, and so it is necessary to request data again to prevent transactions from using inconsistent data. Accessing storage devices is expensive and aborted transactions that run again are, in essence, duplicating information retrieval. However, sometimes only a small proportion of the re-retrieved data has changed. Retrieving data that has not changed is, therefore, a waste

of disk access. This is more serious in distributed databases when the data to be retrieved does not exist in the same machine, and is therefore more expensive in terms of incurring communication costs.

To improve performance, technique has been developed to only retrieve those items of data that have changed. Such a technique is called virtual execution, which allows an aborted transaction to continue reading the data it requires, and that data is cached locally so that upon rerun it does not need to waste resources reading the data again. Using this ”pre-fetched” data can lead to significant performance improvements, as there should be no disk I/O overhead involved in rerunning a read transaction where the data is already cached. However, there is now an issue with consistency when considering a transaction that is rerunning with pre-fetched data. Clearly some of the pre-fetched data may have been modified at the server since it was read to the local cache, which would result in the transaction running with inconsistent data. Concurrency control techniques must be applied to overcome this problem [15].

1.5 Research Contributions

This thesis introduces a novel Read-Write-Validate transactional phase sequence combined with virtual execution to render the conventional OCC approach appropriate for mobile device environments. The proposed approach presented in two contexts:

- Firstly, it is show that implementing the proposed approach on the mobile devices themselves can improve contention issues with shared resources on that device, such as the solid-state disk. [16][17].
- Secondly, it is further shown that the implementation of the proposed approach in client-server model based on a broadcast datacycle approach for wireless environment is efficient [18].

The results show that, with the proposed approach, overall system performance is improved, and the number of transactions that miss their deadlines due to concurrency issues is reduced. The number of transactions requiring a restart is reduced, and so less energy is used in re-accessing a resource or in retransmitting data a second time [17][18].

The benefits gained by the contribution made in this thesis are summarised below.

1. *Transaction Lifespan Minimization*

   The lifespan of a transaction is the time between the start of a transaction and when it commits or the end of the write phases. The validation phase adds a non-deterministic timing period to the lifespan of the transaction. Therefore, the reordering of phases in the proposed approach removes from the transaction's lifespan the non-deterministic timing of the validation phase.

2. *The Blocking of Concurrent Transactions is Eliminated*

   In the conventional OCC approach, non-conflicted transactions executing in the read phase will eventually be blocked after having been validated while the validating transaction executes in the validation and write phases. This temporary blocking is essential to prevent non-conflicted transactions from entering a conflict state. Using the proposed approach, none-conflicted transactions no longer have to be blocked from progressing and yet database consistency is still maintained.

3. *Newly Starting Transactions are Never Blocked*

   Newly starting transactions are those which may start execution while another transaction is executing in the validation or write phase. In the conventional OOC approach, such transactions will be temporally blocked until the validating transaction commits, in order to prevent them from entering a conflict state. In the proposed approach, newly starting transactions no longer have to be blocked from progressing and yet database consistency is still maintained.

4. *Earlier Visible Updates*

   In the proposed approach, write operations become visible to concurrent transactions earlier, affording more likelihood of reading up-to-date data and thus reducing the opportunity for conflict to occur. This is because the reordering of the validation and write phases guarantees that all new updates have already been made before the validation phase starts.

5. *Energy Efficiency Improvement*

Virtual execution allows those transactions that have been aborted to re-execute using in-memory values as opposed to reading directly from the persistent storage. This improves the proposed approach, because accessing a conventional hard disk drive is expensive in terms of power usage given that the disk must attain read speed and the appropriate data sector mist be found. Even solid-state drives are significantly more expensive to access compared to local memory. So, the reduction made in disk access leads to a reduction in energy consumption. The energy savings will be even greater if the transaction reads from a remote server over a wireless connection.

6. *Reduction of Risk of Conflict*

Rerunning transactions is quicker than those in their initial run since there is no persistent storage access which in turn increases the chance of transaction commitment. This is because transactions in rerun become ready to enter the critical section for write and validation phases in a shorter time. The shorter read phase in a rerun reduces the risk of conflict with other transactions occurring.

1.6 Publications

The contributions made in this thesis have been published in two conference papers, one workshop paper and one poster. One of the conference papers won the (best paper award) at The 13th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), 2013. In addition, a journal paper is produced and invited to be submitted to Information Sciences journal, and a survey paper is in preparation. Details of the poster and the published papers are given below:

1. K. Solaiman, M. Brook, G. Ushaw, and G. Morgan, "Optimistic Concurrency Control for Energy Efficiency in the Wireless Environment" in the 13th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), pp. 115-128. Springer International Publishing, 2013.

2. K. Solaiman, M. Brook, G. Ushaw, and G. Morgan, "A Read-Write-Validation Approach to Optimistic Concurrency Control for Energy Efficiency of Resource-Constrained Systems", in the 9th International Wireless Communication and Mobile Computing Conference (IWCMC), IEEE, pp. 1424-1429, 2013.

3. K. Solaiman and G. Morgan, "Later Validation/Earlier Write: Concurrency Control for Resource-Constrained Systems with Real-Time Properties," in 30th Symposium on Reliable Distributed Systems Workshops (SRDS), IEEE, pp. 9-12, Oct. 2011.

4. K. Solaiman and G. Morgan, "Later Validation/Earlier Write: Concurrency Control for Resource-Constrained Systems with Real-Time Properties", Poster Session at Computing Department, Newcastle University, 2012.

1.7 Thesis Structure

The rest of this thesis is organised as follows:

- Chapter 2 covers the background to the proposed approach. It starts with a description of concurrency control problems, and the concept of database consistency, and introduces centralized, distributed, mobile and real-time database types. Then previous research related to the proposed approaches is discussed, focussing on optimistic concurrency control techniques and aspects. Finally, an introduction to mobile computing, including caching and broadcast datacycle, is provided.
- Chapter 3 describes the proposed approach in detail and extensively discusses its advantages and disadvantages. Then the Read-Write-Validation protocol and its pseudo code algorithm are presented to describe how the protocol works. The Read-Write-Validation protocol deals with concurrently running transactions accessing shared data at a single mobile device. Then the Distributed Read-Write-Validation protocol is presented including pseudo code algorithms, to describe how the proposed protocol works. The Distributed Read-Write-Validation protocol is designed to control numerous mobile transactions accessing a centralised database at the server.
- Chapter 4 provides a description of the implemented simulations used to evaluate the performance of both proposed protocols, the Read-Write-Validation protocol and the Distributed Read-Write-Validation protocol. In addition, the results collected from the simulation experiments are provided.
- Chapter 5 draws the conclusions of the thesis and suggests.

Chapter 2

Background and Related Work

This chapter introduces the background and concepts necessary to understand the contribution of the thesis. The present study is primarily concerned with improving performance via the concurrency control mechanisms employed to govern the read/write ordering of concurrent transactions. The research covers both local and geographically dispersed clients, and so the architectures and the techniques employed within them to achieve improved transactional performance are described. Since performance impacts more acutely on those databases that have time requirements (real-time), one section is devoted to these approaches. This chapter starts with an introduction to database consistency, and then presents information concerning centralized, distributed, mobile and real-time databases. After that, caching and rerun policy enhancements are explained. This is followed by a discussion of the previous research relevant to the proposed approach. A description of optimistic concurrency control techniques is followed by a discussion of the trade-off of optimistic concurrency control aspects considered in the literature. Finally, mobile computing and broadcast wireless environments are introduced.

2.1 Database Consistency

A database is an organized collection of data items; each item could be in the form of a record, page, data structure, picture, or text (the general term data item is used throughout this thesis). Each single data item has a unique identifier, and the database is managed by a database management system (DBMS), which is a special application designed to interact with the users. In order to improve performance, applications in real life are allowed to run concurrently which may lead to multiple accesses to shared data simultaneously. Such multiple accesses are not secure and may lead to unexpected results. Two examples of these database anomalies are explained below:

Examples of Database Anomalies

This section illustrates two examples of database anomalies described in the literature [19][6]. Both examples explain the process of accessing one bank account for deposit and withdrawal operations by multiple users.

Example 1: Lost Updates

Consider that two customers - C1 and C2 - deposit money in the same bank account at approximately the same time. The deposit method works as follows:

```
Deposit (amount, account_number) {
temp = read(accounts[account_number];
temp= temp+ amount;
write(accounts[account_number],temp);
}
```

As illustrated in Figure 2.1, C1 reads the balance of the account (£200) and then adds £ 150 to the local copy of the balance (temp) to make it £350. C2 read the balance as well, which was still £200, and adds £50 to C2's local copy, making it £250. Then, C1's update of £350 was written back to the original database. Subsequently, C2's update of £250 was also written back to the original database (the same account). At this point, an incorrect state has resulted and £150 has been lost: the correct balance should be £400.

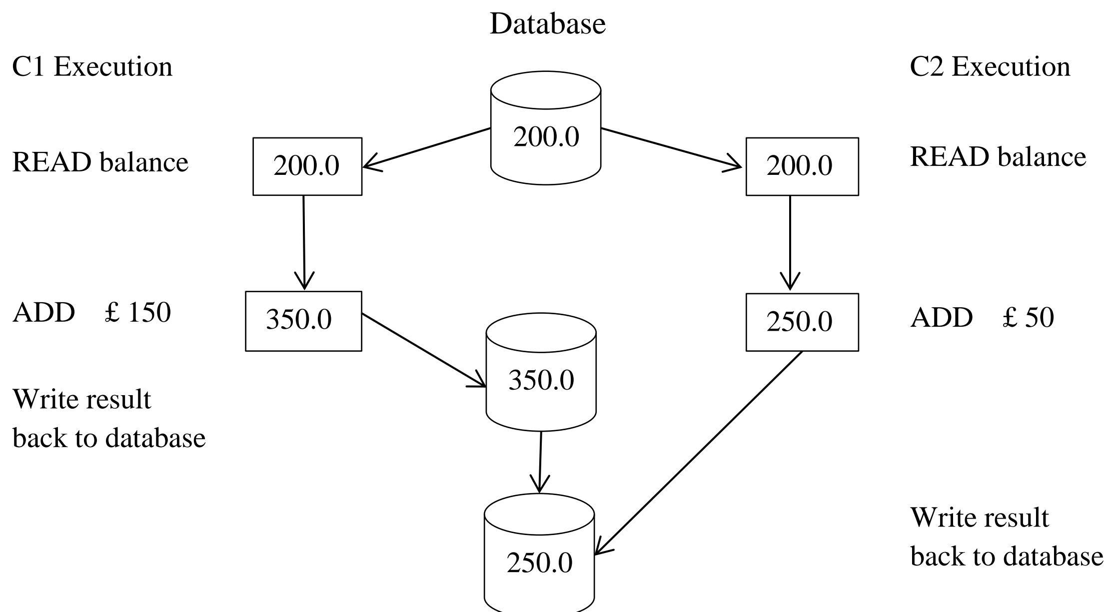


Figure 2.1 Lost update anomaly [6]

Example 2: Inconsistent Retrievals

Suppose that the two customers C1 and C2 simultaneously execute the following transactions T1 and T2 respectively:

1. T1 transfers £1000 from a checking account to the same person's saving account.
2. T2 prints the total balance of the both accounts (checking and savings).

As illustrated in Figure 2.2, T1 reads the balance of £1200 from the checking account and subtracts £1000 from it; the result is that £200 will be written back to the database. Then, at approximately the same time, T2 reads the balances of both accounts and then prints the total. T1 continues and reads the balance of the savings transaction and adds the £1000 to the previous balance. The new balance of the savings account will be £1500, which will be updated to the original database. This time, the final result placed in the database is correct, but the execution is incorrect because the total balance printed by T2 is £700, whereas the real total balance is £1700.

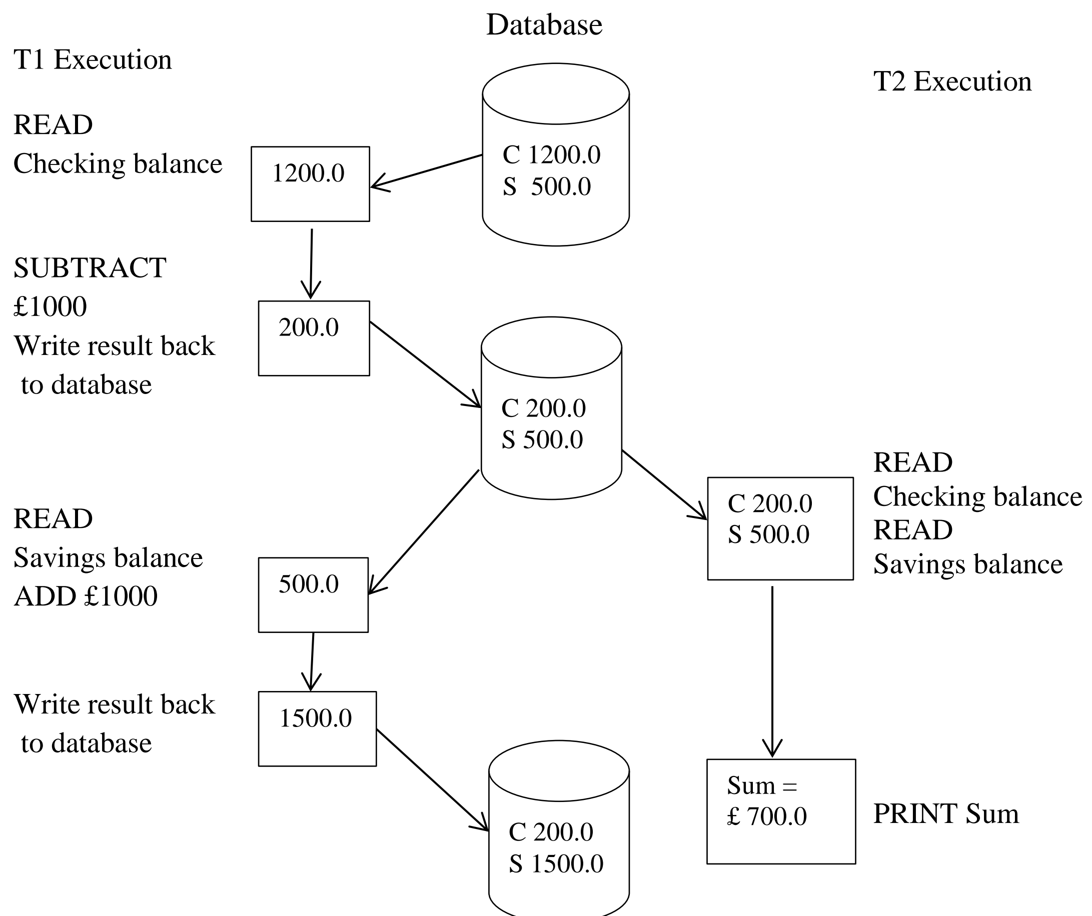


Figure 2.2 Inconsistent retrievals [6]

When all operations of a transaction are executed before or after all other transaction operations, the execution is called serial and database consistency is maintained. However, if the operations of more than one transaction are interleaved, their execution may lead to a state of inconsistency. Interleaved operations of one or more transactions are called schedules.

2.1.1 Serial Schedule

A schedule is serial if its operations consist of all the operations of one transaction, then all the operations of another transaction, and so on. Interleaved operations from different transactions are not allowed. In other words, if a schedule consists of a number of transactions T1, T2, T3,…Tm, then, for every i=1 to m-1, the transaction Ti is completed before the next transaction Ti+1 starts.

Consider the following example: T1 and T2 are two concurrent transactions.

T1 = {R1 (a), R1 (b)}
T2 = {W2 (a), W2 (b)}.
Histories:
H1 = {R1 (a), R1 (b), W2 (a), W2 (b)}
H2 = {W2 (a), W2 (b), R1 (a), R1 (b)}

H1 and H2 indicate the order of the execution of operations in transactions T1and T2. Both H1 and H2 are serial because in both histories no operations were interleaved and the effect of the schedule on the database will be equivalent to schedule T1, T2 in the case of H1 and T2, T1 in the case of H2. However, in the following history:

H3= {R1 (a), W2 (a), W2 (b), R1 (b)}

H3 is not serial because here, operation R1 (a) precedes operation W2 (a) through the data item a, which means that T1 precedes T2. On the other hand, operation W2 (b) precedes operation R1 (b) through the data item b, which means that T2 precedes T1. Therefore, H3 is not serial.

2.1.2 Serializable Schedule

A schedule is serializable if it has the same effect on the database as other serial schedule of the same transactions. Therefore, if the serial schedule maintains database consistency, then the serializable schedule also maintains database consistency.

Consider again the following histories from the previous example:

H4= {R1 (a), W2 (a), R1 (b), W2 (b)}
H5= {W2 (a), R1 (a), W2 (b), R1 (b)}

In H4, operation R1 (a) precedes W2 (a) through data item a, which means that T1 precedes T2; and R1 (b) precedes W2 (b) through data item b, which means that T1 precedes T2. Therefore, the effect of the execution of H4 is equivalent to the effect of the serial execution H1. Thus, H4 is serializable.

In H5, operation W2 (a) precedes R1 (a) through data item a, which means that T2 precedes T1; and W2 (b) precedes R1 (b) through data item b, which means that T2 precedes T1. Therefore, the effect of the execution of H5 is equivalent to the effect of the serial execution H2. Therefore, H5 is serializable as well.

2.1.3 non-serializable schedule

H3 in the previous section is an example of non-serializable history:

H3= {R1 (a), W2 (a), W2 (b), R1 (b)}

As stated in the previous section, T1 precedes T2 because of the operation R1 (a) precedes operation W2 (a) through the data item a, and T2 precedes T1 because of operation W2 (b) precedes operation R1 (b) through the data item b. Therefore, the effect of the execution of H3 is not equivalent to the effect of any serial execution, thus, H3 is non-serializable.

The concept of serializability is a popular correctness criterion that has been used in concurrency control field. Serializability means that the effect of certain schedules on the database state is equivalent to at least one serial schedule of the same transactions [20][19][6].

2.2 Database Architecture

A database is an organized collection of data. Depending on the method use to stor such data, databases can be classified into three categories: centralized, distributed, and mobile databases.

2.2.1 Centralized Database

A centralized database is a database located and maintained in one location, where access may be performed via a communications network. Banking systems are an example of centralized databases, in which processing is performed in a mainframe, and clients use online banking for their transactions. Reservation systems could be another example of centralized databases, due to its advantage of preventing the double booking problem [166]. Such systems become more complicated in distributed database environments due to double booking issues.

Centralised Database Architecture

Figure 2.3 illustrates a model of a centralised database architecture, which consists of four sites connected via a network, and the database resides at only one site (site 4 in this example). Therefore, site 4 will be responsible for database management and processing requests from other sites [21].

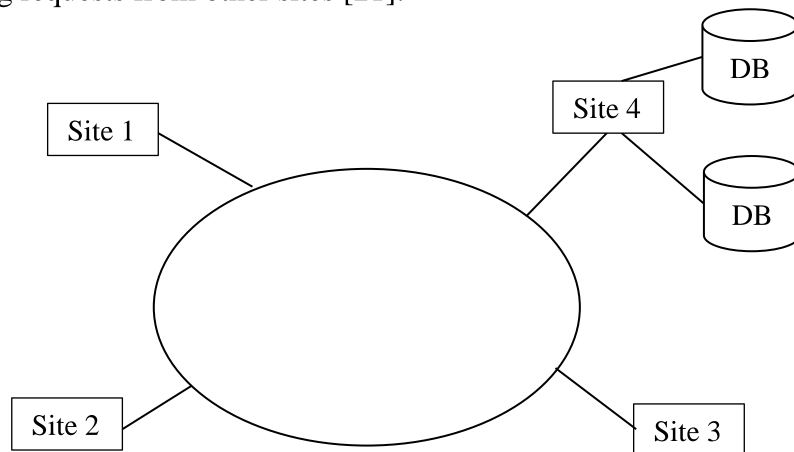


Figure 2.3 Model of centralized database architecture in a network

2.2.2 Distributed Database

A distributed database is a database physically stored across multiple computers connected to each other via a network, and these computers may be located in the same physical location or dispersed in multiple locations. In both cases, the distributed database will operate logically as a single database. Distributed databases may be managed by several database management systems (DDBMSs), where one coordinates each remote site. Therefore, each site of the distributed database system is designated to be capable of administering its local database if connections with other sites have failed, and this is known as local autonomy. On the other hand, when distributed database sites

are successfully connected to each other, the system must provide location transparency, which means that users can retrieve or update data from any site without prior knowledge of its location, so that all data in the distributed database should appear to be one logical database existing at one site.

Distributed Database Architecture

Figure 2.4 illustrates a model of a distributed database architecture, which consists of four sites connected via a network and the database is distributed between these sites. Local applications are executed at one site using data stored in the same site, not requiring data from other sites. Global applications however require data stored in other sites. Sites may have identical software, in which case the system is known as a homogeneous DDBMS, or different software in a heterogeneous DDBMS [21][22]. The client-server model is a popular modern type of architecture, providing service to clients via a communications network. Clients request a server's content or service functions and wait for the server's response. Other types of distributed models, such peer-to-peer, are beyond the scope of this thesis.

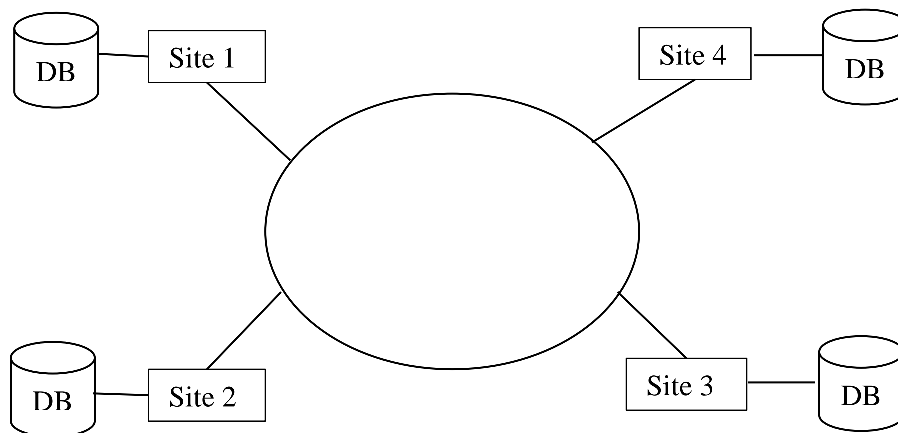


Figure 2.4 Model of distributed database architecture

2.2.3 Mobile Database

Developments in wireless networks, and mobile computing devices such as smartphones, tablets and PDAs have made mobile applications achievable and practical for use as stand-alone applications or in accessing remote applications. This has led to the necessity for mobile or ubiquitous databases. A mobile database is a small database residing on a mobile computing device, giving the ability to handle local queries without connectivity [23][24][25]. Due to the limited storage capacity of mobile devices, the entire database

is stored in the server. The mobile devices downloading requires data in its local storage, so that locally existing desired data will add great benefits to applications running on a mobile device. This is especially important in the case of disconnection events, either by undesired interference or in an effort to save battery energy. Meanwhile, if some local data has been updated by mobile applications, new updates have to be transmitted back to the server in order to maintain consistency [26]. Nowadays, the primary type of storage used in mobile devices is flash memory, which is non-volatile and has several benefits compared to a conventional disk. The rapid increase in capacity at affordable prices has made flash memory widely used in mobile devices and even in modern notebooks [27].

However, mobile environments involve substantial constraints in comparison with non-mobile environments. Energy consumption is one important issue in modern powerful portable devices; as a general rule, the more advances are made in mobile hardware and the applications that need to be executed on it, the more energy consumption is required. Communication disconnection is another serious challenge in mobile computing, where for example, a wireless signal can suffer interference, for instance from electronic noise or tall buildings. In addition, the restricted bandwidth of wireless networks and limited resources in portable computing devices, add further constraints to mobile environments [28][29].

2.3 Real-time Database

Real-time database systems (RTDBSs) have become very important over the past two decades due to their significance use in a wide range of operations. Increases in computer speed and capacity have led them to be integrated into our society and to employ many different applications, for example in stock markets, banking, reservation systems, multi-media, telephone switching systems and military command and control management. In many of these examples, real-time databases manage time-constrained data and time-constrained transactions. For example, in stock market programs, current prices have to always be current, and must be no more than a few seconds old to be considered valid. In addition, transactions operated using these data have time-constraints in terms of reading and analysing information in the database. Therefore, the goal in real-time database use not only depends on logical computation carried out as in conventional databases, but also requiring the timing constraints of data and transactions [30][31][32][33].

By contrast, in conventional database, timeline constraints are not taken into account, and correctness depends on logical computation only. The main performance criteria in conventional databases are to achieve reasonable throughput or to minimize average response time. Meanwhile the scheduling of transactions is achieved by either fairness or resource consumption criteria, such as giving priority to transactions which have made the most progress toward their end [32][34].

2.3.1 Timeline Requirements

In real-time databases all traditional database requirements are maintained, which preserve the logical consistency of data and transactions, for example in granting the serializability of transactions and operations on data items. They can also require the temporal consistency of transactions and data. These requirements are summarised below [30].

- *Logical Consistency of Transactions*

  This controls the values produced by transactions. For example, serializability, as discussed in section 2.1.2, is correctness criteria for the logical consistency of transactions and has been widely used in traditional database systems.

