PERFORMANCE ANALYSIS OF MIXTURES OF FIXED AND MOBILE TRANSACTIONS OVER WIRELESS COMPUTING ENVIRONMENTS

By

AHMAD H. ALQEREM

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy in Computer Science

Loughborough University

Acknowledgment

I am truly fortunate to have received the love and support of so many friends and colleagues throughout my graduate career and it would be impossible to properly acknowledge them all. The Computer Science Department at Loughborough University is an exciting, rewarding place to study. Many thanks to the faculty, graduate students and staff who make it so.

It has been a great privilege for me to work with Dr Walter Hussak, an exceptional researcher and teacher, who introduced me to concurrency control theory as well as database systems and mobile computing; He has been extraordinarily patient and supportive, having been always available for discussion and responding speedily to research reports. I would like to take this opportunity to thank him for his continued encouragement and guidance throughout the course of my research. My sincere thanks go to Dr Helmet Bez for his constructive comments and suggestions.

I am grateful to my parents and family, especially to my father hussien who built in me the resolve to fight it out, and to my mother fattom who was always there when I needed her. Their relentless encouragement was perhaps the single most influential factor in my education since childhood. And thanks to my brothers (wajeah, nazeah, abu hanean, belal and tareq) and sisters (rokaya, asma and kawlah) for everything they had to offer.

Finally, I am grateful for the invaluable love, understanding and support shown by my wife Manal and my son Hadi during my graduate life.

Abstract

As technological advances are made in software and hardware, the feasibility of accessing information "any time, anywhere" is becoming a reality. In a mobile computing environment, a potentially large number of mobile and fixed users may simultaneously access shared data; therefore, there is a need to provide a means to allow concurrent management of transactions. Specific characteristics of mobile environments make traditional transaction management techniques no longer appropriate. This is due to a fact that the ACID properties of transactions are not simply followed, in particular the atomicity property. Thus, transaction management models adopting weaker forms of atomicity are needed.

In the first part of this thesis, a performance evaluation of three common execution strategies for mobile transactions, the mobile host strategy (MHS), the fixed host strategy (FHS), and the combined host strategy (CHS), is conducted. A MHS strategy determines that the execution of mobile transactions always take place at the mobile host, a FHS strategy determines that the execution of mobile transactions always take place at the fixed host, and a CHS strategy provides the flexibility to execute mobile transactions at both hosts. The significant contribution of the evaluation is that the effects of the presence of fixed host transactions are identified and included in the evaluation. To do so an execution framework for mobile transactions is proposed. The main underlying feature of the framework is the relaxation of the basic atomicity property for transactions. This is achieved by defining a mobile transaction as a set of subtransactions where each subtransaction consists, in turn, of basic and complementary subtransactions. The execution place of basic and complementary subtransactions is based on the execution strategy in operation. As wired and wireless environments are integrated, the choice of execution strategy is critical for the performance of the system. Our results show that neither a MHS nor a FHS are optimal in all situations and the wasted wireless resources can be substantial. A combined strategy CHS at least matches the best performance of the FHS and MHS and shows better performance than both in many cases.

In the second part of this thesis we extend the two fundamental approaches of locking-based and optimistic concurrency control for standard concurrent transaction environments, to the case of environments containing mixtures of standard (fixed host) and mobile transactions. Recent other studies have suggested that the optimistic concurrency control (OCC) protocols outperform the locking-based protocols in mobile database systems (MDBS). However, the OCC protocols suffer from the problem of multiple transaction restarts. The restarting problem is more intensified in mixed transactions environments where both fixed and mobile transactions coexist in the system. We propose effective concurrency control approaches for such environments to address this problem. Two approaches namely Lock-Mix and OCC-Mix are given - the former is a lock based approach which utilizes features of both the OCC and two-phase locking protocols, whereas the latter is an optimistic-based approach which combines optimistic and timestamp protocols. The main objectives of these approaches is to overcome the limitations of the wireless environment by avoiding restarts and blocking of mobile transactions by other fixed or mobile transactions while still providing the opportunity for the fixed transactions to finish their execution.

The characteristics of the Lock-Mix and OCC-Mix approaches are examined in detail. A simulator is built and experiments are conducted. The results show that the performance of these approaches is consistently better than using a traditional concurrency protocols over a wide range of system settings. In particular, these approaches provide a more significant performance gain in mixed transaction environments.

Table of contents

List of Figures

List of Tables

LIST OF ACRONYMS

ACID	Atomicity, Consistency, Isolation, And Durability.
FT	Fixed Transaction
MT	Mobile Transaction
MDB	Mobile Database System
MH	Mobile Host
FH	Fixed Host
GDB	Global Database
GT	Global Transaction
MST	Mobile Sub Transaction
JT	Joey Transaction
KM	Kangaroo Model
KT	Kangaroo Transaction
LLT	Long-Lived Transactions
LTM	Local Transaction Manger
MH	Mobile Host
MSS	Mobile Support Station
BSC	Base Station Controller
MSC	Mobile Switching Centre
MTM	Mobile Transaction Manager
MU	Mobile Unit
TM	Transaction Manager
OCC	Optimistic Concurrency Control
2PL	Two Phase Locking
DAA	Data Access Agent
PCR	Power Consumption Ratio
FHS	Fixed Host Execution Strategy
MHS	Mobile Host Execution Strategy
CHS	Combined Host Execution Strategy
Treq	Transaction Request
Dreq	Data Request
CC	Concurrency Control
SGs	Serialization Graphs
TSO	Time Stamp Ordering
MGL	Multi-Granularity Locking
DM	Data Manager
SGT	Serialization Graph Testing
DFV	Dynamic Finite Versioning
TSH	Time Stamp History.
TL	Transaction Length

CHAPTER 1

Introduction

1.1 Atomicity relaxation

Transaction processing in mobile databases imposes further limits on both functionality and performance over those of traditional transaction management. Transaction management requirements in mobile database systems are different from those in conventional database systems. Many new mobile applications require advanced transaction processing, where traditional transaction properties, especially the atomicity and isolation properties, are challenged. The flat transaction structure is often abandoned in favour of some form of nested transaction model, atomicity is replaced by "semantic atomicity", and serializability (which is the correctness criterion guaranteeing isolation) is replaced by some non-serializable correctness criterion. There are a variety of concurrency control techniques guaranteeing different flavours of non-serializability. These techniques often use semantic information about transactions and/or objects to guarantee correctness for concurrent transactions while at the same time allowing more concurrency.

One particular problem in mobile applications is the need to support long-lasting transactions. The duration of a long-lasting transaction may cause serious performance problems if it is allowed to lock resources until it commits. This may either force other transactions to wait for resources for an unacceptable long time, or it may increase the likelihood of transaction abort. Aborting a long-lasting transaction may have a negative effect on both response time and throughput. If the long transaction has a flat structure, a failure will cause the whole transaction to be undone and possibly re-executed. This is a very expensive recovery strategy, especially if the failure occurred after executing most of the transaction.

1.1.1 Mobile transaction processing

To support mobile transactions, the transaction processing models should accommodate the limitations of mobile computing, such as unreliable communication, limited battery life, low bandwidth communication, and reduced storage capacity. Mobile computations should minimize aborts due to disconnection. Operations on shared data must ensure correctness of transactions executed on both stationary and mobile hosts. The blocking of a transaction's executions on either the stationary or mobile hosts must be minimized to reduce communication cost and to increase concurrency. Proper support for mobile transactions must provide for local autonomy to allow transactions to be processed and committed on the mobile host despite temporary disconnection.

Semantic based transaction processing models [58, 59] have been extended for mobile computing in [60] to increase concurrency by exploiting commutative operations. These techniques require caching a large portion of the database or maintaining multiple copies of many data items. In [60], fragmentability of data objects have been used to facilitate semantic-based transaction processing in mobile databases. Each fragmented data object has to be cached independently and manipulated synchronously. That is, on request, a fragment of data object is dispatched to the MH. On completion of the transaction, the mobile hosts return the fragments to the BS. Fragments are then integrated in the object in any order and such objects are termed as *re-orderable* objects. This scheme works only in the situations where the data objects can be fragmented like sets, stacks and queues.

In [61], the concept of transaction proxies is introduced to support recovery. For each transaction submitted to a MH, a dual transaction called *proxy* is submitted to the base station. The proxy transaction includes the updates of the original transaction. Proxy transactions take periodic backups of the computation performed at mobile hosts. Similar to the above, in [62], the notion of *twin transaction* was introduced which essentially replicates the process of executing transactions. In the twin transaction model, each write's request will be mirrored and two equivalent transactions will be created. In this way, if a mobile host is disconnected, the transaction execution can still proceed.

Dynamic object clustering has been proposed in mobile computing in [63, 64]. It assumes a fully distributed system, and the transaction model is designed to maintain the consistency of the database. The model uses weak read, weak-write, strict-read and strict-write. The decomposition of operations is done based on the consistency requirement. Strict-read and strict-write have the same semantics as normal read and write operations invoked by transactions satisfying ACID properties. A weak-read returns the value of a locally cached object written by a strict-write or a weak-write. A weak-write operation only updates a locally cached object, which might become permanent on cluster merging if the weak-write does not conflict with any strict-read or strict-write operation. The weak transactions use local and global commits. The local commit is the same as pre-commit of [30] and global commit is the same as final commit of [30]. However, a weak transaction after local commit can abort and is compensated. In [30], a pre-committed transaction does not abort; hence requires no undo or compensation. A weak transaction's updates are visible to other weak transactions whereas pre-writes are visible to all transactions.

An `open' nested transaction model has been proposed in [25] for modelling mobile transactions as a set of subtransactions. They introduce reporting and co-transactions. A reporting transaction can share its partial results, can execute concurrently and can commit independently. Co-transactions are like co-routines and are not executed concurrently. The model allows transactions to be executed on disconnection. It also supports unilateral commitment of subtransactions, compensating and non-compensatable transactions. The author claims that the model minimizes wired as well as wireless communication cost. However, not all the operations are compensated [25], and compensation is costly in mobile computing.

A kangaroo transaction (KT) model was given in [31]. It incorporates the property that transactions in a mobile computing hop from a base station to another as the mobile unity moves. The mobility of the transaction model is captured by the use of split transactions [26]. A split transaction divides ongoing transactions into serializable subtransactions. An earlier created subtransaction is committed and the second subtransaction continues its execution. The mobile

transaction is split when a hop occurs. The model captures the data behaviour of the mobile transaction using global and local transactions. The model also relies on compensating transactions in case a transaction aborts.

1.1.2 Limitations of the existing works:

Actually, there is a big gap between academic research and commercial products on mobile transactions. In academic research, the mobile support stations play very important roles in the processing of mobile transactions. While in commercial products, mobile hosts and database servers communicate directly, i.e., the role of the mobile support stations does not exist. Moreover, commercial products mainly focus on disconnected transaction processing, while the mobility of mobile hosts is not taken into consideration. The following are the main limitations of the existing work regarding the academic research in the mobile transaction field.

• The lack of some fundamental support for mobile transactions is an issue. There are different views what a mobile transaction is. Many models consider mobile transactions as transactions that are submitted to or initiated from the mobile hosts other models require that mobile hosts must take part in the execution of mobile transactions [MB01]. These different attitudes cause incompatibility and incoherence between mobile transaction processing systems.

• The common architecture of mobile transaction environments relies heavily on the mobile support stations that are stationary and wired connected with the database servers. A difficulty is to extend the capacity of mobile transaction processing systems. The bottleneck problem can occur when there are many mobile hosts within a mobile cell and the distribution of the transaction processes among mobile hosts must be carried out through the mobile support stations.

• The lack of specific detail for how the is the execution is take place at both host (MH, FH) which is consider as an assumption by some of mobile transaction models found in the literature.

• Most of the previous work in transactions model did not consider the effect of fixed transactions on mobile and vies versa.

• Sharing partial results among mobile transactions is not fully dealt with. The existing approaches like delegation operations that support sharing of data among transactions may not be adequate because it requires a tight cooperation between delegator and delegatee transactions. Furthermore, the issue of distributed transaction execution among mobile hosts has not been addressed.

1.1.3 Transaction decomposition

Decomposing the transaction into a number of subtransactions is one way of dealing with these problems. Each subtransaction is typically executed and committed/aborted independently of the top level transaction. This makes it possible to abort some part of a transaction without aborting the whole long-lasting transaction. An aborted subtransaction can be re-executed or an alternative subtransaction can be executed instead of the one that failed. Such a transaction execution gives a semantic form of atomicity, instead of the conventional strict form of atomicity. Decomposing transactions may also positively affect the concurrency of transactions. Since subtransactions commit independently of the top level transaction, locks held by the subtransaction may be released at subtransaction commit time. If the unlocked resources are made publicly available at subtransaction commit time, this will increase concurrency in the system. However, early release of locks will also affect recovery, in that conventional rollback is no longer sufficient. Compensation is needed if the results of a subtransaction are made publicly available before commit of the top level transaction. Decomposition of transactions is one approach, used in several transaction processing techniques, to adapt transaction processing to the mobile computing environment.

Wide area and wireless computing suggest that there will be more competition for shared data since it provides users with the ability to access information and services through wireless connections that can be retained even while the user is moving. Further, mobile users will have to share their data with others. Those users may have access to the shared data by reliable wired communication (i.e.

fixed host transactions) or unreliable wireless communication (i.e. mobile host transactions). The task of ensuring consistency of shared data becomes more difficult in mobile computing because of limitations of wireless communication and restrictions imposed due to mobility and portability [24].

Access to the future information systems through mobile computers will be performed with the help of mobile transactions. Research in mobile database systems (MDBS) has received a lot of interest in the past decade [47, 48, 49, 50, 51, 52, 53]. Transactions in a mobile database system are usually associated with relaxations of ACID properties because of their long-lived nature and limitations inherited from the wireless environment. Any abortion of a mobile transaction may result in a high cost associated with scarce wireless resources, whilst fixed transactions might only cause a little degraded level of system performance. In the past two decades, researchers have proposed various transaction models either to tackle the isolation and atomicity properties of mobile transaction or guarantee the consistency of mobile transaction. Most of the previous studies in mobile database systems assume that the system consists of only one single type of transaction (i.e. mobile transaction), without paying any attention to the effect of interaction between both types of transactions sharing the same databases simultaneously. In these studies, different assumptions are made on the system and transaction models, e.g., the execution strategy and structure of the transactions, so that different scheduling techniques can be engineered to satisfy the different requirements of mobile transactions. However, little work has been done in the development of an integrated approach that can handle mobile database systems satisfactorily containing a *mixed* population of fixed and mobile transactions simultaneously.

As the fixed and mobile transactions continue to interact, it's become apparent that the dichotomy must be resolved not only at the atomicity relaxation level, but at the lower, isolation levels as well. As an initial step in this direction is the first phase of this study which aim to propose a unified framework which has been used later as a basis for evaluating a different execution strategies and then go more in depth into isolation level in the second phase of this study.

1.2 Concurrency control

Many variations of concurrency control (CC) schemes have been introduced to improve concurrency and system performance [68], [69], [70], [71], [72] and [73] in conventional database environments. Recently, there have been several studies dealing with CC in mobile databases addressing the transaction scheduling aspect [74], [75],[76], [77], [78] and [79] These CC schemes are mainly extended or adapted from existing CC schemes, such as two phase locking, optimistic CC schemes, and multiversion schemes, to mobile environments. Some of the CC schemes are more naturally adaptable to mobile requirements, while others are less so. For example, it's not practical for a mobile transaction to hold the lock on a certain data item and then disconnect, as this will cause performance degradation for both mobile and other fixed host transactions running concurrently on the system and sharing the same database access [80].

Several techniques have been developed from conventional CC schemes to cope with wireless requirements, especially for transaction processing in broadcasting environments, such as Multi-Version Broadcast [74], Serialization Graph [77] and [81], Broadcast Concurrency Control with Time Stamp Interval (BCC-TI) [79] and Certification Report [76]. The analysis and drawbacks of these methods can be found in [82]. Some of these methods only support client read-only transactions [76], and some of them could have substantial processing overhead [77]. Protocol inefficiency is also introduced in some of these solutions because of the support of strict global serializability as the correctness criterion. It becomes necessary to design a new broadcast concurrency control protocol to overcome these drawbacks. Most importantly, in order to improve transaction processing efficiency, we need to overcome the problems caused by global serializability which is very difficult to achieve in distributed broadcast environments.

In other types of mobile computing environments such as distributed and multi-database systems, most existing work concentrates on proposing a mobile transaction model and an execution framework for the model over these environments and uses the same traditional concurrency techniques at the database server.

No work has been done in the field of processing of mixtures of transactions. One of the main objectives of this thesis is to address this problem. We will propose a suitable concurrency approaches that take into account the wireless constraints of the mobile computing environment and, at the same time, cater for fixed host transactions which use tolerable resources. In addition to the wireless requirements, other major differences exist between conventional and mobile database systems. For example, while transaction response time and throughput are usually the performance metrics to measure conventional database systems, additional performance measures will be introduced to distinguish between the performance of different concurrency protocols in mobile database systems with mixture of transactions.

1.2.1 Concurrency control background

Concurrency control schemes have the responsibility to ensure that although transactions are executed concurrently with interleaving operations, the committed or certified transactions can be ordered or given a certification time stamp ordering so that the net effect on the database is equivalent to the execution of these transactions in a serial order. Note that, after a transaction is committed, its effect on the database becomes permanent. The CC design space can be classified, into optimistic and pessimistic approaches, along several dimensions: conflict detection, conflict resolution, serialization rule and order, and run policy. Such a classification provides us with a nice framework from which various techniques for achieving serializability and improving performance can be better illustrated.

1.2.1.1 Conflict detection

There are two ways to detect a conflict: either before the data item access or after the data item access. The former is referred to as the pessimistic approach. The latter is referred to as the optimistic approach, where checking for serializability is done later at the certification time. Several mechanisms can be used to facilitate the detection process, such as locks, time stamps and serialization graphs (SGs) [83]. For each of these mechanisms, there is one type of pessimistic CC scheme using it exclusively for conflict detection. For example, two-phase locking (2PL) schemes use locks, time stamp ordering (TSO) schemes use time stamps, and

serialization graph testing (SGT) schemes use serialization graphs. For optimistic concurrency control (OCC) [73], however, any one of the three mechanisms can be used to detect conflicts at the certification time. For a lock based scheme, a lock must first be obtained before an access is made to a data item. However, different access modes and compatibility matrices can be used to allow for more concurrency. For example, read (shared) and write (exclusive) modes are often introduced to distinguish between read and write accesses in a 2PL scheme. A read lock is only compatible with other read locks; a write lock is incompatible with any other lock. When locks are used by an OCC scheme, an additional type of lock mode, referred to as a weak lock, can be introduced as well [84], [72]. A weak lock, which is in contrast to the normal (strong) lock, is only used to indicate that a data item is being accessed. It is compatible with other weak locks and the strong locks of shared mode, but is incompatible with the strong locks of exclusive mode. In addition, an extension can also be made in 2PL to allow each transaction to use a different data item size that is most appropriate for its execution. This is referred to as multi-granularity locking (MGL) [85]. For a TSO scheme, each transaction is assigned a time stamp before execution. For each data item access, the transaction's time stamp is checked against the time stamp of the last transaction that has accessed the same data item to determine whether the predefined time stamp order can be maintained. For an SGT scheme, the CC manager maintains an SG that represents the execution ordering of all the transactions in the history, and checks for cycles. In general, information about a transaction cannot be deleted at commit time. It can be deleted only after the committed transaction will not, at any time in the future, be involved in a cycle of the SG.

1.2.1.2 Conflict resolution

Once a conflict is detected, a conflict resolution mechanism needs to be put in place. A conflict resolution mechanism needs to decide which candidate transaction(s) (the lock requester or lock holders) to penalize, and choose an appropriate action and a suitable timing for the action. There are two possible actions that are most frequently used: blocking (or wait) and abort (or restart). (Two other alternatives are multiversioning, and dynamically readjusting the

serialization order, which will be discussed later.) If a conflict is detected before the data item access, either blocking or abort can be used to resolve the conflict. However, if a conflict is detected after the data item access, only abort is appropriate. An alternative to blocking for conflict resolution is delaying commit. That is to say, instead of waiting for a lock, the transaction is waiting for the commit order. This can be implemented in a multiversion scheme and other locking schemes like the ordered-sharing schemes in [86]. As to a suitable timing for an action, it is immediate for blocking, but it can be either immediate or delayed, e.g., delayed until the certification time, for abort.

We consider different performance goals in processing fixed and mobile transactions at the same time. The performance goal in processing mobile transactions is to minimize the amount of wasted wireless resources, while the performance goal in processing fixed transactions is to maximize the system throughput (or to minimize the mean response time, for example).

As astute readers may notice, conventional concurrency control protocols used in scheduling mixture of mobile and fixed transactions often ignore the performance requirements of fixed transactions. For example, most of the traditional concurrency control protocols resolve access conflicts based on the idea of both transactions type experience the same type of resources. When they are applied to a mixture environments, it is likely to waste more wireless recourses due to blocking and restarting overhead of mobile transactions, due to data conflicts. This may cause the poor response times of fixed host transactions too. How to explore the balance and trade-off between the performances of the two types of transactions is one of the main objectives in the second phase of this study.

1.3 Thesis contributions

The following is a summary of the major contributions of this dissertation:

- We propose an execution framework for mobile transaction processing and analyze the impact on mobility of mobile transaction processing under different execution strategies.
- We develop a model representing mixed transactions systems which captures both fixed and mobile characteristics of transactions. The model provides a basis for the performance study of general concurrency control approaches in such mixed environments.
- We study the effect of bandwidth variability on transaction processing in mixed transactions systems. Mainly, we present the effect of using a 2-phase locking protocol mathematically and analyse it by estimating the delay caused by the mobile transaction on other fixed and mobile transactions.
- We propose two adaptable concurrency control approaches suitable for mixtures of transactions. One is based on locking, called the 'Lock-Mix' approach; the other is based on optimistic concurrency control paradigms and is called the 'OCC-Mix' approach.
- We conduct extensive evaluations, through simulation experiments, of the concurrency control approaches. The results demonstrate that using these approaches reduces wastage of scarce and expensive resources of mobile computing environments. In particular, we show that these approaches provide significant performance gains over an optimistic concurrency control protocol for environments with mixtures of mobile and fixed transactions under a wide range of operating conditions.

The publications arising from the work of the thesis are given in the appendix.

1.4 Thesis organization

The remainder of this thesis is organized as follows. In Chapter 2, we first revisit the basic concepts of database transactions, and discuss how these concepts are achieved in practical systems. Next, we briefly go through the architecture of transaction processing systems in centralized and distributed environments. We survey several selected transaction models and transaction processing systems that have been purposely developed to support transaction processing in mobile environments. We recap some traditional transaction models whose features could be used in mobile environments. Traditional transaction models are reviewed along with a discussion of their importance to mobile environments. Mobile transaction models and mobile transaction processing systems, that have been developed recently, are surveyed in Chapter 2. We also review the data management issues in mobile computing environments. In Chapter 3, we exploit a particular relaxation of ACID properties, commonly used in mobile transaction models when building execution frameworks. We evaluate the impact of mobility and disconnection, on processing of mobile transactions under different execution strategies, through simulation, and discuss the results. In Chapter 4 the basic concurrency problem for environments with mixtures of fixed host and mobile transaction systems is defined. We consider the effect of bandwidth variability on scheduling a mixture of mobile and fixed host transactions, and use this to motivate our proposed concurrency control approaches in Chapter 5. The proposed concurrency control approaches, those of 'Lock-Mix' and 'OCC-Mix', presented in Chapter 5. In Chapter 6, we investigate the performance of these approaches in mixed transaction environments for different percentages of mobile transactions. The performance of these protocols are evaluated and compared against an OCC protocol under different workload and operating conditions. We show that wastage of wireless resources is reduced substantially by applying these approaches. Finally, Chapter 7 summarizes the salient results of our research, and gives a concluding discussion.

[1]CHAPTER 2

TRANSACTIONS PROCESSING BACKGROUND

In this chapter, we first revisit the basic concepts of database transactions in section 2.1, and discuss how these concepts are achieved in practical systems. Next, in section 2.2, we briefly go through the architecture of transaction processing systems in centralized and distributed environments. Then, in section 2.3, mobile database systems are described. The difference between the mobile and distributed computing is explained in section 2.4. We survey several selected transaction models and transaction processing systems that have been purposely developed to support transaction processing in mobile environments. We will also recap some traditional transaction models whose features could be used in mobile environments. Traditional transaction models are reviewed in section 2.5. We discuss why they are important, and how these models can be used in mobile environments. Mobile transaction models and mobile transaction processing systems that have been developed recently are surveyed and commented on in section 2.6. Finally, we review the data management issues in mobile computing environments in section 2.7.