- *Logical Consistency of Data*

  A range of data constraints require to be maintained in most traditional database systems, in order to ensure the logical consistency of data. For instance, database items should not have negative values.

- *Temporal Consistency of Transactions*

  The temporal consistency of transactions is controlled by timing constraints such as start time, period of execution and deadline. These timing constraints can be divided into three categories: hard, firm and soft. Failure to satisfy timing constraints is considered to be a violation of consistency and an appropriate recovery procedure has to be performed by the database management system.

- *Temporal Consistency of Data*

  Data temporal consistency concerns the age of data, and whether it is still considered to be valid, reflecting the current state of the data, or out of date it

might where have changed. In the previous example, prices in stock market programs have to be always current, for example, no more than a few seconds old, to be considered valid.

2.3.2 Handling Late Transactions

As previously mentioned, a primary performance measure in real-time databases is timeline level and not throughput or response time as in conventional databases. Therefore, transaction management becomes a scheduling issue, in which priority is considered and attention given to those transactions struggling to meet their deadlines, in order to minimize the number of late transactions. Earliest deadline scheduling policy is used to give priority to transactions that are closest to expiration [35], which leads to noticeable improvements in real-time environments. In addition, several deadline-cognizant methods have been introduced in the literature in order to achieve optimal performance, such as always sacrifice, OPT-wait and no sacrifice policies [36][13][37][38][39][6].

2.4 General Enhancements

Each access to a conventional hard disk is expensive in terms of both power usage and time, as the disk is spun up to speed and the relevant data sector located. Solid-state drives are also significantly more costly to access compared to local memory. Furthermore, with a soled-state drive, it is not possible to overwritten data straightaway, out of place update mechanism is applied. So, to update an data item, the whole block where such an item is located must be erased (‘Bulk erase’) and then the whole block rewritten with the new updated item [40]. Such access costs further increase when communication costs are incurred in network environments, such as when clients access remote data at the server. Therefore, reducing the number of times that a disk is accessed will improve performance and reduce the energy consumed. Caching and rerun policy are general enhancement methods used for such purposes[15][41].

2.4.1 Caching

Caching is an important technique that is used in many areas of computer science, such as in the CPU cache, disk cache or web cache. Data is simply stored in a local memory for future use. Cached data is usually a replica of the original data located elsewhere (EX. server), or it might be values that have been computed earlier. If new data

requested exists in the cache, then it will be read from the cache more quickly and cheaply. If new data is requested that does not exist in the cache, then it will need to be obtained from the original source or recomputed, which again incurs extra time and cost [41][42].

1 Cache Replacement Strategy

Due to cache size limitations on the client's side, space could be exhausted quickly. A replacement strategy is used to clear some space in the cache for new requested data. The decision about which data should be removed from the cache could be influenced by several factors including the following [43][44]:

- Recency: period of time since the last reference to the data item.
- Frequency: number of times that data item has been referenced.
- Cost of fetching: cost of obtaining data item from the original place.
- Size: size of data item.
- Expiration time: period of time before the data becomes out of date.
- Modification time: period of time since the last modification to the data item.

(Least recently used) LRU and (least frequently used) LFU are two well-known replacement strategies [43][42][45]. LRU looks backward to the history of data stored on the cache and removes an item that has not been used for the longest period of time. LFU again looks backward to the history of data stored on the cache and removes the item that has been least frequently referenced.

2 Client-Server Caching

Caching data in a client-server architecture is very important for improving performance. It can be implemented by two methods: intra-transaction caching and inter-transaction caching: In the former, data is stored within a single transaction boundary and discarded after the transaction commits. This is a simple method and cache management is performed by the clients themselves. In inter-transaction caching, data is stored across transaction boundaries, which requires more sophisticated techniques to be used in order to maintain the consistency of the cached data [46][47][45]. Even though caching incurs extra overheads for maintaining the consistency of cached data, it involves many benefits to the system, including the following [48][49][43]:

- Reduced reliance on the server, a decrease in network traffic and message processing time and cost overheads, which consequently leads to a reduction in system latency and response time.
- Allows better utilization of resources presented to clients.
- The server can manage a larger number of clients in the system.

3 Transactional Cache Consistency

Caching introduces multiple copies of the same data items, and is similar to replication. Therefore, consistency between these redundant copies has to be maintained. The following are some techniques that have been introduced to ensure the consistency of cached data.

Avoidance versus Detection

In avoidance-based approaches, stale or out-of-date data is not allowed to exist in a client's cache. Therefore, transactions never have a chance to read stale data. Avoidance-based approaches uses a read one/write all (ROWA) technique to make sure that all replicas of updated data items are the same when updated transactions commit. Based on the ROWA technique, transactions are read from the local copy in the client's cache and all copies updated in the system. In contrast, in detection-based approaches, stale data is allowed to exist temporarily in the client's cache. Therefore, checking the validity of cached data is mandatory and is performed by each transaction before it is allowed to commit In comparison with the ROWA technique, the detection-based approach is simple, because a consistency action only includes the server and a single client [45][50].

Invalidation versus Propagation

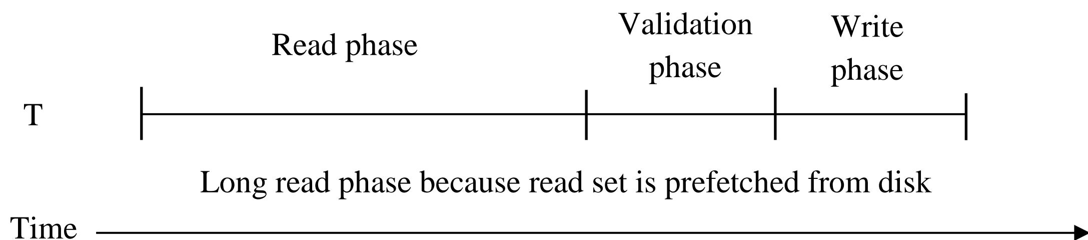
Two kinds of techniques to maintain cache consistency can be used in a client's cache when a notification update arrives from the server: invalidation and propagation.

Invalidation is a technique of removing the stale copy from the client cache when the original copy at the server is updated. Therefore, invalidated data will be inaccessible for any subsequent transaction. Subsequent transactions interested in accessing invalidated data have to obtain an up-to-date copy from the server. Information needed for cache invalidation is broadcast from the server via invalidation messages, and this requires that the commitment of updated transactions is delayed until all client caches have been invalidated. This is considered to be a scalability weakness.

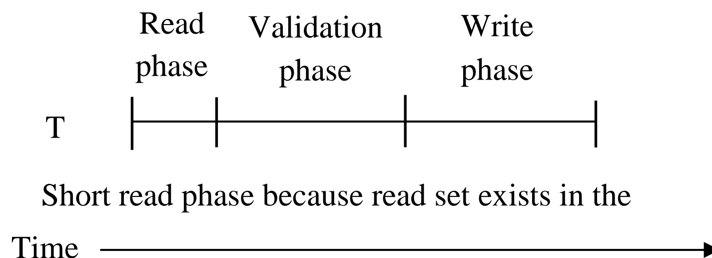
Propagation is a technique of sending new updates to clients when updated transactions commit at the server; this consequently replaces stale copies with new updated copies. Therefore, cached client data will be accessible for any subsequent transaction.[50][45]

2.4.2 Virtual Execution

Virtual execution is a concurrency control technique that allows conflicting concurrently running transactions to continue execution virtually, in order to prefetch all required data in its private workspace in memory (first run), which is illustrated by the figure 2.5 a. When a transaction has finished the virtual execution, it aborts and reruns using the pre-fetched data stored in the memory from when it first read (rerun), which is illustrated by the figure 2.5 b. Then, if transactions enter a state of conflict within the rerun, it immediately aborts and reruns again [51] [52][15].



a) First run in virtual execution environments



b) Rerun in virtual execution environments

Figure 2.5 transactional phases in virtual execution environments

Analysis has shown that virtual execution techniques that utilize OCC perform better if transactions are allowed to reach the end of their read phase before being aborted [53][54]. This is intuitively logical, since as transactions that have been aborted early would not have retrieved all the required data to be ready locally for the rerun phase. Cached values from the write sets of committed transactions together with read sets from

currently executing transactions populate a local buffer to the transaction management system. This can improve performance if overheads associated with persistent store access are significant. Therefore, distributed data stores [55] and real-time databases [15] have made use of such techniques. There is typically no disk I/O overheads required for the transaction during rerun, as the data has already been pre-fetched. Therefore, considerable battery power savings can be gained by deploying such a technique on mobile devices [16][17]. The following two important issues need to be considered in a virtual execution environment.

1. An access invariant property has to be guaranteed when using this approach, which means that any two executions of the same transaction must always access the same data items, even if these executions are separated by other conflicted transactions [15][51].

2. An issue of consistency arises for a transaction that operates using pre-fetched data. It may be that some of the pre-fetched data has since been modified. This will result in the rerun transaction operating with inconsistent data. A further concurrency control technique is clearly required to overcome this problem. [15][51]

It is important also to mention that, in conventional optimistic concurrency control methods, conflict resolution can be classified into two approaches: kill-based or die-based.

1. Kill-based approaches resolve conflicts between the validating transaction and conflicted concurrently running transactions by aborting conflicted concurrently running transactions and preceding the validating transactions to commit.

2. Die-based approaches, in contrast, resolves conflict by aborting the validating transaction and continuing the execution of conflicted concurrently running transactions.

When a virtual execution environment is deployed with a kill-based approach, it becomes logically equivalent to a die-based approach [51]. That is because the validating transaction continues execution towards the write phase as seen in the kill based approach, yet at the same time concurrently conflicted running transactions continue executions as described in the die-based approach.

2.5 Related Work

This section presents a survey of previous studies in the areas of optimistic concurrency control and broadcast datacycle approaches.

2.5.1 Optimistic Concurrency Control (OCC) Techniques

OCC techniques are classified into three categories: forward and backward oriented validation, serialization graph and timestamp. These techniques are described below:

1. Forward and Backward Oriented Validation

Härder [56] proposed two schemes for the validation phase: backward oriented optimistic concurrency control (BOCC) and forward oriented optimistic concurrency control (FOCC):

- *Backward Oriented Optimistic Concurrency Control. This* operates by comparing the read set of a validating transaction with the write sets of all currently executed transactions that have finished the read phase before the validating transaction. If conflict is identified then the only way to resolve it is restarting the validating transaction in its entirety. T_v is the validating transaction and Ti is the currently running transactions that have finished the read phase before T_v.

It is important to note that the WSs of the overlapping commit transaction have to be saved until their last current transaction has completed.

Algorithm 1 Backward Oriented Optimistic Concurrency Control

1: valid = TRUE;
2: for each T_i (i=1,2,…,n)
3:     If $RS(T_v) \cap WS(T_i) \neq \{\}$ then
4:         Valid = FALSE;
5:     Endif;
6: Endfor;
7: If valid then commit;
8:     Else abort;
9: Endif;

RS – read set and WS – write set.

- *F o r w a r d O r*

iented Optimistic Concurrency Control. This based on comparing the write set of a validating transaction with the read sets of all currently runing transactions that have yet to finish the read phase. When a conflict is found, FOCC provides a degree of flexibility in that a number of resolution policies are possible. These may:

1. Delay the validating transaction and restart the validation phase at a later time.
2. Abort all conflicting transactions and allow the validating transaction to commit.
3. Abort the validating transaction.

It is this flexibility in resolution policies which has made FOCC the focus of further research [57][58][59][60][61][62][63].

T_v is the validating transaction and Ti is the active transactions.

Algorithm 2 Forward Oriented Optimistic Concurrency Control

```
1:    valid = TRUE;
2:    for each Ti (i=1,2,…,n)
3:        If WS(Tv) ∩ RS(Ti) ≠ { } then
4:            Valid = FALSE;
5:        Endif;
6:    Endfor;
5:    If valid then commit;
6:    Else  resolve the conflict;
```

RS – read set and WS – write set.

However, aborting the validating transaction is expensive because such transactions have used resources and completed execution. The never abort validating (NAV) transaction strategy ensures that these resources will not be wasted by guaranteeing that the validating transaction commits [33]. However, a major drawback of FOCC is that concurrent transactions have to be blocked in their read phase while the validating transaction is executing in the validation and write phases. This blocking significantly degrades the performance of the system.

2. Timestamp Technique

Timestamp (TS) is a unique number associated with each transaction at the beginning of its execution. TSs do not necessarily reflect the actual times of the generation of transactions but are important in that they reflect their order. Therefore, they must be issued in ascending order. Two methods could be used to generate TSs. The first method is taking the system clock as the TS. This is reasonable, but with this method the scheduling should not be quicker than the system clock in ordere to prevent the possibility of generating the same TS to two different transactions. The second method is to use the counter as the TS generator. Therefore, each new transaction receives the previous TS but increased by 1. The rule here is: for each transaction T, if T_i starts after T_j then the TS of T_i must be higher than the TS of T_j.

In addition to the data, three pieces of information need to be associated with each data item: two TSs (RT and WT) and one additional bit (C) [34]:

1. RT (a), read time of data item a; this timestamp refers to the highest timestamp of the transaction that has read a.
2. WT (a), write time of data item a; this timestamp refers to the highest timestamp of the transaction that has written a.
3. C (a), commit bit of data item a, which is set to true if the most recent transactions that wrote object a have already committed. This information is used to eliminate the reading of dirty data [34].

The timestamp approach has been widely studied in the literature [16-18][19-22]. It shows a high degree of concurrency, guarantees a deadlock-free property, and provides a relatively smaller number of unnecessary rollback overheads. In contrast, the major disadvantage of the timestamp approach is the large overheads associated with timestamp management, especially when database is geographically distant from the client, communication between clients and the server is needed for every read operation to keep track of both read and write timestamp; which further increase timestamp overhead [64][21][18].

3. Serialization Graph Testing (SGT) Technique

The SGT scheduler maintains a serialization graph of the history representing the execution it controls. During the execution, the scheduler maintains the SG by adding edges between concurrent transaction nodes corresponding to all read and write

operations requested, without consideration of SG being acyclic. When a transaction T finishes execution and the scheduler receives a request to commit T, then it checks if T lies on a cyclic SG. If so, then this indicates that there has been a conflict operation inserted into the schedule, and some resolution policy needs to be applied to resolve this conflict. If not, then the schedule is still serializable and T can commit safely. The SGT scheduler theoretically maintains the serializability of the schedule. However, in practice, it is very expensive to maintain SG overheads; and yet, checking for cycles adds extra cost to this technique [19][65][66].

2.5.2 Aspects of Optimistic Concurrency Control

Optimistic approaches have the potential to provide greater performance than pessimistic approaches, particularly, in real-time databases even given high data contention over a wide range of resource availability levels. This is because, in the optimistic approach, conflict resolution is delayed until the times at which transactions commit, which helps in making better conflict decisions. Therefore, optimistic algorithms ensure that no transactions which are likely to miss their deadlines prevent other transactions execution in the system. Therefore, this section devoted to explored fifteen aspects of OCC studied in the literature. These include correctness criteria, conflict detection and resolution, unnecessary rollback, transaction length and starvation problems, back-off policy, partial rollback, read-only transaction considerations, transaction arrival rate, database granularity, static/dynamic data access schemes, silent/broadcast commit, speculative CC, deadline-cognizance and virtual execution.

1. ***Correctness Criteria***

Serializability is the fundamental approach of correctness criteria to concurrency control. The serializability of a schedule means that its outcome, the transformation of a database state, is equivalent to at least one serial schedule [19]. Although serializability has been widely adopted in concurrency control [19][5][20][6], it is considered to be strong correctness criteria in certain circumstances, for instance in some commercial applications. Alternative weaker correctness criteria have been proposed in order to increase system performance [67][68][69][70][71][72][73][74]. For example, if a list of products is retrieved according to price, but which is just about to be updated with a new product, the new product may not appear in the list. However, it will appear in the list later.

2. Conflict Detection

The conflict detection process in concurrency control is classified in two classes: pessimistic and optimistic approaches.

- The pessimistic approaches detection may be performed before accessing conflicted data item. Here transactions require data items to be locked to before read operations. Therefore, if one of these data items has already been locked by another transaction, then these two transactions are in conflict and one of them is aborted in order to resolve this conflict. Aborting conflicted transactions in the early stages, based on this strategy, obviously reduces the amount of wasted work and saves resources. On the other hand, aborting transactions early because of conflicts with other concurrently running transactions which may abort later is a big disadvantage. Some conflicts are reconcilable and may be resolved without aborting conflicted transactions[75][6].

- The optimistic approaches detection is performed after accessing a conflicted data item. The transaction reads all required data items in the read phase without restriction. Afterwards, in the validation phase, conflicts between transactions which have already accessed shared data are detected and resolved. The advantages and disadvantages of this strategy are opposite to pessimistic approaches. [56][75] [76][77][56] [19].

3. Conflict Resolution

Resolving a conflict by aborting one of two conflicted transactions can be an expensive solution if the rate of conflicts is high. In some cases, conflicts can be resolved efficiently without aborting either of the conflicting transactions. For example, consider two concurrent transactions - T1 and T2:

T1: R1 (a), W1 (a), R1 (b), W1 (b), C1

T2: R2 (b), R2(c), C2

Suppose they execute as in the following history:

H1: R1 (a), W1 (a), R2 (b), R1 (b), W1 (b), C1, R2(c), C2.

Based on the forward validation approach, T2 is conflicted with T1 in a read-write conflict (T2 is read-only transactions). However, T2 should not be aborted if there are no more conflicts between T1 and T2. Consistency is still maintained with serialization

order T2⟶T1. This kind of conflict is a reconcilable conflict. However, now consider the following execution history:

T2: R2 (b), W2 (b), R(c), C2

And the H2: R1 (a), W1 (a), R2 (b), W2 (b), R1 (b), W1 (b), C1, R(c), C2.

In this case, T2 conflicts with T1 with both read-write and write-write conflicts, which is irreconcilable, and aborting T2 is necessary to preserve consistency. Therefore, conflicts between concurrent transactions can be divided into two types: reconcilable and irreconcilable conflicts [6].

- Reconcilable conflicts are conflicts between two concurrent transactions resulting from the occurrence of read-write conflicts only. Therefore, conflict resolution can be performed without aborting conflicted transactions.
- Irreconcilable conflicts are conflicts between two concurrent transactions resulting from the occurrence of both read-write and write-write conflicts. Therefore, conflict resolution requires the restarting one of the transactions involved in this conflict. In this case, transaction priority, length, deadline and the amount of transaction execution already completed have to be considered when resolving irreconcilable conflicts [78][76][6][79].

4. Unnecessary Rollback

Rollback overhead is the major problem in the OCC approach. This problem can be worsened if the scheduler aborts transactions which should not be aborted which is termed unnecessary rollback. This happens when conflicts between concurrent transaction and the validating transaction occur after the end of the write phase of the validating transaction [57]. For example, consider the scenario of the validation based on Kung and Robinson [8], as illustrated in Figure 2.6.

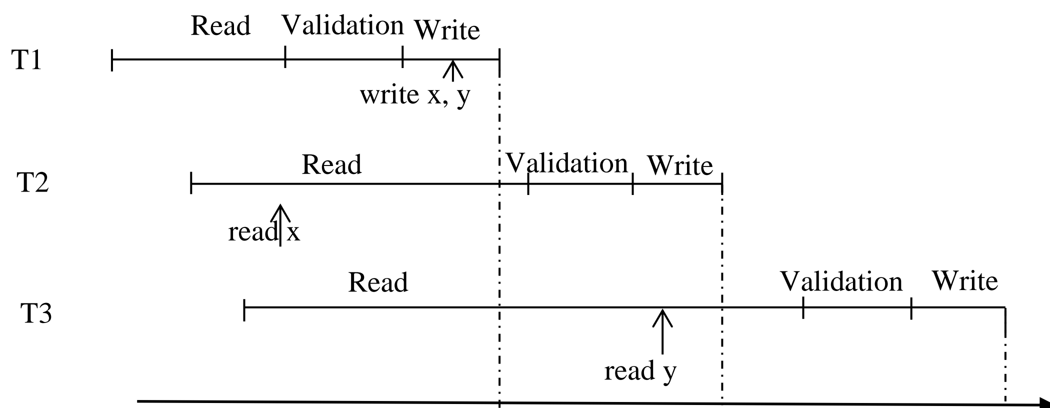


Figure 2.5 Illustration of serious and non-serious conflicts

Figure 2.6 shows a scenario of three concurrently running transactions: T1, T2 and T3. T1 updates data items x, y. T2 and T3 read data items x and y respectively. T2 reads data item x before the new update has been written by transaction T1. Therefore, if T1 has committed, T2 must be aborted to resolve this conflict. This is a so called 'serious conflict'. T3 reads data item y after the new update has been written by transaction T1. Therefore, if T1 committed, T3 is still valid in the serialization order T1 ⟶ T3, and T3 should not be aborted. This is a non-serious conflict. As this example illustrates, conflicts between transactions which are running can be classified into two categories of serious or non-serious conflicts.

- *A serious conflict* is one which occurs between the concurrent transaction and the validating transaction, before the end of the write phase of the validating transaction. This conflict may transform the database into a state of inconsistency by producing unexpected results, and conflict resolution has to take place to preserve database consistency [57].

- *A non-serious conflict* occurs between a concurrent transaction and the validating transaction after the end of the write phase of the validating transaction. This conflict does not affect database consistency, and so there is no need to perform conflict resolution [57][78][6][64][80].

5. Transaction Length and Starvation Problem

Transaction length indicates the number of data items which need to be accessed during the transaction's execution time. As the length of the transaction increases, the probability of conflict increases as well due to the following reasons [77]:

- A long transaction takes a longer time to execute, which increases the chance of becoming conflicted with other concurrently running transactions.
- A long transaction accesses a larger number of elements, which increases the probability of conflicts of these elements with other concurrent transactions.

High contention and hotspot data items, which are those accessed more frequently, also increase the chances of conflict. Therefore, long transactions are likely to be repeatedly restarted, which is called starvation. Extra consideration is needed when designing OCC protocols for long transactions in order to have as similar a chance of committing as regular ones, [77][57]. A simple solution to the starvation problem is to

give priority to starved transactions, or the whole database may be blocked to give a chance for a starved transaction to be able to commit [8]. Also, starvation could be managed by limiting the number of concurrently running transaction [81][82][83]. Other solutions to starvation problems have also been suggested [84][57][85].

6. Back-off Policy

Restarting conflicted transactions directly may increase the probability of conflict occurring again, especially if concurrently running transactions are accessing the same hotspot data items. A period of waiting time (back-off) before restarting an aborted transaction reduces the probability of the same conflict recurring. However, especially in real time systems, a long delay may lead to failure to meet deadlines. Back-off with reasonable time allowed for an aborted transaction has shown improvement in some concurrency control protocols [86][63][59]. Back-off policy is demonstrated in the following example.

Suppose that four transactions - T1, T2, T3 and T4 - are concurrently running and all conflict with each other, as illustrated in Figure 2.7.

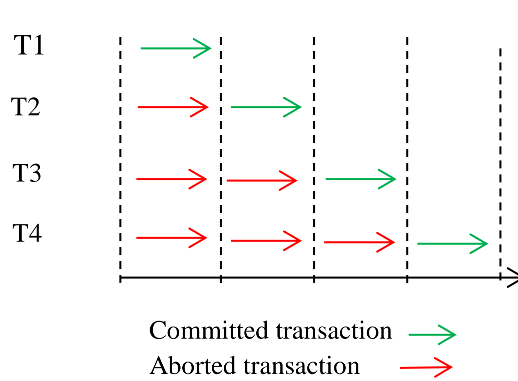


Figure 2.7 (a)

Frequently rolled back scenario

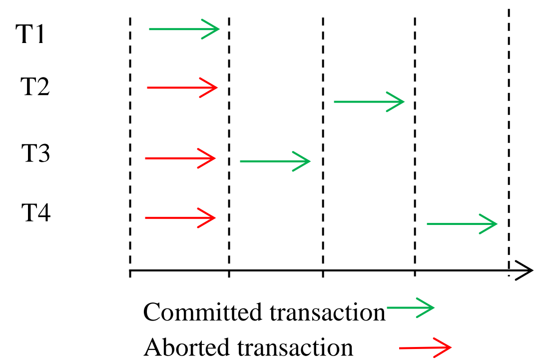


Figure 2.7 (b)

Transactions backed-off scenario

In Figure 2.7 (a), transactions T3 and T4 are frequently rolled back before they get the chance to commit. This is expensive wasted work and wasted use of resources. However, using a back-off policy as illustrated in Figure 2.7 (b), aborted transactions are backed off for different appropriate periods of time. This leads to a reduction in the number of transaction restarts.

7. Partial Rollback

Rolling back conflicted transactions increase OCC overheads because they have already executed, and the resource usage of the aborted transactions are lost; this is a wasted execution. Wasted execution arises if a rolled-back transaction has done most of its work and was near to its completion. Partial rollback is a technique which has been introduced to reduce the wasted execution caused by restarting a conflicted transaction, and involves rolling back only a conflicted part of the conflicted transaction instead of rolling back the entire transaction. This consequently reduces the amount of execution which needs to be re-performed when a conflict occurs [87]. This can be achieved by using checkpoints at the level of the transaction; at these points a transaction can roll back and re-establish its execution. Therefore, if a conflict has occurred, a conflicted transaction will be partially rolled back from the most recent checkpoint [87][88].

8. Read-only Transaction Considerations

A read-only transaction, query, is a transaction that does not update the database; in other words, it is a transaction whose write set is empty and it has no write phase. For example, in a transaction which checks a given balance in a bank, there are no withdrawal or deposit operations. Such a transaction is a typical read-only transaction. Read-only transactions are very important because they constitute the major proportion of typical transactional traffic [57][68][89]. Therefore, giving some flexibility for read-only transactions can have a great impact on system performance, especially for query applications. A simple possible technique is to delay the write phase of an update transaction if it conflicts with a read-only transaction. However, this solution might produce a long delay in updating transactions. Multiversion is another concept used to support read-only transactions, when the system keeps a number of versions of the same data items. Therefore, a read-only transaction can always maintain a consistent view by reading suitable versions of data items [57]. Further schemes proposed in the literature have given special treatment to query transactions [89][90][91][92].

9. Transaction Arrival Rate

The probability of conflicts between concurrently running transactions will increase as the number of them accessing shared data items increases. Therefore, controlling transaction arrival rate by minimizing the number of running transactions that retrieve the same data will obviously lead to the reduction of conflicts and thus roll-back

overheads. This is simply achieved by blocking some of those transactions at the beginning of their execution. On the other hand, blocking transactions in real-time databases is undesirable. Therefore a good balance has to be struck in order to gain optimum performance [93][94].

10. Database Granularity

Transactions in OCC protocols back up data items in their private workspace in the memory. So, the size of the data will be considered as granular as word, page, or object is an important issue concerning the consumption of memory the main space. When a smaller data item is used as a granule, more memory space will be saved. A balance needs to be found between memory efficiency and database granularity when designing OOC protocols [95]. On the other hand, in locking-based concurrency control protocols, grouping several data items as one granule could be beneficial in some circumstances. For instance, if a transaction needs to access the whole database, then it would be better to request one single lock to lock the entire database instead of requesting locks for each data item separately, which would consume much more time and resources [96][97].

11. Static/Dynamic Data Access Schemes

In concurrency control, accessing data items can be divided into two schemes: static and dynamic data access.

- *In static data access schemes*: all required data items will be read at the beginning of the transaction's execution. This gives more flexibility in designing a validation mechanism, because all accessed data will be known in advance. On the other hand, read data will be held for a longer time, which leads to an increase in system contention. [19] [98][99][100]

- *Dynamic data access schemes*: require data items to be read as they are needed during transaction execution. As opposed to static access schemes, dynamic schemes reduce data contention because data items are held for shorter periods of time. However, dynamic schemes are more complicated from the perspective of validation, because the read sets of transactions keep changing during transaction execution [98][99][100].

12. Silent/Broadcast Commit

The transaction commit can be divided in to two schemes: silent and broadcast commit.

- *Silent commit scheme*: In this scheme, the transaction commit is not advertised to other concurrently running transactions. Therefore, the latter continue execution until the validation phase, where they become aware of conflicts. Delaying the restarting of conflicted transaction leads to an increase in wasted transaction executions [98][99][100].
- *Broadcast commit scheme*: Conversely, here the, transaction commit is advertised to all concurrently running transactions in order to abort conflicted transactions earlier and thus reduce wasted executions and available resources. The broadcast commit has the advantage that conflicted transactions do not continue execution in vain and waste system resources, which consequently leads to performance improvements [98][99][100].

13. Speculative CC

Speculative CC techniques use redundant transactions to start as early as possible on an alternative schedule if conflict is expected. This redundant transaction is called a transaction shadow. If conflict in an original transaction is resolved and the original transaction is successfully committed, then the transaction's shadow must be discarded. On the other hand, if the original transaction fails to commit, then the transaction's shadow is adopted instead of restarting the original conflicted transaction from the beginning. For example, consider that T1 and T2 are concurrently running transactions under a broadcast commit scheme, as illustrated in Figure 2.8. [101]

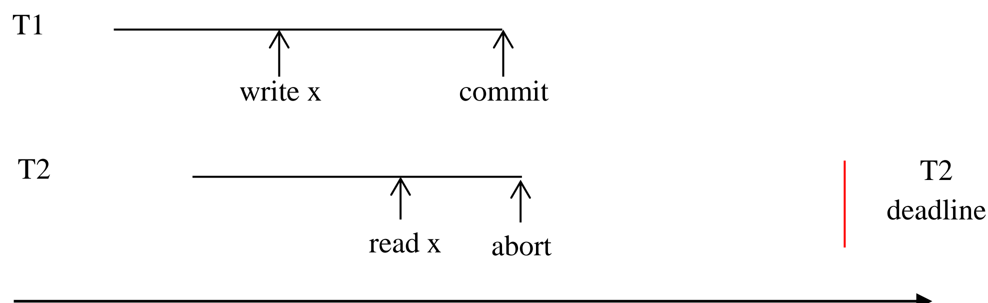


Figure 2.8 Transaction management
under an OCC broadcast commit scheme

Figure 2.8 shows a simple scenario of two concurrently running transactions. T2 is conflicted with T1 in data item X, and T2 reads an updated value of X made by T1. This is unlike the pessimistic approach, which would block T2 until such a conflict is resolved, and also unlike an optimistic approach which would ignore the expected conflict. A speculative approach would make a copy of T2 (shadow), starting the execution at a different time. Both transactions - T2 and T2's shadow - will be allowed to run concurrently at different points of execution, but only one of them is allowed to commit. Although both transactions (T2 and its shadow) may see different versions of data items in their read operations, both transactions are exact replicas of each other because both are performing the same operations. Figure 2.9 illustrates a scenario when T2 reaches the commit time before T1, where T2 successfully commits and T2's shadow aborts.

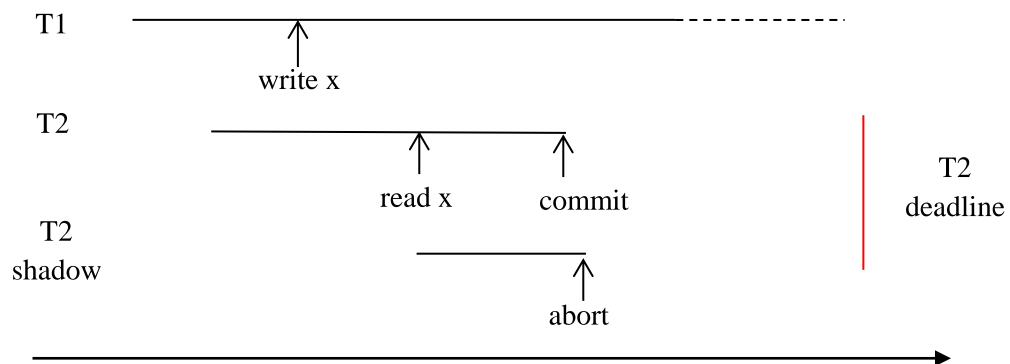


Figure 2.9 Schedule with
an undeveloped possible conflict.

Figure 2.10 shows another scenario, when T1 reaches the commit first. T2 must abort because of a conflict with T1. In this case, T2's shadow will be adopted and will continue execution instead of restarting T2 from the beginning. Speculative concurrency control offers a better opportunity for real-time transactions to become committed before their deadline expires. However, this advance is not gained at no cost; it requires extra memory and processing resources when transactions succeed in committing and other shadows are discarded. Speculative concurrency control has been intensively studied in the literature [101][102][103][104][105][106][107][108][109][110].

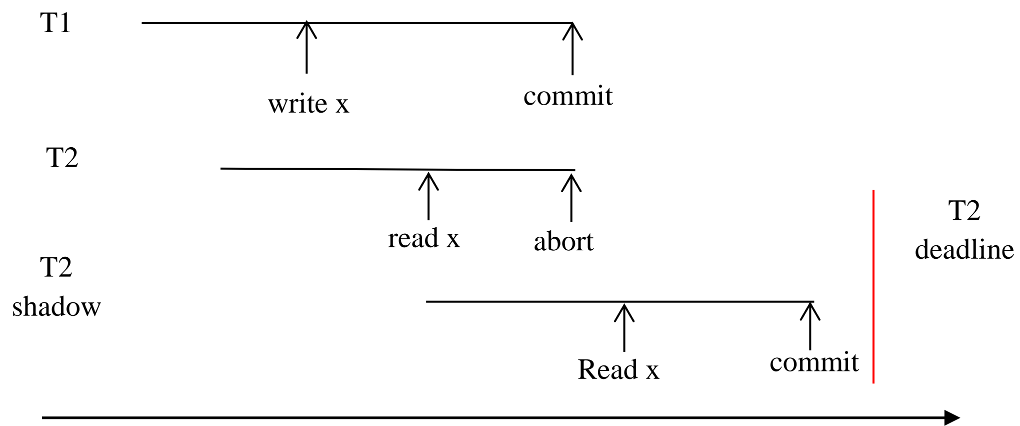


Figure 2.10 Schedule with a developed conflict.

14. Deadline-Cognizant

Timeliness is a primary performance measure in real-time database systems, in contrast to conventional database systems which use response time and the throughput as main performance measures. The main goal of real-time systems is to minimize the number of transactions that cannot meet their deadlines. Therefore, priority is a key factor that needs to be taken into account when dealing with scheduling in real-time systems. Intensive research studies have been conducted in order to determine the optimal deadline-cognizance [6][39][35][38][37][13][36]. Three important policies for deadline-cognizance are reviewed below.

- **OPT- Sacrifice:** When a transaction reaches the validation phase, it starts validation operations with all concurrently running transactions. If a conflict is detected with at least one transaction with a higher priority, then the validating transaction aborts. Otherwise, the validating transaction proceeds to commit and all conflicted transactions must restart. The goal of this strategy is to help higher priority conflicted transactions to meet their deadlines [11][111][112][113]. However, two problems arise with the OPT- sacrifice policy:

    1. There is a potential problem of wasted work, which results from aborting some conflicted transactions on behalf of the validating transaction, then aborting the validation transaction itself afterwards. In this case, the abortion of the transactions which are conflicted with

the aborted validating transaction is unnecessarily and the work that has been made by these aborted transactions is wasted work [111][11].

2. A problem of mutual restarts arises when priority reversal is allowed, based on a dynamic transaction priority assignment scheme [114]. For instance, if transaction T1 restarts on behalf of T2, because T2's priority is currently higher than that of T1, then T2 at a later time restarts on behalf of T1, because T1's priority is now higher than that of T2. This fluctuation in transaction priority assignment causes the pair of transactions to continue aborting each other, which affects the progress of both transactions and consequently degrades the whole system performance.[113][6][11]

- ***OPT-Wait*****:** This scheme is an updated version of OPT-sacrifice, with the addition of waiting mechanism. When the transaction reaches the validation phase, it starts validating with concurrently running transactions. If conflict with a higher priority transaction is detected, the validating transaction does not restart immediately as in OPT-sacrifice; instead it is put on hold, waiting for a higher priority transaction to commit. The waiting transaction consequently restarts if a conflicted higher priority transaction successfully commits. But if the latter is aborted, then the waiting transaction will be allowed to proceed. OPT-wait has several advantages over OPT-sacrifice, including the problem of wasted work as mentioned earlier, because restarts occur only at the commit time of a higher priority transaction. In addition, the problem of mutual restarts is eliminated, because fluctuations of transaction priority do not lead to transactions aborting [111][11][6]. On the other hand, OPT-wait also has some negative features which can be summarized as follows.

  1. When a waiting transaction successfully commits after a period of waiting time, it will abort all lower priority conflicting transactions at a later time; this will increase the chance of failure of these transactions in meeting their deadlines [11][111].

2. Waiting transactions may develop new conflicts, consequently leading to an increase in the number of restarts; this may become significant if the data contention rate is high [11][111].

- ***No Sacrifice*****:** in this policy, which is also known as the never abort validating (NAV) transaction strategy [85], a validating transaction guaranteed to commit when it reaches the validation phase mean that all conflicted transactions have to be aborted [6]. Although, priority is not considered in this policy, it has great benefits as summarized next

  1. A validating transaction has already used all the resources it needs and has done all the work. Therefore, aborting a validating transaction will be very expensive in terms of resource use and computational costs. The NAV strategy guarantees that the resources utilized by a validating transaction are not lost [85].

  2. NAV eliminates the wasted work which results from aborting some conflicted transactions on behalf of a validating transaction which aborts later, thus avoiding considerable performance degradation [111] [11].

  3. A no sacrifice policy prevents problems related to priority-driven scheduling, such as mutual restarts, the starving of low priority transactions, and the extra cost of priority assignment and management[6].

No sacrifice policy has been evaluated in previous work [115][113] [116] and the results reported were generally relatively good. It outperforms other deadline-cognizant polices under a variety of operating conditions [6].

2.5.3 Data Delivery Choice

Data is delivered between clients and the server by three different methods: pull-based , push-based and hybrid delivery [117][118].

1 Pull-based Delivery

In pull-based delivery, data transmission between clients and the server is based on a request/response structure. When a client requires a data item not existing in its local cache, it sends a request to the server for such an item. In response, when the server receives the data request, it retrieves the data item and transmits it to the requesting client. This approach is illustrated in Figure 2.11.

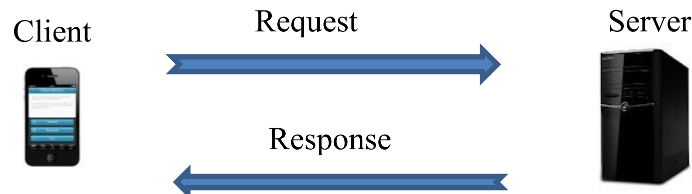


Figure 2.11 Pull-based delivery [117]

The pull-based approach works well if network disconnections are rare and the number of clients of whom the server manages to respond to their requests within the expected time intervals is relatively limited. In contrast, pull-based approaches have noticeable scalability limitations, which are summarised as follows:

- Increasing numbers of clients leads to an increased number of requests sent to the server, which can exceed the connection limit.

- Increasing numbers of requests transmitted to the server can rapidly lead to a bottleneck if the request rate exceeds the upper limit of the server's service rate.

- The client requires a backchannel to make requests to the server since asymmetric environments with uni-directional communication are not suitable. This increases the energy consumption needed for upstream communication between clients and the server, which significantly drains battery energy in battery-powered devices because sending data consumes more energy than receiving it.

These limitations mean that push-based delivery is not suitable in mobile computing environments where network disconnections frequently occur either due to interference

or in order to save battery energy. Moreover, the number of mobile clients who the server is required to serve may be relatively enormous.

2 Push-based Delivery

Here, server is repeatedly cycling through the entire database and broadcasting it to all clients. A client needing to instigate a read transaction simply waits for the relevant piece of data to come around in the broadcast cycle, and there is thus no need for the client to transmit a read request to the server. Equally, the server does not need to respond to read requests from clients, as it never receives any, as illustrated in Figure 2.12.

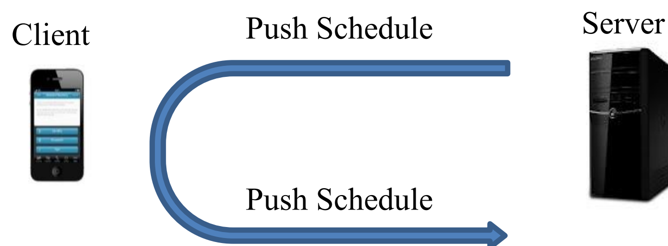


Figure 2.12 Push-based delivery [117]

This approach is particularly applicable when a large number of clients must read a relatively small database. For read transactions, the push-based approach is expandable to any number of clients with no degradation of performance. Complications arise when write transactions need to be incorporated [119][120]. The following advantages that can be gained from a push-based delivery approach.

- *Scalability:* The load in the network is reduced and becomes independent of the number of clients, which gives a greater ability to scale since the server can support more clients before overloading.

- *Lower bandwidth utilization*: avoiding the upstream bandwidth from the client to the server makes this approach attractive for asymmetric environments.

- *Energy efficiency*: preventing requests sent by clients to servers has a big impact in saving battery energy in battery-powered devices.

Push-based delivery approaches have fallen out of favour in recent decades as efforts were devoted towards synchronous clouds and server farms using pull-based

delivery approaches. The rapid developments in computing and communication landscapes, and the availability of high-bandwidth links has led to a reevaluation of the ways data should be delivered between computers. This is particularly important given the innovations in information-feed applications such as traffic information systems, stock market monitors, live audio and video telecasts, battlefield applications, news delivery, video-on-demand and other entertainment delivery applications [117], which usually deal with enormous numbers of clients. In addition, the increased use of mobile applications running on portable smart devices has brought attention back to push-based delivery approaches.

3 Hybrid Approaches

These are combinations of pull and push approaches, and are also known as interleaved-push-and-pull (IIP) approaches [121]. The server regularly broadcasts hot data (that frequently used by clients), based on the push approach, and cold data needs to be requested by clients via a back-channel, based on the pull approach. The hybrid approach represent a compromise between the various advantages and disadvantages of previous data delivery approaches [121].

2.5.4 Broadcast Datacycle Approach

Broadcast datacycle for asymmetric communication environments continuously broadcast all data items in the database to all connected mobile devices, using single or multiple wireless channels. Clients listen to this broadcast stream and access the required data as it is broadcast, if it does not exist in the local memory or disk. Therefore, the number of mobile devices does not affect access time, since it is read-only. Read transactions are expandable to any number of clients with no degradation of performance. Which are outweighing write transactions in many applications in a wireless environment. For example, information-feed applications such as an online stock-trading application involves far more transactions resulting from a user checking or tracking stock prices than those instigated by a user purchasing or selling stock. The broadcast datacycle approach [120] is an established solution for this type of application, and it has recently been the subject of further work to establish it as a viable option for mobile environments [122][123][124][125].

In contrast, conventional concurrency control techniques unsuitable, for many reasons [126]. For example, using a concurrency control protocol based on locking

techniques could lead to the server being swamped with lock requests. Similarly, for timestamp-based techniques, communication between clients and the server is needed for every read operation in order to keep track of both read and write timestamps; this can be unwieldy in broadcast environments. Optimistic concurrency control is found to be more convenient in such environments.

2.5.5 Indexing on the Air

Indexing is a technique used to speed up searching operations. It is widely used in traditional database systems and storage. However, in a broadcasting Datacycle approach, air channels support only sequential access. Therefore, clients would need to scan all of the data blocks in order to find the desired data item. Without indexing, half of the broadcast datacycle will be scanned on average to reach the desired data item, as illustrated in Figure 2.13.

Figure 2.13 Broadcast datacycle with no indexing [127]

This is inefficient in terms of the energy consumption of mobile devices. However if the client knows in advance where the desired data is located in the broadcast datacycle, a CPU could switch to doze mode for most of the time, and only stay active when the desired data is expected to arrive. This would lead to considerable energy savings because a CPU in active mode consumes much more energy than in doze mode. Two important parameters need to be considered when studying indexing in a broadcast environment:

- *Access Time*: this is the average time from the point that a client requests data until the point that the client downloads that data.
- *Tuning Time*: this is the time spent by the client listening to the channel and waiting for desired data. Listening to the channel needs the client to be in active mode, consuming more energy.

Three indexing strategies are summarised below: tune_opt indexing, (1,m) indexing and distributed indexing. These can be used in broadcast data cycle environments in order to improve performance and save energy [127][128][129].

Tune_opt indexing

The index is broadcast at the beginning of every broadcast cycle, as depicted in Figure 2.14. A client tuning in to the channel at the beginning of the next broadcast needs to be able to read the index in order to locate the position of the desired data. This strategy provides the longest access time, since the client must wait until the beginning of the following broadcast even if the desired data is in front of it [127][128][129].

Figure 2.14 Broadcast datacycle in tune_opt strategy [127]

(1,m) indexing

Here the index is broadcast in m time in each broadcast datacycle, and each data block has information about the next index allocation, as seen in Figure 2.15. Therefore, when a client starts tuning into the current block, it reads the information concerning the nearest index allocation. Then it switches to doze mode until the beginning of the next index. From the next index, the client reads the relevant data location, and then goes into doze mode again until the data of interest has arrived. The (1,m) indexing strategy offers great energy savings but has to send the entire index several times which increases the broadcast datacycle length [127].

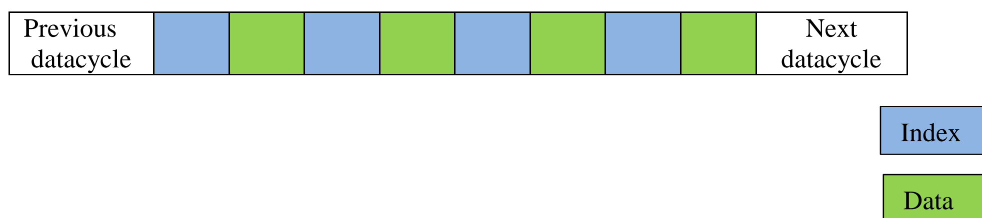


Figure 2.15 Broadcast datacycle in (1,m) strategy [127]

Distributed indexing

Distributed indexing partially replicates the index with data segments in each datacycle. Only a portion of the index attached at the front of each data segment indexes. Unlike

(1,m) indexing strategy, which attaches the whole index to the front of each data segment. This obviously reduces the datacycle length generated by the (1,m) indexing strategy. The distributed indexing strategy makes energy savings similar to the (1,m) indexing strategy, and it outperforms the (1,m) strategy in terms of access time, especially if the size of the index segment is large [127][128]. Many studies on indexing strategies have been published [130][131][132][133][134][135][136][137][138][139].

2.5.6 Broadcast Disks

A broadcast disk is a broadcast datacycle technique in which the entire database content is repeatedly and continually broadcast from the server to clients. Clients read the required data from the broadcast channel as a disk. This is different from a conventional broadcast datacycle in the sense that data is broadcast with different disks of varying speed and size. Data stored in the faster disks will be broadcast more frequently than that stored in slower disks. In a conventional flat approach, as illustrated in Figure 2.16, the expected waiting time for any data item is the same, approximately half the broadcast period [119][117].

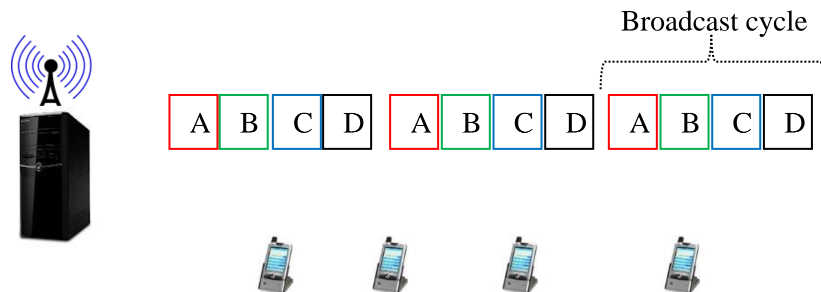


Figure 2.16 Flat broadcast approach

However, in real-life, data is not accessed uniformly, and a subset of data (hot spots) will be accessed more frequently. A server can speculate on the frequency of access to data by clients monitoring the previous history of a client's activity or by generating a summary of the client's intended future use. The server can then broadcast different items at different frequencies in order to satisfy client demand. A simple scenario with different broadcast programs for three data sets (in this case pages) is illustrated in Figure 2.17. Program (a) is a flat broadcast, in which each page is broadcast only once

in each broadcast cycle. Program (b) is a skewed broadcast in which page A is broadcast twice sequentially, with B and C broadcast once each time cycle. Program (c) is a multi-disk broadcast in which page A is broadcast twice as often as D and C, but interspersed between them. The prosperity of program (c) is equivalent, as if the page A was stored on a disk spinning in double speed as the disk in which pages B and C are existing [140][117]. Broadcast disks is attractive for information-feed applications such as traffic information systems, stock market monitors, live audio and video telecasts, battlefield applications, news delivery, video-on-demand and other entertainment delivery applications [117], which usually deal with enormous numbers of clients.

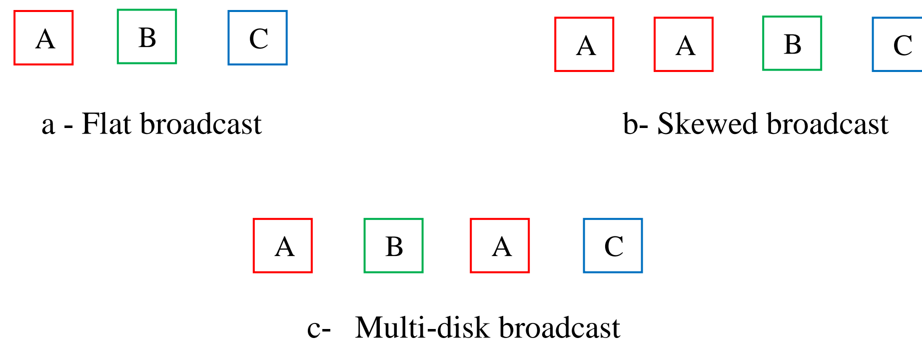


Figure 2.17 Three different broadcast programs [117]

Broadcast Disks Generation Program Example

A broadcast disks generation program from a previous study [119] is demonstrated in Figure 2.18, which apportions all of the data to three disks. Data in each disk is partitioned into chunks, and the chunks in different disks can be of different sizes. Data in the first disk will be broadcast more frequently than the data in the other disks (with double the frequency of data in the second disk and four times the frequency of data in the third disk). Each datacycle, or major cycle, contains four minor cycles, and each minor cycle contains one chunk of each disk. It is important to note that, adding more pages to faster disks results in more delay to the pages on the slower disks. Therefore, it is preferred for fast disks to have fewer pages than slower disks.