2.1 Database and transaction concept

A *database* is a collection of data items that is gathered over a period of time, and safely stored for further examination or analysis [1]. A database is usually accompanied by a data structure and a set of constraint rules that specify what information a data item represents. For example, in an employee database, the employee age is an integer number and must be greater than eighteen and less than sixty five. *A database state* is a collection of all the stored data values of all the data items in the database at a specific time [2]. A *consistent state* of a database is a database state in which all the data values fulfil all the constraint

[1] This is a background chapter "book work" see reference [71]

rules of the database. A set of operations is usually provided to support users in retrieving or modifying data items in the database. These provided operations can be simple, for example read and write operations, or more complex operations, for example deletion or modification operations. To assist users to perform much more complex operations rather than reading from and writing to the database, a piece of specialized software called a *database management system* (DBMS) is accommodated in the database. In general, a DBMS not only provides an easy-to-use and friendly interface to users for accessing and manipulating the database, but also manages all the database operations. In addition, the DBMS also protects the database from unauthorized users.

2.1.1 Database transactions

Users can interact with the database by one or many database operations. The database operations can be gathered together to form a unit of execution program that is called a *transaction* [3]. In other words, a transaction is a logical execution unit of database operations. A transaction transforms the database from one consistent state to another consistent state. Figure 2.1 presents a programming model of a transaction.

```
Begin_transaction (initial_consistent_state)
One or more database operations
If (reach new_consistent_state) Then
Commit_transaction (new_consistent_state)
Else
Abort_transaction (initial_consistent_state)
```

Figure2. 1: Transactional programming model

A transaction program starts from an initial consistent state of the database by invoking a *Begin_transaction* method call. After that, one or a set of database operations of the transaction program are executed. When these database operations are completed, i.e., a new consistent database state is established as designed, the transaction program saves this new consistent state into the database by calling the *Commit_transaction* method. The *Commit_transaction* call ensures

that all the database operations of the transaction program are successfully executed and the results of the transaction are safely saved in the database. If there is any error during the execution of the transaction program, the initial consistent state of the database is re-established by the *Abort_transaction* call. The *Abort_transaction* call indicates that the execution of the transaction program has failed and this execution does not have any effect on the initial consistent state of the database. The transaction is said to be *committed* if it has successfully executed the *Commit_transaction* call, otherwise it is *aborted*. A transaction is called a *read-only transaction* if all of its database operations do not alter any database state.

2.1.2 The ACID properties

In a database system, there may be a large number of transactions that are executed concurrently, i.e., the shared data items in the database are read and possible written by many transactions at the same time. Each transaction must ensure that it always preserves the consistency of the database system. In order to retain and to protect the consistency of the database system, transactions will have the following ACID (Atomicity, Consistency, Isolation, and Durability) properties [3]:

- **Atomicity.** Either all database operations of a transaction program are successfully and completely executed, or none of the database operations of this transaction program are executed.
- **Consistency.** A transaction must always preserve and protect the consistency of the database, i.e., it transforms the database from one consistent state to another. In other words, the result of a transaction that has committed fulfils the constraints of the database system.
- **Isolation.** An on-going transaction must not interfere with other concurrent transactions, or be able to view intermediate results of other concurrent transactions. In other words, a transaction is executed as if it is the only existing execution program on the database system at any given time.

• **Durability.** The result of a transaction that has successfully committed is permanent in the database. The consistent state of the database is always survived despite any type of failures.

The ACID properties of a transaction ensure that: (1) a transaction always keep the database in a consistent state, (2) a transaction does not disturb other transactions during their concurrent execution processes, and (3) the consistent state of the database system that is established by a committed transaction withstands software or hardware failures. In order to achieve the ACID properties, normally, two different sets of protocols named *concurrency control protocols* and *recovery protocols* are needed [2].

2.1.3 Concurrency control of transactions

In this section, we discuss the problems that can occur in a database system in which there are many transactions being executed concurrently. In other words, we answer the question of why there is a need of concurrency control in the database system. We also review different techniques that ensure the correctness of transaction execution. To illustrate and to simplify the analyses without losing generality, we assume that each transaction possesses the following characteristics:

• Transaction T_i starts by a *Begin_transaction* call that is denoted by B_i.

• A database operation $Op_i(X)$ on a data item X is either a read operation $R_i(X)$ or a write operation $W_i(X)$. In general, more complex operations on a database system can be modelled via read and write operations.

• Transaction T_i ends by either a *Commit_transaction* call denoted by C_i, or an *Abort_transaction* call denoted by A_i.

Some typical problems which are caused by the concurrent execution of transactions are: lost update, dirty read, and unrepeatable read [3]. These problems are presented in Figure 2.2

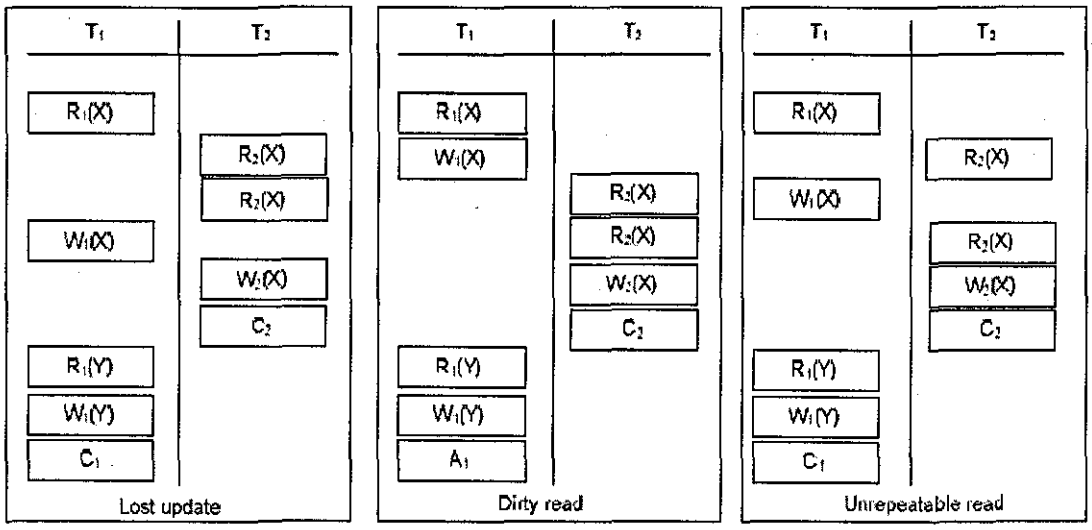


Figure2. 2: Concurrency problems

First, the lost update occurs when two transactions *T1* and *T2* try to write the same data item *X*. In the figure, transaction *T2* overwrites the value of data item *X* that was, prior to that, written by transaction *T1*. The dirty read occurs when transaction *T2* reads the value of data item *X* that is written by transaction *T1* before the transaction *T1* commits. If transaction *T1* aborts, transaction *T2* has been operating on an invalid data value. Finally, the unrepeatable read happens if a transaction executes the same read operation at different times, and obtains different data values. In Figure 2.2, the read operations of transaction *T2* return two different values of *X*: before and after the write operation of transaction *T1*.

The concurrency problems can be solved if the DBMS can schedule these database operations of transactions in an execution order in which no transaction interferes with other, i.e., fulfils the isolation property of transactions. The execution order that sequentially contains all the database operations of all concurrent transactions is called the *schedule* or *history* of transactions [4]. The order of database operations of one transaction must be retained in the schedule of all transactions. A schedule is a serial schedule if, for any pair of transactions, all the database operations of one transaction follow all the database operations of another transaction. In other words, the isolation property of transactions is ensured in a serial schedule. Figure 2.3 (we omit the commitment and the abortion operations of transactions in the schedule) presents the possible serial schedules of transactions *T1* and *T2*.

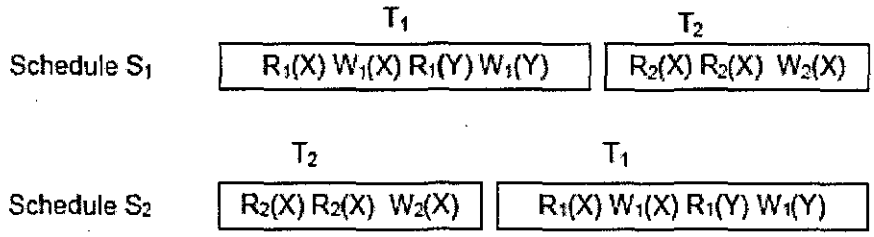


Figure2. 3: Serial schedules

The main disadvantage of the serial schedule is that transactions must be executed serially, i.e., the concurrent execution of transactions does not exist in a serial schedule. This may decrease the performance of the database system. To deal with this drawback, the concept of *serializable schedule* [4] is normally used. A schedule is serializable if it is equivalent to a serial schedule. The remaining question is how to determine if a schedule is a serializable schedule. In other words, we need to clarify the "equivalent" term. Two examples of the equivalent serializability are: *conflict serializability* and *view serializability* [1]. Conflict serializability is based on the concepts of conflicting operations. The idea behind conflicting operations is that: for two sequentially executed operations Op_1 then Op_2 that belong to two transactions T_1 and T_2, respectively, if their order is interchanged, i.e., Op_2 then Op_1, the results of at least one of the involved transactions will possibly be changed. In other words, two database operations that belong to two different transactions are conflicting if they access the same data item in the database and at least one of them is a write operation [1]. Two consecutive operations, which are not in conflict, can be swapped or interchanged in a schedule without any effect on the transaction behaviour. Two schedules are said to be *conflict equivalent* if one can be turned into another by swapping the pairs of non-conflict operations [1]. A schedule is *conflict serializable* if it is conflict equivalent to a serial schedule. Figure 2.4 illustrates some conflict serializable (CS) schedules. Both the schedules CS_1 and CS_2 (in Figure 2.4) are conflict serializable with the serial schedule S_1 (in Figure 2.3), while the schedule non-CS_3 is not conflict serializable. Moreover, the schedule CS_1 can be turned into the schedule CS_2 by sequentially swapping pairs of non-conflict operations ($W_2(X)$, $R_1(Y)$), ($W_2(X)$, $W_1(Y)$), and ($R_2(X)$, $R_1(Y)$).

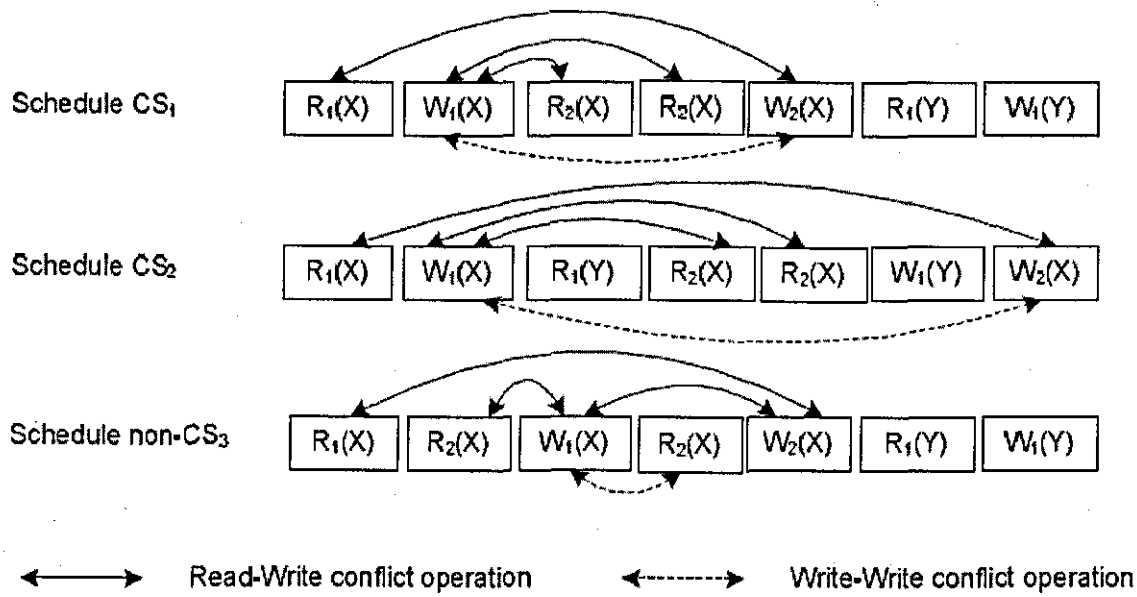


Figure2. 4: Conflict serializable and non-conflict serializable schedules

A schedule *S* can be validated if it is conflict serializable by analyzing a *serialization graph* [4]. A serialization graph (SG) is a directed graph that is constructed in two steps as follows:

1. Each node labelled T_i in the SG represents a corresponding transaction T_i in the schedule *S*.

2. For any pair of operations, Op_i and Op_j, that conflict in the schedule *S*, where Op_i precedes Op_j, add an edge from T_i to T_j in the SG.

The schedule *S* is conflict serializable if the constructed SG has no cycles [4]. In Figure 2.5, the serialization graphs of schedules CS_1, CS_2 and *non-*CS_3 (in the Figure 2.4) are constructed. For schedules CS_1 and CS_2, the corresponding SG do not contain any cycle, i.e., the schedules are conflict serializable. On the other hand, the SG of the schedule *non-*CS_3 does contain a cycle $T_1 \rightarrow T_2 \rightarrow T_1$, i.e., it is not conflict serializable.

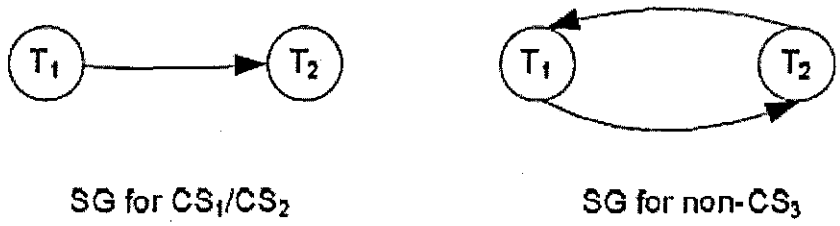


Figure2. 5: Serialization graph

View serializablity is a weaker condition that guarantees that a schedule is serializable. Two schedules S_1 and S_2 are said to be view equivalent if the following conditions hold: (1) any read operation in either schedule returns the same data value, and (2) if a write operation $W_i(X)$ is the last operation on data item X in S_1, $W_i(X)$ must also be the last operation on X in S_2 [1]. Thus, the view equivalent conditions ensure that (1) all the transactions read the same data values, and (2) the final database states are identical. If a schedule is view equivalent to a serial schedule, it is said to be *view serializable*. Figure 2.6 illustrates a view serializable schedule. The serial schedule S_1 presents the sequential order schedule of transactions T_1, T_2, and T_3. The schedule VS_2 is not a conflict serializable schedule because of conflicting operation pairs $((W_1(X), R_2(X))$ and $((W_2(Y), W_1(Y))$. However, the schedule VS_2 is a view serializable schedule because: (1) all the read operations $R_1(Y)$, $R_2(X)$ and $R_3(X)$ return the same data values of data items Y and X as in the serial schedule S_1; and (2) all the write operations $W_1(X)$ and $W_3(Y)$ are the last write operations on the data items X and Y as in the serial schedule S_1. The main disadvantage of view serializability is that, verifying view serializable schedule problem has been shown to be a NP-complete problem, i.e., it is not likely that a polynomial time algorithm for this problem will be found [5].

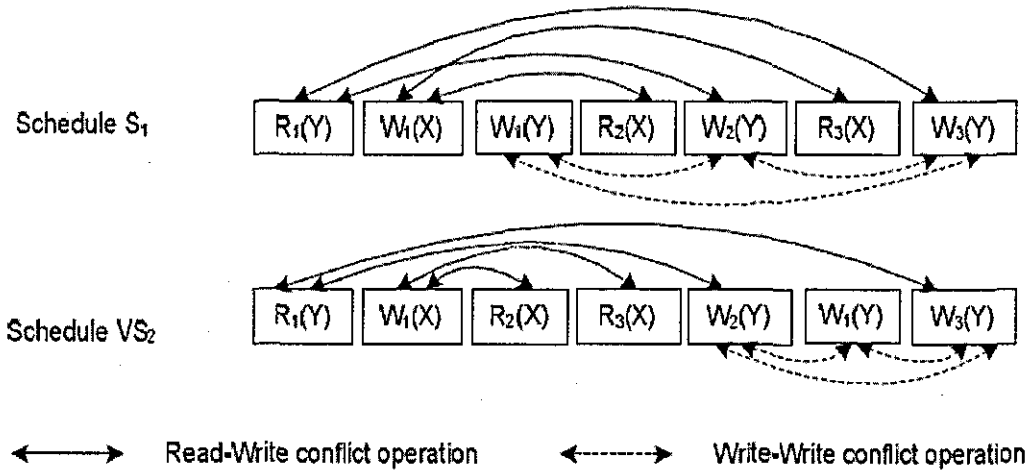


Figure2. 6: View serializable schedule

To assure that a schedule *S* is serial equivalent, the database system must keep track of conflicting operations in the schedule *S*, construct the SG of the schedule *S*, and check for a cycle in the constructed SG. This process repeats every time when a new database operation arrives to the database system, and requires a lot of computing resources and processing time. Due to the overhead of checking serialization graphs, one normally requires that a completion of the execution schedule of all committed transactions is available before the verifying algorithm can be carried out. This is not true in real-world transaction processing systems where transactions are dynamically and continuously submitted to the transaction processing system. Concurrency control protocols, in fact, do not check for serializability, but are used to ensure that a sequence of executable database operations submitted from on-going transactions can form a serializable schedule. There are two main approaches for concurrency control protocols [1]: *pessimistic* and *optimistic*. For the pessimistic approach, a database operation is checked if it could cause a non-serializable schedule before it is executed. The database operation is rejected, i.e., the transaction is aborted, if it may potentially lead a schedule into a non-serializable schedule. For the optimistic approach, the submitted database operation is immediately executed as if there is no conflict between this database operation and database operations of other transactions. When a transaction begins to commit, a certification process, in which the transaction will be validated against other transactions, is carried out. If none of

the database operations of this transaction breaks the serializability, the transaction is allowed to commit, otherwise the transaction is aborted. Locking and timestamp ordering protocols are two common concurrency control protocols that are mostly used in the pessimistic approach. Concurrency control by the locking protocol requires that a transaction must request an appropriate lock on a data item before its database operation can be accepted for executing. In other words, a lock plays a role as an execution license for the database operation. One usually applies two types of lock: *shared* (read) and *exclusive* (write) [3]. A shared lock can be granted to many transactions at the same time, while an exclusive lock can only be assigned to one transaction at a time (see Table 2.1 for the lock compatibility matrix which shows what kind of lock combination are allowed or not). Serializability among transactions can be guaranteed by a *2-phase locking* (2PL) protocol [4]. The 2PL protocol requires that a transaction must obtain all its locks (in *growing phase*) before it can release any lock (in *shrinking phase*). Strict 2PL is a locking protocol that only allows a transaction to release exclusive locks after it has committed or aborted. Concurrency control by using timestamp ordering guarantees serializability among transactions based on the following time quantities: (1) the starting time or timestamp of each transaction *TS*, and (2) the read and write timestamp values for each data item *X*, denoted by *Read_TS(X)* and *Write_TS(X)* respectively. These read or write timestamp values correspond to the timestamp value of the latest transaction that successfully reads or writes the data item *X*. A timestamp can be a computer system clock or any logical counter maintained by the database system. When a transaction submits a database operation on a data item *X*, the timestamp *TS* of the transaction will be checked against the current read *Read_TS(X)* and write *Write_TS(X)* timestamp values of the data item. The outcome of this timestamp checking procedure is either the database system accepts the submitted database operation and the new timestamp value is updated for *X*, or the transaction is aborted.

Table2. 1: Lock compatibility matrix

Lock hold / Lock request	Shared	Exclusive
Shared	Yes	No
Exclusive	No	No

The optimistic approach for concurrency control was first proposed in [6]. There are several methods to carry out the certification process of a transaction, for example serialization graph testing (SGT) [4] or validation [7]. The SGT method dynamically builds a serialization graph SG between transactions when a conflicting operation is carried out. When a transaction T_i requests to commit, the SGT method checks if the transaction T_i belongs to a cycle of the SG. If it does, the transaction T_i is aborted; otherwise the transaction T_i passes the certification procedure and will be allowed to commit. The validation method is based on the concepts of conflicting operations to ensure that the scheduling of a transaction T_i is serializable in relation to all other overlapping transactions T_j, which have not committed when the transaction T_i begins [8]. Figure 2.7 illustrates a validation process of transaction T_3 (time proceeds from left to right). When transaction T_3 requests to commit, the validation process will check to ensure that the database operations of transaction T_3 do not conflict with the database operations of transactions T_1, T_2 and T_4.

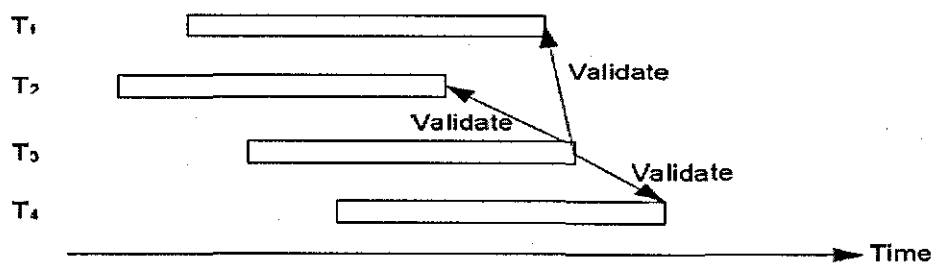


Figure2. 7: The validation procedure of a transaction

Every concurrency control protocol has disadvantages. Transactions in a database system that uses locking protocols can suffer from deadlocks or long blocking periods [1]. Timestamp ordering protocols can decrease the performance of the

transaction processing system if there is a high conflict among transactions [9], i.e., many transactions must abort or roll back. In the guard-after approach, work that has been done and system resources might be wasted if transactions are aborted. Concurrency control in a database system can apply either one or a combination of these concurrency control protocols.

2.1.4 Recovery concepts

The objective of recovery protocols is to enforce the atomicity and durability properties of transactions [2]. The atomicity property requires that either all or none of the database operations of a transaction is carried out. The durability property requires that the results of committed transactions, i.e., consistent database states, survive any kind of failure. In this section, we first study different types of failures that could happen in a database system. Later, we review different recovery techniques that allow the database system to recover from failures. Normally, databases are stored on non-volatile media systems like magnetic or optical disks, and are further backed-up by one or more safe storage systems [5]. During the execution of transactions, data items are loaded and temporarily stored in computer memory that is volatile storage. There are two main types of failures of a database system: *catastrophic* and *noncatastrophic* [1]. A catastrophic failure happens when there is a breakdown in data storage systems, for example a hard disk crashes. A catastrophic failure can be recovered if there is a sufficient database system backup. Non-catastrophic failures do not affect the non-volatile database storage system, i.e., only data in the volatile storage such as memory is lost. The non-catastrophic failures include transaction and computer system malfunctions. Failures of transactions might be caused by logical faults of data or transaction programs or by the database system. Computer system malfunctions could be caused by errors in the operating systems or applications. A recovery support system will keep track of and record the progress of the execution of transactions by periodically writing important information like data modifications, commitments or abortions of transactions to a logbook, which is stored in the non-volatile storage system. These log records will be used to re-establish a consistent database state if any failure occurs. There are two main

recovery techniques - those of *undo* and *redo* [4]. These two approaches support database systems to reconstruct consistent database states when there is any failure in the database system. However, they are different in their *logging* strategies. The undo logging strategy records in the non-volatile logs, the former consistent database state before the database state was changed by a transaction. The redo logging writes to the non-volatile logs, the new consistent database state that the database system will have after the updated transaction commits. Figure 2.8 compares these two logging strategies. The undo technique supports database systems to reconstruct the previous consistent database states when a transaction fails. The database system behaves as if no database operation of the aborted transaction has been executed. In other words, the undo technique is used to clean up the presence of data values of uncommitted transactions in the database system. For the undo approach, the new database state must be written to the database system after the undo logs have been written to non-volatile storage [3]. The redo technique ensures that the database system reproduces the database states that are the results of successfully committed transactions. The redo approach, therefore, will ignore any uncompleted transaction. Before the new data values are written to the database system, all the redo log records must be written to non-volatile storage [3]. A recovery support system can combine (which is also the normal case) both undo and redo approaches so that it can decrease the work lost by failures.