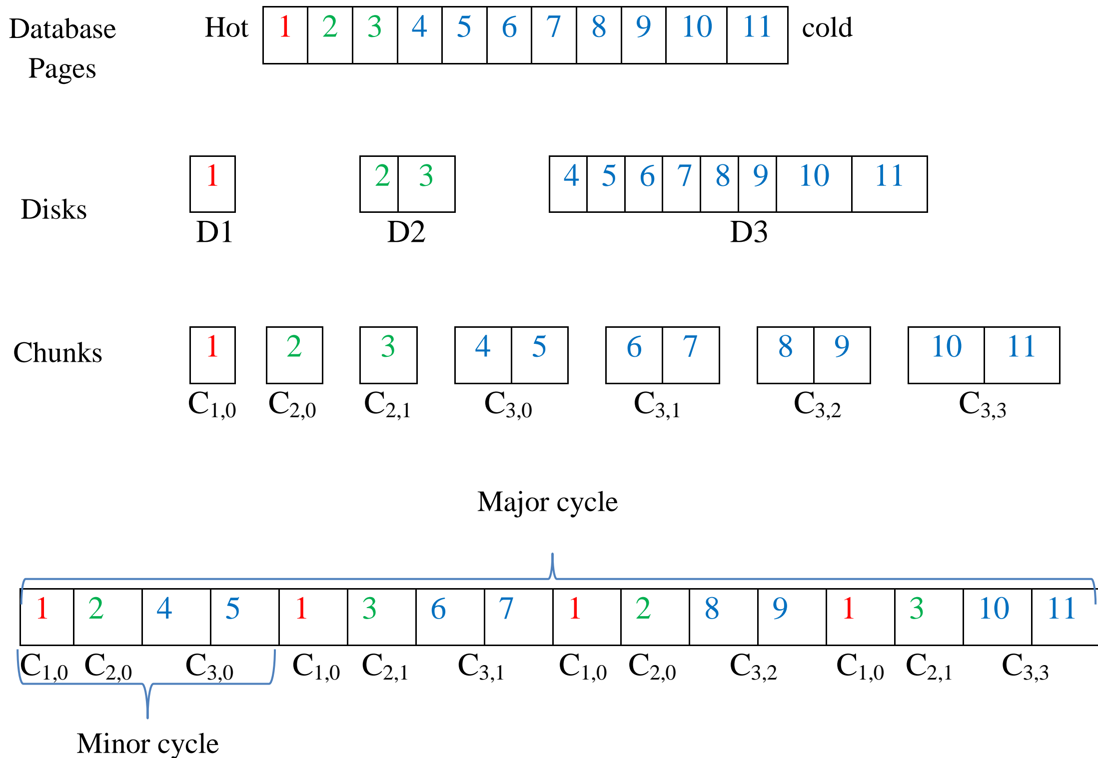


Figure 2.18 Broadcast disks generation program [119]

2.5.7 Fault-tolerance in Broadcast Disks

Data transmission in wireless environments is not always safe; signals can be affected by noise on the air, which may corrupt the data transmitted. If some desired broadcast data is corrupted, the client has to wait for the next broadcast data cycle to receive it correctly. This causes further delay to data transmission, which may be tolerable in conventional applications, but increasing latency in real-time applications may lead to deadline being missed. Some ordinary error detection techniques such as cyclic redundant code can be used to overcome data transmission failure [141]. However, this technique is a relatively simple to implement, but other advanced techniques [142] can distinguish between corruption occurring in the data itself and corruption occurring in the index buckets along the path of the search. The proposed techniques can overcome some types of corruption to indexes and continue searching in the current broadcast data cycle, instead of starting the search from scratch in the next broadcast data cycle, this maintain considerably lower access time. Various further error detection techniques have been introduced in the literature to deal with data transmission failures in broadcast environments they can be found in [143][144][145][146].

2.5.8 Forward and Backward OCC (FBOCC)

The forward and backward OCC (FBOCC) is a distributed concurrency control algorithm suitable for governing transactions in wireless broadcast environments [147][148]. It consists of three validation stages, the first of which involves partial backward validation at a client, and the second and third stages involve forward validation and final partial backward validation at the server. The three validation stages are described in detail in the following sections.

- *Partial backward validation* compares the write set of committed transactions at the server with the read set of running mobile transactions at the client at the beginning of every datacycle. These include both read-only and updated mobile transactions. Any conflicted mobile transaction will be aborted. Successfully validated read-only mobile transactions will proceed to commit locally. Successfully validated mobile updated transactions are sent to the server to be validated globally.

  The pseudo-code for partial backward validation at the client is presented below:

  Algorithm 3 Partial backward validation

```
1:      PartialBackwardValidation(Tm) {
2:          if ((CD(Ci) ∩ RS(Tm)) ≠ {}) then
3:              abort(Tm)
4:          else
5:              record the value of Ci,
6:              Tm is allowed to continue;
7:          endif
8:      }
```

  RS – read set and WS – write set.

  T_m – transaction generates and executes at the clients.

  $CD(C_i)$ – the set of data items which was updated.

- *Forward validation* is performed at the server between the write set of validating transactions at the server (this could be a server transaction or a mobile update transaction submitted by a mobile client for global validation), and the read set

of transactions at the server (which includes transactions generated and executed at the server, and update transactions generated and executed at the clients, then sent for global validation at the server). Conflicted transactions at the server will be aborted and restarted at the server, and conflicted mobile update transactions will be aborted and restarted at the client. If the validating transaction is successfully validated then it commits, and its write set will be added to the control information table, to be broadcast in the following broadcast cycle.

The pseudo-code for forward validation is presented bellow using the same notation as explained in the section on partial backward validation.

Algorithm 4 Forward validation

```
1:     validate(Tv){
2:         if (Tv is a mobile update transaction) then
3:             FinalValidation (Tv);
4:             If (return fail )then
5:                 Abort (Tv); exit;
6:             End if
7:         End if
8:         For each Tj (j= 1,2,.....,n) {
9:             if ((WS(Tv) ∩ RS(Tj)) ≠ {}) then
10:                abort (Tj);
11:            endif
12:        }
13:        Commit WS(Tv) to database;
14:        CD(Ci) = CD(Ci) U WS(Tv);
15     }
```

RS – read set and WS – write set.

$CD(C_i)$ – the set of data items which was updated.

- *Final partial backward validation* has to be performed at the server for mobile update transactions before starting forward validation. This final partial backward validation is needed in cases of existing update transactions committed at the server since the last backward validation performed at the

client. Final validation results are also broadcast with a control information table, as acknowledgement for mobile clients.

The pseudo-code for final validation using the same notation explained in the section on partial backward validation is presented below:

Algorithm 5 Final validation

```
1:     FinalValidation(Tm){
2:         For each Ti ( i= 1,2,...,n) {
3:             If (RS(Tm) ∩ WS(Ti) ≠ {}) then
4:                     Return fail;
5:         }
6:         Return success;
15     }
```

Example of interaction between server and mobile client

Figure 2.19 illustrates the schedule of the following set of transactions:

Transactions at the server: U_1: r (a) w (a) U_5: r (q) w (q) U_6: r(y) w(y)

Transaction at mobile client: Q_2: r (a) r (b) r (c) Q_3: r (p) r (q) U_4: r (x) r (y) w (y)

From Figure 3.19 the following execution scenario is concluded:

- After Q_2 has read data items a and b from the broadcast cycle, C_{i-1}, a is updated by U1 before Q2 reads c, which is caught by partial backward validation. Therefore Q2 aborts.

- Q3 successfully passes partial backward validation and commits.

- U4 passes the partial backward validation in broadcast cycle Ci and is then sent to the server for validation (final and forward validation). Because Y has been updated by U6 before U4 reaches the validation point, U4 fails to pass the final validation and aborts.

- U1, U5 and U6 successfully pass the forward validation and commit.

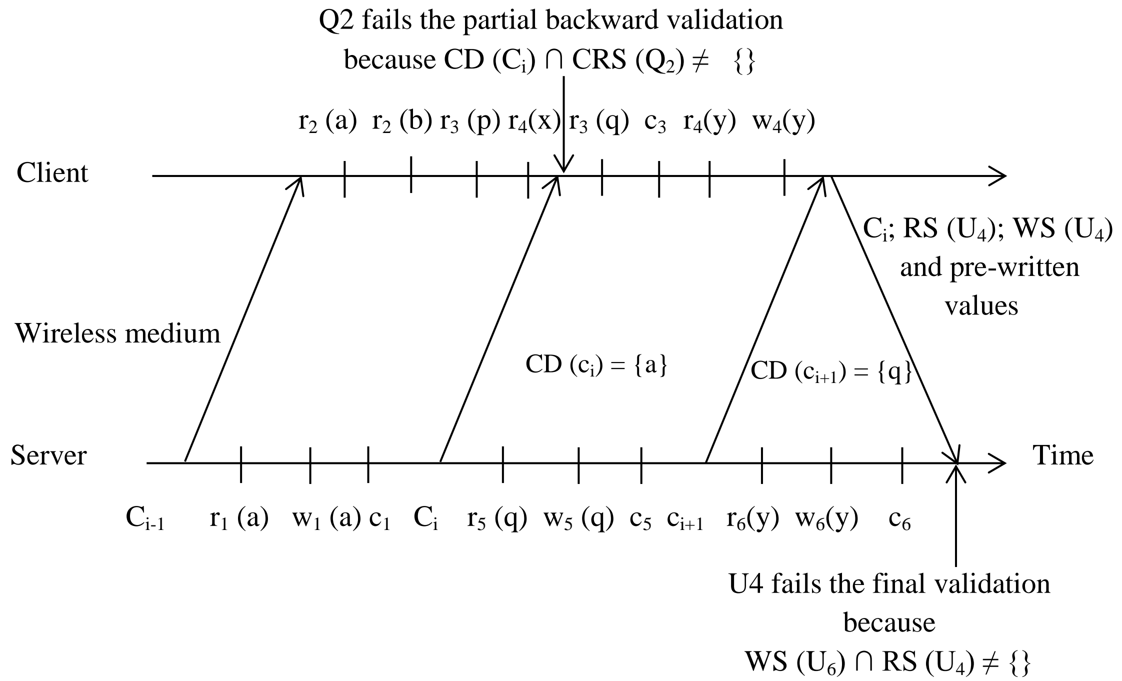


Figure 2.19 Transaction execution schedule [147]

As a result, all committed transactions based on FBOCC are serializable. FBOCC also minimizes the use of the uplink channel. This is because, firstly, validated and committed read-only transactions are locally at clients, and these constitute the majority of mobile transactions. Secondly, update transactions are validated and aborted locally at clients, which means that update transactions are more likely to pass the validation and write phases at the server. The FBOCC is suitable for concurrency control in wireless broadcast environments for many reasons [126] and is widely deployed [147][59][61][60][63].

2.5.9 Discussion

Serializability is a standard correctness criterion for many mobile applications such as mobile stock trading, and the inability to maintain it may lead to serious financial consequences [149][150]. However, maintaining serializability in mobile environments is facing new challenges due to the resource constraints of mobile computing devices and the nature of its use anytime and anywhere. Access efficiency and power limitations are the two major challenges in mobile wireless environments. Limited upstream communication capacity from a mobile device to the server makes conventional concurrency control techniques inappropriate for such environments. Using upstream

communication is also very expensive in terms of battery power consumption [147][151]. In addition, disconnection issues mean that mobile devices may struggle to cope because of undesired signal interference, or when users seek to reduce energy consumption [127]. Furthermore, the cost of validation overhead which, required to be relatively low in order to appropriate the mobile resources constraints. These challenges lead conventional concurrency control approaches to be less applicable to mobile environments [59][63][126]. The following discussion addressing the weakness of the conventional OCC techniques regarding these challenges.

Locking-based concurrency control request locks for each data item read in a transaction, including read-only transactions in order to detect data conflicts. In mobile environments such techniques would require extensive use of client to server communication and would overload the server with lock/unlock requests [126][59][63][152].

The timestamp based OCC has a relatively expensive validation cost at triple at forward and backward oriented validation [64]. Timestamp management also involves other large overheads, since each data access requires its timestamp to be updated [64][24]. It is therefore infeasible in mobile environments since it requires client to server communication, which leads to high levels of inefficiency in terms of resource utilization and energy use [126][150][59][126].

OCC based on serialization graph testing is very expensive in terms of validation costs. In practice it is very expensive to maintain the serialization graphs of concurrently running transactions, and further overhead needed for cycle checking which adds even more cost and energy drain [19][152][153].

Forward validation schemes [56] are a good concurrency control approach for mobile environments for many reasons [126], and are widely deployed in wireless broadcast environments [147][59][60][63]. It involves relatively low-cost validation, at one-third of the timestamp validation cost [7]. In addition, it has the ability to be combined with virtual execution environments to reduce the rollback overhead, which is a battery-friendly advantage [18][17]. It is therefore argued here that the forward

validation scheme is a suitable OCC approach for governing transactions operating in mobile environments.

Conventional OCC protocols were originally designed to work in conventional database environments. Mobile environments, however, have different features, which mean that conventional OCC protocols are not suitable for mobile environments. Therefore, there is still gap in concurrency control for mobile environments need to be covered in order to satisfy its requirement.

The contribution made in this thesis involves a novel departure from existing techniques by redesigning forward validation schemes in order to make them more appropriate for use in mobile environments. In the proposed new approach, the order of the traditional transactional phases sequence read-validation-write [8] is changed. The write phase now follows the read phase with the validation phase occurring after the write phase [16]. The combination of the new order of transactional phases with virtual execution can provide a solution appropriate to the constraints of mobile devices and mobile broadcast environments. Chapters 3 and 4 demonstrate that the new approach is capable of improving overall system performance and the likelihood that transactions will complete within their specified deadlines [17][18].

2.6 Summary

Background information in previous research relevant to this thesis is reviewed in this chapter. It introduces databases and showed how their consistency may interfere concurrent execution. It is explained how database consistency can be maintained by enforcing serializable schedules. In addition, various database types are introduced, including centralized, distributed, mobile and real-time databases. Furthermore, caching and rerun policies used to enhance system performance are explained. Following this, the main existing OCC techniques are reviewed, and significant aspects of OCC are investigated to identify the strengths and weakness of existing OCC protocols. Further topics regarding mobile environments are covered, including data delivery methods, broadcast datacycles and forward and backward OCC. The following two chapters introduce the contributions made by this thesis.

Chapter 3

The Read-Write-Validate Approach

3.1 Introduction

Millions of smartphones and tablet devices are being used for increasingly complex tasks. As mobile applications become achievable and practical for use as stand-alone applications or to access remote applications. multiple applications run in parallel on mobile devices, raising issues of sharing resources such as processors, memory access, solid state disk access and network connections [154]. In addition, many mobile applications require asymmetrical channels between clients and the server, whereby the frequency of read transactions requested by the client is significantly higher than the number of write transactions. Taking the example of a stock trading application; there are far more transactions involving a read-only checking of stock prices, compared to the number of transactions involving sales or other events requiring update transactions (that is, users typically check share prices far more than they buy shares). A common implementation of this type of application involves the use of a broadcast disk protocol [119], whereby the database is repeatedly broadcast to the clients in its entirety. This approach means that there is no requirement for the client to send a read request to the server; the client simply waits for the requested piece of data to appear in the cycled transmission, and the server does not have to respond to individual client requests to send data. Clearly, this greatly reduces the amount of traffic on the network, and the number of requests which the server must process. This type of approach is particularly useful when a relatively small database must be read by numerous clients.

Conventional OCC is a well-understood solution in this type of situation [8]. However, these protocols place a strain on the mobile device's battery due to the cost of validation and of duplicating information retrieval associated with aborted transactions. In addition, they tend to involve the heavy use of the network in both directions to request and validate read transactions, which renders the approach less applicable to mobile networks [126] due to limited uplink bandwidth and battery life. Forward

validation schemes [56] provide relatively cheaper validation costs [64]. They have been extended so as to be suitable for mobile broadcast environments [147], and have been widely adopted in subsequent research [59][63][61][155][60][156]. Therefore, a novel OCC approach that employ forward validation schemes to address the real-time requirements of mobile devices and mobile broadcast environments is proposed in this thesis.

The proposed approach is a combination of virtual execution policy and a novel transactional OCC phase's order in which the write phase occurs before the validation phase. When transactions are in a rerun state, we can offset their validation until after the write phase. There are important benefits of this approach, for example writes may become visible to transactions in the read phase earlier, affording more likelihood of reading up-to-date data from disk. Also, overall blocking is reduced (in the original OCC protocols, transactions in the read phase need to be blocked as a transaction commits changes to the database - such blocking is not required in the proposed approach, as out-of-date reads are caught by the later validation step). The proposed approach is explored in this chapter in two contexts. Firstly, it is show that implementing it on the mobile devices themselves can improve contention problems due to shared resources on that device. Secondly, it is further show that the proposed protocol represents an efficient implementation for client-server models based on a broadcast datacycle for a wireless network, which is now receiving renewed interest due to its potential for energy efficiency in the field of mobile communications [122]. The results show that with the proposed approach the number of client-server transactions which miss their deadlines due to concurrency issues is reduced. The number of transactions requiring restarts is also reduced, so less energy is used in retransmitting data or in accessing a resource a second time.

This chapter introduces the proposed Read-Write-Validation approach, and discusses the correctness justification for reordering the transactional phases. Then, an extensive discussion of the advantages and disadvantages of the Read-Write-Validation approach are presented, and additional enhancements of the approach are introduced. Then, the Read-Write-Validation protocol for governing transactions operating on databases residing on mobile devices themselves is explained. Finally, the distributed Read-Write-Validation protocol for governing transactions operating in client-server models based on wireless broadcast environment is presented.

3.2 Read-Write-Validate Approach

The proposed approach fundamentally changes the order of the traditional transactional phases in conventional OCC [8]. The write phase now follows the read phase, with the validation phase occurring afterwards as illustrated in Figure 3.1. Such the transaction now commits after the write phase finishes and before validation phase starts.

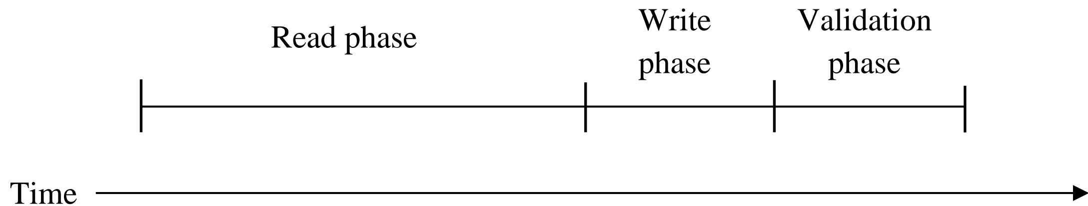


Figure 3.1: Read-Write-Validate approach phases.

Both write and validation phases are collectively considered a single critical section, allowing only one transaction to execute in either phase [8][56]. The validation phase executes based on a forward validation scheme [56]. Validation with this scheme is achieved by identifying the intersection between the validating transaction's write set WS (Tv) and every read set of all concurrently running transactions RS (Ti):

$$WS\ (Tv) \cap RS\ (Ti) \neq \{\}$$

If the above holds true (i.e., there is intersection), then a conflict has occurred and a resolution policy is needed in order to resolve it. In this approach, The never abort validating (NAV) transactions strategy [85] is the only conflict resolution policy that is applied. This guarantees that a transaction entering the critical section will commit. This requires transactions conflicting with the validating transaction to be aborted. NAV is important in the sense that the validating transaction has used the resource and completed its execution; it will be very expensive to abort such transaction. In addition, NAV gains in importance in the proposed approach because the write phase occurs before the validation phase. Therefore, data will be updated and accessible by other transactions at the time of validation, which makes aborting validating transactions more expensive and more complicated. The new transactional phase's ordering is combined with the virtual execution technique to allow for much greater performance. The combination with virtual execution technique will be described in detail in section 3.6 Read-Write-Validate enhancement.

3.3 Justifying the Read-Write-Validate Approach

Using a critical section around the write and validation phases, the ordering becomes trivial as system correctness is guaranteed (as serial schedule) in either scheme. However, without using forward validation coupled with the never abort validating (NAV) transactions strategy, it would be more costly to employ the Read-Write-Validate approach; here if a validating transaction is aborted it is expensive to undo the changes made during the write phase. This would also result in an increased number of conflicts due to other transactions having accessed the same data needing to be aborted. Another advantage of this strategy of NAV transactions is that the resources utilized by a validating transaction are not wasted [85].

In, addition, real-time centralized transactional databases need to handle transactions with timing constraint in the form of deadlines. Factors such as system contention have a direct impact on satisfying transactional deadlines; such factors occur during validation. Therefore, it is acknowledged that, in the traditional OCC phase ordering, the validation step introduces a degree of non-determinism with regards to how long writes will take to become visible in the database (delaying entering the write phase). The validation phase is required to ensure system correctness with regards to transactions that are still executing, rather than providing a direct benefit to the validating transaction itself. If the write phase occurs before the validation phase then non-deterministic timing constraints of the validation phase are removed, allowing a transaction to commit sooner.

3.4 Advantages of the Read-Write-Validate Approach

The Read-Write-Validation approach provides substantial advantages leading to the achievement of significant improvements in system performance. These advantages including the minimization of transaction lifespan , eliminating the blocking of concurrent transactions, newly starting transactions are never blocked and updates are visible earlier. The advantages achieved by the Read-Write-Validate approach are described in the following sections.

3.4.1 Transaction Lifespan Minimized

Transaction lifespan is the transaction's execution time, which is the time between the transaction beginning its execution and committing. In conventional OCC approaches, the lifespan of transaction T includes the non-deterministic timing of the validation phase period, which is the period of validating other concurrently running transactions in order to ensure system correctness rather than providing direct benefit to validating transaction itself. The additional non-deterministic timing of the validation phase in conventional OCC is illustrated in Figure 3.2 by the line marked in red.

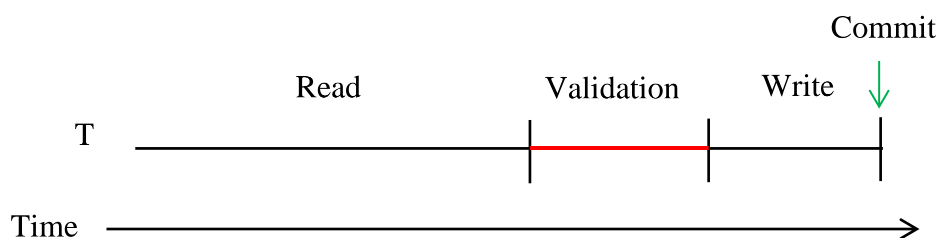


Figure 3.2 Transaction
lifespan in conventional OCC

In the Read-Write-Validate Approach, the non-deterministic timing of the validation phase period is removed from the lifespan of the transaction, as shown in Figure 3.3. The validation phase of transaction T executes after it commits. This is an important benefit in real-time systems where the validation phase introduces non-deterministic timing constraints, which may effect on satisfying transactional deadlines.

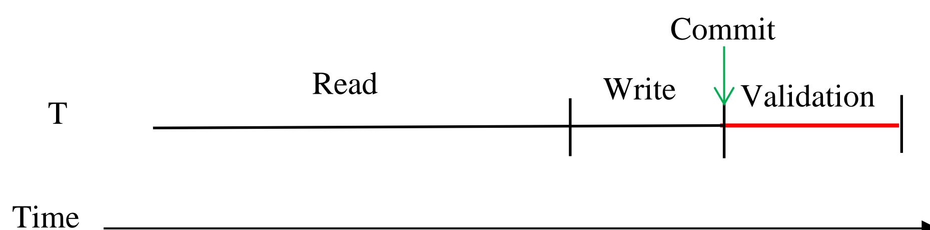


Figure 3.3 Transaction
lifespan in Read-Write-Validate approach

3.4.2 Blocking of Concurrent Transactions Eliminated

In the conventional OCC approach, non-conflicted transactions executing in the read phase will eventually be blocked while the validating transaction executes in the validation and write phases, in order to be prevented from entering a conflict state after validation. If non-conflicted concurrently running transactions are allowed to continue execution after they have been validated, they may potentially enter a conflicted state. This arises if a value read by non-conflicted transactions in the read phase is shared with the write set of a validating transaction. As a result, non-conflicted concurrently running transactions must be blocked after having been validated, until the validating transaction commits, in order to guarantee database consistency. Although non-conflicted transactions constitute the majority of transactions contentious workload in OCC transactional systems, blocking them continually is considered to be a great weakness of conventional OCC.

In the Read-Write-Validate approach, non-conflicted transactions are no longer blocked from progressing, and yet database consistency is still maintained, since transactions waiting to enter the critical section are not considered blocked. Concurrently running transactions are allowed to continue execution while the validating transaction execute in both the validating and write phases. If concurrently running transactions enter a conflict state while the validating transaction is writing, such a conflict will eventually be detected in the deferred validation phase. If non-conflicted concurrently running transactions do not enter a conflict state while the validating transaction is writing, such transactions will successfully pass validation along with the validating transaction, and continue execution without affecting database consistency. The following two scenarios illustrate how the blocking of concurrently running transactions is removed in the Read-Write-Validate approach.