Initial states	T_1	Undo Log	Redo Log
X = 10 Y = 20			
	Read(X)	START T_1	START T_1
	X = X + 10		
	Write (X)	<T_1, X, 10>	<T_1, X, 20>
	Read(Y)		
	Y = Y - 10		
	Write (Y)	<T_1, Y, 20>	<T_1, Y, 10>
	C_1	COMMIT T_1	COMMIT T_1

Figure2. 8: Undo logging against redo logging

In Figure 2.8, for the undo approach, if transaction T1 aborts after it has modified the value of data item Y, the recovery system can re-establish the initial database states by two logging records <T1,X,10> and <T1,Y,20>. For the redo approach, if a failure occurs after transaction T1 has committed, the database system will re-produce the committed values of transaction T1 based on two logging records <T1,X,20> and <T1,Y,10>. If a new failure happens when the database system is being recovered from previous failures, the recovery procedure has to be able to restart as many times as needed. This feature is called *idempotent* [3], i.e., the results of the re-executed recovery procedure are independent of the number of times that they are repeatedly executed. When a transaction is aborted, its effect on the database system will be rolled back. If a transaction commits, its results are permanent by the durability property. In other words, a committed transaction does not rollback. A schedule S is said to be *recoverable* if no transaction T in S commits until all transactions T' that have updated data items that T reads have committed or aborted [4]. A serial schedule is, therefore, always recoverable. Note that a serializable schedule does not forbid a transaction T_i to read from a data item X that is modified by an uncommitted transaction T_j (see Figure 2.2, *dirty read* problem). Recovery techniques make no attempt to support the serializability of transactions [1]. Figure 2.9 illustrates the recoverable against serializable schedules. Schedule S_3 is a recoverable schedule because the transaction T_2 that reads a new value of data item X modified by transaction T_1 commits after transaction T_1 has committed. Schedule S_4 is a serializable but non-recoverable schedule because transaction T_2 commits before T_1 commits.

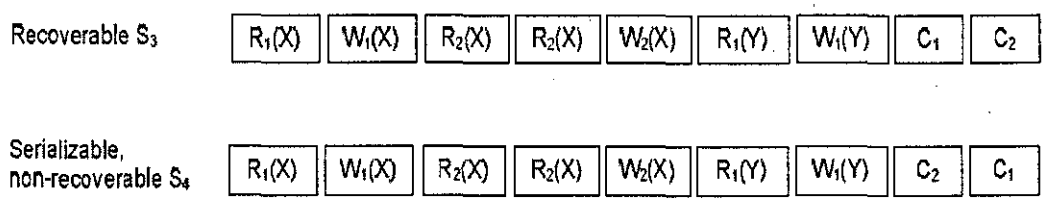


Figure2. 9: Recoverability versus serializability

In a recoverable schedule S, if a transaction T_i reads data values that are written by an uncommitted transaction T_j, then if transaction T_j aborts, T_i must also abort.

The aborting of transaction *Ti* could subsequently cause other transaction *Tk* to abort if the transaction *Tk* has been reading data values that are modified by the transaction *Ti*. This aborting can recur to many other transactions. This phenomenon is called *cascading abort* and is illustrated in Figure 2.10. Unfortunately, a recoverable schedule does not prevent the cascading abort problem. Therefore, a stronger condition that only allows a transaction to read data values, which are modified by committed transactions, is needed. An *avoid cascading abort* schedule only allows a transaction to read data values that are written by a committed transaction. Furthermore, a *strict schedule* only allows a transaction to read or write data items that are modified by committed transactions [4].

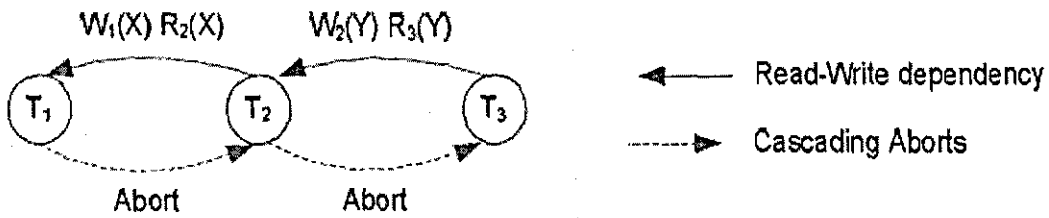


Figure2. 10: A cascading abort scenario

2.2 Transaction processing systems

In this section, we will first discuss the basic and essential components of a transaction processing system that manages the execution of transactions on a transaction-oriented database system. Later, we review the architecture of distributed transaction processing systems.

2.2.1 Essential components of a transaction processing system

A transaction processing system plays a role as a mediator that accepts transaction requests from users, dispatches these requests to the database system, coordinates the execution of the involved transactions, and forwards transaction results to the original acquirers. Figure 2.11 illustrates an interaction model for a transaction-oriented database system. The common programming model for a transaction-oriented database system is the client-server model [3, 10]. Users or clients interact with the database system by submitting their transaction processes that consist of one or many database operations to the transaction processing system.

The transaction processing system will coordinate and manage the execution of these transaction processes by subsequently sending these database operations to the database system. The database system will carry out the actual execution of the submitted database operations. Finally, the transaction results that reflect the consistent states of the database system are returned to the clients.

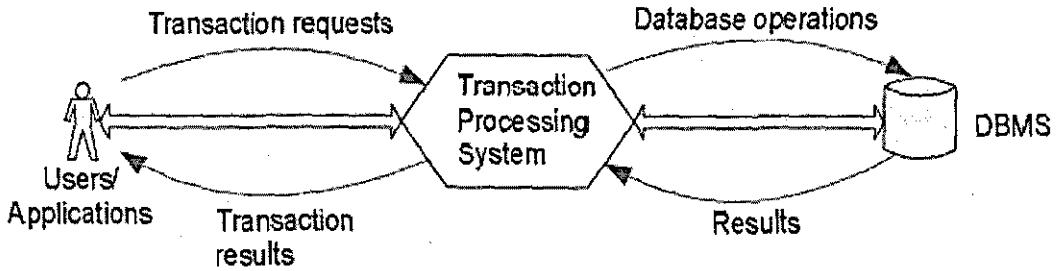


Figure2. 11: Dataflow of transaction-oriented database systems

To protect the integrity constraint of the database system, the transaction processing system must ensure that the ACID properties of transactions are fulfilled. In order to achieve this, a set of essential components that includes a transaction manager, a scheduling manager and a log manager are deployed [3]. Additional components such as a communication manager or other resource managers can also be employed by the transaction processing system. However, in this section, we will focus our discussion on the three essential components. Figure 2.12 presents the roles of the transaction processing system components.

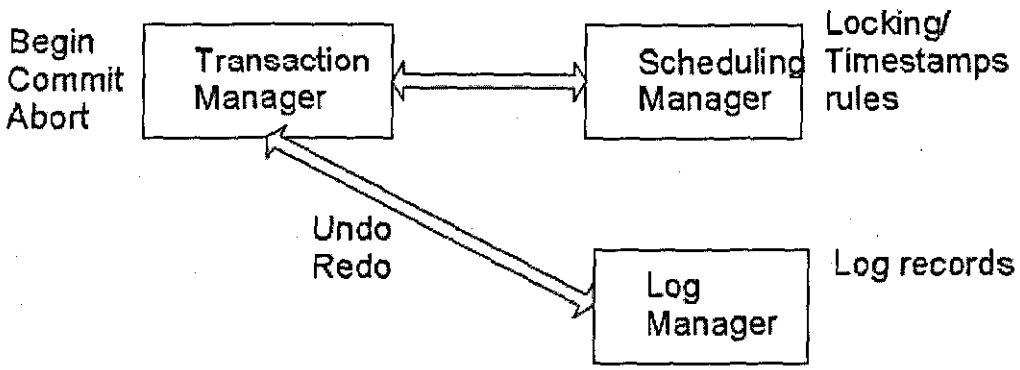


Figure2. 12: Transaction processing system components

The role of each transaction processing component is described as follows:

Transaction manager. The role of the transaction manager is to orchestrate the execution of transactions [3]. Via the help of the scheduling and log managers (explained below), the transaction manager takes care of all important operations of transactions such as begin, read, write, commit, and abort (or rollback). If the execution of a transaction is distributed to many different resource managers, the transaction manager will act as the coordinator of the involved participants.

Scheduling manager. The scheduling manger manages the order of execution of the database operations. Usually, the scheduling manager makes use of concurrency control protocols, for example locking or timestamp protocols, in order to control the execution of transactions. Thus, the scheduling manager supports the isolation and consistency properties of transactions. Based on the applied concurrency control protocol, the scheduling manager will determine an execution order in which the submitted database operations will be carried out. For example, if a locking protocol is used, the scheduling manager will decide whether a lock request will be granted to the acquired transaction or, if a timestamp protocol is applied, the scheduling manager will assess if a submitted operation will be allowed to be carried out.

Log manager. The role of the log manager is to support the database system to recover from failures. The log manager keeps track of the changes of the database states by recording the history of transaction execution. Depending on the deployed recovery strategies, for example *undo* and/or *redo*, the log manger will record necessary information in a non-volatile logbook. The log manager ensures the atomicity and the durability properties of transactions.

The cooperation among the transaction manager, the scheduling manger and the log manager will assure that the ACID properties of transactions in a transaction-oriented database system will be fulfilled.

2.2.2 Distributed transaction processing systems

In the previous section, we have discussed the essential components of a transaction processing system where data is stored in one database system. In this section, we will consider a distributed database system where data is distributed

among different computers [11]. A distributed transaction processing system is a collection of sites or nodes that are connected by communication networks (see Figure 2.13).

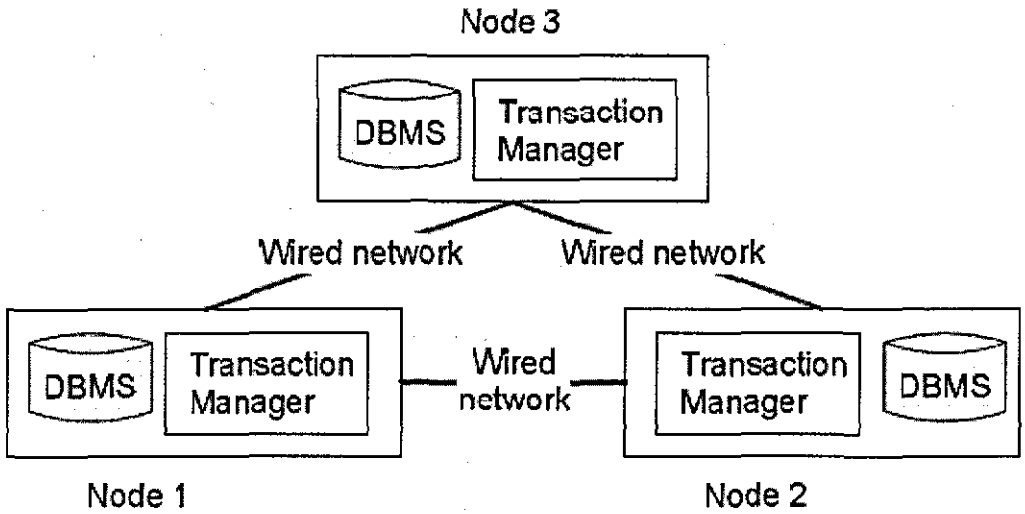


Figure2. 13: Distributed transaction processing systems

The communication networks are usually reliable and high speed wired networks, like LANs or WANs. At each node in a distributed system, there is a local database management system and a local transaction processing system (TPS) that operates semi-independently and semi-autonomously. An execution of a transaction in a distributed database system may have to spread to be processed at many sites. The transaction managers at different sites in a distributed transaction system cooperate for managing the transaction execution processes. Transactions in a distributed system can be categorized into two classes: local transactions and global transactions. Consequently, there are two types of transaction manager in a distributed transaction processing system: a local transaction manager and a global transaction manager [12]. Local transactions are submitted directly to local transaction managers (Figure 2.14). Local transactions only access data at one database system at one site, and are managed by the local transaction manager. On the other hand, global transactions are submitted via the global transaction manager. A global transaction can be decomposed into a set of subtransactions; each of which will be submitted and executed as a local transaction at a local database system [13, 12]. Therefore, the execution of a global transaction can involve accessing data at many sites, and be under the control of many local

transaction managers. A successful global transaction must meet both the integrity constraints of local databases and the global constraints of the distributed database system.

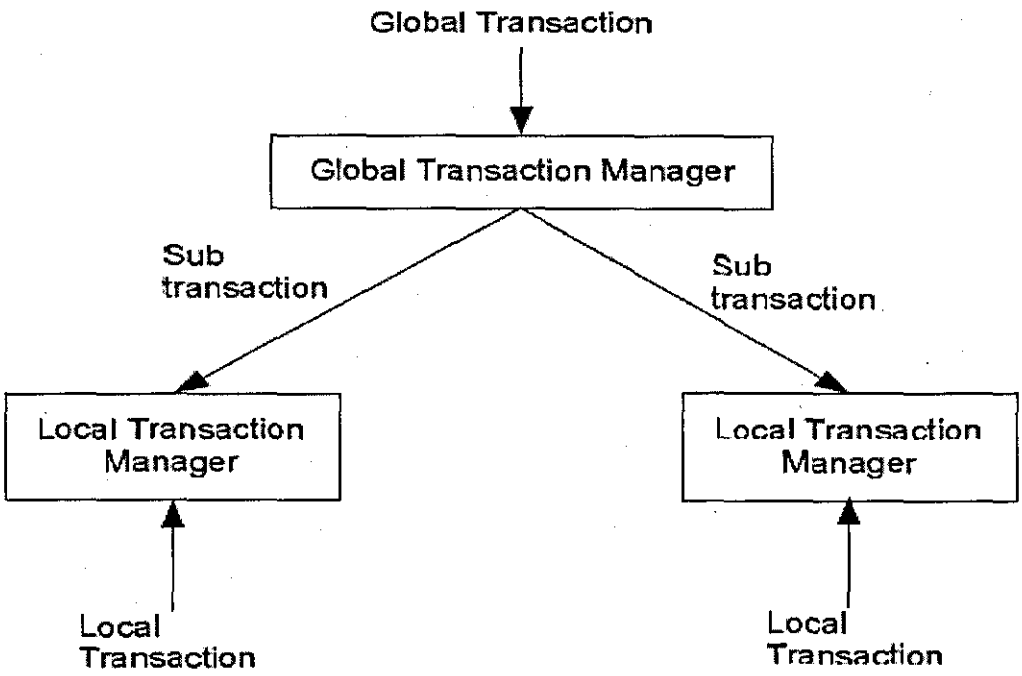


Figure2. 14: Local and global transactions

Some of the potential advantages of distributed transaction processing systems are: (1) higher throughput for transaction processing, and (2) higher availability than the centralized transaction processing system [3]. However, distributed transaction processing systems also introduce many challenging issues, for example disconnections in communication between computing sites or concurrency control across computing sites. These problems could cause data inconsistency among database systems, and aborts of on-going transactions. Consequently, more complicated concurrency control protocols or transaction commitment protocols are needed [4], for example distributed 2-phase locking and 2-phase commit protocols. Moreover, the heterogeneous characteristic of the distributed system must also be taken into consideration [3, 8] - for example different database systems or operating systems.

2.3 Mobile database architecture

In a mobile computing environment (see Figure2.15), the network consists of fixed hosts (FHs), mobile units (MUs) and base stations (BSs) or mobile support stations (MSS). MUs are connected to the wired network components only through BS via wireless channels. MUs are battery powered portable computers, which move around freely in a restricted area, which we refer to as the "geographical region" (G). For example in Figure2.15, G is the total area covered by all BSs. This cell size restriction is mainly due to the limited bandwidth of wireless communication channels. To support the mobility of MUs and to exploit frequency reuse, the entire G is divided into smaller areas called cells. A particular BS manages each cell. Each BS will store information such as user profile, login files, and access rights together with user's private files. At any given instant, an MU communicates only with the BS responsible for its cell.

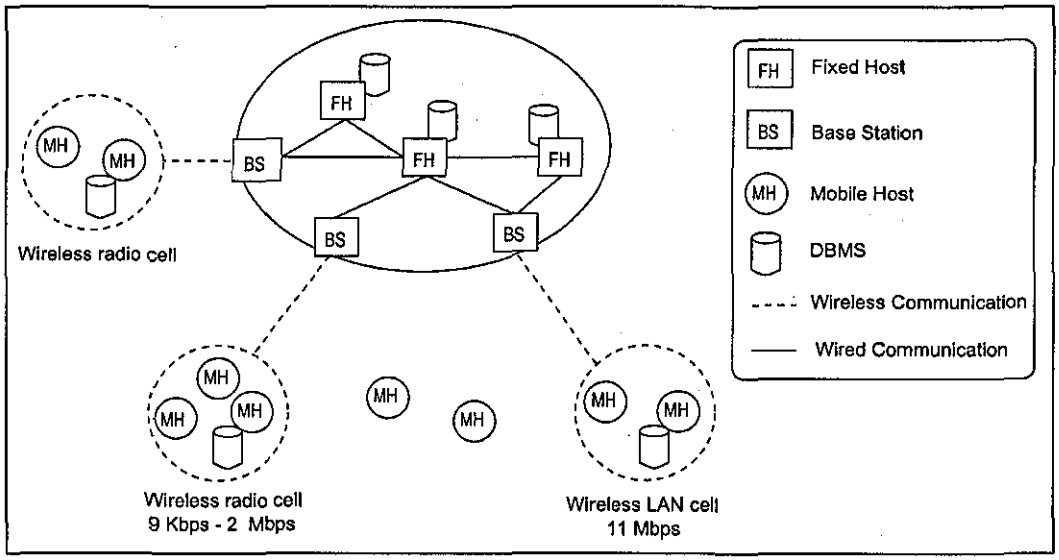


Figure2. 15: mobile data base architecture

The mobile discipline requires that an MU must have unrestricted movement within G (inter-cell movement) and must be able to access desired data from any cell. An MU changes its location and network connections while computations are being processed. While in motion, a mobile host retains its network connections through the support of BSs with wireless connections. The BSs and FHs (fixed hosts) perform the transaction and data management functions with the help of

database server (DBS) components to incorporate database processing capability without affecting any aspect of the generic mobile network. DBSs can either be installed at BSs or can be a part of FHs or can be independent to BS or FH. BSs will provide commonly used application software so that a mobile user can download the software from the closest FH and run it on the palmtop or execute it remotely on the FH. Thus, the most commonly used software will be fully replicated. A mobile host may play a different role in a distributed system. A MU may have some server capability to perform computations locally using local concurrency control and recovery algorithms. Some MUs may have a very slow CPU and very little memory and thus, act as an I/O device only. Thus, they depend on some FHs. Within this mobile computing environment, shared data are stored and controlled by a number of DBSs. When an MU leaves a cell serviced by a BS, a hand-off protocol is used to transfer responsibility for the mobile transaction and data support to the BS of the new cell. This hand-off involves establishing a new communication link. It may also involve migration of in-progress transactions and database states from one BS to another. The entire process of handoff is transparent to a MU and is responsible for maintaining end-to-end data movement connectivity. Three essential properties pose difficulties in the design of applications for mobile computing environments: wireless communication, mobility, and portability [14]:

Wireless Communication: mobile computers rely heavily on wireless network access for communication. Lower bandwidths, higher error rates, and more frequent spurious disconnections often characterize wireless communication. These factors can in turn lead to an increase in communication latency arising from retransmission, retransmission time-out delays, error control protocol processing, and short disconnections. Mobility can also cause wireless connections to be lost or degraded. A mobile user may travel beyond the coverage area or may enter an area of high interference. Thus, wireless communication leads to challenges in the areas of:

1- Disconnection: Wireless networks are inherently more prone to disconnection. Since computer applications that rely heavily on the network may cease to function during network failures, proper management of

disconnection is of vital importance in mobile computing. Autonomy is a desirable property that allows the mobile client to deal with disconnection. The more autonomous a mobile computer is, the better it can tolerate network disconnection. Autonomy allows the mobile unit to run applications locally. Thus, in environments with frequent disconnections, it might be better for a mobile device to operate as a stand-alone device. In order to manage disconnection, a number of techniques such as caching, asynchronous operation and other software techniques may be applied. Maintaining cache consistency is difficult however, since disconnection and mobility severely inhibit cache consistency. Cache consistency techniques employed in traditional architectures designed for fixed hosts may not be suitable for the mobile computing environment. Asynchronous operations can be used to mask round-trip latency and short disconnections. Software techniques such as prefetching and delayed write-back can also be used to minimize communication, thus allowing an application to proceed during disconnection by decoupling the communication time from the computation time of a program [15]. Delayed write back takes advantage of the fact that data to be written may undergo further modification. Operation queuing can also help; operations that cannot be carried out while disconnected can be queued and done when reconnection occurs.

2-Limited Bandwidth: Wireless networks deliver lower bandwidth than wired networks. Cutting-edge products for portable wireless communication achieve only 1 megabit per second for infrared communication, 2 Mbps for radio communication, and 9–14 kbps for cellular telephony. On the other hand, Ethernet provides 10Mbps, fast Ethernet and FDDI, 100 Mbps, and ATM (Asynchronous Transfer Mode) 155 Mbps (37). Available bandwidth is often divided among users sharing a cell. Thus, bandwidth utilization is of vital importance. Software techniques such as compression, filtering, and buffering before data transmission, can be used to cope with low bandwidth. Other software techniques such as perfecting and delayed-write back that are used to cope with disconnection can also help to cope with low bandwidth. A large dynamically changing number of mobile clients are a characteristic

of a mobile computing environments. Thus, bandwidth contention is a problem. Caching can help to reduce bandwidth contention, which also helps to support a disconnected operation.

3- High Bandwidth Variability: bandwidth may vary many orders of magnitude depending on whether a mobile client is plugged in or communicates via wireless means. Bandwidth variability is treated by traditional existing systems as exceptions or failures [15]. However, this is the normal mode of operation for mobile computing. Applications must therefore have the ability to adapt to the available bandwidth and should be designed to run on full bandwidth or minimum bandwidth.

Mobility: The ability to change location while retaining network connection is the key motivation for mobile computing. As mobile computers move, they encounter heterogeneous networks with different features. A mobile computer may need to switch interfaces and protocols; for example a mobile computer may need to switch from a cellular mode of operation to a satellite mode as the computer moves from urban to rural areas or from infrared mode to radio mode as it moves from outdoors to indoors. Traditional computers do not move, therefore, certain data that are considered to be static for stationary computing becomes dynamic for mobile computing. For example, a stationary computer can be configured to print from a certain printer attached to a particular print server, but a mobile computer needs a mechanism to determine which print server to use. A mobile computer's network address changes dynamically. Its current location affects configuration parameters as well as answers to user queries. If mobile computers must serve as guides, location–sensitive information may need to be accessed. Thus, mobile computers need to be aware of their surroundings and have the ability to find location dependent information automatically and intelligently while maintaining system privacy. Mobility can also lead to increased network latency and increased risk of disconnection. Cells may be serviced by different network providers and may employ different protocols. The physical distance may not reflect the true network distance and therefore a small movement may result in a much longer path if a cell or network boundary is crossed. Transferring service connection to the nearest server is desirable but this may not be possible if load balancing is a

key priority. Security considerations exist because a wireless connection is easily compromised. Appropriate security measures must be taken to prevent unauthorized disclosure of information. Encryption is necessary to ensure secure wireless communication. Data stored on disks and removable memory cards should also be encrypted. The amount of data stored locally should be minimal. Backup copies must be propagated to stationary servers as soon as possible as is done in replicated systems.

Portability: Designers of desktops take a liberal approach to space, power, cabling, and heat dissipation in stationary computers that are not to be carried about. However, designers of mobile computers face far more stringent constraints. Mobile computers are meant to be small, light, durable, operational under wide environmental conditions, and require minimal power usage for long battery life. Concessions have to be made in each of the areas to enhance functionality. Some of the design pressures that result from portability constraints include:

a) Low Power: Batteries are the largest single source of weight in portable computers. Reducing battery weight is important, however too small a battery can undermine the value of portability leading to: i) frequent recharging, ii) the need to carry spare batteries, or iii) make less use of the mobile computers. Minimizing power consumption can improve portability by reducing battery weight and lengthening the life of the battery charge. Chips can be designed to operate at lower voltages. Individual components can be powered down when they become idle. Applications should be designed to require less communication and computation. Preference should be given to listening rather than transmitting since reception consumes a fraction of the power it takes to transmit.

b) Limited User Interface: Display and keyboard sizes are usually limited in mobile computers as a consequence of size constraints. The amount of information that may be displayed at a time is limited as a result. Present windowing techniques may prove inadequate for mobile devices. The size constraint has also resulted in designers abandoning buttons in favour of

analogy input devices for communicating user commands. For instance, pens are now the standard input device for PDAs because of their ease of use while mobile, their versatility, and their ability to supplant the keyboard.

c) Limited Storage capacity: Physical size and power requirements effectively limit storage space on portable computers. Disk drives, which are an asset in stationary computers, are a liability in mobile computers because they consume more power than memory chips. This restricts the amount of data that can be stored on mobile devices. Solutions include compressing files systems, accessing remote storage over the network, shared code libraries, and compressing virtual memory. Table 2.2 summarizes these issues and their effect on traditional issues of concern in database environments.