- *Conventional OCC Scenario*

  Figure 3.4 shows a scenario where three concurrent transactions, T1, T2 and T3 are running. T1 finishes the read phase before T2 and T3 and consequently starts validation against T2 and T3. T2 is a conflicted transaction because it is reading the shared data x and y, and this conflict is detected and T2 is aborted. The non-

conflicted transaction T3 successfully passes validation, but T3 has to be blocked while T1 is completing the rest of its validation phase along with other concurrent transactions and its entire write phase in order to ensure that T3 will not enter an inconsistent state after having been validated against T1. T3 will resume execution after T1 commits and leaves the critical section. Such blocking is important when transaction read sets are dynamic, (which means that it is not known in advance when the transaction starts. Therefore, the scheduler has to block all non-conflicted concurrently running transactions after they have been validated, to make sure that none of them will enter a state of inconsistency.

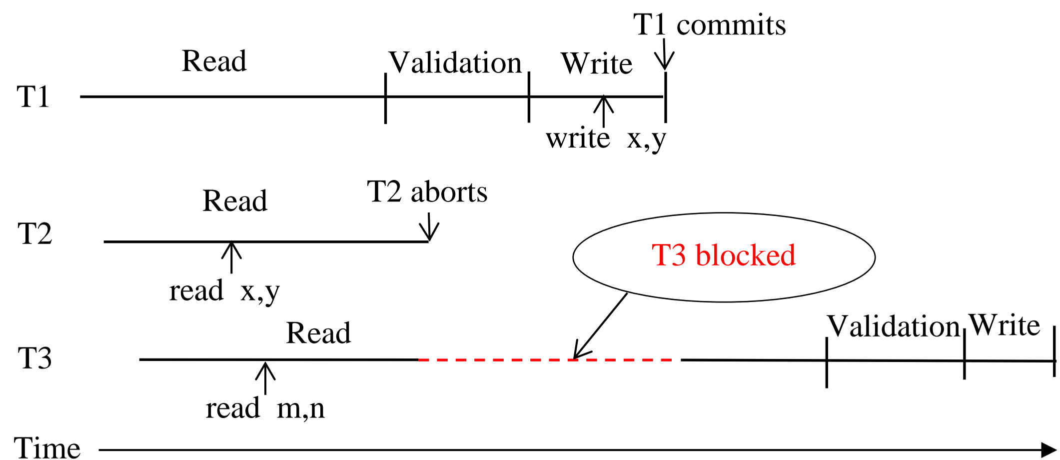


Figure 3.4 The occurrence of
blocking in the conventional OCC approach

- Read-Write-Validate Approach Scenario

Figure 3.5 shows the same scenario running three concurrent transactions T1, T2 and T3. The validating transaction T1 starts executing its write phase before the validation phase, based on the Read-Write-Validation approach. Both T2 and T3 continue execution while the validating transaction writes. When T1 starts the validation phase, the conflicted transaction T2 will be detected and aborted. If T3 enters a conflict state while T1 performs the write phase, that is not a problem, since such a conflict will be detected by validation afterwards. Otherwise, the non-conflicted transaction T3 will successfully validate against

the validating transaction T1, and it continues execution without affecting database consistency. Therefore, non-conflicted transactions like T3 will benefit from not being blocked during T1's validation and write phase period.

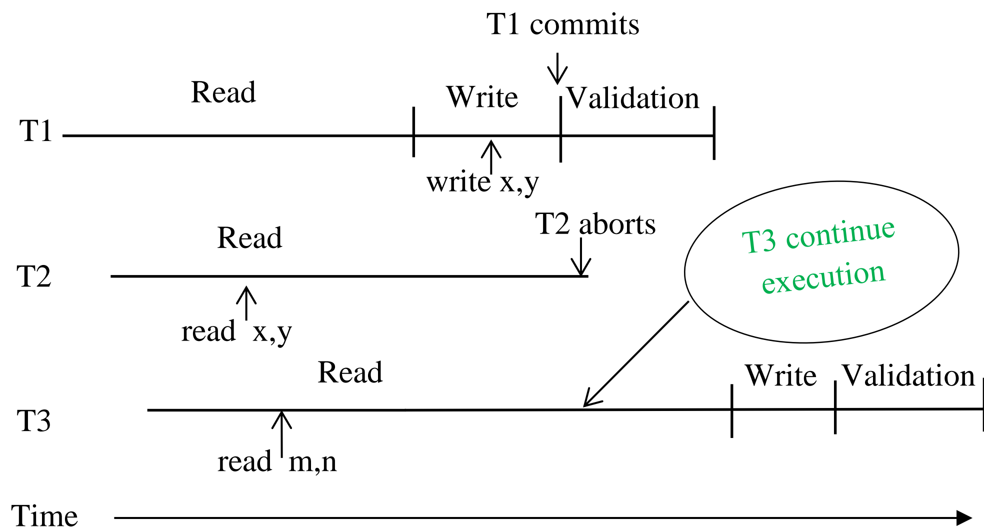


Figure 3.5 Blocking eliminated in Read-Write-Validate approach

3.4.3 Newly Starting Transactions Never Blocked

Newly starting transactions are those, which may start execution while another transaction is executing in the validation or write phases.

In the conventional OCC approach, such transactions will be temporally delayed until the validating transaction commits. This is trivial for the following reasons:

- If the newly starting transaction starts while the validating transaction is executing in the validation phase, it will not be in conflict with the validating transaction because it has only just started and its read set will be empty. However, it has to be blocked to make sure that it will not become involved in a state of inconsistency, as described in the previous section.

- If the newly starting transaction starts while the validating transaction is exacting in the write phase, obviously this transaction will not have any chance to be validated in the future against the currently validating transaction which has passed the validation phase already. Therefore, newly starting transactions

must be blocked until the current validating transaction finishes the whole updating process and commits, in order to guarantee database consistency.

The Read-Write-Validate approach has the advantage of allowing newly starting transactions to continue execution straightway without affecting database consistency. This advantage is explained in the following.

- If the newly starting transaction starts execution while the validating transaction running in the validation phase, which takes place after the write phase, at this point the database will already be updated. Therefore, newly starting transactions will never enter a conflicted state, and no validation is required for such new transactions.

- If the newly starting transaction starts execution while the current validating transaction is running in the write phase, the newly starting transaction continues execution and causes no problem. If the newly starting transaction was a conflicting transaction, that is also not a problem because such a conflict will be detected later at the validation phase and the conflicted transaction will be aborted. If the newly starting transaction was actually a non-conflicting transaction, which is in fact expected to constitute the majority of the contentious workload in OCC environments, then such a transaction will pass the validation phase successfully and benefits from not being blocked while the current validating transaction is in its write and validation phases.

The following two scenarios illustrate how newly starting transactions gain these benefits from the proposed Read-Write-Validate approach:

- *Newly Starting Transactions in the Conventional OCC Scenario*

  Figure 3.6 illustrates a scenario where three concurrent transactions, T1, T2 and T3 are executing based on conventional OCC. T1 entered the validation phase before T2 and T3 were started. T2 started execution while T1 was executing in the validation phase. Unfortunately, T2 has to be blocked, as described earlier, until T1 commits in order to prevent T2 from entering a state of inconsistency.

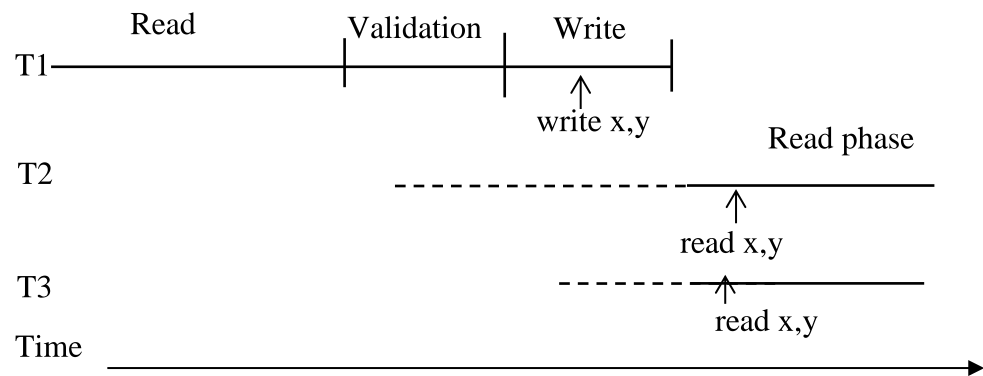


Figure 3.6 Blocing in newly starting transactions in the conventional OCC approach

T3 started execution while T1was executing in the write phase, and it also has to be blocked until T1 commits to ensure consistency. If T3 continued execution during T1's write phase, and T3 comes into conflict with it, such a conflict will not be detected because T1 has already completed the validation phase.

- *Newly Starting Transactions in Read-Write-Validate Approach Scenario*

Figure 3.7 illustrates an example similar to that discussed in the previous section, with an additional transaction T4 and using the Read-Write-Validate approach. T1 entered the validation phase before T2, T3 and T4 started. T2 and T3 started execution while T1 was still executing in the write phase. As opposed to conventional OCC, T2 and T3 will not be blocked, and will continue execution during T1's write phase. Then, at T1's validation phase which occurs afterward, there will be two possibilities as illustrated below:

1. If the newly starting transaction is non-conflicting, such as transaction T2 and which constitutes the majority of the contentious workload, it benefits from not being blocked during T1's write phase because it will successfully pass validation against the currently validating transaction T1 and therefore continue execution.

2. If the newly starting transaction is a conflicting transaction such as transaction T3, it will be detected in T1's validation phase and will abort.

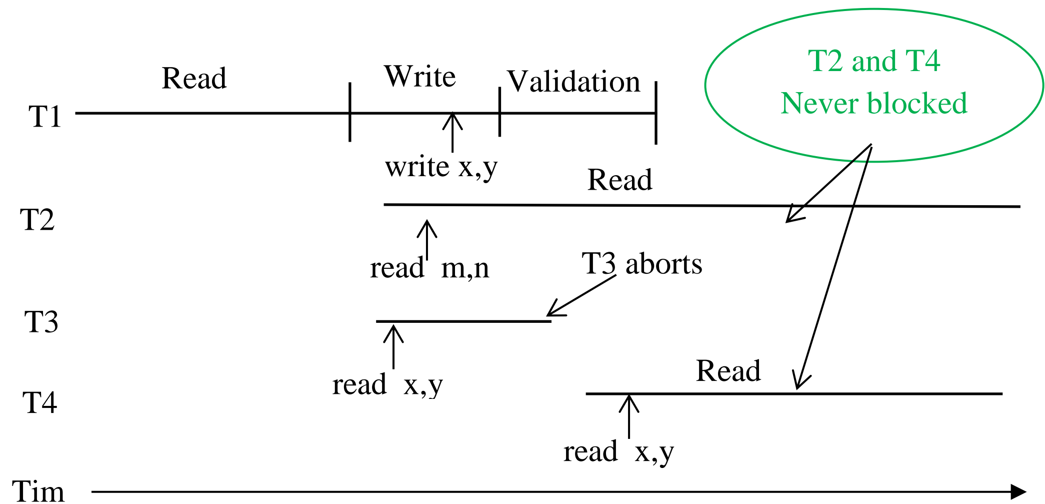


Figure 3.7 New transactions are never blocked in in Read-Write-Validate approach

Transaction T4, which started during T1's validation phase, continues execution without the need for validation, because all of T1's updates have already been transferred to the database. Therefore, T4 will never have the chance to become conflicted with T1.

3.4.4 Earlier Visible Updates

In the Read-Write-Validation approach, writes become visible earlier, affording more likelihood of reading up-to-date data and thus reducing the opportunity for conflicts to occur. This is because the reordering of the validation and write phases guarantees that all new updates are made before the validation phase starts. The following two scenarios running two transactions, T1 and T2, based on the conventional OCC approach and the Read-Write-Validate approach clarify this issue.

- *Late Visible Updates in Conventional OCC Scenario*

  Figure 3.8 illustrates two concurrently running transactions, T1 and T2. T1 entered the validation phased before T2. T2 was blocked while T1 executed the

validation and write phases. After T1 commits, T2 resumes execution and reads the new T1 updates of x,y after the blocking period which is equal to approximately T1's validation and write phases.

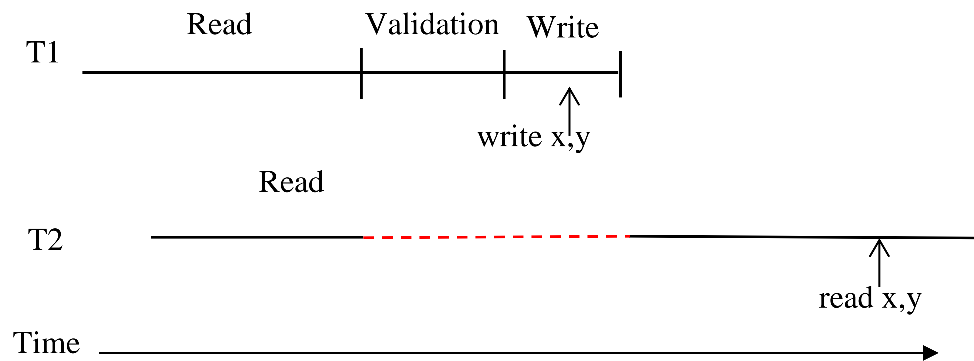


Figure 3.8 Late visible
updates in conventional OCC approach

- *Earlier Visible Updates in Read-Write-Validation Approach Scenario*

  Figure 3.9 shows the same scenario of concurrent transactions T1 and T2 using the Read-Write-validation approach. T2 will not be blocked, as described earlier in previous sections, while T1 is executing its write and validation phases. T2 continues execution and reads the new T1 updates of x,y while T1 is still running in the validation phase. T2 is not considered to be in conflict with T1 even if it could read T1's updates during its validation phase because T1 is already committed by that time, and the validation performed by T1 is needed in order to keep other concurrently running transactions consistent. Therefore, the time of T2's execution during T1's write phase and part of T1's validation phase benefits T2 which reaches the operation of reading x,y earlier. This time is illustrated in T2's execution time in the green segment. The fact that T2 does not need to wait for the rest of the T1's validation to be finished (illustrated in T2's execution time in the blue segment), together gives T2 the ability to see and use T1's new updates earlier.

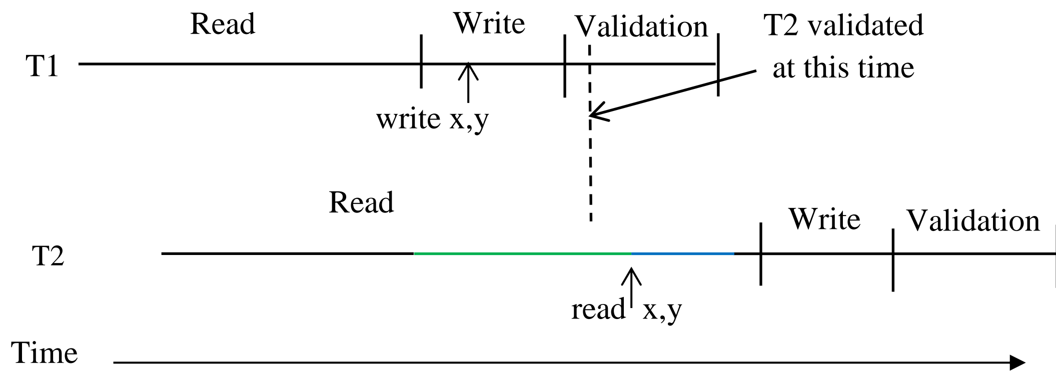


Figure 3.9 Earlier visible updates
in the Read-Write-Validate approach

3.5 Disadvantages of Read-Write-Validate Approach

The advantages of the Read-Write-Validate approach described in the previous section are not without cost. There is a price to pay in order to gain these advantages. The cost can be summarised in two points. There is a longer wasted execution and a critical section constraint, these are discussed below.

3.5.1 Longer wasted execution

If a conflicted transaction aborts as soon as the conflict is detected in the validation phase, then placing validation phase before the write phase as in the conventional approach is beneficial in the sense that conflict detection will occur earlier. In other words, there is no need to wait until the entire write phase period is completed to identify conflicts. This minimises the amount of wasted work resulting from that conflicted transaction, compared to aborting a conflicted transaction after the write phase execution period as in the Read-Write-Validate approach. The following two scenarios illustrate this problem.

- Wasted Execution in Conventional OCC Approach

  Figure 3.10 shows the scenario of two concurrently running transactions,T1 and T2. T1 entered the validation phase and started validating against T2. Due to a conflict with T1, T2 aborts. The amount of work which has been done by T2 is

considered to be a wasted execution (Marked by the blue segment in T2's execution line).

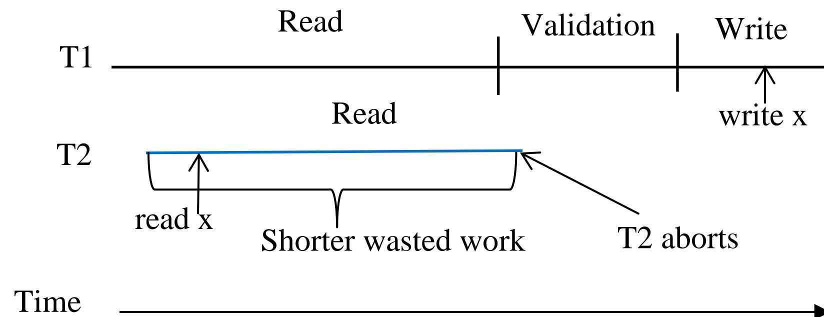


Figure 3.10 Short period of wasted execution in the conventional OCC approach

- Read-Write-Validation Approach Scenario

The phase reordering in the Read-Write-Validate approach means the validation phase occurs after the write phase. Therefore, conflicted transactions will continue execution in the entire period of the write phase before they are detected. This consequently increases the amount of wasted execution of conflicted transactions, which are shown by the blue and red segments in T2's execution line in Figure 3.11. Phase reordering benefits non-conflicted transactions, which constitute the majority of the contentious overload in OCC main assumption. In contrast, it a has negative impact in case of conflicted transactions, because it gives them the chance to continue execution for a longer period before being aborted.

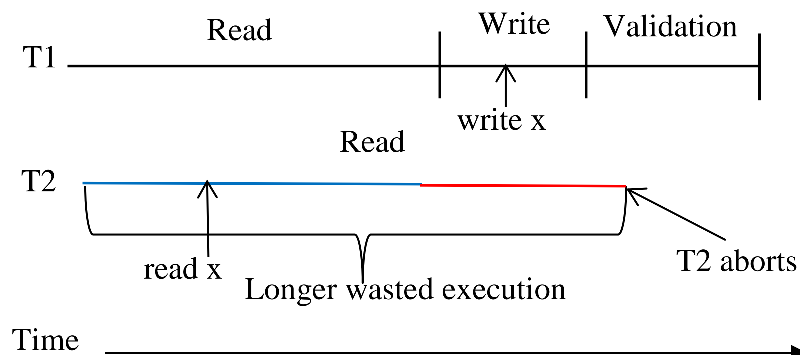


Figure 3.11 Longer period of wasted work in the Read-Write-Validate approach

However, this potential disadvantage of the Read-Write-Validate approach is solved by combining Read-Write-Validate with virtual execution, which is discussed later in this chapter in section 3.6.

3.5.2 Critical Section Constraint

The critical section constraint of the write and validation phases is considered to be a scalability drawback of the Read-Write-Validate approach, which is inherited from the original forward and backward validation approaches. It is needed to ensure database consistency. If more than one transaction enters the write or validation phase at the same time, a state of inconsistency might occur. However, this constraint is also solved in the distributed version of the Read-Write-Validate Approach introduced in section 3.9.

3.6 Read-Write-Validation Enhancement

The virtual execution approach described in section 2.4.2 is applicable with the Read-Write-Validate approach. The combination of both approaches is beneficial and fixes the problem of longer period of wasted execution discussed previously in section 3.5.1. Furthermore, it adds two important advantages to the Read-Write-Validate approach: reduces the risk of conflict and improves energy efficiency. These additional advantages are described in the following.

3.6.1 Energy Efficiency Improvement

Virtual execution allows those transactions that have been aborted to re-execute using in-memory values as opposed to reading directly from the persistent store. Cached values from the write sets of committed transactions together with read sets from currently executing transactions populate a buffer local to the transaction management system. This improves the proposed approach because accessing a conventional hard disk drive is expensive in terms of power usage, as the disk must attain read speed, and the appropriate data sector be found. Even solid-state drives are significantly more expensive to access compared to the local memory. Clearly, a reduction in the

frequency of transactions that must be restart will reduce the number of times a disk is accessed, leading to a reduction in energy usage.

3.6.2 Reduction of Conflict Risk

In virtual execution, rerun transactions are quicker than those in their initial run, as there is no access to the persistent store. So, transactions in rerun become ready to enter the critical section for the write and validation phases in a shorter time, which increases the probability of transaction commitment and obviously reduce the risk of being conflicted with other concurrently running transactions

3.6.3 Wasted Execution Elimination

In the virtual execution environment, a conflicted transaction in the first run does not abort directly even if a conflict is detected (as explained previously in section 2.4.2,). Instead, it continues execution to prefetch all of the read set data to the main memory. In this sense, the problem of increasing wasted execution discussed earlier in section 3.5.1, which results from deferring the validation phase until after the write phase, no longer exists. In fact, there is no wasted execution in the first phase because such an execution will be used to prefetch the read set data. Conflicted transactions only restart in the rerun phase. The time at which conflict is detected (whether early or late detection) then makes no difference. Such a benefit is illustrated in following scenario.

Figure 3.12 shows two concurrently running transactions T1 and T2, based on the Read-Write-Validate approach combined with virtual execution. T1 entered the write and validation phases before T2. T1 detected a conflict with T2 while T2 was still executing in the read phase. As opposed to the previous scenarios, T2 will not be aborted even if it is a conflicted transaction. T2 continues execution until the end of the read phase to prefetch all T2's read set data, then it aborts and it rerun using the in-memory data prefetched by T2 and the write set of T1. Therefore, first phase execution is not considered wasted work and the conflicted detection time makes no difference in the virtual execution environment.

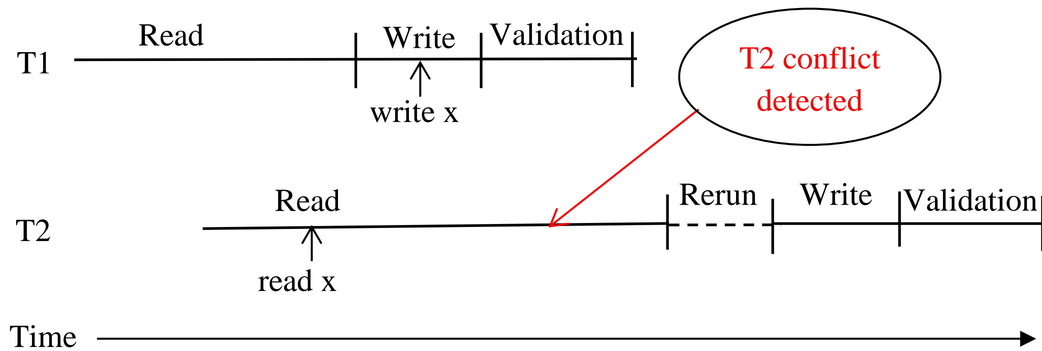


Figure 3.12 Wasted executions no longer exist in the virtual execution environment.

3.7 Coping with System Failures

System failure occurs in a situation when the state of transactions is lost. For instance, power loss may cause a complete loss of the main memory content, or software errors may overwrite parts of the data in the main memory. In such system failures, the following recovery techniques are performed to ensure database consistency:

- Restart all transactions that have not yet been committed from scratch. Therefore, they will read from the database directly, as if they were in the initial run.
- If a transaction was executing in the validation phase, then this transaction has already committed and written the new updates to the database. Therefore, there is no need to execute the validation phase again because concurrently running transactions will be restarting, reading the updated data directly from the database.
- If a transaction was executing in the write phase, the recovery of such a transaction will use the logging techniques described in a previous study [34].

3.8 Read-Write-Validation Protocol

The Read-Write-Validate protocol explained in this section is a straightforward implementation of the Read-Write-Validate approach. Its aim is to govern concurrent transactions running on the mobile devices themselves, in order to improve issues of

contention with shared resources on that device, such as the solid-state disk. The Read-Write-Validate protocol and the pseudo-code of the validation algorithm are illustrated below.

3.8.1 Protocol Description

A transaction that reaches the end of the read phase enters a pre-commit set (PCS). One member of the PCS may be chosen by the scheduler to enter the write phase. The earliest deadline policy [35] is employed to give priority to transactions that are closest to expiration.

Transactions that either are in the read phase or are members of the PCS may be aborted and rerun if they are found to be in conflict with a validating transaction. The validating transaction is guaranteed to commit. Therefore, any other transactions that are in conflict with it must be rerun. A transaction that is in its initial run will complete the read phase, regardless of whether or not it is in conflict, and enter the PCS. Allowing conflicted transactions to complete the read phase improves performance because the persistent data store is only accessed once per read operation [53]. A transaction that is rerun will have a local copy of all the required data for it to attempt execution again.

A forward validation scheme is employed which, during the validation phase of Tv, checks if there is an intersection between the write set WS (Tv) with any read set RS (Ti) for all running transactions.