Table2. 2: Characteristics of Mobile Environment

Mobile Environment Characteristics	**Resulting Issues**
1- wireless Connection	Disconnection
	Communication Channel
	High cost
	Network
	Low Data Rate
2- Mobility	Motion Management
	Location-Dependent Data
	Heterogeneous Networks
	Interfacing
	Data-Rate Variability
3-Portability	Limited Resources
	Limited Energy Sources
	User Interface

2.4 Mobile computing verses distributed computing

A mobile computing system is a dynamic type of distributed system where links between nodes in the network change dynamically. Thus, we cannot rely on a fixed network structure. A single site cannot play the role of co-ordinator as in a centralized system. The mobile host and FHs also differ in computational power and memory. The distributed algorithms for mobile environments should be structured such that the main bulk of the communication and computation costs are borne by the static portion of the network. In [10], there is the idea of associating with each mobile host a proxy on the static network, thus decoupling mobility from the design of the algorithm. Many of the solutions for distributed computing problems may not work in the mobile computing area. Table 2.3 summarises some differences between these environments.

Table2. 3: Distributed environments versus mobile environments

	Distributed	**Mobile environments**
Computing hosts	Stationary sites. Powerful computing capacity. Reliable computing hosts	Mobile and non-mobile hosts. Limited computing capacity of mobile hosts. Less reliable computing hosts.
Network connectivity	Wired and high-speed networks. Reliable networks	Wireless, unstable and low speed networks Unreliable, error-prone, frequent and long disconnection periods.

In a mobile environment, a DBMS also needs to be able to recover from site, network and transaction failure, as in the case of distributed systems. However, the frequency of most of these failures increases and mobility complicates the recovery. Site failures at a MU may be frequent due to limited battery power. Also, a MU may be in *doze mode* (shutdown), which cannot be treated as failure. Furthermore, mobility may force more logging in order to recover from failure. Caching at a MU is an interesting idea to optimize the use of wireless connections by increasing availability. Its application in the WWW environment is very useful

where size of data in enormous. However, maintaining cache consistency is an important objective and different consistency requirements can be used depending on the applications. Caches need to be updated frequently and thus, need new update protocols. Replication in mobile environment certainly increases availability but may need certain weaker consistency criteria [16]. Also, replication schemes for distributed systems may not be directly applicable here and there is a need for dynamic replication schemes [17]. Another important area is query processing. In mobile environments, queries may need to be distributed at least in two places. Part of query may be executed at a MU and another part may be at a FH with the help of the DBS. Another interesting issue is location-dependent query processing in mobile environments where queries return results according to the location [18, 19]. The same query may return different results at different locations. Here, replication of data has a different meaning to that in traditional distributed databases where all copies of data objects keep the same consistent values. In location-dependent data management, the same object in different locations may have different values but still these values are considered as consistent. For example, a tax object has different values in different states in the United States. The most important remaining issue is transaction processing in such environments. Transaction failures may increase due to the possibility of a problem during hand-off when the MU moves between cells. An MU failure creates a partitioning of the network, which in turn complicates updating and routing algorithms. Another major difference lies in the transaction model. Unlike a distributed transaction, a mobile transaction is not identified by a cell or a remote site. It is identified by the collection of cells it passes through. A distributed transaction is executed concurrently on multiple processors and data sets. The execution of the distributed transaction is co-ordinated fully by the system including concurrency control, replication and atomic commit. A mobile transaction, on the other hand, is executed sequentially through multiple base stations, and on possibly multiple data sets, depending on the movement of the MU. The execution of the mobile transaction is thus not fully co-ordinated by the system. The movement of the MU controls the execution.

2.5 Traditional transaction models

As the transaction environment evolves from the centralized environment to distributed and mobile environments, the properties and the structure of transactions change. However, several basic transaction models are indispensable. In other words, they are still useful and applicable in the new mobile environments. In this section, we will review the following transaction models:

- *Flat* transaction model [20]
- *Nested* transaction model [21]
- *Multilevel* transaction model [22, 2]
- *Sagas* transaction model [23]
- *Split and Join* transaction model [24]
- *Flexible* transaction model [3]

For each transaction model, we briefly describe the transaction model, the properties and discuss how the features of the transaction model could be used in mobile environments.

2.5.1 Flat transaction model

Description. The flat transaction model [20, 3] presents the simplest transaction structure that fully meets the ACID properties. Figure2.16 illustrates the structure of a flat transaction. The building block of a flat transaction, between Begin and Commit /Abort operations, contains all the database operations that are tightly coupled together as one atomic database operation. A flat transaction begins at one consistent database state, and either ends in another consistent state, i.e., the transaction commits, or remains in the same consistent state, i.e., the transaction aborts.

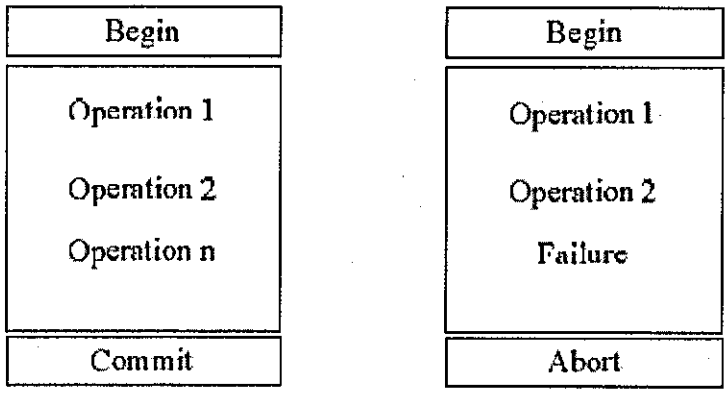


Figure2. 16: Flat transaction model

Transaction properties. The flat transaction model fully meets the standard ACID properties. A flat transaction is fully isolated during its execution, and any failure causes the whole transaction to abort. The results of a committed flat transaction are durable and permanent.

Usefulness for mobile environments. Due to the strict ACID properties, the flat transaction model is not suitable in mobile environments. However, the flat transaction model plays an important role for building more advanced transaction models. For example, a complicated transaction model can consist of a set of smaller flat transactions. The flat transaction model can be easily supported at the application programming level.

2.5.2 Nested transaction model

Description. The nested transaction model [21] defines the concepts and the mechanisms for breaking up the large building block of a flat transaction into a set of smaller transactions, called *subtransactions*. Thus, the nested transaction model has a hierarchical tree structure that includes a top-level transaction and a set of subtransactions (either parent or children transactions). Subtransactions at the leaf level of the transaction tree are flat transactions.

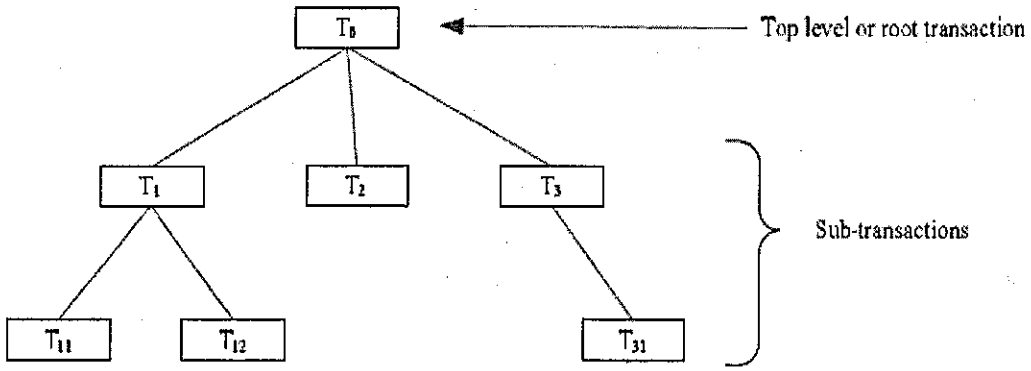


Figure2. 17: Nested transaction model

Transaction properties. The nested transaction model has the following characteristics. First, children transactions are flat transactions. Second, the children transactions start after their parent has started, and can autonomously commit or abort. However, the results of the committed children transactions do not take effect until their parent transactions commits. In other words, the nested transaction only commits when the top level transaction commits. And third, when a child transaction commits, its results become visible to its parent transaction. If a parent or the top-level transaction aborts, all the subtransactions must abort, regardless of their states.

Usefulness for mobile environments. The concept of the nested transaction model can be applied in mobile environments, especially for decomposing a large transaction into subtransactions which can be carried out concurrently.

2.5.3 Multilevel transaction model

Description. The multilevel transaction model [22, 2] is looser than the nested transaction model in terms of the relationship between parent and children transactions. Subtransactions in the multilevel transaction can commit or abort independently of their parents. This is supported by the concepts of compensating transactions. We will briefly discuss the concept of *compensating transactions,* and are opposite to *contingency transactions* (see Figure2.18). Compensating transactions [3] are designed to undo the effect of the original transactions that have aborted. They are triggered and started when the original transactions fail. Otherwise, the compensating transactions are not initiated. Once a compensating transaction has started, it must commit. In other words, the compensating

transactions cannot abort. If a compensating transaction fails, it will be restarted. Contingency transactions [2] are designed to replace the task of the original transactions that have failed. Contingency transactions are also triggered by the failures of the original transactions. Note that it is not always possible to specify the compensating or contingency transactions for an original transaction.

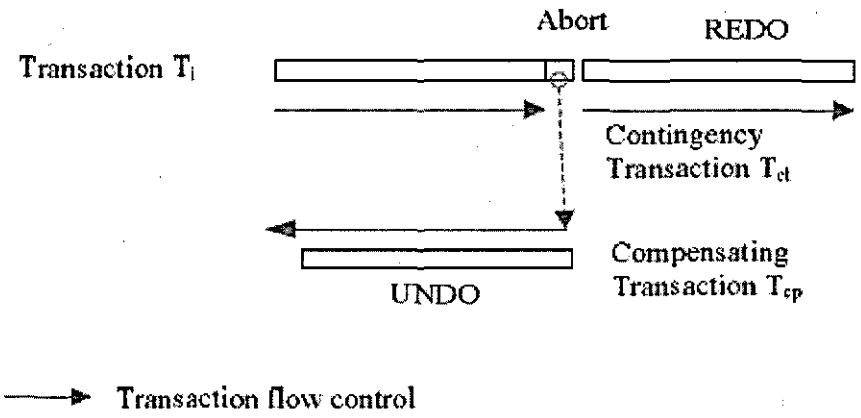


Figure2. 18: Compensating and contingency transactions

Transaction properties. The isolation property is relaxed in the multilevel transaction model. The committed results of subtransactions are visible to other transactions. The atomicity property is ensured by means of compensating transactions.

Usefulness for mobile environments. The multilevel transaction model is applicable in mobile environments. It not only relaxes the isolation property of transactions but also provides a flexible recovery mechanism by means of the compensating and contingency transactions.

2.5.4 Sagas transaction model

Description. The Sagas transaction model [23] also makes use of the concept of compensating transactions to support transactions whose execution time is long. A Sagas transaction consists of a consecutive chain of flat transactions S_i that can commit independently. For each flat transaction S_i, there is a compensating transaction CP_i that will undo the effect of transaction S_i if transaction S_i aborts. A compensating transaction CP_i in the Sagas chain is triggered by the associated transaction S_i or the compensating transaction CP_{i+1}. If the Sagas transaction

commits, no compensating transaction CP_i is initiated (see Figure2.19), otherwise the chain of compensating transactions is triggered (see Figure 2.20).

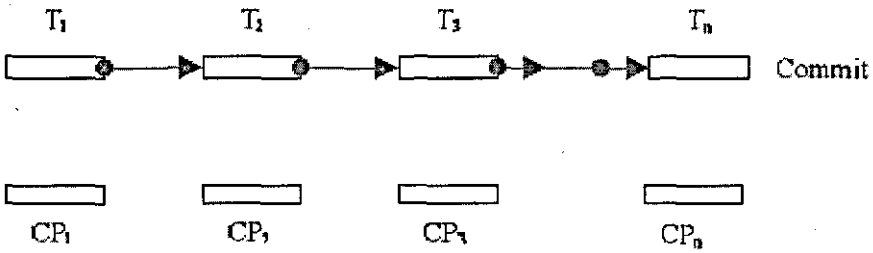


Figure2. 19: A successful Sagas

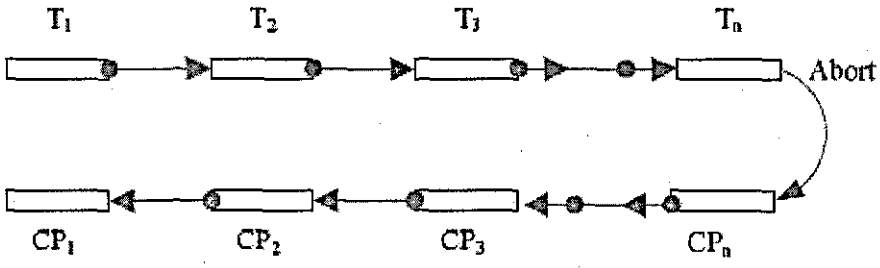


Figure2. 20: An unsuccessful Sagas

Transaction properties. The unit of control of a Sagas transaction is the whole transaction chain. Sagas relaxes the isolation property by allowing component transactions S_i to commit. The atomicity property of Sagas is achieved by the commitment of the last transaction component S_n in the chain or by the backward execution of the compensating transaction chain.

Usefulness for mobile environments. The Sagas transaction model is useful in mobile environments because of its ability to support transactions that are long-lived. The isolation property is also compromised. Therefore, the concept can be used to support sharing of data during the execution of mobile transactions. Moreover, it is possible to modify the Sagas model so that we can minimize the losing of useful work when a component transaction S_i aborts, for example by deploying contingency transactions instead of compensating transactions. The main drawback of Sagas is the dependence on the previous component transactions in the chain.

2.5.5 Split and Join transaction model

Description. The Split and Join transaction model [24] was proposed to support the open ended activities that are associated with transactions. The Split and Join transaction model focuses on activities that have uncertain duration, uncertain developments, and are interactive with other concurrent activities. The main idea is to divide an on-going transaction into two or more serializable transactions, and to merge the results of several transactions together as one atomic unit. In other words, the Split and Join transaction model supports reorganizing the structure of transactions (as illustrated in Figure 2.21).

Transaction properties. The Split and Join transaction model divides the accessed data set of a transaction into different subsets that will be used by newly created and serializable transactions. The goal is to commit part of the original transaction and to make committed results or resources available to other transactions.

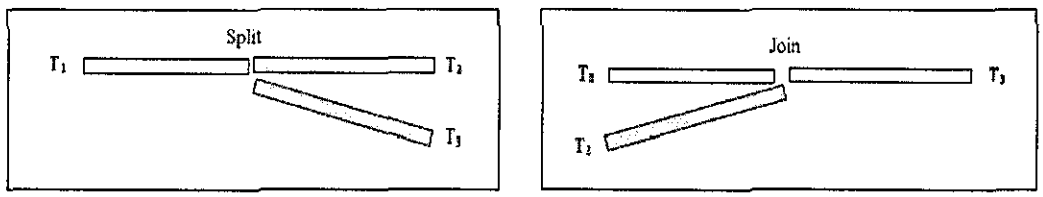


Figure2. 21: Split and Join transaction model

Usefulness for mobile environments. The Split and Join transaction model benefits transactions in mobile environments in terms of dynamic re-structuring of transactions.

2.5.6 Flexible transaction model:

The flex transaction model [4], used in the InterBase project at Purdue University, relaxes the atomicity and isolation properties of subtransactions to provide users with increased flexibility in specifying transactions. This model was proposed for multidatabase transaction management to provide an extended transaction model. The main features of this model are:

- It allows the user to give a set of acceptable states, i.e., it allows specification of a set of functionally equivalent subtransactions. This allows failure tolerance since

it takes advantage of the fact that a given function can be executed in more than one way.

- Users can define the execution order of transactions in terms of internal dependencies and external dependencies of transactions.
- It allows the concept of mixed transactions. This allows compensatable and non compensatable transactions to coexist.
- It allows the user to control the isolation granularity of a transaction through use of compensating transactions.

.Transaction properties. The isolation is relaxed in the flexible transaction model by supporting alternative executions for transaction. The atomicity property is ensured by means of compensating transactions.

Usefulness for mobile environments. The flexible transaction model is applicable in mobile environments. It not only relaxes the isolation property of transactions but also provides a flexible recovery mechanism by means of the compensating transactions and alternative executions. This extended model is particularly useful where local autonomy is of concern.

Table2. 4: Comparison of traditional transactional models

Transactional Support	Compensating Transactions	Sub- Transactions	Pre-Commit	ACID Relax. (*)
Flat	X	X	X	X
Nested	X	√	X (1)	X
Multilevel	√	√	X (1)	X
Sagas	√	√	√ (2)	ACI
Split	X	√	√	ACI
Flexible	√	√	√	ACI

(*) In this column we show the letter of the ACID acronym that is relaxed or eliminated from the corresponding model. For instance, if A appears, it means that the Atomicity property is relaxed.
(1) Only the top level and the siblings can access the changes made by a lower level commit operation.
(2) If compensating transactions are used, it is possible to pre-commit in order to publish the changes made to the outside world.

2.6 Mobile transaction models

We have reviewed several traditional transaction models whose features are still useful in mobile environments. The traditional transaction models, however, do not have the ability to deal with other challenging requirements of mobile transactions, such as supporting the mobility of transactions and coping with disconnections. Consequently, there are many advanced transaction models that have been developed to particularly support mobile transactions. In this section, we will review several selected mobile transaction models that have the ability to efficiently support mobile transactions. The follows mobile transaction models will be surveyed:

- *Report* and *Co-transaction* model [25]
- *Pro-motion* transaction model [26]
- *Two-tier* transaction model [27]
- *Weak-Strict* transactions model [28]
- *Pre-write* transaction model [29, 30]
- *Pre-serialization* transaction model [13]
- *Kangaroo* transaction model [31]
- *Moflex* transaction model [32]

For each model, we describe the transaction model and its properties, and then we address how the model: (1) handles the *mobility of transactions*, (2) deals with *disconnections*, and (3) supports *distributed transaction execution* among mobile and non-mobile hosts.

2.6.1 Reporting and Co-transaction model

Description. The *Reporting* and *Co-transactions* transaction model [25] is based on a two-level nested transaction model (see Figure 2.22). A reporting transaction T_R shares its partial results with a top-level transaction S by delegating its operations. The delegation process can happen at any time during the execution of transaction T_R. A co-transaction is a reporting transaction but it cannot continue

executing during the delegation process. Thus, the co-transaction behaves as a co-routine, and resumes execution when the delegation process is completed.

Transaction properties. The top-level transaction is the unit of control, and atomic subtransactions are compensatable transactions. A Reporting transaction that is compensatable does not have to delegate all of the committed results to the top-level transaction when it commits. Subtransactions that are non-compensatable delegate all of their operations to the top-level transaction when it commits.

Mobility. The locations of mobile hosts are determined via the identification of mobile support stations. However, the model does not mention explicitly what happens when mobile hosts move from one mobile cell to another.

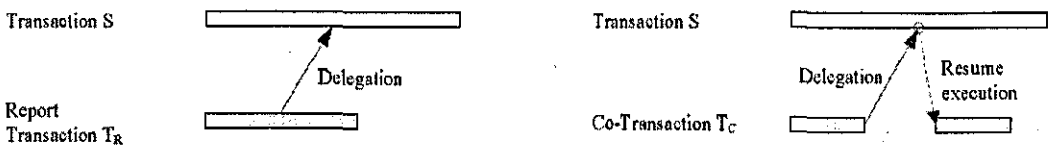


Figure2. 22: Reporting and Co-transaction

Disconnection. Delegation operations require a tight connectivity between the delegator (i.e., Report and Co-transaction) transactions and the delegatee transaction (i.e., the top level transaction). Therefore, disconnection is not supported in this model.

Distributed execution. The model supports distributed transaction processing among mobile hosts and fixed hosts where the network connectivity among these hosts is assumed to be available when it is needed.

2.6.2 Pro-motion transaction model

Description. The Pro-motion transaction model [26] is a nested transaction model. The Pro-motion model focuses on supporting disconnected transaction processing based on the client-server architecture. Mobile transactions are considered as long nested transactions where the top-level transaction is executed at fixed hosts, and subtransactions are executed at mobile hosts. The execution of subtransactions at mobile hosts is supported by the concept of *compact* objects (see Figure 2.23).

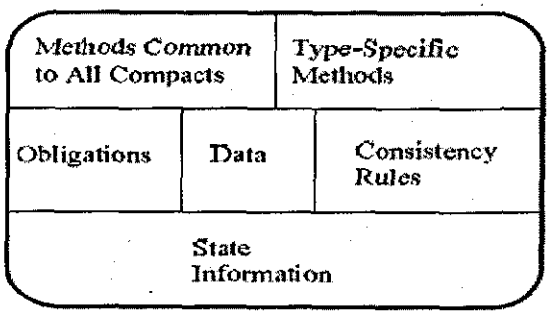


Figure2. 23: Compacts as objects

Compact objects are constructed by a compact manager at database servers. Necessary information is encapsulated within a compact object. The compact objects are co-managed by the compact managers (residing at the database servers), the mobility managers (at the mobile support stations), and the compact agents (at the mobile hosts). The compact object plays a role as a contractor that supports data replication and consistency between mobile hosts and database servers. When a mobile host is disconnected, the compact agent takes responsibility for managing all local database operations of mobile transactions at the mobile host. When the mobile host reconnects to database servers, the compact objects are verified against global consistency rules before the locally committed mobile transactions are allowed to commit. Figure 2.24 shows the architecture of the Pro-motion transaction model. Transaction processing consists of four phases: *hoarding, disconnected, connected,* and *resynchronization.* Shared data is downloaded to the mobile host in the hoarding phase. When the mobile host is disconnected from the fixed host, transactions are disconnectedly executed at the mobile host. If the mobile host connects to the fixed database, the transactions are carried out with the support of the compact manager. When the mobile host reconnects to a fixed host, the results of local transactions are synchronised with the database.

Transaction properties. The Pro-motion transaction model supports ten different levels of isolation. Transactions are allowed to locally commit at mobile hosts; the committed results of these transactions are made available to other local transactions. However, the local committed results must be validated when the mobile hosts reconnect to the database servers. Therefore, the durability property

of transaction is only ensured when the transaction results are finally reconciled at the fixed database.

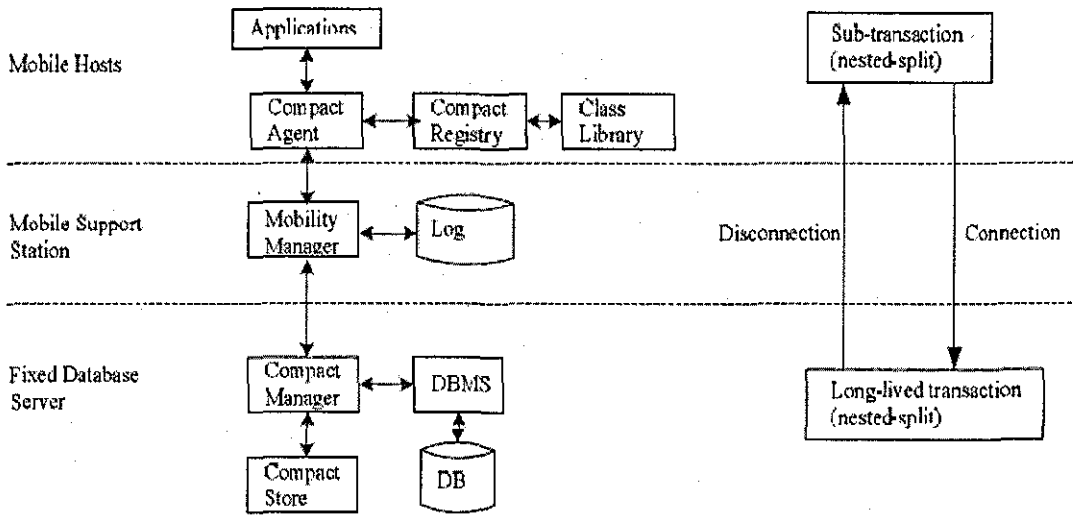


Figure2. 24: Pro-motion transaction architecture

Mobility. Though the mobility manager supports communications between the mobile host and the database servers, how the Pro-motion transaction model supports transaction mobility is not explicitly discussed.

Disconnection. The Pro-motion transaction model supports disconnected transaction processing via the support of compact objects. When the mobile host is disconnected from the fixed database, the subtransactions are split and executed at the mobile host (these split subtransactions are not joined when the mobile host reconnects to the fixed database). Disconnected transaction processing is a dominant transaction processing mode in Pro-motion even when the mobile hosts are able to connect to the database server. Therefore, the Pro-motion transaction model requires high-capacity mobile resources at the mobile hosts.

Distributed execution. Transactions are mostly executed at mobile hosts and the results are reconciled at the database servers. Therefore, distributed transaction processing is not strongly supported by the model.

2.6.3 Base-Tentative transaction model

Description. The Base-Tentative transaction model [27] is based on a data replication scheme. For each data object, there is a master copy and several replicated copies. There are two types of transaction: *Base* and *Tentative*. Base

transactions operate on the master copy; while tentative transactions access the replicated copy version. A mobile host can cache either the master or the copy versions of data objects. While the mobile host is disconnected, tentative transactions update replicated versions. When the mobile host reconnects to the database servers, tentative transactions are converted to base transactions that are re-executed on the master copy. If a base transaction does not fulfil an acceptable correctness criterion (which is specified by the application), the associated tentative transaction is aborted. The two-tier transaction model is shown in Figure 2.25.

Transaction properties. Tentative transactions locally commit at the mobile host on replicated copies, and the committed results are made visible to other tentative transactions at that mobile host. The final commitments of those tentative transactions are performed at the database servers.

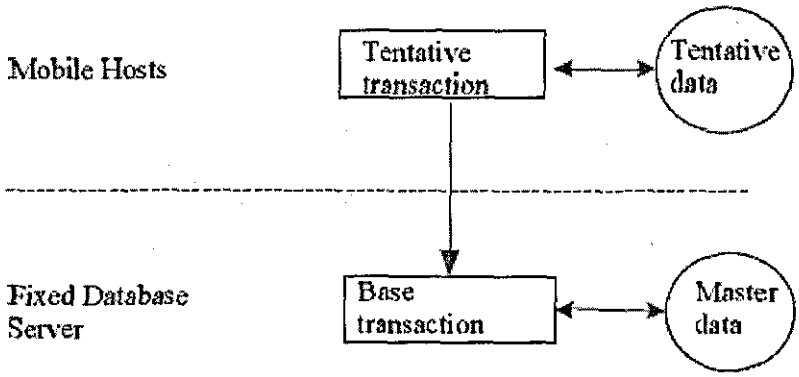


Figure2. 25: Two-tier transaction model

Mobility. The Two-tier transaction model does not support the mobility of transactions.

Disconnection. While the mobile hosts are disconnected from the database servers, tentative transactions are locally carried out based on replicated versions of data objects.

Distributed execution. Two distinct transaction execution modes are supported: connected and disconnected. Transactions are tentatively carried out at

disconnected mobile hosts, and re-executed as base transactions at the database servers.

2.6.4 Clustering transaction model

Description. The Weak-Strict (also called Weak-Strict model [28] consists of two types of transaction: *weak (or loose)* and *strict*. These transactions are carried out within the *clusters* that are the collection of connected hosts which are connected via high-speed and reliable networks. In each cluster, data that is semantically related is locally replicated. There are two types of a replicated copy: *local consistency (weak)* and *global consistency (strict)*. The weak copy is used when mobile hosts are disconnected or connected via a slow and unreliable network. Weak and Strict transactions access weak and strict data copies, respectively. Figure 2.26 presents the architecture of this transaction model. When mobile hosts reconnect to database servers, a synchronization process reconciles the changes of the local data version with the global data version.

Transaction properties. Weak transactions are allowed to commit within its cluster, and results are made available to other local weak transactions. When mobile hosts are reconnected, the results of weak transactions are reconciled with the results of strict transactions. If the results of a weak transaction do not conflict with the updates of strict transactions, weak transactions are globally committed; otherwise they are aborted.

Mobility. The concept of transaction migration is proposed to support the mobility of transactions, and to reduce the communication cost. When the mobile host moves and connects to a new mobile support station, parts of the transaction that are executed at the old mobile support stations are moved to the new one. However, no further details about the design or implementation are given.

Disconnection. The Weak-Strict transaction model supports transaction processing in disconnected and weakly connected modes via weak transactions.

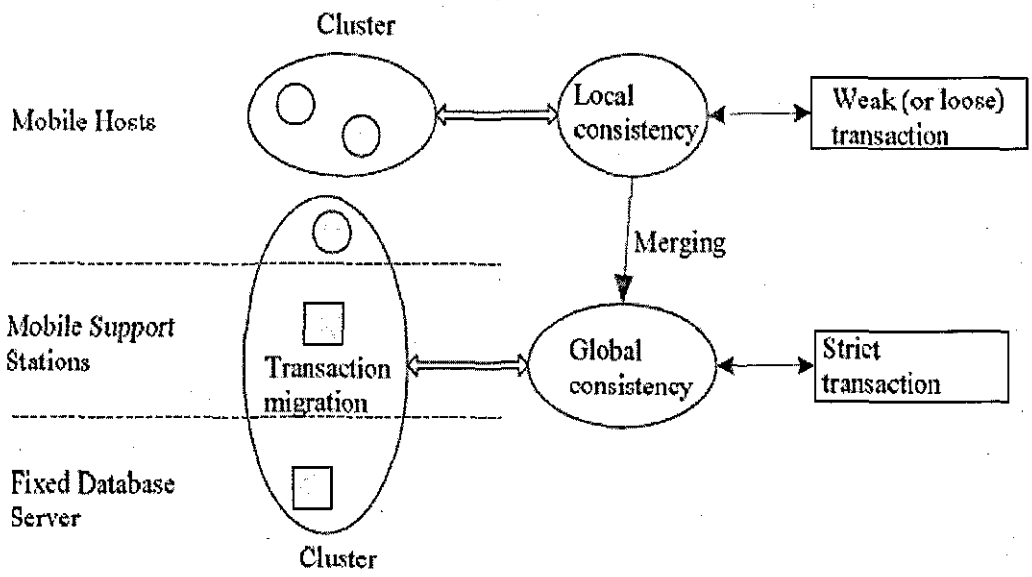


Figure2. 26: Weak-Strict transaction model

Distributed execution. Transaction execution processes can be distributed between the mobile host and the database servers within a cluster that the mobile host participates in. However, the distributed transaction processing among mobile hosts in a cluster is not discussed.

2.6.5 Pre-write transaction model

Description. The Pre-write transaction model [29, 30] was proposed to increase data availability in mobile environments. Mobile transactions are transactions that are initiated at the mobile host. The Pre-write transaction model aims to increase data availability at mobile hosts. This is achieved by allowing a transaction on a mobile host to submit pre-write operations that write the updated data values, and then issue a pre-commit state to the mobile support station. After that, the rest of the mobile transaction can be carried out and finally committed at fixed hosts. The small variation, which is specified by the applications, between the pre-committed result and the final committed result is acceptable. Pre-committed data values are accessible to other transactions via pre-read operations. Two different types of lock, the *pre-read* and *pre-write* locks, are introduced to support the new operations. Mobile transactions are not allowed to abort after they have submitted pre-commit operations to the mobile support station. This mobile transaction model can be used to support mobile hosts which have little or no capacity for transaction processing.

Transaction properties. After a mobile transaction submits a pre-commit request, the pre-write values of the mobile transaction are made available to transactions. The pre-committed mobile transaction is not aborted in any case. The final commits of mobile transactions will be carried out by fixed hosts. The final commits and the pre-committed data values may not be identical.

Mobility. The roles of the mobile support station are to accept and to process pre-write and pre-commit operations submitted from the mobile host. When moving into a new mobile cell, a mobile transaction connects to the mobile support station in order to submit its pre-write and pre-commit operations.

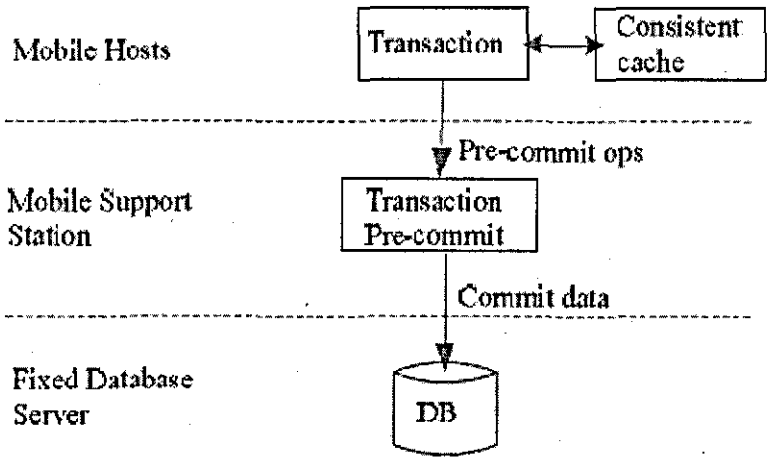


Figure2. 27: Pre-write transaction model

Disconnection. Disconnected transaction processing is supported in the Pre-write transaction model. The mobile transaction is executed at the mobile host until the pre-commit state is reached.

Distributed execution. The major part of the mobile transaction is migrated to the fixed hosts via the mobile support station to be executed there. The mobile host partly takes part in the execution process until the pre-commit states of the mobile transaction are achieved. After this, the mobile host plays no role in the execution of the mobile transaction.

2.6.6 Pre-serialization transaction model

Description. The Pre-serialization transaction model [13] is built on top of local database systems. Mobile transactions (also called global transactions) are submitted from mobile hosts through the global transaction coordinators that

reside at the mobile support stations. The mobile transactions are entirely processed at local database systems (see Figure 2.28). At each node (or site), there is a site manager that administrates all the transactions executed at that node. When a global transaction is prepared to commit, a global transaction coordinator will carry out an algorithm, called the Partial Global Serialization Graph algorithm, which detects any non-serializable schedule among the mobile transactions. If there is a cycle in the graph, i.e., the schedule is non-serializable, the mobile transaction is aborted.

Transaction properties. Each subtransaction of a global transaction is managed by the local transaction manager. The global serializable graph of transactions is constructed by collecting sub-graphs from the local sites. The atomicity property of the global transaction is relaxed through concepts of vital and non-vital subtransactions. If a vital subtransaction aborts, its parent transaction must abort. However, the parent transaction does not abort if a non-vital subtransaction aborts. When a subtransaction commits at the local database system, the results are made visible to other transactions at this local database system.

Mobility. The global transaction coordinators that reside at the mobile support stations support the mobility of mobile transactions. This is done by transferring the global data structure from one global transaction coordinator to another as the mobile host moves from one mobile cell to another.

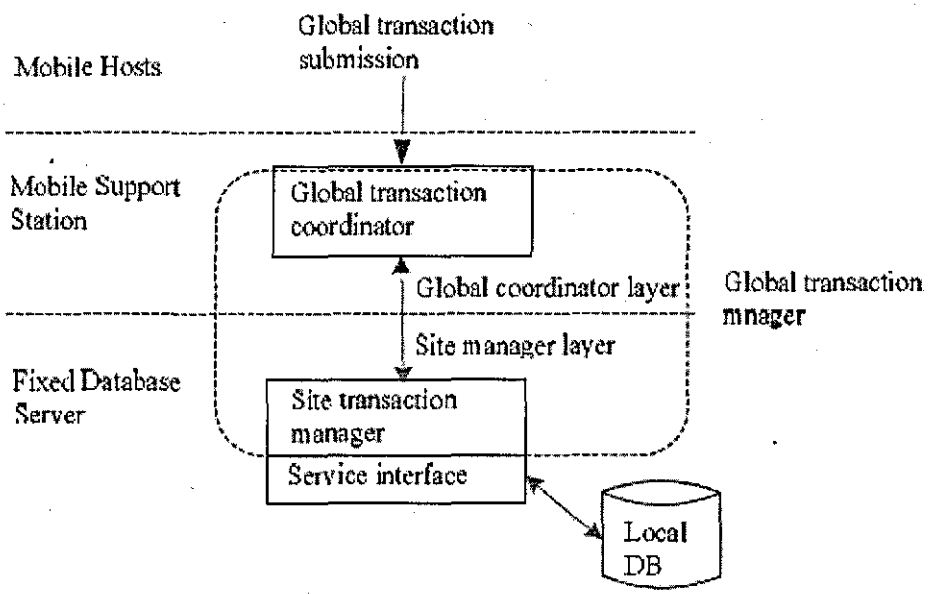


Figure2. 28: Pre-serializable transaction model

Disconnection. Mobile transactions are submitted from a mobile host, and subtransactions are executed at local database servers. When the mobile host is disconnected, the global transaction is marked as disconnected if the disconnection is known and planned. The execution of the global transaction is still carried out at the local database servers. On the other hand, if the disconnection is unplanned, the global transaction is suspended. The global transaction is resumed when the mobile host reconnects to the mobile support station.

Distributed execution. Mobile transactions are submitted from mobile hosts, and the entire transactions are distributed among local database servers through the support of mobile support stations. The mobile hosts do not take part in the execution process.

2.6.7 Kangaroo transaction model

Description. The Kangaroo transaction model [31] is designed to capture the movement behaviour and the data behaviour of transactions when a mobile host moves from one mobile cell to another. This transaction model is built based on the concepts of global and split transactions in a heterogeneous and multi-database environment. The global transaction is split when the mobile host moves from one mobile cell to another, and the split transactions are not joined back to the global transaction. The Kangaroo transaction model assumes that the mobile transactions may start and end at different locations. The characteristics of the Kangaroo transaction model are (see Figure 2.29 for the architecture of the Kangaroo transaction model):

- Mobile transactions that include a set of subtransactions called global and local transactions are initiated by mobile hosts. These mobile transactions are entirely executed at the local database servers that reside on the fixed and wired connected networks.
- The execution of a Kangaroo subtransaction in each mobile cell is supported by a Joey transaction that operates in the scope of the mobile support station. The

Joey transaction has the role of a proxy transaction that supports the execution of the subtransactions of the Kangaroo transaction in the mobile cell.

• The movement of the mobile host from one mobile cell to another is captured by the splitting of the on-going Joey transaction at the old mobile support station and the creating of a new Joey transaction at the new mobile support station. The execution of the Joey transaction is supported by the Data Access Agents (DAA) that act as the mobile transaction managers at the mobile support stations.

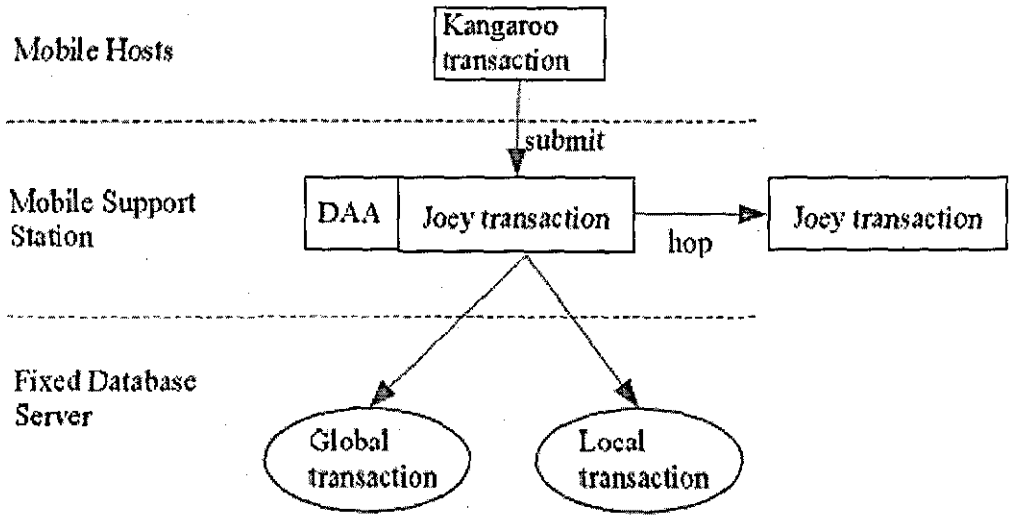


Figure2. 29: Kangaroo transaction model

Transaction properties. The Kangaroo transaction is the basic unit of computation in mobile environments. The serializability of mobile transactions is not guaranteed, and there is no dependency among Joey transactions, i.e., each Joey transaction can commit independently. Two transaction processing modes, which are *compensating* and *split modes*, are supported by the model. In the compensating mode, when a failure occurs, the entire Kangaroo transaction is undone by executing compensating transactions for all those Joey transactions. In the split mode, the local DBMS takes responsibility for aborting or committing subtransactions.

Mobility. The Kangaroo transaction model keeps track of the movement of mobile hosts via the support of the DAA that operates at the mobile support station. In other words, the mobility of mobile hosts is captured on the condition that the mobile hosts always may communicate with the mobile support stations. While

mobile hosts move from one mobile cell to another, the hand-off processes are carried out by the DAAs.

Disconnection. Disconnected transaction processing is not considered in the Kangaroo transaction model. The processing of Kangaroo transactions is entirely moved to the fixed database servers for execution.

Distribution. The mobile transactions are initiated at the mobile hosts, and executed entirely at fixed hosts. Transaction results are forwarded back to the mobile hosts. The Kangaroo transaction model has shown that the structure of mobile transactions at the specification and execution phases (with the dynamic support of Joey transactions) can be different because of the mobility behaviour, i.e., fast or slow movements, of the mobile host.

2.6.8 Moflex transaction model

Description. The Moflex transaction model [32] is an extension of the Flex transaction model [33] to support mobile transactions. The Moflex model is built on top of multi-database systems and based on the concepts of split-join transactions. The main characteristics of a Moflex transaction are:

- A Moflex transaction consists of compensatable or non-compensatable subtransactions initiated by the mobile host. These subtransactions are submitted to the mobile transaction manager (MTM) that resides at the mobile support station. The MTM will send these subtransactions to the local execution monitor (LEM) at local database systems for execution. Figure 2.30 presents the architecture of the Moflex transaction model.

- Each Moflex transaction T is accompanied by a set of success and failure transaction dependency rules, hand-over control rules (see Table 2.5), and acceptable goal states. External factors include the execution time, cost and execution location of transactions. Furthermore, joining rules are provided to support the join of the split subtransactions (subtransactions are split when the mobile host moves from one mobile cell to another).

Transaction properties. The mobile transaction managers make use of the two-phase commit protocol to coordinate the commitment of the Moflex transaction. The Moflex transaction commits when its subtransactions that are managed by a

MTM have reached one of the acceptable goal states, otherwise it is aborted. A compensatable subtransaction is locally committed, and the results are made visible to other transactions. For noncompensatable subtransactions, the last mobile transaction manager, which corresponds to the end location of the mobile host, plays the role of the committing coordinator.

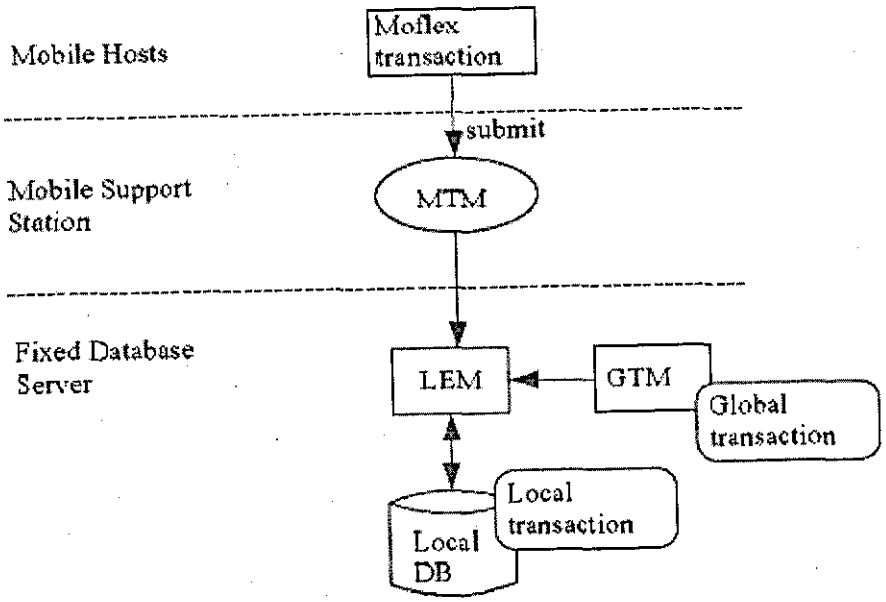


Figure2. 30: Moflex transaction model

Mobility. The mobility of transactions is handled by splitting the subtransaction which is executed on the local database at the current mobile cell, as the mobile host moves from one mobile support station to another (with the support of the mobile transaction manager). Hand-over control rules must be specified for each subtransaction (see Table 2.5). If a subtransaction is compensatable and location independent, it will be split into two transactions; one will continue and commit at the current local database, the second will be resumed at the new location. If the subtransaction is location dependent, at the new location, the subtransaction must be restarted. If a subtransaction is noncompensatable, the subtransaction is either restarted as a new one in the mobile cell if it is location dependent, or continued if it does not depend on the location of the mobile host.

Table2. 5: Hand-over control rules of subtransactions

	Compensable	**Non compensable**
Location independent	split_resume	continue
Location dependent	restart, split_restart restart	restart

Disconnection. The Moflex transaction model does not support disconnected transaction processing. The Moflex transaction model requires network connectivity between the mobile host and the mobile support stations during the execution process.

Distributed execution. The execution of a Moflex transaction is transferred to local database systems at fixed hosts to be carried out there. The Moflex transaction model provides a framework to specify the execution of transactions in mobile environments. The main drawback of the Moflex transaction model is that the specification of mobile transactions must be fully specified in advance. Therefore, the Moflex transaction model may not have the capacity to deal with unexpected or unplanned situations.

Table2. 6: Comparison of mobile transactions models

		Reporting /Co Transaction	Pre serialization	Weak/Strict	Two-Tier Renlication	Promotion	Moflex	Kangaroo
Physical Aspect	Mobility Support	Yes	Yes	No	Yes	Yes	Yes	Yes
	Disconnection Support	No	No	Yes	Yes	Yes	No	No
	Replication Support	No	No	Yes	Yes	Yes	No	No
Transaction Execution	Place of Execution	Both	FH	MH /FH	MH	MH	MSS	MSS
	Compensating transaction	Yes	No	Yes	No	Yes	Yes	Yes
	Execution conditions	No	No	No	No	No	Yes	No
Adaptability	Location dependent support	No	No	No	No	No	Yes	No
	Acceptable final states	No	No	No	No	No	Yes	No
ACID Properties	Atomicity	Yes	No	No	Yes	Yes	Yes	Yes
	Consistency	Yes	No	No	Yes	Yes	No	No
	Isolation	Yes	Yes	No	Yes	Yes	No	No
	Durability	yes	No	No	Yes	Yes	Yes	No
Basic extended transaction		nested	Flat	------	------	Open nested/split	Flexible	Open nested

2.7 Mobile data management

In this section, we will discuss some of the important data management issues with respect to mobile computing. Data management in mobile computing can be described as global and local data management. Global data management deals with network level issues such as location, addressing, replication, broadcasting, etc. Local data management refers to the end user level that includes energy efficient data access, management of disconnection and query processing.