$WS(Tv) \cap RS(Ti) \neq \{\}$

This includes transactions executing in the read phase and members of the PCS. If an intersection (i.e., a conflict) is found, then:

- If the conflicting transaction Ti is in the initial run, it is allowed to proceed to the read phase and is marked for rerun. Ti enters the PCS upon completing the read phase but is not eligible to enter the write phase. At this point, Ti is updated with the values from other transactions it has conflicted with and will be rerun.

- If Ti is in rerun then it is aborted. At this point, RS (Ti) is updated with WS (Tv), so that it can be rerun again with the updated read set.

Newly started transactions may start the read phase at any time. The correctness of system execution is ensured as follows:

- If a transaction enters the read phase while the validating transaction is writing, there is the possibility of reading inconsistent data. This will be detected when the validating transaction finishes the write phase and enters the validation phase.
- If a transaction enters the read phase while the validating transaction is validating, then any reads are made against the updated values from the persistent store (as validation occurs after the write phase). Any transactions entering the read phase at this point do not need to be validated against the currently validating transaction.

3.8.2 Pseudo-code

Pseudo-code illustrating the execution of the validation phase is presented next.

Conventions used in the pseudo-code

- Active Transactions (AC) – This is the set of all currently running transactions. It includes transactions in the read phase and those waiting to enter the write and validation phases.
- Conflicted Set (CS) – CS (T_i) contains the updated read values from any validating transactions that T_i has conflicted with. Each item (O_k) in CS (T_i) is cached until RS (T_i) can be updated. These values are cached rather than the read set of T_i being directly update to make it clear that the writes would not be automatically updated. If RS (T_i) are chosen to update directly, RS (T_i) can be updated when T_i has finished the initial run or, if it is in rerun, when it is aborted. Upon updating, CS (T_i) is discarded.

The assumption is that a transaction executing in the read phase reads the required data and performs any required computation. Similarly, a transaction in the

write phase updates any values that were written to during its read phase. The scheduler handles rerunning identified transactions along with updating the read sets for conflicting transactions.

The pseudo-code of the validation algorithm is presented below:

Algorithm 6 Validation phase

```
1:   for each Ti in AC do
2:      if ((WS(Tv) ∩ RS(Ti)) ≠ {}) then
3:         for each Ok in (WS(Tv) ∩ RS(Ti)) do
4:             update Ok in CS(Ti);
5:         end for
6:         if Ti in initial run then
7:             mark Ti for rerun;
8:         else
9:             update Ti with CS(Ti), rerun Ti;
10:        end if
11:     end if
12:  end for
13:  discard WS(Tv);
```

3.9 Distributed Read-Write-Validate Protocol

Section 3.8 showed that this approach is applicable to OCC on resource-constrained devices such as smart-phones. Now, this work is extended to the wireless broadcast datacycle model for mobile network applications. Earlier studies on transaction processing in wireless environments focused on read-only transactions [89][157][158][159], which is applicable to conventional information services such as weather and traffic information. However, update transactions must be considered as well in complex mobile applications such as mobile e-commerce [152]. Therefore, the Distributed Read-Write-Validate Protocol aims to improve the overall performance of the system, including update transactions. The results in chapter 4 show that, with this technique, the overall performance of the system is increased and the number of client-server transactions which miss their deadline due to concurrency issues is reduced [18].

The validation process in the distributed Read-Write-Validate Protocol is performed in two stages: the validation stage at the client and the validation stage at the server. Both validation stages are described below.

3.9.1 Validation Stage at Client

The validation stage at the client is performed using partial backward validation for all client transactions. All running transactions at the clients (i.e. both read-only and update transactions) will be validated at the beginning of every broadcast cycle by performing backward validation with the write set of the committed transactions at the server. Conflicted transactions will be marked for rerun, but will continue execution until the end of the read phase using the rerun policy. When a conflicted transaction reaches the end of the read phase, it simply updates the conflicted data items in memory and is rerun without accessing the persistent store. Previous studies [160][54] have shown that optimistic concurrency control performs better if transactions are allowed to reach the end of their read phase before being aborted. This is intuitive, since transactions that have been aborted early would not have retrieved all the required data to be ready locally for the rerun phase. Read-only transactions which are not conflicted can proceed and commit locally at the client. Non conflicted update transactions will be sent to the server to be globally validated.

Partial Backward Validation Pseudo-code

Conflicted Set (CS) – Given CS(T_m), this contains the updated values from C_i and that T_m has been found to conflict with. Each item (O_k) in CS(T_m) is cached until RS(T_m) can be updated with these updated values. These values are chosen to be cached rather than directly updating the read set of T_m in order to make it clear that the writes would not be automatically updated. If RS(T_m)) are chosen to be updated directly, RS(T_m) can be updated when T_m has finished the initial run or, if it is in rerun, when it is aborted. Upon updating, CS(T_m) is discarded.

It is assumed that a transaction which is executing in the read phase reads the required data and performs any necessary computation. Similarly, a transaction which is in the write phase will update any values that were written to during its read phase. The

scheduler will handle rerunning transactions that have been marked for rerun, along with the process of updating the read sets for conflicting transactions.

The pseudo-code for partial backward validation is presented below:

Algorithm 7 Partial backward validation

```
1:   PartialBackwardValidation(Tm){
2:       if ((ControlInfo(Ci) ∩ RS(Tm)) ≠ {}) then
3:           for each Ok in (ControlInfo(Ci) ∩ RS(Tm))
4:               update Ok in CS(Tm);
5:           if Tm in initial run then
6:               mark Tm for rerun;
7:           else
8:               update Tm with CS(Tm), rerun Tm;
9:           endif
10:      else
11:          record the value of Ci,
12:      endif
13:  }
```

T_m – transaction generates and executes at the clients.

ControlInfo(C_i) – the set of data items which was updated.

3.9.2 Validation Stage at the Server

The validation stage at the server is performed in two steps: 1) final partial backward validation; 2) Read-Write-Validate. Both steps are described below:

- ***Final Backward Validation Algorithm***

Update transactions have to perform final backward validation with any possibly committed transactions after the update transaction has finished partial validation at the client, and before starting Read-Write-Validation validation at the server [147][59]. The

results of this validation (to commit or abort) will also be included in the information table as acknowledgment to the mobile client for further actions.

The final backward validation pseudo-code is as follows:

Algorithm 8 Final backward validation

```
1:    FinalValidation(T_m) {
2:        For each Ti ( i= 1,2,...,n) {
3:            If (RS(T_m) ∩ WS(T_i) ≠ {}) then{
4:                Return fail;
5:                Break;
6:            }
7:    }
```

- ***Read-Write-Validate Algorithm***

One of the transactions which is ready to commit will be chosen to enter the write phase by the scheduler. The earliest deadline policy [35] is employed to give priority to transactions that are closest to deadline expiration. Once this transaction has completed the write phase, it performs forward validation against all concurrently running transactions at the server [16][18]. This includes locally generated transactions and update transactions that have been received from clients for global validation. Any locally generated conflicted transactions will be marked for rerun. They will continue executing until the end of the read phase in the first run as described previously [18][15][16]. Conflicted updating mobile transactions will be aborted and rerun again at the client. When a validating transaction finishes the write and validation phases, the write set will be broadcast in the next broadcast datacycle with the control information table. The control information table is a table consisting of the write sets of committed transactions at the server (new updates), which is used for partial backward validation at clients to keep mobile transactions consistent. In addition, it contains final validation results of mobile transactions (performed at the server) as acknowledgement to the mobile clients for future actions. In other words, control information table provide mobile clients with all information it need to maintain consistency.

The pseudo-code for Read-Write-Validate uses the same notation explained in the section on partial backward validation, and is presented as follows:

Algorithm 9 Read-Write-Validate validation

```
1:   validate(Tv){
2:       if (Tv is a mobile update transaction) then
3:           FinalValidation (Tv);
4:           If (returen fail )then
5:               Abort (Tv); exit;
6:           End if
7:       End if
8:       Commit WS(Tv) to database;
9:       ControlInfo(Ci) = ControlInfo(Ci) U WS(Tv);
10:      For each Tj (j= 1,2,.....,n) {
11:          if ((WS(Tv) ∩ RS(Tj)) ≠ {}) then
12:              if ( Tj is not mobile update transaction) then
13:                  for each Ok in (WS(Tv) ∩ RS(Tj)) {
14:                      update Ok in CS(Tj);}
15:                  if (Tj in initial run) then
16:                      mark Tj for rerun;
17:                  else
18:                      update Tj with CS(Tj), rerun Tj;
19:                  endif
20:              else
21:                  abort (Tj);
22:              endif
23:          endif
24:      }
25:  }
```

The proposed approach is orthogonal to the back-off method [59] and OCC for the broadcast disk scheme [63]. That is to say, both of these approaches can be combined with the proposed approach.

3.10 Summary

This chapter has introduced the Read-Write-Validate approach, which involves a novel order of transactional phases in OCC. The proposed approach changes the order of the traditional read/validation/write phases; write now follows the read phase with validation occurring after the write phase. The combination of the proposed approach with virtual execution environments brings substantial benefits for resource-constrained devices in terms of performance, including throughput, response time and late transaction rate, and also in terms of the efficiency of energy use (battery utilization).

The proposed approach is explored in this chapter in two contexts:

- Firstly, the Read-Write-Validate protocol is suitable for mobile devices which are resource constrained such as smart phones and tablets. The Read-Write-Validate protocol improves issues of contention with shared resources on such devices.
- Secondly, it is then adopted in a distributed Read-Write-Validate protocol, which is suitable for client-server models based on a wireless broadcast datacycle which are receiving renewed interest due to the potential for increased energy efficiency in the field of mobile communications. The distributed Read-Write-Validate protocol improves issues of contention in both the server and client devices.

This chapter provides clear explanations of the pseudo code of algorithms to show how these protocols work. The next chapter concentrates on the evaluation of the proposed approach. It includes descriptions of the simulation tool used for the evaluation, the system model and settings implemented in the simulation and the results gathered from the simulation experiments.

Chapter 4

Evaluation

The evaluation reported in this thesis focuses on improvement in performance associated with the proposed approach, which includes the assessment of throughput, response time and miss rate measurements. Energy efficiency evaluation is beyond the scope of the thesis and experiments regarding this will be carried out in further work. In this chapter, two simulation implementations are performed in order to evaluate the contributions provided by the present research. The first simulation is used to evaluate the Read-Write-Validate protocol, and the second simulation evaluates the distributed Read-Write-Validate protocol. Both simulations are presented below.

4.1 Read-Write-Validate Protocol

In this section, a brief introduction of the simulation tool used to evaluate the Read-Write-validate protocol is presented. Then the simulation model which demonstrates the Read-Write-validate protocol and the parameters used in the simulation is explained. Finally, the results of a comparison between the Read-Write-validate protocol with the new ordering of phases (read-write-validation) and the forward validation protocol with conventionally ordered phases (read-validation-write) are presented.

4.1.1 Simulation Tool

The simulation of the Read-Write-Validate protocol was implemented using SimJave, which is a simulation package used to build working models of complex systems. It is a public source discrete event toolkit produced by Fred Howell and Ross McNab at the Department of Computer Science, University of Edinburgh. SimJave consists of three packages: eduni.simjava, eduni.simanim and eduni.simdiag.[161][162]

- eduni.simjava: The purpose of this package is implementing standalone java simulation code.
- eduni.simanim: this is integrated with the previous package to visualize the simulation by providing a skeleton applet.
- eduni.simdiag. This package's purpose is to give Simjava the ability to display results in graphic form.

Building a simulation is based on breaking the systems down into different entities, and extending Simjava classes to simulate the behavior of such entities. The communication between these entities is performed by scheduling events.

4.1.2 Simulation Model and Setting

A simulation model is produced that matches closely accepted designs published in the literature [64][9]. A few modifications are introduced to this design to accommodate the rerun of transactions and the format of the proposed protocol. The model investigates different performance characteristics of the proposed protocol compared to those of a forward validation approach in a virtual execution environment. A range of results are presented highlighting the performance benefits of the Write-Read-validate protocol.

The simulation model consists of a single-site database system operating with a shared-memory multiprocessor. It contains two disks and two CPUs with a queue per disk and a shared queue for the CPUs. The simulation parameters shown in Table 1 were taken from previous simulation experiments [64][40][163]. The transaction size remains the same for every transaction and the write set is assumed to be a subset of the read set. When the transaction performs a read, a 36μs cost is incurred to access the disk and a further 1.5μs for processing the page. A write costs 200μs with 36μs to read the page beforehand. When the transaction enters the write phase, 200μs per write is incurred. The disk access probability is used for a page being present inside the buffer. For rerun transactions this probability is zero, as the page is present in memory. The validation cost is based on the number of transactions that have to be validated, with a unit cost of 0.5μs. Deadline assignment, as described elsewhere [78], is controlled by the minimum and maximum slack factor parameters that provide a lower and upper bound for a transaction's slack time. The following formula from the study cited above [78] is used when calculating a transaction's deadline:

Deadline = AT + uniform (Minimum Slack, Maximum Slack) * ET

In the formula, AT and ET denote arrival and execution times respectively. As deadlines must be calculated prior to execution, ET is an estimated value based on transaction size, disk access and CPU access, which is equal to 1250 μs when transactions execute with no contention.

Parameter	**Value**
Pages in database	5000
Transaction size	12-page read set 4-pages write set
disk access (read) disk access (write)	36 μs 200 μs
CPU access	1.5 μs
disk access probability (1st run)	0.5
disk access probability (rerun)	0
Minimum slack factor	2
Maximum slack factor	8
Validation cost (per transaction)	0.5 μs
Transaction arrival rate	1 per 1000 μs to 1 per 200 μs

Table 1. Simulation parameters used in

the evaluation of the Read-Write-Validate protocol

Each simulation was performed using the same parameters for 10 random number seeds. Each run consisted of 10000 transactions. To allow the system to

stabilize, the results from the first few seconds were discarded. Mean values are presented for the performance metrics analysed in all experiments.

Two experimental sets are presented. The first set of experiments was based on the assumption that 50% of execution transactions are updating transactions. The second set of experiments was based on the assumption that 75% of the execution transactions are updating transactions. The percentage of update transactions was increased in the second set of experiments in order to determine the behavior of the proposed approach in high contention environments. Results from each set of experiments include the average response time, throughput and number of late transactions. In each graph, results are presented for the two protocols. The first is the Read-Write-Validate protocol introduced in the previous chapter using the new ordering of phases (read-write-validation) which is termed LV in the figures. The other protocol is forward validation using the conventional ordering of read-validation-write phases, which is abbreviated to FV in the figures.

4.1.3 Simulation Results

The results of the series of experiments are presented in the following:

Experimental set 1:

The first set of experiments was based on the assumption that 50% of transactions are updating transactions, and the results are illustrated in Figures 4.1-4.3.

Figure 4.1 shows throughput for an increasing rate of transactions. Throughput is measured as the number of committed transactions, with the commit occurring at the end of the write phase for both types of phase ordering. All protocols share a common progression when contention is low, and are still manageable using the concurrency control protocol. Therefore, as the number of transactions input to the system increase, the throughput of the system also is increases. However, when the point is reached where the level of contention is too high and cannot be handled by concurrency control, throughput starts to degrade. The number of transactions missing their deadline, shown in Figure 4.3 also has an impact on throughput as these transactions are aborted and will never commit. As the rate increases, the number of late transactions increases as throughput falls. Figure 4.1 clearly demonstrates that the proposed approach is more efficient in handling high level of contention among transactions, reaching its highest

point at about 3600 transactions per second. In comparison, the conventional OCC approach reaches the highest point at about only 2000 transactions per second.

Figure 4.2 shows the average response time for an increasing rate of transactions. The response time is only for transactions that successfully commit and, as the rate of transactions input to the system increases, the response time increases due to high contention. The results in figure 4.2 illustrate that when the transaction rate is less than 1500 per second, both approaches have the same response time due to low contention. Then from 1500 to 5000 transactions per second, the proposed approach has a lower response time than the conventional approach. This indicates the effect of the advantages gained by the proposed approach, as presented in chapter 3 which, including that the cost of the validation phase does not affect the transaction's commit time and also that concurrent running transactions do not suffer temporary blocking when another transaction is validating. Above 5000 transactions per second, the average response time is similar for both protocols. The response time stabilizes around 4500 microseconds due to deadline assignment, where only transactions that have a sufficiently large deadline will be able to commit. Regardless of the benefits of the proposed approach, transactions at this level of contention expire during the initial run of the read phase.

Figure 4.3 shows the percentage of transactions which miss their deadlines. All protocols have no late transactions if contention is low and they are still manageable using concurrency control. When the point is reached where contention is too high and can hardly treated adequately using concurrency control, the late transactions rate starts to rise. Figure 4.3 illustrates that both protocols have no late transactions when transaction contention is low, as the rate of the transactions being input to the system is less than 2000 per second. Then, when the rate rises to more than 2000 transactions per second, the rate of late transactions with the conventional OCC protocol start to rise, which indicates its inefficiency in coping with such level of contention. However, the proposed approach manages higher level of transaction contention, with no rise in late transactions until about 3700 transactions per second, which indicates its ability to deal with high transaction contention. Then each protocol, at its peak, has a high percentage (around 80%) of missed deadlines. With high levels of system contention, transactions experience longer delays in accessing the disk and the CPU. This results in transactions

being more likely to miss their deadlines during the read phase and never entering the validation and write phases.

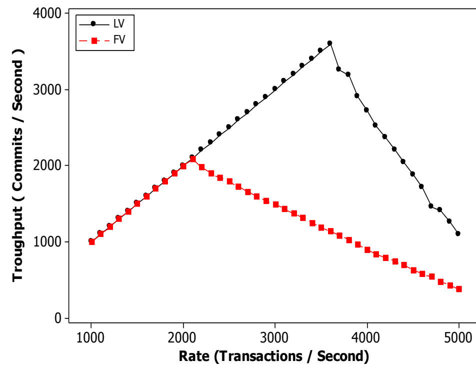


Figure 4.1: Throughput with 50% of update transactions

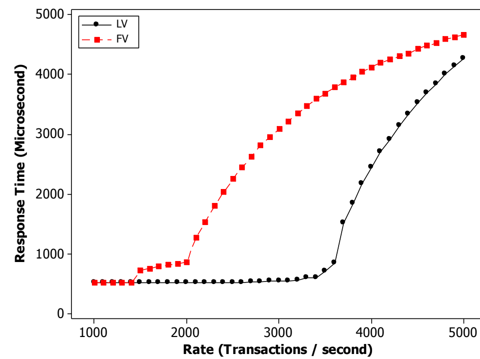


Figure 4.2 Average response times with 50% of update transactions

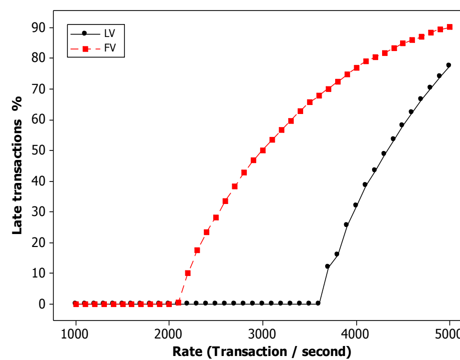


Figure 4.3 Late transactions with 50% of update transactions

Experimental set 2:

The second set of experiments was based on the assumption that 75% of execution transactions are updating transactions, and the results are illustrated in Figures 4.4-4.7.

Figure 4.4 shows the throughput for increasing rates of transactions. Both protocols share a common progression when contention is low and are still controllable by the concurrency control protocol. Thereafter, as the number of transactions input to the system increases, the throughput of the system increases. However, when the point is reached where contention is too high and can hardly be treated properly by concurrency control, the throughput starts to degrade. The numbers of late transactions

shown in Figure 4.6 strongly impacts on throughput, since these transactions are aborted and will never commit. As the transaction rate increases, the number of late transactions increases as throughput decreases. However, Figure 4.4 also shows that the proposed approach is efficient in coping with higher contention among transactions. It reaches the highest point at about 3400 transactions per second, while conventional OCC approach only reaches the highest point at about 2600 transactions per second. The plateau shown around 3500 transactions per second represents a bottleneck in the critical section in the write and validation phases. This is not considered a problem, since in real systems read-only transactions constitute the majority of typical transactional traffic [68][14]. The graph still illustrates that the Read-Write-Validate protocol sustains a higher level of throughput compared to the other approach.

Figure 4.5 shows the average response time for an increasing rate of transactions. As the rate increases, transaction response time increases due to high level of contention. The figure demonstrates that at rates less than 1500 transactions per second, both approaches have the same response time due to low contention. Then, from 1500 until 5000 transactions per second, the proposed approach has a lower response time than the conventional approach. This demonstrates the effect of the advantages gained by the proposed approach, which include that the cost of the validation phase does not affect the transaction's commit time, and that concurrent running transactions do not suffer temporary blocking when another transaction is validating. Above 5000 transactions per second, the average response time is similar for both protocols. The response time stabilizes at around 4500 microseconds due to deadline assignment, where only transactions that have a sufficiently large deadline will be able to commit. Regardless of the benefits of the proposed approach, transactions expire at this level of contention during the initial run in the read phase. The jump at a rate of ~3400 and then a decline at ~3700 is explained by the plateau in Figure 4.4. First the response times increase because executing transactions need to wait before they are able to enter the write and validation phases, and then response times decline due to the increased miss rate at that arrival rate as shown in figure 4.6.

Figure 4.6 shows the rate of late transactions for an increasing rate of transactions. Both protocols have no late transactions when contention is lower at 2600 transactions per second. Then, as the transaction rate increase, the number of late

transactions with the conventional OCC protocol starts to sharply rise, which indicates its inefficiency in dealing with high contention. However, the proposed approach still shows no late transactions until about 3500 transactions per second, which indicates its ability in dealing with high contention. Each protocol, at its peak, has a high proportion of missed deadlines at around 80%. With a high level of system contention, transactions experience longer delays in accessing the disk and the CPU. This results in transactions being more likely to miss their deadlines during the read phase and thus never entering the validation and write phases.

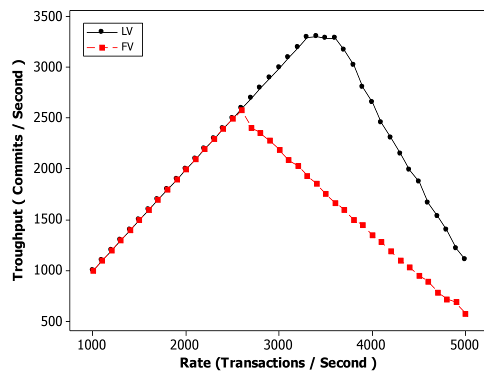


Figure 4.4 Throughput with 75% of update transactions

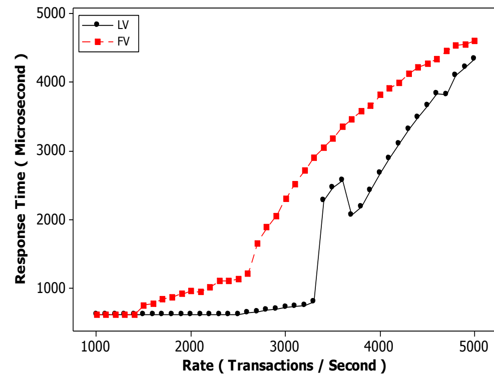


Figure 4.5 Average response times with 75% of update transactions

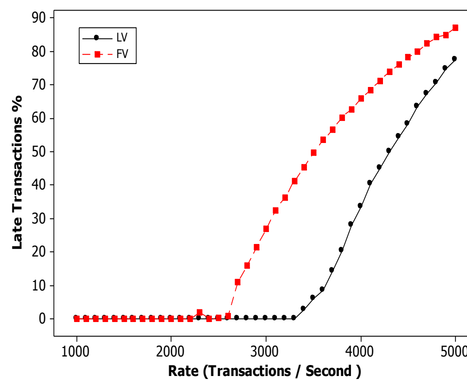


Figure 4.6 Late transactions with 75% of update transactions

From these previous results, it can be concluded that significant improvements in throughput, response time and late transaction rates are gained when deploying the proposed approach to control transaction contention on mobile devices.

4.2 Distributed Read-Write-Validation Protocol

This section describes experiments using a simulation model carried out in order to evaluate the distributed Read-Write-Validate protocol. The simulation is built using Simjava, which is the same simulation tool used in the previous evaluation of the Read-write-validate protocol (section 4.1). Descriptions of the simulation model and the parameters used in the simulation are presented first. Then the results are discussed, comparing the performance of the proposed simulated model with that of the simulation of the original forward and backward optimistic concurrency control (FBOCC) [147].

4.2.1 Simulation Model and Setting

A simulation model has been developed that is based on the model presented in previous studies [147][63][59][126]. The arrival rate of transaction at the server has been increased by a factor of 100x to a value representative of current applications. The model was also extended slightly in order to accommodate the rerun of transactions and the format of the distributed Read-Write-Validate protocol, in order to conduct a meaningful comparison. The model investigates the different performance characteristics of the proposed protocol versus FBOCC in a virtual execution environment. A range of results is presented which highlight the performance benefits of the distributed Read-Write-Validate protocol. The simulation model consists of a server, a client, and the broadcast disk structure. Only one client was used in the simulation, in order to provide a direct comparison with the existing work, which is built upon broadcast disk implementations where the read transaction is carried out entirely at the client (so that the number of clients is irrelevant), and where mobile update transactions are relatively rare. The server executes the server's transactions based on Read-Write-Validate algorithms. Deadline assignment, as explained elsewhere [78] is controlled by the minimum and maximum slack factor parameters that provide a lower and upper bound for a transaction's slack time. The deadline of transactions is calculated using the following formula [78]:

Deadline = AT + uniform (minimum slack factor, maximum slack factor) * ET

In the formula, AT and ET denote arrival and execution times respectively. Execution time is estimated using the values of transaction length, CPU time and disk access (mean inter-operation delay in mobile transactions). Table 2 shows the parameters

which were used during the simulation experiments. The time unit is in bit-time, which is the time to transmit a single bit. For a broadcast bandwidth of 64 kbps, 1 M bit-time is equivalent to approximately 15s.