2.7.1 Cache consistency

Caching of frequently accessed data plays an important role in mobile computing because of its ability to alleviate the performance and availability limitations during weak-connections and disconnections. Caching is useful during frequent relocation and connection to different DBSs. In wireless computing, caching of frequently accessed data items is an important technique that will reduce contention on the small bandwidth wireless network. This will improve query response time, and support disconnected or weakly connected operations. If a mobile user has cached a portion of the shared data, he may request different levels of cache consistency. In a strongly connected mode, the user may want the current values of the database items belonging to his cache. During weak connections, the user may require weak consistency when the cached copy is a quasi-copy of the database items. Each type of connection may have a different degree of cache consistency associated with it. That is, weak connection corresponds to a "weaker" level of consistency. Cache consistency is severely hampered by both the disconnection and mobility of clients since a server may be unaware of the current locations and connection status of clients. The server can solve this problem by periodically broadcasting the actual data, invalidation report (reports the data items which have been changed), or even control information such as lock tables or logs. This approach is attractive in mobile environments since the server need not know the location and connection status of its clients and clients need not establish an up link connection to a server to invalidate their caches. There are two advantages of broadcasting. First, a mobile host saves

energy since they need not transmit data requests and second, broadcast data can be received by many mobile hosts at once with no extra cost. Depending upon what is broadcast, appropriate schemes can be developed for maintaining consistency of data of distributed systems with mobile clients. Given the rate of updates, the trade-off is between the periodicity of broadcast and divergence of the cached copies that can be tolerated. The more the inconsistency tolerated the less often the updates need to be broadcast. Given a query, the mobile host may optimize energy costs by determining whether it can process the query using cached data or transmit a request for data. Another choice could be to wait for the relevant broadcast. Cache coherence preservation under weak-connections is expensive. Large communication delay increases the cost of validation of cached objects. Unexpected failures increase the frequency of validation since it must be performed each time communication is restored. An approach that only validates on demand could reduce validation frequency but this approach would worsen consistency since it increases the possibility of some old objects being accessed while disconnected. In Coda [28], during the disconnected operation, a client continues to have read and write access to data in its cache. The Coda file system allows cached objects in the mobile host to be updated without any co-ordination. When connectivity is restored, the system propagates the modifications and detects update conflicts. The central idea is that of caching of data and key mechanisms, which include three states: hoarding, emulation and reintegration, for supporting disconnected operations. The client cache manager while in the hoarding state relies on server replication, but is always on the alert for possible disconnection and ensures that critical objects are cached at the time of disconnection. Upon disconnection, it enters the emulation state and relies solely on the contents of the cache. Coda's original technique for cache coherence while connected was based on callbacks [29]. In this technique, a server remembers that a client has cached an object, and promises to notify it when the object is updated by another client. This promise is called callback, and the invalidation message is a callback break. When a callback break is received, the client discards the cached copy and refetches it on demand. When a client is disconnected, it can no longer rely on callbacks. Upon reconnection, it must revalidate all cached objects before

they are checked for updates at the server. Cache invalidation strategies will be affected by the disconnection and mobility of clients. The server may not have information about the live MUs in its cells. In [36] there is a taxonomy of different cache invalidation schemes and a study of the impact of client's disconnection times on their performance. The issue of relaxing consistency of caches is addressed. Quasi-copies whose values can deviate in a controlled manner are used. There is a categorization of MUs, based on the amount of time spent in sleep mode, into sleepers and workaholics. Different caching schemes turn out to be effective for different populations. Broadcast with timestamps are proved to be more advantageous for frequent queries with a low rate of updates provided that the units are not workaholics. In [37] a technique is proposed to decide whether some items in the cache can still be used by the MU even after it is connected to the server. The database is partitioned into different groups and items in the same group are cached together to decrease the traffic. Thus, the MU has to invalidate only the group rather than individual items.

2.7.2 Data replication

The ability to replicate the data objects is essential in mobile computing to increase availability and performance. Shared data items have different synchronization constraints depending on their semantics and particular use. These constraints should be enforced on an individual basis. Replicated systems need to provide support for disconnected mode, data divergence, application defined reconciliation procedures, optimistic concurrency control, etc. Replication is a way by which the system ensures transparency for mobile users. A user who has relocated and has been using certain files and services at the previous location wants to have his environment recreated at the new location. Mobility of users and services and its impact on data replication and migration will be one of the main technical problems to be resolved. In [38], caching of data in mobile hosts and the cost of maintaining consistency among replicated data copies is discussed. It allows caching of data to take place anywhere along the path between mobile/fixed servers and clients. It determines via simulations which caching policy best suits given mobility and read/write patterns. A general model for maintaining consistency of replicated data in distributed applications is considered

in [39]. It defines a casualty constraint, a partial ordering between application operations, such that data sharing is achieved by defining groups requiring it, and it broadcasts data to the group. Each node processes the data according to the constraints. In [40], it has been argued that traditional replica control methods are not suitable for mobile databases and the authors have presented a virtual primary copy method. In this method, the replica control method decides on a transaction-by-transaction basis whether to execute that transaction on a mobile host's primary copy or virtual primary copy. This method requires a transaction to be restarted when the mobile host disconnects. Also, when a mobile host reconnects, it either has to wait for the completion of all transactions executed on a virtual copy before synchronizing itself with the rest of the system or all running transactions will have to be restarted. The work [41] presents an analysis of various static and dynamic data allocation methods with the objective of optimizing the communication cost between a mobile computer and the stationary computer that stores on-line database. They consider one-copy and two copies allocation schemes. In the static scheme, the allocation scheme remains unchanged, whereas in the dynamic scheme, the allocation method changes based on the number of reads and writes. If in the last k requests there are more reads at MU than writes at the stationary computer, it uses two copy schemes. Otherwise it uses one-copy schemes. A new two-tier replication algorithm is proposed in [42] to alleviate the unstable behaviour observed in the update anywhere-anytime-anyway transactional replication scheme when the workload scales up. Lazy master replication that is employed in the algorithm assigns an owner to each object. The owner stores the object's correct value. Updates are first done by the owner and then propagated to other replicas. The two tier scheme uses two kinds of nodes: mobile nodes (may be disconnected) and base nodes (always connected). The mobile nodes accumulate tentative transactions that run against the tentative database stored at the node. Each object is owned by either the mobile node or the base node. When the mobile node reconnects to the base station, it sends replica updates to the owner mobile node, the tentative transactions and their input parameters to the base node. They are to be re-executed as base transactions on the master version of data objects maintained at

the base node in the order in which they are committed on the mobile node. If the base transaction fails its acceptance criterion, the base transaction is aborted and a message is returned to the user of the mobile node. While the transaction executed on the objects owned by the mobile nodes are confirmed, those executed on the tentative objects have to be checked with nodes that hold the master version. A dynamic replication scheme which employs user profiles for recording users' mobility pattern, access behaviour and read/write patterns, and actively reconfigures the replicas to adjust to the changes in the user locations and systems is proposed in [39]. There the concept of open objects is devised to represent a user's current and near future data requirements. This leads to a more precise and responsive cost model to reflect changes in access patterns.

2.7.3 Query processing

Query processing in mobile computing environment involves two types of queries. The first type of query may involve only the content of databases. Another type may involve queries which may include location-dependent data and furthermore data that depend on the direction of movement. Thus, queries may introduce new parameters regarding query optimization. Location data may change during query evaluation. Queries may be answered in an approximate way due to fast changing location data. The question is how to keep track of the value of a query involving broadcast data in constantly changing environments? Another issue is querying the broadcasted data. What is the best execution plan for a query that involves data broadcast on different channels? What should be the organization of the broadcast data so that the energy spent on the client's side is minimized? Which information should be broadcasted and which should be provided on demand? How to keep track of continuous query in a constantly changing environment [43] is also an issue. Should queries be answered approximately [44] as in the case of querying update intensive data such as the location data? Since the location information may be incomplete, new models of query answering that include data acquisition at run time are needed [45]. The work [46] presents a query processing facility suitable for mobile database applications. The query model, called query by icons (QBI) considers the inherent limitations of mobile environments. It allows the construction of a database query

with no special knowledge of how the database is structured and where it is located. The tools assist in the formation of the query during disconnections. A query is formulated in an incremental manner without accessing actual data in the remote database to materialize intermediate steps. Data are accessed and transmitted back to the mobile computer only when a complete query is materialized. In [45], a query-processing model for mobile computing using summary databases (database stored in some predefined condensed form) is presented. The concept hierarchies are used to generate summary databases from the main database in various ways. This would enhance availability and provide a more optimal use of data during periods of disconnection and enable efficient utilization of low bandwidth and restricted memory size.

CHAPTER 3

EVALUATION OF TRANSACTION EXECUTION STRATEGIES

This chapter is organized as follows. Section 3.1 discusses the general architectural context, execution model and modes of operation of mobile transactions. In section 3.2 we discuss the differences in transaction processing approaches for mobile transactions. Section 3.3 describes our chosen architecture for mobile databases giving assumptions over this architecture which are used for our evaluation purposes. Section 3.4 presents three strategies for execution of mobile transactions to be evaluated in the remainder of the chapter. A framework for mobile transaction execution used in the evaluation process is described in section 3.5. Section 3.6 describe the simulation model, system parameters and discuss the results.

3.1 Mobile transaction context

3.1.1 Architectural context

In classic client-server architectures, functions of each actor are statically defined. In the absence of failures it is assumed that neither the client and server locations nor the connection between them change. In mobile environments, however, the distinction between clients and servers can be temporarily blurred resulting in an extended client server model [10]. Architectural choices impact application design and data management. In particular, transaction execution on a mobile host (MH) is only possible if the MH provides some minimal capabilities. Data stored on MHs (memory/disk) generally come from database servers. Therefore, MH work

must remain consistent with the database server. Mobile clients may vary from thin to full clients, depending on their characteristics as follows:

1- Thin client architecture Here, clients require running operations on servers. This architecture is especially suitable for dumb terminals or small PDA applications. Thin client resources are limited (e.g. small screen size, small cache memory, limited bandwidth). For this architecture, the server is in charge of all computations while clients only display text and graphics, play audio and compressed video, capture pen input, etc.

2- Full client architecture In mobile environments, clients can be forced to work in disconnected mode or with weak connections (due to low bandwidth, high latency, or high costs). Full clients emulate server functions to enable application execution without being strongly connected to remote servers. Full clients are usually portable computers with enough resources to execute applications.

3- Flexible client-server architecture This generalizes both thin and full client approaches. The roles of clients and servers as well as the application functionalities can be dynamically relocated. The distinction between clients and servers may be temporarily blurred for performance and availability purposes.

4- Client-agent-server architecture This three-tier model introduces an agent or proxy located on the fixed network. Agents are used in a variety of roles acting as a surrogate of one or several mobile hosts or being attached to a specific service or application (e.g. database server access). Several proposals concerning mobile transactions management adopt this architecture.

3.1.2 Execution models

Mobile transactions involve MHs and fixed hosts (FHs). Servers generally run on FHs (wired network) and MHs can be simple clients with some server capabilities. According to client capabilities five execution models have been defined [54]. The first three models involve one MH whereas the fourth and fifth ones involve several MHs.

3.1.2.1 Complete execution on the wired network

Here, the mobile transaction (MT) is initiated by a MH but executed entirely on FHs. This is the classical query shipping approach where the data server executes update/query requests and sends results to the client. Examples, in a mobile context, where this execution scenario is appropriate are location dependent queries, e.g. hotels located within a radius of 5 miles, and updates, e.g. booking a room in one of those hotels; see [31] and [32]. In this context it is also suitable to execute transactions on a large data set.

3.1.2.2 Complete execution on a MH

In this case the MT is initiated and executed on the MH. This model requires the MH to have all relevant data and enough "server" capabilities to execute its local transaction. The autonomy of the MH allows it to keep working even though connections with the server are not available. Reconciliation procedures are necessary to integrate MHs work in the database server located on FHs. In most cases, some final work has to be done on the database server even if the MT is executed on the MH. For instance, consider a salesperson having on his/her MH all the required data related to the products he/she sells (available stock, price, etc.). The work done autonomously (sales) on the MH is integrated afterwards at the main server.

3.1.2.3 Distributed execution between a MH and the wired network

This model is very flexible as it allows distributing transaction execution between the MH and the database server(s) on the wired network. This distribution may be motivated by resource availability (e.g. data, power on the MH) or for optimization reasons. Server capabilities are required on the MH as well as minimum communications with the server during transaction execution. The sales example above may also require a distributed execution scenario. As far as the salesperson has the product information on his/her MH, he/she could sell products without connection to the database server warehouse store. Nevertheless, the payment procedure may require a connection to the bank server to check the client's credit. The sale is done by a distributed transaction having one subtransaction executed on the MH and another one executed on the bank server.

3.1.2.4 Distributed execution among several MHs

This case is very ambitious and difficult but interesting. The objective is to provide a "peer to peer" approach. MHs act as servers for other MHs so that the execution of a MT is distributed among several MHs. The idea is that depending upon MH location; it could be interesting to ask a "neighbour" MH to act as a data server or as a service provider. Here "neighbour" means closer in terms of communication than the database server. As an example, consider two salespeople working in the same geographical area that need to share some data in a cooperative way without referring to the main database server. To support this execution model, particular features are required to allow MHs to be aware of each other. Base stations (BSs) could play an important role by maintaining specific database catalogues allowing MHs to know the data available in the area. These catalogues should be updated and forwarded across BSs according to MHs movements. In ad-hoc networks [55], MHs interact by establishing a point-to-point connection by passing BSs. Since no BS is involved, each MH has to maintain its own database catalog and to allow its neighbours to access it. Distributed transaction execution among several MHs is particularly oriented towards dynamic network configurations.

3.1.2.5 Distributed execution among MHs and FHs

This is the fully distributed scenario where MT execution is distributed among several mobile and fixed hosts. This approach is an extension of the previous scenario and is oriented towards cooperative work as well as to multidatabases including MHs participating in the global execution. Electronic commerce is a promising application where small devices will engage in commercial transactions among themselves (execution model 2.2.2.4), with BSs or with remote hosts reached through a combination of wireless and wired infrastructure (this execution model) [56]. For instance, participant MHs could be in an open air trade fair where suppliers, manufacturers, retailers and customers would meet to see the latest products available. Customers may want to buy some of the products after seeing them and consulting an online catalog available at the fair. Upon the receipt of an order, the merchants could contact one another to order parts and locate supplies. With the appropriate devices and transactional support, all these

operations could be performed on site and be uploaded later onto the company's servers.

The five execution models introduced in this section cover all the possibilities involving mobile or fixed hosts. Nevertheless, current proposals in the literature concern only the first three models. Participation of several mobile hosts in the same distributed transaction has not been developed yet.

3.1.3 Modes of operations

In mobile computing, there are several possible modes of operations [54] as compared to a traditional distributed system where a host may operate in one of two modes - either connected to the network or totally disconnected. The operation mode in mobile computing may be one of the following:

- Fully connected (normal connection);
- Totally disconnected (e.g., not a failure of the MH);
- Partially connected or weak connection (a terminal is connected to the rest of the network via low bandwidth).

In addition, for conserving energy, a mobile computer may also enter an energy conservation mode, called *doze state*. A doze state of the MH does not imply the failure of the disconnected machine. In this mode, the clock speed is reduced and no user computation is performed. Most of these disconnected modes are predictable in mobile computing. Protocols, as given below, can be designed to prepare the system for transitions between various modes. A mobile host should be able to operate autonomously even during total disconnection.

- A *disconnection protocol* is executed before the mobile host is physically detached from the network. The protocol should ensure that enough information is locally available (cached) to the mobile host for its autonomous operation during disconnection. It should inform the interested parties for the forthcoming disconnection.
- *A partially disconnection protocol* prepares the mobile host for operation in a mode where all communication with the fixed network is restricted. Selective caching of data at the host site will minimize future network use.
- *Recovery protocols* re-establish the connection with the fixed network and resume normal operation.

• *Hand-off protocols* refer to the crossing of boundaries of a cell. State information pertaining to the mobile host should be transferred to the base station of the new cell.

3.2 Mobile versus fixed transactions

A transaction in mobile environments is different to transactions in centralized or distributed databases in the following ways:

1. The mobile transactions might have to split their computations into sets of operations, some of which execute on a mobile host while others on a stationary host. A mobile transaction shares its state and partial results with other transactions due to disconnection and mobility.
2. Mobile transactions require computations and communications to be supported by stationary hosts.
3. When the mobile user moves during the execution of a transaction, it continues its execution in the new cell. The partially executed transaction may be continued at the fixed local host according to the instruction given by the mobile user. Different mechanisms are required if the user wants to continue its transaction at a new destination.
4. As the mobile hosts move from one cell to another, the states of transactions, states of accessed data objects, and the location information also move.
5. The mobile transactions are long-lived transactions due to the mobility of both the data and users, and due to the frequent disconnections.
6. The mobile transactions should support and handle concurrency, recovery, disconnection and mutual consistency of the replicated data objects.

3.3 Mobile Database Architecture

As discussed in Chapter 2, there are different models for mobile computing systems. Among them, we choose the model most likely to be a computing environment for mobile database systems. In this system, some computers are fixed and some are mobile. This architecture is widely accepted in existing research for mobile database applications [50], where a global database is distributed among the fixed network nodes. An example of such architecture is

shown Figure 3.1. In this system architecture, all the network nodes in the wired part are fixed units. Mobile units retain communication through the wireless links. Some fixed units, called base stations or mobile support stations, have special functionality with a wireless interface to communicate with mobile units. Each mobile station provides networking services for all the mobile units within a given geographic area called a cell. In other words, each cell has a base station and mobile units within that cell access data on remote nodes through the local base station.

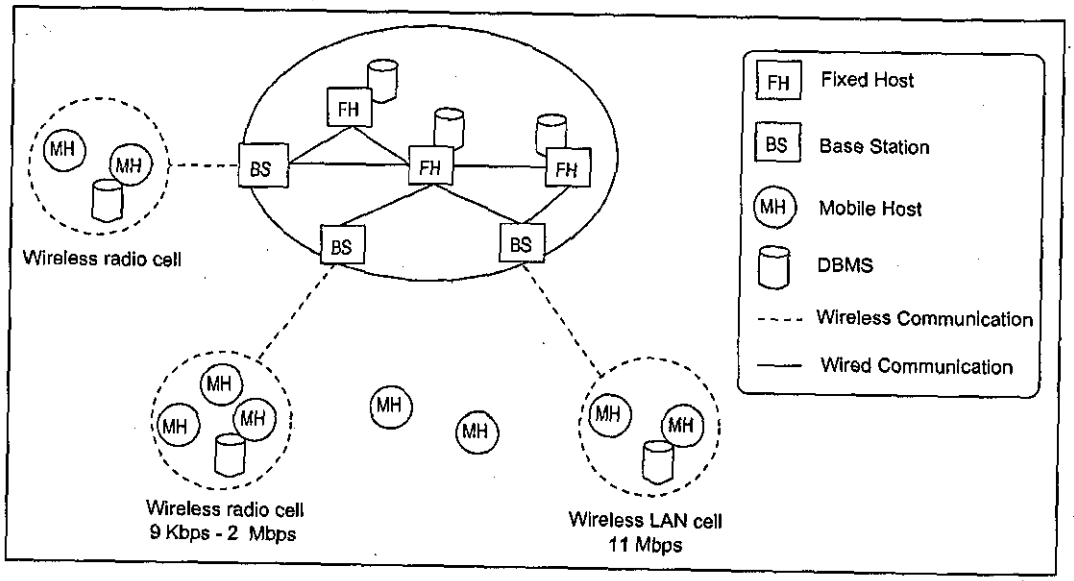


Figure3. 1: Architecture of mobile database systems

A global database (GDB) is defined as a finite set of data items. A GDB is partitioned among fixed hosts as well as mobile hosts (GDB = MDB$\cup$FDB). The data items on fixed hosts make up the fixed database (FDB) of the GDB (i.e. FDB =$\cup$ FDB_i) and the data items on mobile hosts make up the mobile database (MDB) of the GDB (i.e. MDB = $\cup MDB_i$). Moreover, the data items on a single mobile host are called a *mobile part* of the MDB. Accordingly, a data item in the FDB is referred to as a fixed data item and a data item in the MDB is referred to as a mobile data item. A data item in the MDB is owned by a mobile host and a data item in the FDB is owned by a fixed host. We further assume that each mobile host has local storage and computing capability, and is able to estimate and exchange its status regarding the information of the environment current state.

The fixed data items are replicated by caching on a mobile host, whereas the mobile data items have a full replication on a fixed host. Moreover, a mobile transaction does not access the data on other mobile units ($MDB_i \cap MDB_j = \{\}$ for all i, j such that $i \neq j$). That is, it can access only local mobile data items and fixed data items.

3.4 Execution Strategies:

3.4.1 Fixed Host Execution Strategy (FHS)

In this strategy, the execution of mobile transactions, with a copy of required data on the mobile host, is transferred to the fixed host. The fixed host coordinates the execution of the transaction on behalf of the mobile host and returns the final results back to the mobile unit. As such, a mobile transaction can be processed just like a traditional distributed transaction in a client server model. The only difference is that the propagation of updates and the return of results to a mobile host may be delayed as a consequence of wireless communication. This execution strategy is relatively simple. Its main advantages are:(1) maintaining strict data consistency because an entire transaction is managed and executed on a fixed host - traditional transaction schemes can be applied, (2) reducing battery consumption because the operations are shifted to the fixed network, and so computation tasks on a mobile host are avoided and (3) the level of concurrency at the fixed host is increased due to avoidance of long delays resulting form poor communication. However, a shortcoming of the fixed host execution strategy is that no autonomous operations are allowed during the disconnection of a mobile unit because required data on the fixed host is not available. This policy is suitable when the entire database is allocated on the fixed machines. In other words, a mobile host only plays the role of a client, or a remote device (i.e. thin client architecture). This strategy has been taken as an underlying assumption, of the processing model, by many researchers.

3.4.2 Mobile Host Execution Strategy (MHS):

To allow continuous computation when a disconnection occurs, the data stored at the fixed hosts can be duplicated on a mobile unit rather than moved to a fixed host. Instead of making a full replication on a local disk, using cached copies is a

common technique to support autonomous operations, increase the availability [64], and minimize network access. Since data involved in a mobile transaction is always locally available, this policy allows transaction processing independent of fixed data services. Hence, the autonomous operations can be carried out at the mobile unit. In addition, this policy uses less battery power compared to the FHS strategy, regarding data transmission, because the network connection between a mobile unit and a fixed host is asymmetric. A fixed host typically has a stronger transmitter and unlimited power. Therefore, transporting data from a fixed host to a mobile unit is likely to be more efficient than transporting data in the opposite direction. There are two methods of transporting data. Firstly, in *planned* mode. In this mode required data is transferred from fixed hosts to a mobile unit before a disconnection occurs. This mode requires that a disconnection protocol knows which data will be used in the near future. Secondly, *non-planned* mode. In this mode, data is downloaded based on current transaction requirements and network conditions. For instance, if a transaction uses data items X and Y, and at this time point the bandwidth is high, then the data will be transferred to the mobile unit. Considering two types of disconnection, the first mode is more suitable for predictable disconnection, while the second mode is suitable for unpredictable disconnection.

3.4.3 Combined Execution Strategy (CHS):

For mobile transactions having a number of subtransactions which pass through different resources availability during their execution life time, neither MHS nor FHS on its own can give better performance in terms of system throughput and battery consumption in all situations. Since the conditions of the mobile environment are dynamically reflected in the transaction processing, an adaptive approach to control the execution of a transaction is desirable - one that takes fixed host and mobile host execution strategies as two basic options for each subtransaction. A decision is made, based on available mobility information, to select an execution strategy for each subtransaction of a mobile transaction in an optimal way corresponding to the current state of the mobile computing environment.

The run-time decision-making based on the current environment state, is the key of the CHS strategy approach [35]. With the CHS strategy, a mobile host always submits transaction requests to the mobile transaction manager (MTM) if the network condition is satisfactory; otherwise it will be processed locally at the MH. In this strategy, each mobile subtransaction consists of a set of database operations (T_{op}), a set of data items on which the operations are performed, i.e. its base set (T_{bs}), the cache status (T_{cs}) of the data items in the base set and the current environment state information (T_{es})of the mobile host. Thus, each mobile subtransaction SMT is identified by the 4-tuple SMT= (T_{op}, T_{bs}, T_{cs}, T_{es}).

The cache status of data items determines the amount of stale data in the cash. The current environment state is determined by several factors. The state factors, of particular interest for CHS, are the available bandwidth, disconnection threshold and battery life of a mobile host. In general, let MSTi be a mobile subtransaction request submitted by a mobile host at a particular time point. Let the tuple < B_i, D_i, E_i > be the current status vector at this time, where Bi, Di and Ei are the bandwidth, disconnection threshold, and battery life of the mobile host, respectively. The basic idea of selecting the execution of each mobile subtransaction as a `Dreq' or `Treq' (see below) transaction will be as follows. If the computing condition is normal (strong connectivity), it should be treated in the same manner as in a conventional environment. That is, submit the transaction to a fixed host to be executed, i.e. as a Treq transaction. If the computing environment suffers from low bandwidth that is below a specified level, then the transaction will be processed locally at the mobile host, i.e. as a Dreq transaction. With the information provided by each subtransaction at a certain time, the decision criteria will be made as follows. First, if the battery life is below a specified level, then the transaction should be executed on a fixed host. Second, if the network bandwidth is weak, i.e. below a specified level, it is treated like a disconnection and so the mobile transaction request will not go to the MTM and instead is scheduled as a Dreq transaction. These two simple cases can be expressed in the following rules:

(1) IF Ei < $E_{threshold}$ THEN schedule as a Treq transaction

(2) IF Bi < $B_{threshold}$ THEN schedule as a Dreq transaction

Other criteria to determine where to execute a mobile subtransaction can depend on the amount of required data transmission. When the network bandwidth is above a specified level, the decision will be made based on the time of downloading/uploading, T_{down}/T_{up}, which is the function of the size of downloading/uploading data and the current bandwidth. Upon the receipt of a transaction request the MTM can determine the amount of data to download/upload from the cache status information. This amount gives the size of stale data cached on the mobile host. If cached data for a current transaction is valid no data will be transported. Therefore, if the size of downloading data from the fixed part to the mobile host is smaller than the size of uploading data from the mobile host to the fixed part the transaction is scheduled as a Dreq transaction; otherwise it is scheduled as a Treq transaction. This is expressed as the following *rule:*

(3) IF $T_{down,} < T_{up,}$ THEN schedule as Dreq transaction ELSE schedule as Treq transaction.

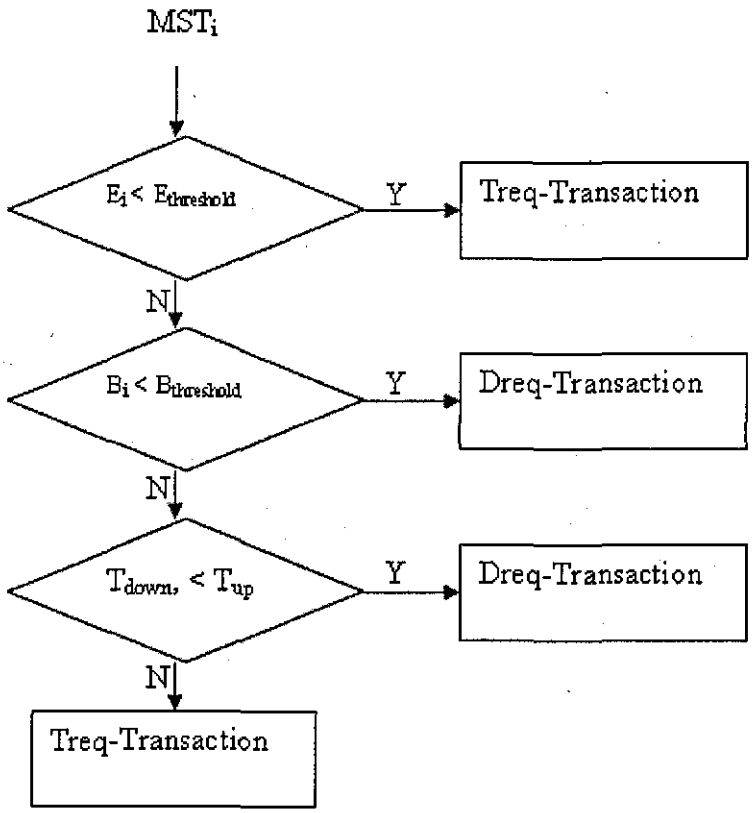


Figure3. 2: Decision flow of CHS

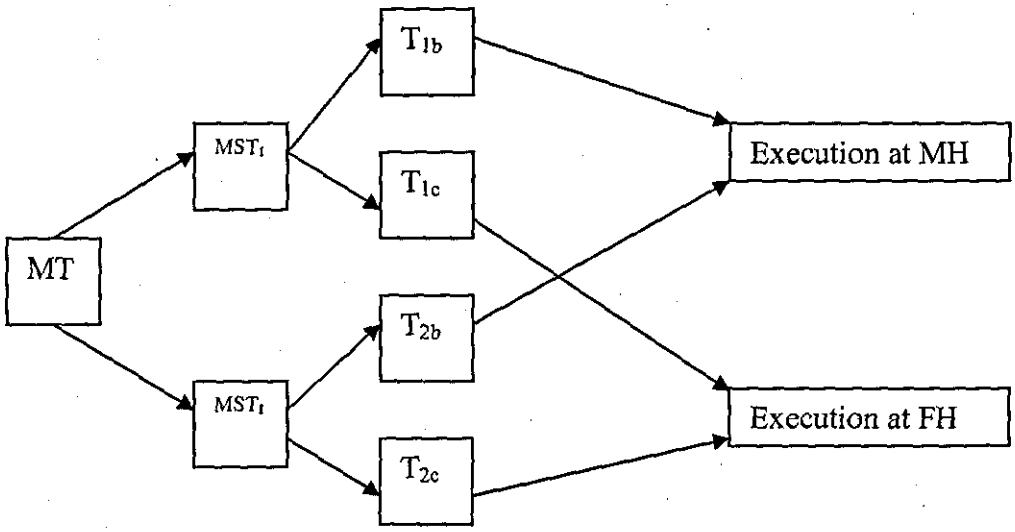
In summary, the above description can be expressed as a decision flow shown in Figure 3.2. In the figure, $E_{threshold}$, and $B_{threshold}$ are two thresholds and indicate satisfactory battery and bandwidth levels, respectively. They can be specified at the time when a mobile host registers, and can be modified during mobile transaction execution.

With the combined strategy, a mobile subtransaction can be scheduled as a data request or a transaction request. In this approach we can gain two main advantages: (1) a mobile transaction can be tentatively committed based on locally cached data and other transactions can be submitted during disconnection so that autonomous operations can be supported. (2) A mobile transaction can be either executed on a mobile host or a fixed host based on a run-time decision.

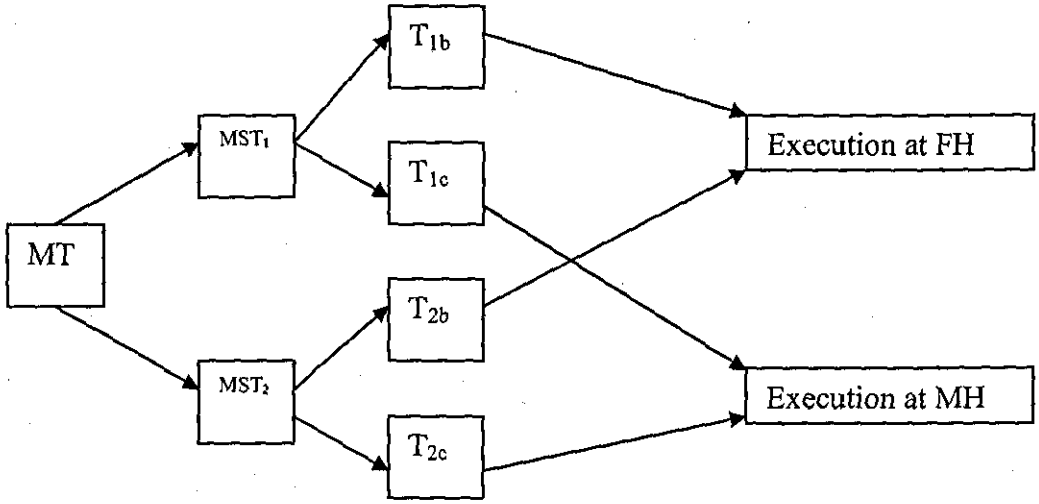
3.5 Execution Framework

There have been many different models proposed for Mobile Transactions (MT) [54]. In these models mobile transactions are supposed to be decomposed into a set of subtransactions which make it possible for the atomicity requirement to be relaxed. The common base between these models is their extension of advanced transaction models. Since our work here concentrates on evaluating execution strategies under different network connectivity conditions, rather than evaluating a specific mobile transaction model, we make a general assumption that a mobile transaction MT is defined as a set of mobile subtransactions MST. The update made by a subtransaction at the execution place should be reflected on the opposite part of the network (i.e. if the execution takes place at the mobile host, the update made should be reflected on the fixed host and vice versa). So, each mobile subtransaction MST is decomposed into two subtransactions: namely, a basic subtransaction T_b and a complementary subtransaction T_c. Corresponding to this, there are two commit points: local commit and global commit, respectively. A local commit records the status of all basic subtransaction processing when it is done, as it may not be possible for the complementary sub-transactions to be issued due to network disconnection at that time. A global commit implies that all complementary subtransactions have been executed and modified data items have been propagated to their master copies. Global commit can happen only when the network is connected. Successful commitment of a

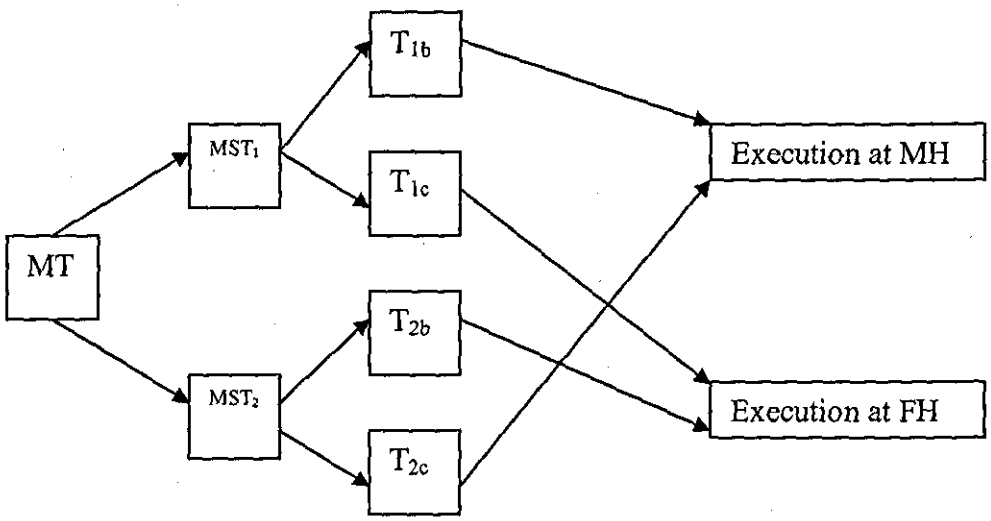
mobile transaction occurs if and only if all of its basic and complementary sub transactions have committed successfully. So, the processing of each mobile subtransaction consists of two phases, a basic subtransaction executes in the first phase of processing a mobile subtransaction. Major transaction operations are performed in this phase. A complementary subtransaction derived from a basic subtransaction occurs in the second phase of mobile subtransaction processing and its execution takes place when the data items on both the fixed part and mobile part are connected. A complementary subtransaction largely performs updates on the data items that are modified by its basic subtransaction and the place where a basic subtransaction executes is always opposite to where its complementary subtransaction executes. In general, each subtransaction of a mobile transaction is scheduled as a data request or a transaction request. The basic subtransaction of a Dreq transaction is processed on a mobile transaction host with cached data from the fixed part. The corresponding complementary subtransaction is processed on a fixed host. Conversely, the basic subtransaction of a Treq transaction is processed on a fixed host with the replication of the data items from the mobile host. The matching complementary subtransaction is processed on the mobile host. In this way, the effect of data modification can be propagated to its master copy. When a mobile subtransaction is issued from a mobile host, its basic subtransaction is processed with cached data if network connectivity between the mobile transaction host and its local base station is unsatisfactory (i.e., poor bandwidth or disconnected). If the required data is not available, the subtransaction is aborted. Otherwise, after completion a complementary subtransaction is issued and put into a queue. When the mobile transaction host restores its connection, the queued complementary subtransactions are first submitted through a base station to a fixed host. If a complementary subtransaction fails, the mobile subtransaction is aborted. Otherwise it waits for global commitment. If network connectivity is high a mobile transaction is started. The location where its basic subtransaction is processed depends on the execution strategy see Figure 3.3 (a)-(c).



(a) MHS strategy



(c) FHS strategy



(b) CHS strategy

Figure3. 3: Execution framework for the three strategies

If the decision is made to use a Dreq transaction, the fixed data items required by the transaction are downloaded from the fixed host (the size of downloaded data is determined by the amount of valid data in the cache of the mobile transaction host). After this, the process of mobile transaction processing is similar to the one above when network connectivity is poor. On the other hand, if the decision is made to use a Treq transaction, the mobile data items required by the transaction are uploaded from the mobile part owned by the mobile transaction host, to a fixed host where the basic subtransaction is processed. Similarly, at this time, if network connectivity between the mobile transaction host and its local base station is below some threshold, the complementary subtransaction will be queued and executed later when network connectivity is improved. If network connectivity remains strong, the complementary subtransaction that performs updates of mobile data items on the mobile transaction host can be issued as soon as the basic subtransaction is completed. The execution of mobile transactions is given in pseudo-code in Figure 3.4.

Mobile transaction Processing

Begin

For each mobile subtransaction MST,

mobile host analyzes the subtransaction MST to build T_{basic} and $T_{complemnetary}$ based on its read and write requests and availability of requested data (we assume that MH has some server capability).

If the subtransaction is scheduled as a Dreq transaction **Then**

Begin

For all data item reads and writes by T_{basic},

MH sends a request to MTM for all the required read values and request for lock. After MTM acquires the necessary locks, it returns the values to the MH. The basic transaction is executed on the MH and $T_{complementary}$ is executed on the fixed host.

End

Else //the subtransaction is scheduled as a Treq transaction//

Begin

For all data item reads and writes by T_{basic},

MH sends a request to MTM for all operations along with their data items.

After MTM acquires the necessary locks, the basic transaction is executed on the FH and $T_{Complementary}$ is executed on the mobile host.

End

If all mobile subtransactions successfully finish their execution **then**

Mobile transaction is committed

End

Figure3. 4: Mobile transaction processing algorithm

3.6 Performance Evaluation

The performance of different execution strategies are evaluated by the means of simulation. The goal of this simulation is studying the performance of three executions strategies for mobile transaction in presence of fixed transactions under different wireless network connectivity. We describe the simulation model, system parameter and discuss the result.

3.6.1 Simulation Model

The general structure is shown in Figure 3.5. The model consists of three servers labeled MH, MTM, and FH. The server MTM stands for the mobile transaction manager related services. Normally, a mobile transaction request gets this service first. Each mobile transaction issues a set of requests and each request represents one of its subtransaction which could be a data request or a transaction request transaction based on the execution strategy being applied. The server MH represents the services provided by a mobile transaction host and the server FH the services provided by a fixed host where transaction coordination takes place. If the subtarnsaction of a mobile transaction is scheduled as a Dreq-transaction, when its basic subtransaction has finished its execution, the complementary subtransaction is put into queue MQ1. If the network is connected, a Dreq complementary subtransaction is submitted to a fixed host for processing and enters a queue FQ1; otherwise it has to wait in MQ1 for network re-connection. If a Dreq complementary subtransaction is successfully finish its execution. The mobile subtransaction is completed; otherwise, the subtransaction fails. This process will be repeated for all sub transactions, when all subtransactions are successfully completed, the entire mobile transaction is committed. For a subtransaction that schedule as Treq, this process is identical; but with different notations of Figure 3.5; that is if the network is connected, a Treq complementary subtransaction is submitted to a mobile host for processing; otherwise it has to wait in MQ2 for network re-connection. The CHS strategy can be viewed as a combination of the two basic strategies MHS and FHS. For the MHS approach, each subtransaction in a mobile transaction goes through four processing steps: data downloading, basic subtransaction processing, complementary subtransaction submission, and complementary subtransaction processing. Thus, there are four simple services. Similarly, the FHS approach has four simple services as well. The difference between the FHS model and MHS model is that the data required by a mobile transaction is uploaded from the mobile host to a fixed host. Besides, the first three simple services, i.e. data uploading, basic subtransaction processing and complementary subtransaction submission, are implemented on a fixed host. Whereas complementary

subtransaction processing is performed by the mobile host that issues the transaction. According to the CHS strategy, when a mobile host is disconnected a transaction is treated as a Dreq transaction and has no data transmission. If the network is connected, the data will be uploaded or downloaded based on current scheduling decision. Generally, the simulation is conducted for the three strategies separately.

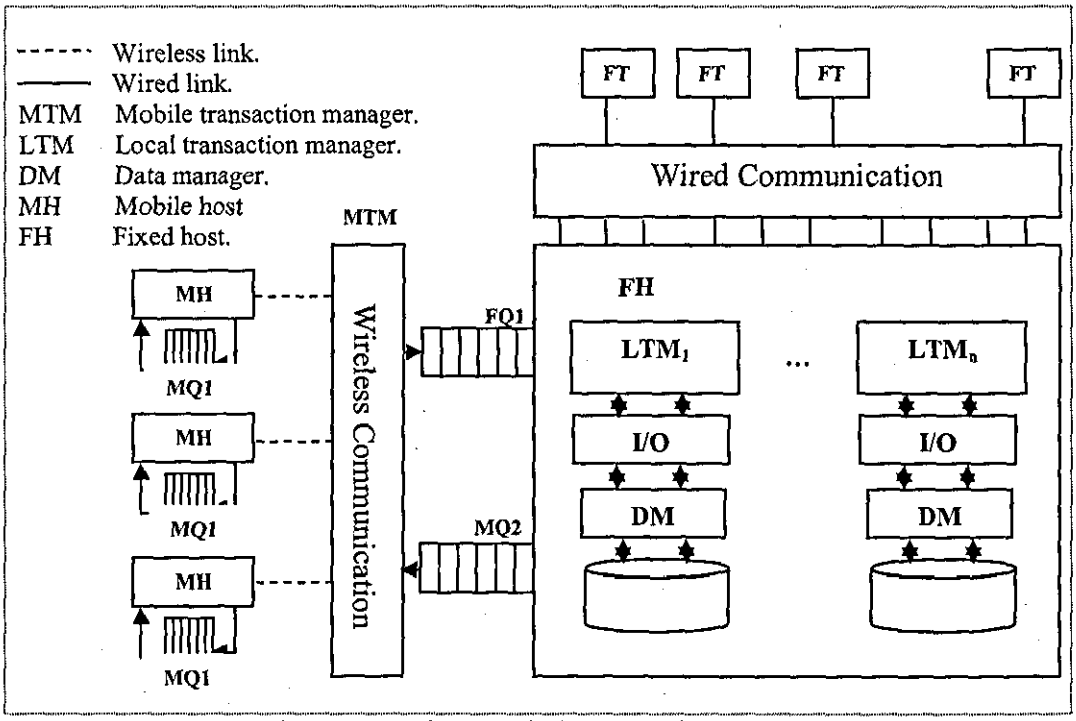


Figure3. 5: The simulation model overview

If the policy is FHS, transaction requests are always scheduled as Treq transactions. If the network is disconnected when a request arrives, this request must give up because its first phase cannot be carried out. After the completion of the first phase, if the network is not connected at this time, the complementary subtransaction cannot be submitted and must wait in the queue. Unlike the FHS approach, the MHS strategy usually does not discard a transaction request. Instead, the first phase of transaction processing is performed by using cached data even if the network is disconnected. However, if the network is disconnected after this phase, the request must be queued for later processing. For the CHS strategy, when the network is disconnected, its work likes an MHS approach; otherwise, it functions as a mixture of the FHS and MHS. Moreover, a complementary subtransaction may or may not be blocked. The network

connectivity directly impacts the performance of transaction processing. To reflect this, the time spent on the completion of a Dreq-transaction can be distinguished into four cases based on the network connectivity during transaction execution. (1) A CC-type transaction is just like a normal transaction where the network is in a fully connected condition. The other three types of transactions indicate that at least one of two communication steps is broken (denoted by the D). (2) CD-type transaction indicates that the network is connected at the time of the data transmission and disconnected at the time of complementary subtransaction submission. (3) DC-type transaction indicates that the network is disconnected at the time of the data transmission and connected at the time of complementary subtransaction submission. (4) DD-type transaction indicates that the network is disconnected at the time of the data transmission and the time of complementary subtransaction submission. The performance of these four types of transaction can be different. The functions to determine the execution and communication time is related to the current state of the environment. In general the service type in the system can be classified into two type communication oriented and processing oriented. For a communication oriented service. The service time consists of a constant communication overhead and the data transport time, which is determined by the size of data and the bandwidth at the time of data transfer. For a processing oriented service, the service time is basically determined by the number of accessed data items, the average number of operations on these items and the amount of time spent in the fixed host which is based on different factors specified in section 3.6.2.

3.6.2 System Parameters

The underlying mobile database system is composed of a number of databases distributed among fixed and mobile hosts. Transaction-generators are responsible for creation of transactions to be executed in the system. Parameters controlling the generation of transactions include the workload (the number of concurrent transactions arrive per unit of time to the system); the number of subtransactions in each mobile transaction; the number of database operations in a subtransaction; the probability of a write operation. The global workload consists of randomly generated local and mobile transactions spanning over a random number of sites.

At each local site, there are a number of local transactions. The local system does not differentiate between the two types (local transaction and mobile subtransaction). Transactions enter the execution phase are subsequently scheduled by acquiring the necessary lock on their data items. If the lock is granted, the operations proceeds through the CPU and I/O queue, and for a mobile subtransaction an abort or commit signal is communicated back to the MTM and the subtransaction terminates. The local system may abort a local transaction or a mobile subtransaction at any time. If a mobile subtransaction is aborted locally, it's communicated to the MTM and the mobile subtransaction will be restarted. The communication between the mobile host and the MTM can occur in both direction using either an uplink channel for uploading data or downlink channel for downloading. The communication overhead for uplink and downlink channels is 15 and 5 respectively. When the disconnection occurs, the mobile unit is disconnected for 10-20 second. In order to effectively evaluate the different execution strategies, several parameter are varied for different simulation runs. Most of these parameters are given in Tables 3.1-3.3 along with their default values.

Table3. 1: Communication Parameters

Communication Parameters	**Default Values**
Transaction workload	50
Data item size	10-20
Mobility timer	5,10,15,..., 50
Disconnection Threshold	100,500,1000
UplinkBW	10-2000
DlinkBW	10-3000
Service time for each communicated message using Uplink	15
Service time for each communicated message using Downlink	5

Table3. 2: Fixed Host Parameter

Fixed Host Parameter	**Default Values**
Number of operation in each local transaction	6- 12
Probability of write operation for local transaction	0.3- 0.5
Lock time for each operation	2
Concurrency protocol	2 Phase Locking

Table3. 3: Mobile Host Parameters

Mobile Host Parameters	Default Values
Mobile transaction percentages	60%
Number of mobile sub transactions	1-5
Number of operations in each sub- transaction	3- 15
Probability of write operation	0.3 – 0.5
Probability of cash hit	0.5 -1.0
Execution cost at the mobile host	2-5
Mobility value of mobile host number of visited MTM	5 – 50

3.6.3 Experiment Results and Discussion

The experiments are designed to study the impact of mobility ratio under different disconnection conditions on the performance measures to compare and contrast the different execution strategies. Figures 3.6 to 3.11 show the impact of changing network conditions on the response time for both mobile and fixed transaction under different execution strategies. In each experiment, 1000 transactions are generated and 60% of these transactions are mobile. The mobility timer is varies from 5 to 50 seconds.