Parameter	Value
Server	
Transaction length	8
Read operation probability	.5
Disk access time	1000
Transaction arrival rate	1 per 20000 to 1 per 1667
Number of database	300
Concurrency control protocol	Distributed Read-Write-Validate OCC
Priority scheduling	Earliest deadline first
Mobile clients	
Transaction length	4
Read operation probability	.5
Fraction of read only transactions	75 %
Minimum slack factor	2 (uniformly distributed)
Maximum slack factor	8 (uniformly distributed)
Mean inter-operation delay	65,536
Mean inter-transaction delay	131.072

Table 2. Simulation parameters in the
evaluation of the distributed Read-Write-Validate protocol

.

4.2.2 Simulation Results

This section presents the results of a series of experiments performed in order to benchmark the distributed Read-Write-Validate protocol. The first set of results presented in Figures 4.7-4.9 demonstrates the performance of transactions generated on the server side. The second set of results presented in Figures 4.10-4.11 demonstrates the performance of transactions generated on the client's side. In each graph, the results are presented of the two protocols: the distributed Read-Write-Validation protocol, abbreviated to DLVEW and the forward and backward optimistic concurrency control protocol termed to FBOCC.

Results set 1:

The first set of results show the throughput, average response time and late transaction rate of transactions generated and performed at the server, which are concatenated with the abbreviation S in Figures 4.7-4.9 below.

Figure 4.7 shows throughput for an increasing rate of transactions at the server. Throughput is defined as the number of committed transactions at the server, with the commit occurring at the end of the write phase for both phase orderings. All protocols share a common progression when levels of contention are low. Then throughput starts to degrade when contention reaches a level at about .2 * $10e^{-3}$ transaction per bit-time in the FBOCC approach. In contrast, the proposed approach is more efficient in handling the high contention of transactions at the server, and achieves further than 4 * $10e^{-3}$ transaction per bit-time. The numbers of late transactions shown in figure 4.9 affect throughput, as these transactions are aborted and will never commit. As the transaction rate increases, the number of late transactions increases and throughput drops.

Figure 4.8 shows the average response time for an increasing rate of transactions. The response time is only included for transactions that successfully commit. As transaction rate increases, the transaction response time increases due to high contention. It can be seen that, between 1 * (10^-4) and 6 * (10^-4) transactions per bit-time, the distributed Read-Write-Validate approach has a lower response time than FBOCC. This indicates the effect of the advantages gained by the proposed approach, as presented in chapter 3 which, including the advantage of offsetting the non-

deterministic period of the validation phase before the write phase in the proposed approach as well as eliminating the temporary blocking of concurrently running transactions when another transaction is validating. The response time stabilizes after 80000 bit-time due to deadline assignment, where only transactions that have a sufficiently large deadline will be able to commit. Regardless of the benefits of the proposed protocol, transaction at this level of contention, expire during the initial run in the read phase.

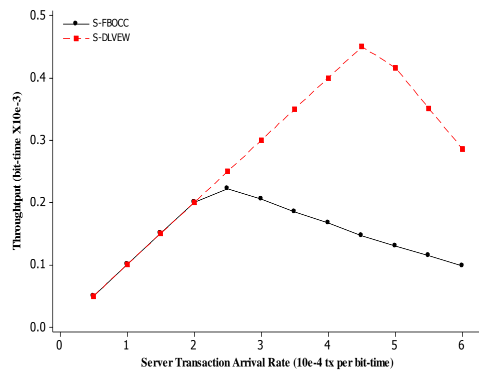


Figure 4.7 Throughput at the server

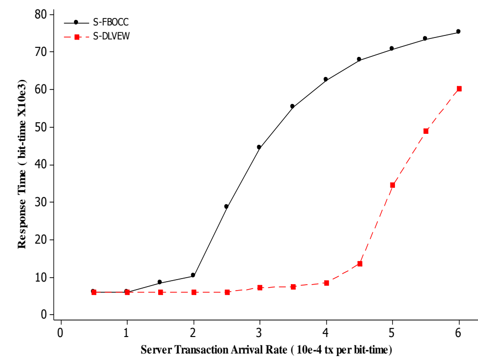


Figure 4.8 Response time at the server

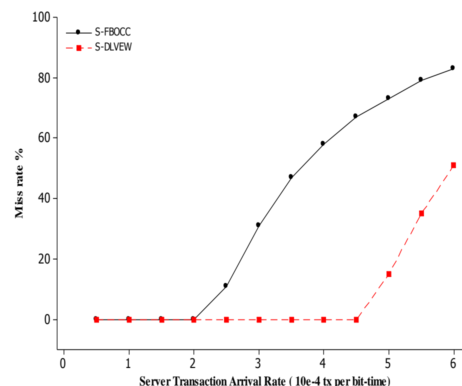


Figure 4.9 Late transactions at the server

Figure 4.9 shows the percentage of late transactions for an increasing rate of transactions. All protocols have no late transactions when the arrival rate of server transactions is low. Then as this rate increases, the percentage of late transactions also increases. Between 2 * (10^-4) and 6 * (10^-4) transactions per bit-time, the distributed Read-Write-Validate protocol has a lower miss rate than FBOCC, which indicates the efficiency of the proposed approach when dealing with high contention environments.

When the transaction arrival rate exceeds 6 *(10^-4) transactions per bit-time, transactions experience longer delays in accessing the disk and the CPU. This results in transactions being more likely to miss their deadlines during the read phase and never entering the validation and write phases.

The results in figures 4.7-4.9 show that adopting the proposed approach at the server can significantly improve server transaction performance, including a throughput increase and reductions in both response time and number of late transactions.

Results set 2:

The second set of results show the throughput and miss rate of mobile transactions generated at clients. Response time results were similar for both protocols, which is satisfactory given the real-time nature of the application domain where applications focus on measuring results of late transactions, which miss their deadlines. Throughput is another performance metric strongly connected to the rate of late transactions rate. In the following figures, mobile transactions are concatenated with the abbreviation MROT indicating mobile read-only transactions, and MUT to indicate mobile update transactions. These results are illustrated in Figures 4.10-4.13 below.

Results for Mobile Update Transactions (MUT)

Mobile update transactions are where the read phase is generated and executed on mobile devices. Then, they are transmitted to the server for global validation with other transactions on the server. The validation and write phases are performed at the server in order to maintain database consistency. Therefore, adopting the proposed approach on the server will directly advantage mobile update transactions during their write and validation phases. Such an assumption is justified by the results presented in Figures 4.10-4.11.

Figure 4.10 shows the throughput of mobile update transactions. All protocols share a common progression when contention at the server is low. Then the distributed Read-Write-Validate protocol demonstrates higher throughput whenever the server transaction arrival rate increases over $2 * 10e^{-4}$ transactions per bit-time. This indicates the advantage of executing the validation and write phases of mobile update transactions under the Read-Write-Validate approach at the server, indicating that the cost of the

validation phase does not affect the transactions commit time. The steady changes in overall trends result from the constant rate of the number of mobile transactions generated at the mobile device, which is specified in table 2. Only the transaction rate at the server increases.

Figure 4.11 shows the rate of late mobile update transactions. All protocols do have some late transactions even when the arrival rate of transactions at the server is low, which indicates the effect of the transmission delay between the clients and the server for those transactions with insufficient deadlines. However, the late transaction rate with the distributed Read-Write-Validate protocol is consistently lower than that with the FBOCC protocol at all levels of contention. This demonstrates the advantage of executing the validation and write phases of mobile update transactions with the Read-Write-Validate approach at the server. The constant difference shown in the graph related to the fact that the number of mobile transactions generated at the mobile device is constant, as stated in table 2. Only the transaction rate at the server increase.

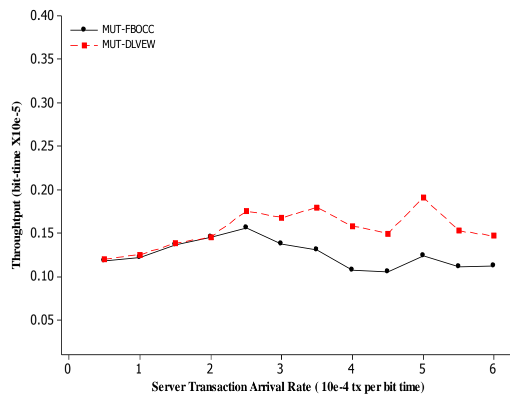


Figure 4.10 Throughput of update transactions at clients

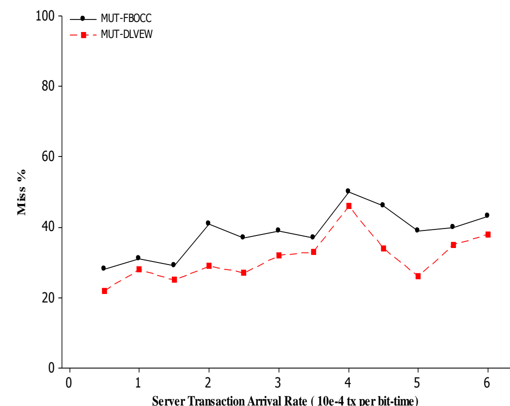


Figure 4.11 Late update transactions at clients

These results illustrated in Figures 4.7-4.9 show that adopting the proposed approach at the server will directly influence mobile update transactions by increasing throughput and reducing the rate of late transactions in the system. This means that the distributed Read-Write-Validate protocol is more appropriate for real-time mobile applications.

Results for Mobile Read-only Transactions (MROT)

Read-only transactions are those which do not update the database; in other words, they are transactions whose write set is empty and have no write phase. Such transactions do not affect database consistency. Therefore, mobile read-only transactions generate, execute, and commit locally on mobile devices without needing to be transmitted to the server for global validation. As a result, adopting the proposed approach on the server will not directly affect these transactions at the clients, as they are validated locally by the backward validation algorithm in both protocols. This is indicated by the results shown in Figures 4.12-4.13, which present the throughput and late transaction rates of mobile read-only transactions. Both figures demonstrate that both the distributed Read-Write-Validate protocol and the FBOCC protocol give similar results as expected. However, the read-only transactions in all protocols will be affected to some extend by the increasing rate of database updates at the server, since the new updates will be constantly broadcast and used for the validation of mobile read-only transactions at the clients. Therefore, as the frequency of database updates at the server increases, the conflict rate among read-only mobile transactions also increases, which consequently leads to increased number of transactions aborting. This explains the overall downward trend of throughput in figure 4.12 and the overall upward trend of late transactions in Figure 4.13. The steady changes in the trends result from the constant rate of the number of mobile transactions generated at the mobile device as specified in table 2. Only the transaction rate at the server is increased.

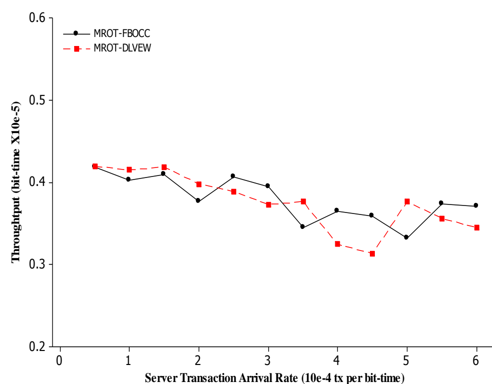


Figure 4.12 Throughput of read-only transactions at clients

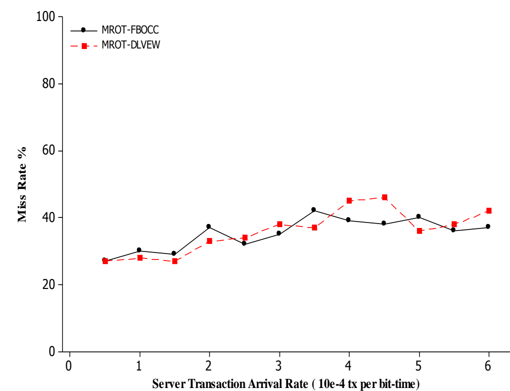


Figure 4.13 late read-only transactions at clients

4.3 Summary

This chapter describes a set of experiments including both centralised and distributed versions of the Read-Write-Validate protocols. These experiments were performed in order to benchmark the Read-Write-Validate approach proposed in this thesis compared to the conventional OCC protocol. The measurements of the performance of the protocols were taken as throughput, response time and the late transaction rate. The analysis of the results gathered from the simulation experiments can be summarised as follows:

- Significant improvements in throughput, response times and the timeliness of the overall system are achieved when the Read-Write-Validate approach is deployed to control access to shared data on mobile devices.

- Significant improvements in throughput, response time and the timeliness of the overall system are achieved at the server, without disrupting mobile transactions running at the clients, when the Read-Write-Validate approach is deployed in server-client models based on wireless broadcast environments.

- Observable improvements are found in the number of late mobile update transactions that miss their deadline due to concurrency issues, and there is a clear increase in mobile update transactions throughput as well when the Read-Write-Validate approach is deployed on client-server models based on wireless broadcast environments. In contrast, read-only mobile transactions running at mobile devices show a similar trend in both protocols regardless of the enhancements made on the server side.

Chapter 5

Conclusions and Future Work

5.1 Introduction

Transactions which are restarted due to being aborted after a conflict with another transaction must access the persistent store more frequently. Each restart represents a drain on time, resources and energy. Virtual execution allows the read phase of a conflicted transaction to complete, and stores the read data locally for reuse when the transaction is aborted. This thesis puts forward the argument that in a virtual environment it is fruitful to bring the write phase forward so that it occurs earlier than the validation phase. This approach, while seemingly counter-intuitive, improves both the throughput and the miss rate of the overall system at the server and clients. Further to this, the approach also improves the energy efficiency of the system, since more transactions meet their deadlines and fewer must be fully restarted since the read phase is often not repeated, and therefore the power consumed in repeating data access is reduced.

This idea has been explored in the context of multiple applications accessing a shared resource on a mobile device [17][16], and in the context of a client-server model based on the broadcast datacycle approach for wireless environments [18]. In both cases, a simulation of the technique is deployed and used to compare the results with those generated using more established FOCC and FBOCC algorithms, where the validate phase occurs entirely prior to the write phase. The results show that the proposed approach significantly improves throughput and the timeliness of transactions achieving their deadlines in the overall system when compared to the conventional approaches.

5.2 Contributions of the Thesis

This thesis introduces a novel Read-Write-Validate sequence of transactional phases combined with virtual execution to give a new OCC approach. The proposed approach is presented in two contexts:

- Firstly, it is shown that implementing it on the mobile devices themselves can improve issues of contention with shared resources on these devices [16][17].
- Secondly, it is further shown that it is an efficient implementation of a client-server model based on the broadcast datacycle approach for wireless environments [18].

5.2.1 Advantages Gained by the Present Research

The advantages given by the contributions made in this thesis are summarised below:

- *Transaction Lifespan Minimized*

The lifespan of a transaction is the time between it starting and committing. With the proposed approaches, the non-deterministic timing of the validation phase period is removed from the transactions lifespan. The validation phase of a transaction executes after the transaction commits. This is can be an important benefit in real-time systems where the validation phase introduces non-deterministic timing constraints that affect transactions satisfactorily meeting their deadline.

- *The Blocking of Concurrent Transaction is Eliminated*

In the proposed approaches, non-conflicted transactions no longer have to be blocked from progressing in order to guarantee database consistency. Concurrently running transactions are allowed to continue execution while the validating transaction executes in both validation and write phases. If concurrently running transactions enter into a state of conflict while the validating transaction is writing, such a conflict will eventually be detected in the deferred validation phase. If concurrently running transactions do not enter a conflict state while the validating transaction is writing, such transactions will successfully pass validation against the validating transaction, and will continue execution, gaining the benefit of not being temporarily blocked during the validating transaction’s write and validation phases.

- *Newly Starting Transactions are Never Blocked*

Newly starting transactions are those which may start their execution while another transaction is executing in the validation or write phases. With the proposed approaches newly starting transactions can continue execution straightaway without affecting database consistency.

- If newly starting transactions start execution while the validating transaction is running in the validation phase, which now is occurs after the write phase, then at this point the database will already be updated. Therefore, newly starting transactions will never enter a conflicted state, and no validation is required for such new transactions.

- If newly starting transactions start execution while the current validating transaction running in the write phase, the newly starting transactions simply continue their execution. If a newly starting transaction was a conflicting transaction, that is also not a problem because such a conflict will be detected later at the validation phase and the conflicted transaction will be aborted. If the newly starting transaction was a non-conflicting transaction, which usually constitute the majority of the contentious workload, it will continue execution and benefit from not being blocked during the validating transaction's write and validation phases.

- *Earlier Visible Updates*

In the proposed approach, writes become visible to concurrent transactions earlier, affording more likelihood of reading up-to-date data and thus reducing the risk of conflict. This is because the reordering of the validation and write phases guarantees that all new updates are already made before the validation phases start. This consequently reduces the risk of becoming conflicted with other concurrently running transactions, which benefits overall system performance.

- *Reduction of Conflict Risk*

Rerun transactions are quicker than those in their initial run, since there is no access to the persistent store, which makes transactions in rerun become ready to enter the critical

section for the write and validation phases in a shorter time. This reduces the risk of becoming conflicted with other concurrently running transactions and increases the chances of transactions committing.

- *Energy Efficiency Improvement*

Virtual execution improves the proposed approach because accessing a conventional hard disk drive is expensive in terms of power usage when the disk must attain read speed and the appropriate data sector to be found. Even solid-state drives are significantly more expensive to access compared to the local memory. Consequently, reducing the number of times that a disk is accessed will reduce the energy consumed. Clearly, a reduction in the frequency of transactions that must be restarted will reduce the number of times a disk is accessed, leading to a reduction in energy usage.

5.3 Future Work

The contribution of this thesis is to provide a novel departure from existing optimistic concurrency control techniques. It opens new doors for future research using this new optimistic concurrency control transactional structure. This section provides suggestions for interesting future research directions related to the contribution proposed in this thesis.

Energy Efficiency Evaluation

The rerun policy adopted in the proposed approach provides the advantage of minimizing persistent store access, which is expensive in terms of power usage. Clearly, a reduction in the frequency of transactions that must be rerun lead to a reduction in energy usage, which is very important in resource-constrained mobile devices [17]. Although this thesis concentrates on evaluating performance improvement, including throughput, response time and late transaction rate, future work should be dedicated to benchmarking the energy efficiency and battery utilization improvements gained by employing the proposed approach in real implementations.

Thick Client Applications

In traditional client-server models, clients have limited resources (thin clients). Their functionality is restricted to sending requests to the server, which is a powerful

computer providing the clients with services. In such models, clients will share the resources of the server. As the number of clients increases, the number of service requests sent to the server increases, which can lead to server bottlenecks. Developments in computing technology such as multi-core processors and memory offered at low cost leads to clients having more powerful hardware (thick clients). Thick clients are capable of providing rich functionality independent from the server, which reduces network latency by caching data at the clients [45]. Running multiple complex tasks in parallel on a thick client's devices raises issues associated with sharing resources such as processors, memory access, solid-state disk access and network connections. Therefore, a concurrency control technique is needed to take full advantage of the thick client's resources, which is another future research path in which, where the proposed approach can be explored.

Software Transactional Memory Considerations

Software transactional memory is a concurrency control mechanism for controlling concurrent access (read/write) to shared memory. Software transactional memory inherits similar properties from conventional database transactions. For instance, transactions in software transactional memory also preserve some ACID properties such as atomicity and isolation [164]. Data consistency needs to be maintained according to a correctness criterion such as serializability in a similar way as in conventional databases. Some types of software transactional memory are designed with a non-blocking property, which adopts the ordering of transaction in conventional optimistic concurrency control where transactions need to be validated before they are eligible to commit [165][164]. It would be well worth investigating the proposed approach with the new transactional phase order in non-blocking software transaction memory, in order to achieve further advances in the field.

References

[1] N. Conway, "Transactions and Data Stream Processing," in *CISC 499*, 2008, pp. 1–28.

[2] V. Kumar, *Performance of Concurrency Control Mechanisms in Centralized Database Systems*. Prentice Hall, New Jersey, 1996.

[3] A. N. Menascé, Daniel A., "Optimistic versus pessimistic concurrency control mechanisms in database management systems Information Systems," 1982, pp. 13–27.

[4] J. W. H. Garcia-Molina, J.U., *Database System Implementation.* Prentice-Hall., 2000.

[5] H. Morgan, K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," vol. 19, no. 11, 1976.

[6] J. Lee, "Concurrency Control Algorithms for Real-Time Database Systems," University of Virginia, 1994.

[7] a. Thomasian and E. Rahm, "A new distributed optimistic concurrency control method and a comparison of its performance with two-phase locking," *Proceedings.,10th Int. Conf. Distrib. Comput. Syst.*, pp. 294–301, 1990.

[8] H. T. Kung and J. T. Robinson, "On Optimistic Control Methods for Concurrency," vol. 6, no. 2, pp. 213–226, 1981.

[9] R. Agrawal, M. J. Carey, and M. Livny, "Concurrency control performance modeling: alternatives and implications," *ACM Trans. Database Syst.*, vol. 12, no. 4, pp. 609–654, Nov. 1987.

[10] J. Lee and S. H. Son, "Performance of Concurrency Control Algorithms for Real-Time Database Systems," *Univ. Virginia, United States*, 1994.

[11] J. R. Haritsa, M. J. Carey, and M. Livny, "Data access scheduling in firm real-time database systems," *Real-Time Syst.*, vol. 4, no. 3, pp. 203–241, Sep. 1992.

[12] M. J. C. and M. L. IHaritsa, Jayant R., "Dynamic real-time optimistic concurrency control," in *Real-Time Systems Symposium, 1990. Proceedings., 11th. IEEE*, 1990, pp. 94–103.

[13] M. L. Jayant H., M.C., "On Being Optimistic about Real-Time Constraints," in *in Proceedings of the 1990 ACM SIGACT-SIGART-SIGMOD Symposium on Principles of Database Systems (PODS)*, 1990, pp. 331–343.

[14] S. h. Lee, J. and Son, "Concurrency control algorhitms for real-time database systems," *Perform. Concurr. Control Mech. Cent. Database Syst. Prentice-Hall, Englewood Cliffs, NJ.*, 1996.

[15] A. Franaszek, P.A., Robinson, J.T., Thomasian, "Access Invariance and Its Use in High Contention Environments," in *proceedings of the 6th International conference on Data Engineering*, 1990, pp. 47–55.

[16] K. Solaiman and G. Morgan, "Later Validation/Earlier Write: Concurrency Control for Resource-Constrained Systems with Real-Time Properties," *2011 IEEE 30th Symp. Reliab. Distrib. Syst. Work.*, pp. 9–12, Oct. 2011.

[17] K. Solaiman and G. M. M. Brook , G. Ushaw, "A Read-Write-Validate Approach to Optimistic Concurrency Control for Energy Efficiency of Resource-Constrained Systems," in *Wireless Communications and Mobile Computing Conference (IWCMC) 9th International. IEEE*, 2013, no. Cc, pp. 1424–1429.

[18] K. Solaiman and G. morgan , M. Brook, G. Ushaw, "Optimistic Concurrency Control for Energy Efficiency in the Wireless Environment," in *in The 13th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP) springer*, 2013, pp. 115–128.

[19] N. Bernstein, P.A., Hadzilacos, V., Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[20] C. H. Papadimitriou, "The serializability of concurrent database updates," *J. ACM*, vol. 26, no. 4, pp. 631–653, Oct. 1979.

[21] and P. V. Tamer èozsu, M., *Principles of distributed database systems. Springer*. 1999.

[22] and T. K. Coulouris, George, Jean Dollimore, *Distributed Systems: Concepts and Design Edition 3*. 2001.

[23] and K. S. Kuramitsu, Kimio, “Towards ubiquitous database in mobile commerce,” in *In Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access, ACM*, 2001, pp. 84–89.

[24] S. Ortiz, “Embedded databases come out of hiding,” in *Computer 33, no. 3*, 2000, pp. 16–19.

[25] and T. N. R. Selvarani, D. Roselin, “A SURVEY ON DATA AND TRANSACTION MANAGEMENT IN MOBILE DATABASES,” *Int. J. Database Manag. Syst. 4.5*, 2012.

[26] et al. Whang, Kyu-Young, “The ubiquitous DBMS,” in *ACM SIGMOD Record 38.4*, 2010, pp. 14–22.

[27] A. Nori, “Mobile and embedded databases,” in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM*, 2007, pp. 1175–1177.

[28] G. S. Welling, “Designing adaptive environment-aware applications for mobile computing,” Diss. Rutgers, The State University of New Jersey, 1999.

[29] J. J. Liu, “Mobile map: A case study in the design & implementation of a mobile application,” Diss. Carleton University, 2002.

[30] V. F. W. DiPippo, Lisa Congiser, “Real-time databases,” *Database Syst. Handbook, Multiscience Press*, pp. 1–57, 1997.

[31] S. A. Aldarmi, “Real-Time Database Systems : Concepts and Design,” 1998.

[32] J. L. and K. Raatikainen, “Optimistic Concurrency Control Methods for Real-Time Database Systems,” Helsinki, 2001.

[33] R. Abbott and H. Garcia-Molina, “Scheduling real-time transactions,” *ACM SIGMOD Rec.*, vol. 17, no. 1, pp. 71–81, Mar. 1988.

[34] J. U. and J. W. H. Garcia-Molina, *DATABASE SYSTEM The Complete Book*. 2009.

[35] M. J. Haritsa, J.R., Livny, M., Carey, “Earliest Deadline Scheduling for Real-Time Database Systems,” in *In proceedings of the 12th Real-Time System Symposium*, 1991, pp. 232–242.

[36] R. K. Abbott and H. Garcia-Molina, “Scheduling real-time transactions: a performance evaluation,” *ACM Trans. Database Syst.*, vol. 17, no. 3, pp. 513–560, Sep. 1992.

[37] M. L. J. R. Haritsa, M. J. Carey, “Value-Based Scheduling in Real-Time Database Systems,” *VLDB J.*, vol. 2, no. 2, p. Number 117–152, 1993.

[38] W. . Liu, C., L ., Layland, J., “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment,” *J. ACM*, vol. 20, pp. 46–61, 1973.

[39] K. V. Datta A., S.H.S., “Limitations of priority cognizance in conflict resolution for firm real-time database systems,” in *IEEE Transaction on Computers*, 2000, pp. 483–501.

[40] Z. Qin, Y. Wang, D. Liu, and Z. Shao, “Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems,” *2012 IEEE 18th Real Time Embed. Technol. Appl. Symp.*, pp. 35–44, Apr. 2012.

[41] M. J. Franklin, *Client data caching: A foundation for high performance object database systems*. Kluwer Academic Publishers, 1996.

[42] and G. G. Silberschatz, Abraham, Peter B. Galvin, *Operating system concepts*. J. Wiley & Sons, 2009.

[43] and L. B. Podlipnig, Stefan, “A survey of web cache replacement strategies,” *ACM Comput. Surv. 35.4*, pp. 374–398, 2003.

[44] B. Krishnamurthy and and J. Rexford, *Web protocols and practice: HTTP/1.1, Networking protocols, caching, and traffic measurement*. Vol. 108. Reading: Addison-Wesley, 2001.

[45] F. Bukhari, “Maintaining Consistency in Client-Server Database Systems with Client-Side Caching,” Newcastle University Newcastle upon Tyne, UK, 2012.

[46] M. J. C. Franklin, Michael J., “Client-server caching revisited,” 1992.

[47] Y. Wang and L. A. Rowe, “Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture,” in *ACM SIGMOD int. Conference on management of Data*, 1991, pp. 367–376.

[48] M. J. Carey and E. Al, “Data caching tradeoffs in client-server DBMS architectures,” in . *Vol. 20. No. 2. ACM,*, 1991, pp. 357–366.

[49] and M.-A. N. Wilkinson, Kevin, “Maintaining consistency of client-cached data,” in *in VLDB*, 1990, pp. 122–133.

[50] M. L. Franklin, Michael J., Michael J. Carey, “Transactional client-server cache consistency: alternatives and performance,” *ACM Trans. Database Syst. 22.3*, pp. 315–363, 1997.

[51] T. A. Franaszek P., R.J., “Concurrency control for high contention environments.,” *ACM Trans. Database Syst.*, vol. 17, no. 2, pp. 304–345, 1992.

[52] and S. S. L. u, Philip S., Daniel M. Dias, “On the Analytical Modeling of Database Concurrency Control,” *J. ACM 40.4*, vol. 40, no. 4, pp. 831–872, 1993.

[53] P. S. Yu and D. M. Dias, “Analysis of hybrid concurrency control schemes for a high data contention environment,” *IEEE Trans. Softw. Eng.*, vol. 18, no. 2, pp. 118–129, 1992.

[54] D. Yu s. P., Dias, M., “Performance analysis of optimistic concurrency control schemes with different rerun policies,” in *Proceedings of the Fifteenth Annual International*, 1991, pp. 294 – 300.

[55] A. Thomasian and S. Member, “Control Methods for High-Performance Transaction Processing,” vol. 10, no. 1, pp. 173–189, 1998.

[56] H. Theo, “Observation on Optimistic Concurrency Control Schemes,” *Inf. Syst.*, vol. 9, no. 2, pp. 111–120, 1984.

[57] R. Unland, “Optimistic concurrency control revisited,” *Arbeitsberichte des Instituts für Wirtschaftsinformatik, Westfälische Wilhelms-Universität Münster, No. 30*, 1994.

[58] J. Huang and E. Al., “Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes,” in *VLDB. Vol. 91*, 1991.

[59] S. Park and S. Jung, “An energy-efficient mobile transaction processing method using random back-off in wireless broadcast environments,” *J. Syst. Softw.*, vol. 82, no. 12, pp. 2012–2022, Dec. 2009.

[60] Y. Lei, X., Zhao, Y., Chen, S., “Concurrency control in mobile distributed real-time database systems,” *Parallel Distrib. Comput.*, vol. 69(10), pp. 866–876, 2009.

[61] W. P. and Y. K. M. Choi, “Two-phase Mobile Transaction Validation in Wireless Broadcast Environments,” in *In Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, 2009, pp. 32–38.

[62] and K.-W. L. Lee, Victor CS and K. Lam, “Optimistic Concurrency Control in Broadcast Environments : Looking Forward at the Server and Backward at the Clients,” *Mob. Data Access. Springer Berlin Heidelb.*, pp. 97–106, 1999.

[63] S. Jung and K. Choi, “A concurrency control scheme for mobile transactions in broadcast disk environments,” *Data Knowl. Eng.*, vol. 68, no. 10, pp. 926–945, Oct. 2009.

[64] J. J. Lee, “Precise serialization for optimistic concurrency control,” *Data Knowl. Eng. Elsevier Sci. B.V.*, vol. 29, no. 2, pp. 163–179, Feb. 1999.

[65] V. C. S. Lee and K. Lam, "Conflict free transaction scheduling using serialization graph for real-time databases," vol. 55, pp. 57–65, 2000.

[66] K. Marzullo, "Concurrency Control for Transactions with Priorities TR 89-996," *Dep. Comput. Sci. Cornell Univ.*, 1989.

[67] H. Berenson, E. O. Neil, P. O. Neil, M. Corp, P. Bernstein, J. Melton, and S. Corp, "A Critique of ANSI SQL Isolation Levels," no. June, pp. 1–10, 1995.

[68] M. Yabandeh and D. Gómez Ferro, "A critique of snapshot isolation," *Proc. 7th ACM Eur. Conf. Comput. Syst. - EuroSys '12*, p. 155, 2012.

[69] T. Riegel, H. Sturzrehm, P. Felber, and C. Fetzer, "From Causal to z - Linearizable Transactional Memory," 2007.

[70] P. L. DANG De-Peng, LIU Yun-Sheng, "Weak Serializable Concurrency Control in Distributed Real-Time Database Systems," vol. 6, no. 4, 2002.

[71] M. a. Bornea, O. Hodson, S. Elnikety, and A. Fekete, "One-copy serializability with snapshot isolation under the hood," *2011 IEEE 27th Int. Conf. Data Eng.*, pp. 625–636, Apr. 2011.

[72] K. Lam and W. Yau, "On using similarity for concurrency control in real-time database systems," *J. Syst. Softw.*, vol. 43, no. 3, pp. 223–232, Nov. 1998.

[73] J. Lindstr, "Relaxed Correctness for Firm Real-Time Databases," in *Embedded and Real-Time Computing Systems and Applications, 2006. Proceedings. 12th IEEE International Conference on. IEEE*, 2006, pp. 82–86.

[74] S. Revilak, P. O'Neil, and E. O'Neil, "Precisely Serializable Snapshot Isolation (PSSI)," *2011 IEEE 27th Int. Conf. Data Eng.*, pp. 482–493, Apr. 2011.

[75] P. S. Yu and S. H. Son, "On real-time databases: concurrency control and scheduling," *Proc. IEEE*, vol. 82, no. 1, pp. 140–157, 1994.

[76] R. K. Lindström J., "Dynamic adjustment of serialization order using timestamp intervals in real-time databases.," in *in In Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications,IEEE Computer Society Press*, 1999.

[77] K. R. and D. T. Jiandong H., J.S., "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes.," in *Proc. 17th Conf. Very Large Databases*, 1991, pp. 35–46.

[78] J. Lee and S. H. Son, "Using dynamic adjustment of serialization order for real-time database systems," *1993 Proc. Real-Time Syst. Symp.*, pp. 66–75, 1993.

[79] R. K. Lindstrom J., "Using Importance of Transactions and Optimistic Concurrency Control in Firm Real-Time Databases," in *in Proc. 7th International Conference on Real-Time Systems and Applications (RTCSA '00)*, 2000, pp. 12–14.

[80] J. Lindstrom, "Optimistic Concurrency Control Methods for Real-Time Database Systems," in *S.o.P.A.R. A-2003-1, Editor.University of Helsinki Department of Computer Science.*, 2001.

[81] A. Peinl, P., Reuter, "Empirical comparison of database concurrency control schemes," in *in Proceedings of the 9th Znternutionul Conference on Very Large Data Bases.*, 1983, pp. 97–108.

[82] R. Pradel, U., Schlageter, G. , Unland, "Redesign of optimistic methods: Improving performance and availability," in *in In Proceedings of the 2nd International Conference on Data Engineering IEEE Computer Society Press*, 1986.

[83] E. R. a. A. Thomasian., "A New Distributed Optimistic Concurrency Control Method and a Comparison of its Performance with Two-Phase Locking.," in *in In Proceedings of Tenth ICDCS.*, 1990.

[84] K. R. and D. T. Jiandong H., J.S., "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes.," in *Proc. 17th Conf. Very Large Databases*, 1991, pp. 35–46.

[85] J. Huang and J. A. Stankovic, "Concurrency Control in Real-Time Database Systems: Optimistic Scheme vs. Two-Phase Locking," 1990.

[86] N. H. Mamun Q. E. K., "Timestamp based optimistic concurrency control.," in *in TENCON2005,*, 2005, pp. 1–5.

[87] T. A., "Checkpointing for optimistic concurrency control methods.," in *Knowledge and Data Engineering, IEEE Transactions on*, 1995, pp. 332–339.

[88] J. G. and A. Reuter, *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers,, 1992.

[89] G. Garcia-Molina, H., Wiederhold, "Read-only transactions in a distributed database.," in *ACM Transactions on Database Systems*, 1982, pp. 209–234.

[90] H.-Y. WANG1 and 3 AND KAM-YIU LAM4, "Mobile Real-Time Read-Only Transaction Processing in Broadcast Disks," vol. 1264, pp. 1249–1264, 2006.

[91] B. Lu, Q. Zou, and W. Perrizo, "A dual copy method for transaction separation with multiversion control for read-only transactions," *Proc. 2001 ACM Symp. Appl. Comput. - SAC '01*, pp. 290–294, 2001.

[92] A. SangKeun Lee, Chong-Sun Hwang and M. Kitsuregawa, "Concise PapersUsing Predeclaration for Efficient Read-Only Transaction Processing in Wireless Data Broadcast," vol. 15, no. 6, pp. 1579–1583, 2003.

[93] S. H. Kwok-Wa L., K.-y.L., "Real-time optimistic concurrency control protocol with dynamic adjustment of serialization order," in *in on Proc of IEEE Real-Time Technology and Application Syniposium*, 1995, pp. 174–179.

[94] P. Pii, G. Britain, C. Engineering, A. Buchmann, and I. Science, "A REAL-TIME CONCURRENCY CONTROL PROTOCOL FOR MAIN-MEMORY DATABASE SYSTEMS +," vol. 23, no. 2, pp. 109–125, 1998.

[95] J. and R. R. Larus, *Transactional Memory.* MORGAN & CLAYPPOOL PUBLISHERS USA, 2007.

[96] H. F. K. Silberschatz, Abraham and and S. Sudarshan, *Database system concepts*, Vol. 4. Hightstown: McGraw-Hill, 1997.

[97] D. Potier and and P. Leblanc., "Analysis of locking policies in database management systems," *Commun. ACM 23.10*, pp. 584–593, 1980.

[98] T. A. Ryu I., "Performance analysis of centralized databases with optimistic concurrency control.," *Elsevier Sci. Publ. B. V.*, vol. 7, no. 3, 1987.

[99] T. Alexander, "Analysis of some optimistic concurrency control schemes based on certification.," in *in In Proceedings of the 1985 SIGMETRICS Conference on Measurement and Modeling of Computer Systems.*, 1985.

[100] T. Johnson, "Analysis of Optimistic Concurrency Contro Revisited.," 1992.

[101] B. Azer, "Speculative Concurrency Control: A position statement," 1992.

[102] B. S. Bestavros A., "SCC-nS: a Family of Speculative Concurrency Control Algorithms for Real-Time Databases," in *in Proc. Third Int'l Workshop Responsive Computer Systems.*, 1993.

[103] B. S. Bestavros A., "Speculative Concurrency Control," in *Computer Science Department Boston University*, 1993.

[104] B. S. Bestavros A., "Time liness via speculation for real-time databasees," in *in In Proceedings of RTSS'94: The 14th IEE Real-Time System Symposium.*, 1994.

[105] B. S. Bestavros A., “Value-cognizant speculative concurrency control,” in *In Proceedings of VLDB 95 : The International Conference on Very Large Databases*, 1995.

[106] B. S. Bestavros A., “Value-cognizant speculative concurrency control for real-time databases.,” *Inf. Syst.*, pp. 21(1):75–101, 1996.

[107] Y. S. G. Jun C. Yan_Li Z., “A New Speculative Concurrency Control Protocol and The Analysis base on Petri Net,” in *in Computer Engineering and Applications*, 2009, pp. 121–123.

[108] J. W. Juna C., Y.f.W., “Concurrency Control Protocol for Real-Time Database and The Analysis Based on Petri Net.,” *Adv. Mater. Res.*, pp. 12–17, 2011.

[109] E. P. Azer B., S.B., “Performance Evaluation of Two-Shadow Speculative Concurrency Control,” 1993.

[110] B. S. and L. A. Haubert, J., “Improving the SCC protocol for real-time transaction concurrency control. in Signal Processing and Information Technology,” in *ISSPIT 2003. Proceedings of the 3rd IEEE International Symposium on*, 2003.

[111] M. C. M. L. Jayant H., “Dynamic Real-Time Optimistic Concurrency Control,” in *in Proceedings of the 11th IEEE Real-Time Systems*, 1990, pp. 94–103.

[112] S. H. S. Lee J., “Performance of concurrency control algorithms for real-time database systems,, K. Vijay, Editor. Prentice-Hall, Inc.,” *Perform. Concurr. Control Mech. Cent. database Syst.*, pp. 429–460, 1996.

[113] M. C. M. L. Jayant H., “Dynamic Real-Time Optimistic Concurrency Control,” in *in Proceedings of the 11th IEEE Real-Time Systems.*, 1990, pp. 94–103.

[114] E. D. Jensen and C. Douglass, “A Time-Driven Scheduling Model for Real-Time Operating Systems,” pp. 112–122, 1985.

[115] J. R. Haritsa, M. J. Carey, and M. Livny, “On being optimistic about real-time constraints,” *Proc. ninth ACM SIGACT-SIGMOD-SIGART Symp. Princ. database Syst. ACM*, pp. 331–343, 1990.

[116] J. Lee, S. H. Son, and C. Science, “AN OPTIMISTIC CONCURRENCY CONTROL PROTOCOL FOR REAL-TIME DATABASE SYSTEMS,” in *In Pro- ceedings of 3rd International Symposium on Database Systems forAdvancedApplications, Daejon, Korea*, 1993, pp. 387–394.

[117] Acharya, “Broadcast Disks: Dissemination-Based Data Management for Asymmetric Communication Environments, PhD dissertation,” Brown Univesity, 1997.

[118] and E. T. Finne, Arild, “Data Management and Concurrency Control in Broadcast based Asymmetric Environments,” 2005.

[119] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, “Broadcast Disks : Data Management for Asymmetric Communication Environments,” no. May, 1995.

[120] A. Herman, G., Gopel, G., Lee, K.C., Weinrib, “The datacycle architecture for very high throughput database systems,” in *Proceedings of the ACM SIGMOD Conference*, 1987, pp. 97–103.

[121] S. Acharya, M. Franklin, and S. Zdonik, “Balancing Push and Pull for Data Broadcast,” pp. 183–194, 1997.

[122] S. A. K. Chitra Manikandan, “Time Stamping Method for Consistent Data Dissemination to Read Write Mobile Clients,” in *International Conference on Computer Communication and Informatics (ICCCI -2012)*, 2012, pp. 10 – 12.

[123] C. L. L.-F. Lin, C.-C. Chen, “Benefit-oriented data retrieval in data broadcast environments,” *Wirel. Networks 16*, pp. 1–15, 2010.

[124] A. S. P. C. K. Liaskos, S. G. Petridou, G. I. Papadimitriou, P. Nicopolitidis, “On the analytical performance optimization of wireless data broadcasting,” *Veh. Technol. IEEE Trans. 59*, pp. 884–895, 2010.

[125] H. S. S.-Y. Yi, “A hybrid scheduling scheme for data broadcast over a single channel in mobile environments,” *J. Intell. Manuf. 23*, pp. 1259–1269, 2012.

[126] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham, “Efficient concurrency control for broadcast environments,” *Proc. 1999 ACM SIGMOD Int. Conf. Manag. data - SIGMOD ’99*, pp. 85–96, 1999.

[127] S. V. and B. R. B. Imielinski, Tomasz, “Energy efficient indexing on air,” in *ACM SIGMOD Record. Vol. 23. No. 2. ACM*, 1994, pp. 25–36.

[128] L. L. M. Chehadeh, Y. C., Ali R. Hurson, “Energy-efficient indexing on a broadcast channel in a mobile database access system,” in *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on. IEEE*, 2000, pp. 368–374.

[129] V. Goel, “Energy Efficient Air Indexing Schemes for single and Multi-level Wireless channels,” pp. 525–530, 2012.

[130] X. Lu and E. Al., “SETMES: a scalable and efficient tree-based mechanical scheme for multi-channel wireless data broadcast,” in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication. ACM*, 2013.

[131] J. Zhong and E. Al., “Evaluation and comparison of various indexing schemes in single-channel broadcast communication environment,” in *Knowledge and Information Systems*, 2013, pp. 1–35.

[132] S. and J. C. Im, “MLAIN: Multi-leveled air indexing scheme in non-flat wireless data broadcast for efficient window query processing,” *Comput. Math. with Appl. 64.5*, pp. 1242–1251, 2012.

[133] et al. Sun, Weiwei, “An automaton-based index scheme for on-demand XML data broadcast,” *Database Syst. Adv. Appl. Springer*, pp. 96–110, 2012.

[134] J. Zhong and E. Al., “Multi-channel energy-efficient hash scheme broadcasting,” in *Proceedings of the 21st international conference on software engineering and data engineering*, 2012.

[135] S. Wang and H.-L. Chen, “Tmbt: An efficient index allocation method for multi-channel data broadcast,” *AINA*, pp. 236–242, 2007.

[136] Q. G. and S. L. J. Xu, W.-C. Lee, X. Tang, “An error-resilient and tunable distributed indexing scheme for wireless data broadcast,” *TKDE, 18(3)*, pp. 392–404, 2006.

[137] E.-P. L. and A. S. Y. Yao, X. Tang, “An energy-efficient and access latency optimized indexing scheme for wireless data broadcast,” *TKDE,18(8)*, pp. 1111–1124, 2006.

[138] B. S. and D. T. A. Waluyo, “Global indexing scheme for location-dependent queries in multi channels mobile broadcast environment,” *AINA*, pp. 1011–1016, 2005.

[139] B. L. and S. P. S. Jung, “A tree-structured index allocation method with replication over multiple broadcast channels in wireless environments,” *TKDE,17(3)*, pp. 311–325, 2005.

[140] R. M. D. Sumari, Putra and and A. R. Rahiman, “A Broadcast Disk scheme for mobile information system,” *J. Comput. Sci. Technol. 10*, 2010.

[141] A. S. Tanenbaum, *Computer Networks*. 1987.

[142] K.-L. Tan and and B. C. Ooi, “On selective tuning in unreliable wireless channels,” *Data Knowl. Eng.*, pp. 209–231, 1998.

[143] A. Bestavros, "AIDA-based real-time fault-tolerant broadcast disks," in *Real-Time Technology and Applications Symposium 1996. Proceedings., 1996 IEEE*, 1996, pp. 49–58.

[144] A. B. Baruah, Sanjoy, "Pinwheel scheduling for fault-tolerant broadcast disks in real-time database systems," in *Data Engineering, 1997. Proceedings. 13th International Conference on. IEEE*, 1997, p. Baruah, Sanjoy, and Azer Bestavros. "Pinwheel sche.

[145] W. et al. Hu, "An on-demand data broadcasting scheduling algorithm based on dynamic index strategy," in *Wireless Communications and Mobile Computing*, 2013.

[146] A. B. Baruah, Sanjoy, "Timely and fault-tolerant data access from broadcast disks: A pinwheel-based approach," in *Proceedings of the workshop on on Databases: active and real-time. ACM*, 1996, pp. 45–49.

[147] V. C. S. Lee, K. Wa Lam, and T.-W. Kuo, "Efficient validation of mobile transactions in wireless environments," *J. Syst. Softw.*, vol. 69, no. 1–2, pp. 183–193, Jan. 2004.

[148] V. C. S. L. and K.-W. Lam and ., "Optimistic Concurrency Control in Broadcast Environments: Looking Forward at the Server and Backward at the Clients," in *in Proceedings of International Conference on Mobile Data Access, Lecture Note in Computer Science*, 1999, pp. 97–106.

[149] A. Bowen, T.F., Gopal, G., Herman, G., Hickey, T., Lee, K.C., Mansfield, W.H., Raitz, J., Weinrib, "The datacycle architecture," in *Communications of the ACM*, 1992, pp. 71–81.

[150] and S. H. S. Lee, Victor CS, Kwok-Wa Lam, "Concurrency control using timestamp ordering in broadcast environments." The Computer Journal," vol. 45.4, pp. 410–422, 2002.

[151] and Z. S. Siau, Keng, "Building customer trust in mobile commerce," *Commun. ACM 46.4*, pp. 91–94, 2003.

[152] and B.-S. J. Choi, Ho-Jin, "A timestamp-based optimistic concurrency control for handling mobile transactions," *Comput. Sci. Its Appl. Springer Berlin Heidelb.*, pp. 796–805, 2006.

[153] et al. Lam, Kam-Yiu, "Concurrency control strategies for ordered data broadcast in mobile computing systems," *Inf. Syst. 29.3*, pp. 207–234, 2004.

[154] A.-P. Bosch, G.; Creus; Tuovinen, “Feature Interaction Control on Smartphones,” in *Industrial Embedded Systems, 2007. SIES ’07. International Symposium*, 2007, pp. 302–309.

[155] et al. Lei, Xiangdong, “Concurrency control in mobile distributed real-time database systems,” *J. Parallel Distrib. Comput. 69.10*, pp. 866–876, 2009.

[156] L. Guohui, Y. Bing, and C. Jixiong, “Efficient Optimistic Concurrency Control for Mobile Real-Time Transactions in a Wireless Data Broadcast Environment,” *11th IEEE Int. Conf. Embed. Real-Time Comput. Syst. Appl.*, pp. 443–446, 2005.

[157] E. Pitoura, “Supporting read-only transactions in wireless broadcasting.,” in *Proceedings of the DEXA98 International Workshop on Mobility in Databases and Distributed Systems*, 1998, pp. 428–433.

[158] P. K. Pitoura, E., Chrysanthis, “Scalable processing of readonly transactions in broadcast push.,” in *Proceedings of the 19th IEEE International Conference on Distributed Computing System*, 1999.

[159] D. Barbara, “Certification reports: supporting transactions in wireless systems.,” in *in Proceedings of 17th International Conference on Distributed Computing Systems*, 1997, pp. 466–473.

[160] D. Yu s. P., Dias, M., “Analysis of hybrid concurrency control schemes for a high data contention environment,” in *in IEEE Trans. Software Eng*, 1992, pp. 118–129.

[161] F. Howell and and R. McNab., “"SimJava: A discrete event simulation library for java.,” *Simul. Ser. 30*, pp. 51–56, 1998.

[162] F. and R. M. Howell, “SimJava.,” *Institute for Computing Systems Architecture, Division of Informatics, University of Edinburgh*, 2000. .

[163] S. Manegold, “Understanding , Modeling , and Improving Main-Memory Database Performance,” CWI Amsterdam, 2002.

[164] and R. R. Harris, Tim, James Larus, *Transactional memory." Synthesis Lectures on Computer Architecture 5.1*. 2010, pp. 1–263.

[165] and W. N. S. I. Herlihy, Maurice, Victor Luchangco, Mark Moir, “Software transactional memory for dynamic-sized data structures,” in *In Proceedings of the twenty-second annual symposium on Principles of distributed computing*, 2003, pp. 92–101.

[166] www.virtualmv.com/wiki/index.php?title=DBMS:Centralised_vs_Distributed