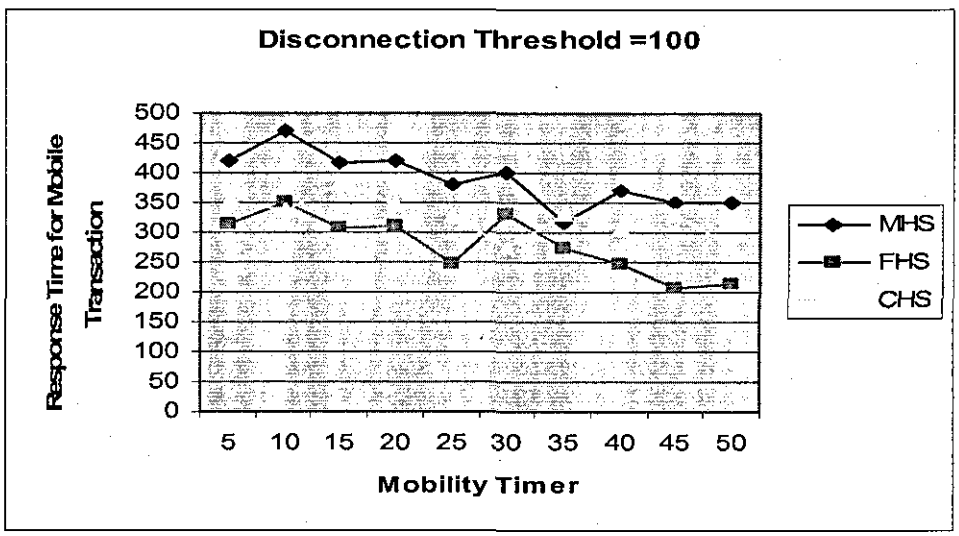


Figure3. 6: Response time for mobile transaction (BW<=100)

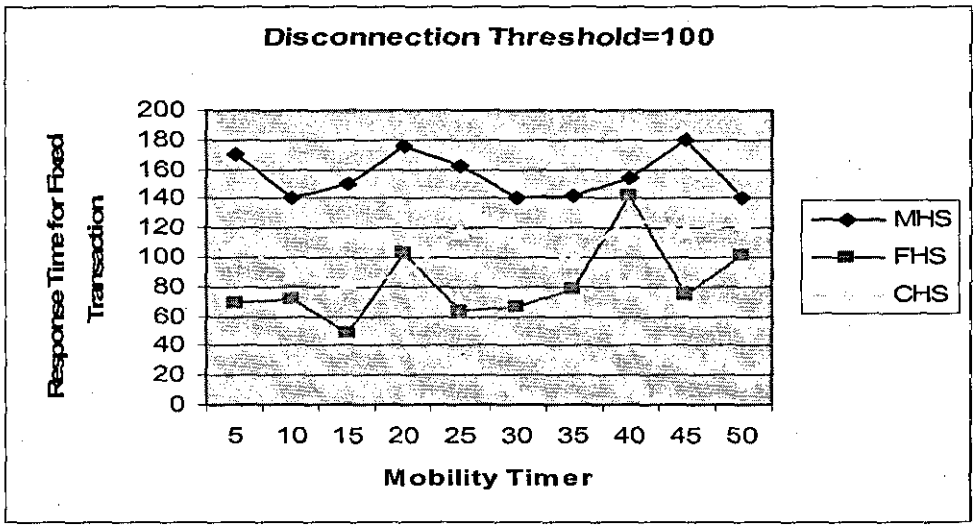


Figure3. 7: Response time for fixed transaction (BW<=100)

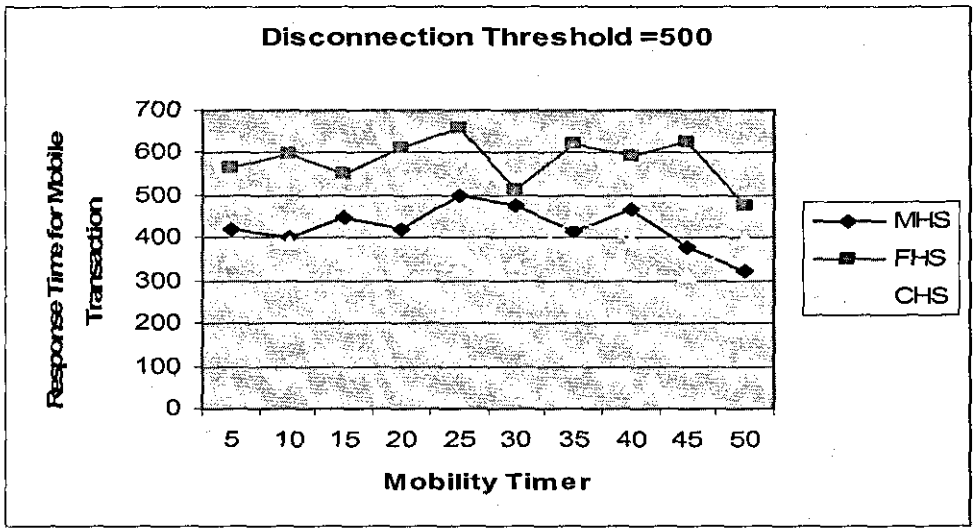


Figure3. 8: Response time for mobile transaction (BW<=500)

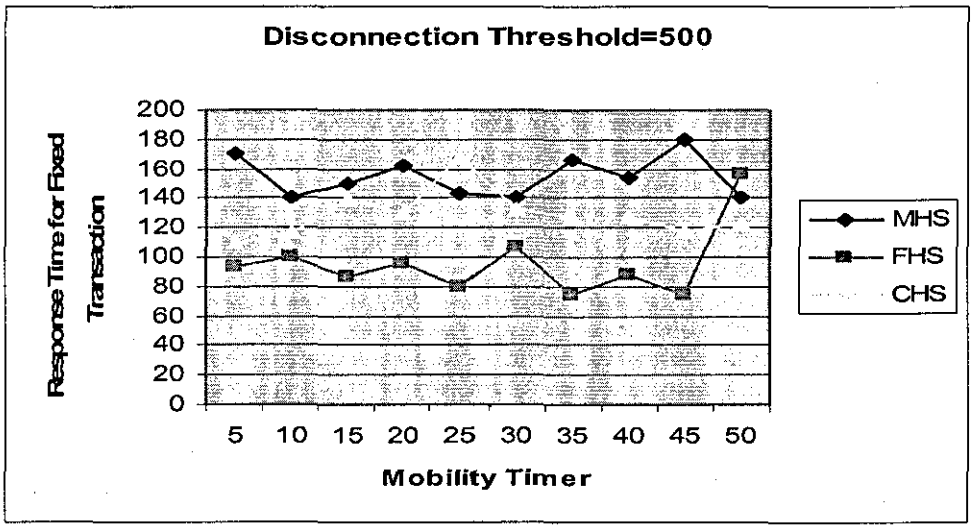


Figure3. 9: Response time for fixed transaction (BW<=500)

Figure 3.6 shows the results when a mobile host has low disconnection (disconnection threshold is BW<=100). It can be seen that among the three approaches, the FHS outperform the other strategies, whereas the MHS is the worst. This is because a mobile subtransaction that schedules as a data request transaction needs more data transmission than if schedule as a transaction request. Moreover, a data request transaction takes a longer time for global commitment. This can be reflected in the response time for the fixed transaction as we can see in Figure 3.7. The CHS strategy is closer to FHS than to MHS which means that most of mobile subtransactions are scheduled as transaction requests. Even though the network is highly connected, the performance of the CHS may not outperform the FHS strategy. This is because the network connectivity is not the only factor which affecting the decision made by the CHS strategy to determine where the mobile subtransaction execution is take place, the size of data to be transmitted and the cash status also affecting the decision made by the CHS. So, if the cache status indicates that cached data for a transaction is available, the transaction may schedule as a data request transaction, even though the network is fully connected. Therefore, unlike the FHS strategy, the sub transactions of the mobile transaction may scheduled in a mixed matter based on the current state of the environment at that time. Figure 3.8 and 3.9 shows the results when a mobile host has medium disconnection (disconnection threshold is BW<=500) for both mobile and fixed transaction, respectively. It can be seen that the response time for all strategies are

negatively affected by increasing the disconnection threshold. However, because of increasing disconnection probability, the performance of the FHS strategy becomes less than the other two strategies and CHS has the best performance. The reason is that when the network disconnection probability increases, the number of mobile subtarnsaction which schedule as data requests increase. On the other hand, the mobile subtransaction at connection time was scheduled as a transaction request, so this environment gives the CHS strategy more chance to exploit the information at the current environment state which makes it better than FHS at the disconnection time and also better than MHS at connection time. In Figure 3.9 the difference between the FHS and other strategies is more obvious than Figure 3.7. Since the disconnection probability is increases, The FHS strategy has more chance to block the basic transaction which comprises the first phase of any mobile subtransaction execution. This will decrease the blocking over head on the fixed transaction at the fixed host. On the other hand, in MHS and CHS strategies, a cached data can be used to execute the basic transaction. So, there is no need to block a mobile subtransaction if the cash status of the required data items is valid. As a consequence, the response time of mobile transactions decreases substantially when compared to the FHS approach. The advantages get by the mobile transaction under MHS and CHS in term of decreasing the response time will be negatively affect the fixed transaction as it have to wait for the basic transactions of the mobile subtransaction until finish its execution at the mobile host which is take a longer time than if the execution is take place at the fixed host. This can be seen in the Figure 3.10 and 3.11 which show the response time for both fixed and mobile transaction when there is a high disconnection, respectively.

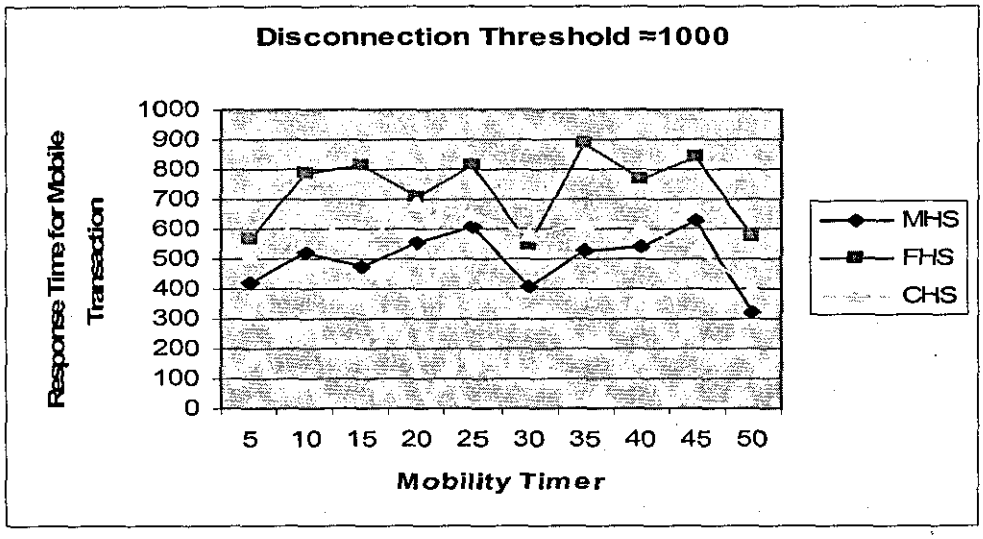


Figure3. 10: Response time for mobile transaction (BW<=500)

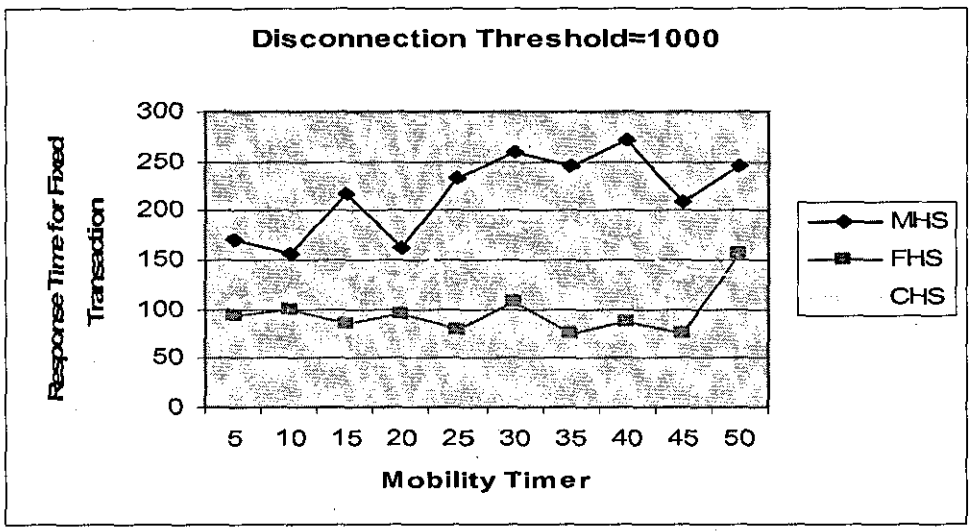


Figure3. 11: Response time for fixed transaction (BW<=500)

The battery of mobile hosts considered to be one of a scarce resource in the mobile computing environment [65], [66], [67], an experiment is conducted to show the difference between three strategies in term of power consumption. Figures 3.12 to 3.14 show the simulation results for different mobility value with different disconnection thresholds. Since the whole mobile transaction take place at the fixed host under FHS strategy, only the communication overhead can affect the power consumption of the mobile host. This justifies the significant difference

between FHS and other strategies. On the other hand, the MHS has the worst effect on power consumption because all mobile transaction processing take place at the mobile host. The difference between the power consumption for the CHS and MHS strategies decrease as disconnection increases since there is more chance for mobile subtransaction to schedule as a data request transaction. Figure 3.12 shows that when the network is strongly connected.

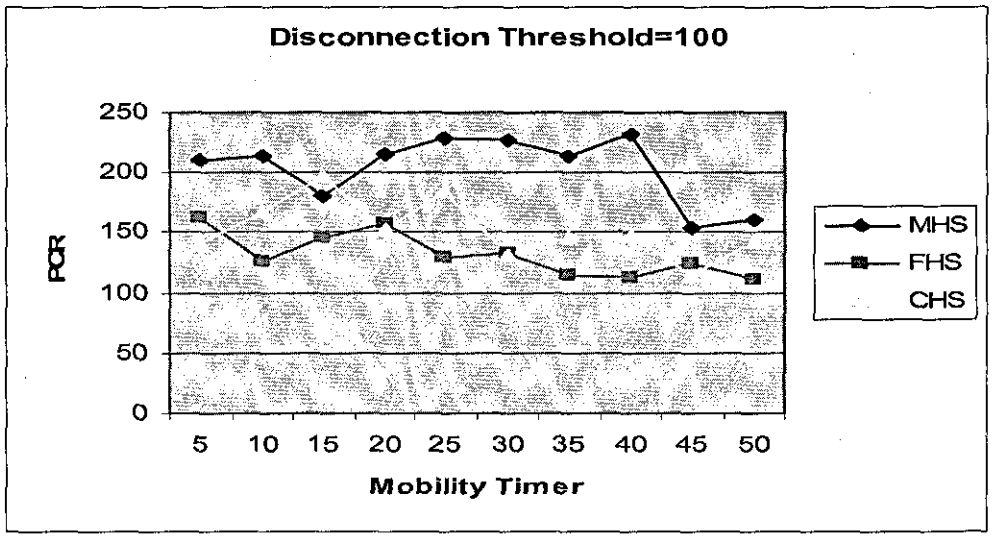


Figure3. 12: Power consumption (BW<= 100)

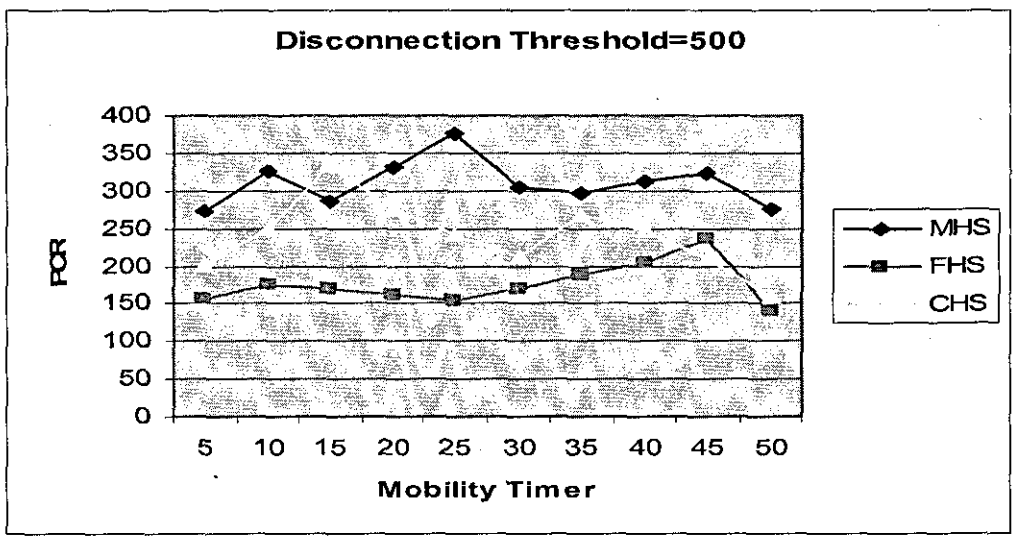


Figure3. 13: Power consumption (BW<= 500)

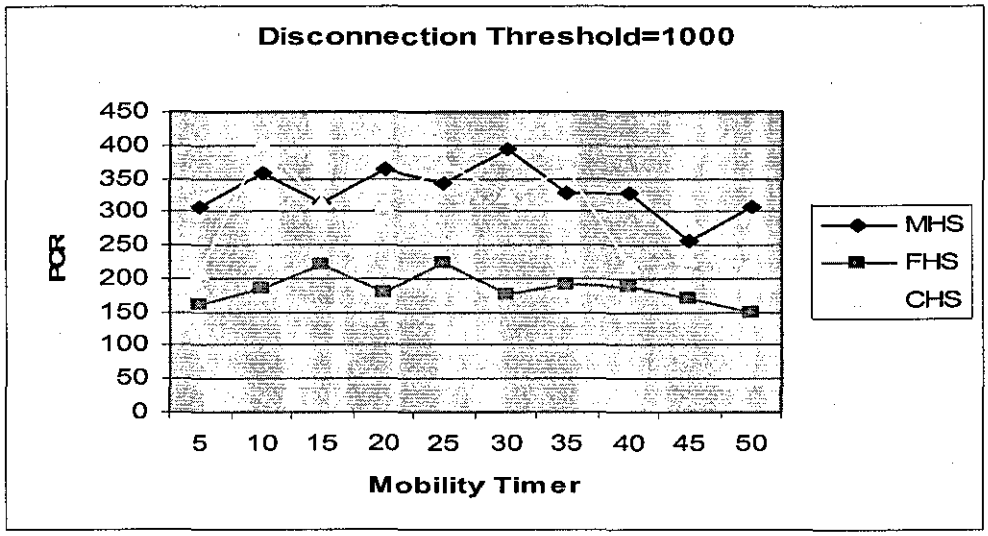


Figure3. 14: Power consumption (BW<= 1000)

The MHS strategy takes more processing time on a mobile host than the CHS strategy. This is because, with the CHS, the number of mobile sub transactions which are scheduled as transaction requests increase so that the time spent on the mobile host is reduced. As the network disconnection increase the CHS strategy still has a better performance in reducing power consumption at the mobile host than MHS. However, as the network disconnection probability increase further, The Power consumption under CHS strategy approach the power consumption of the MHS strategy as we can see in Figure 3.14.

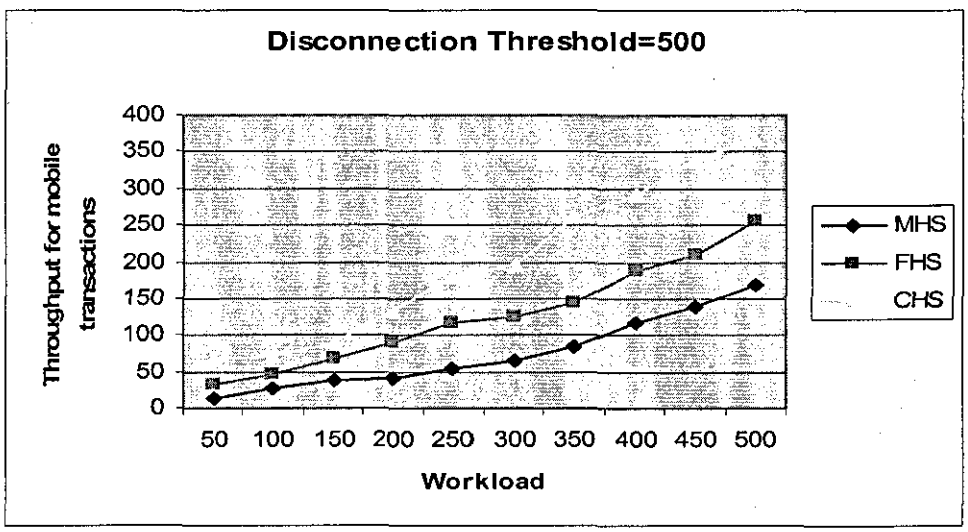


Figure3. 15: Throughput for mobile transactions (BW<= 500)

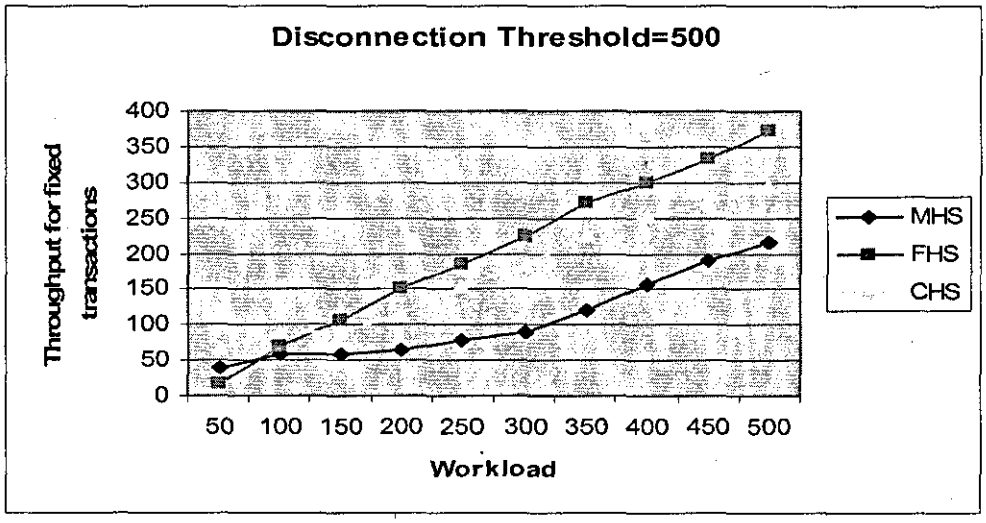


Figure3. 16: Throughput for fixed transactions (BW<= 500)

Figures 3.15 to 3.20 show the experimental results for system throughputs in terms of the number of completed transactions during a time interval. The first experiment show the result under different workload assumptions where the mobility and the disconnected threshold are set to 20 and 500 respectively. The purpose of this experiment is to show how the CHS gives better performance than other strategies as workload increases at the average network connection for mobile transactions and maintain a comparable throughput with FHS for fixed transactions. As we can see in Figure 3.15 the CHS strategy consistently demonstrates better performance than MHS and FHS strategies for a mobile transaction where Figure 3.16 show how the CHS strategy are close to the FHS strategy in term of fixed transaction throughput.

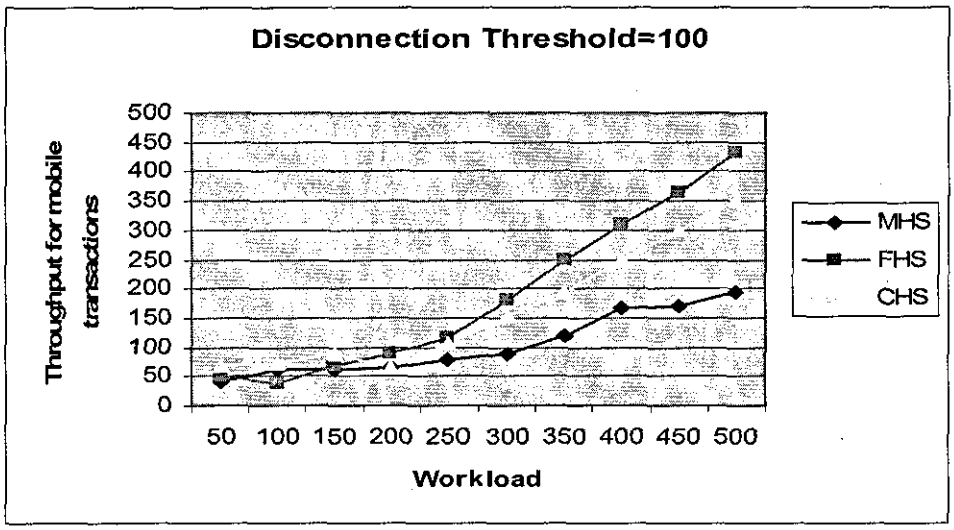


Figure3. 17: Throughput for mobile transaction (BW<= 100)

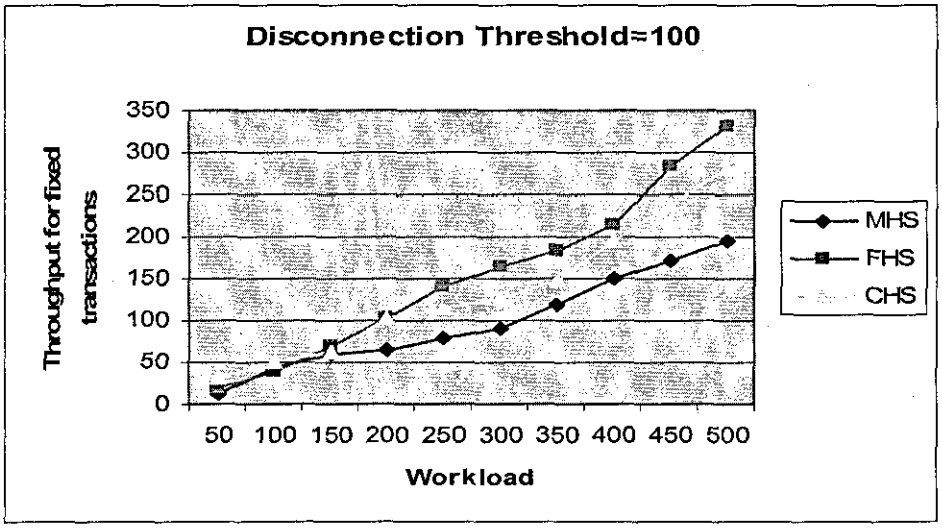


Figure3. 18: Throughput for fixed transactions (BW<= 100)

Figure 3.17 and 3.18 illustrates that with low disconnection; the FHS approach has the highest throughput for both types of transactions where there is an improvement in fixed transaction throughput over the case of medium disconnection. While the MHS the lowest and the CHS approach is in between the two. This result is consistent with the result shown in Figure 3.6 and 3.7 as an approach with a lower throughput can have a longer response time. However, Figure 3.19 shows a degradation of all strategies with high disconnection probability for mobile transactions. The CHS can produce better throughput over

the other two approaches because the basic transaction under the FHS strategy is unable to get through during network disconnection. At the same time, despite that the MHS can carry on transaction processing with cached data it is ignoring the time interval during the connection period of the network. For the throughput of the fixed transaction, FHS strategy still have the superior over the other two strategies and the CHS become closer to the MHS strategy than to FHS as we can see in the Figure 3.20.

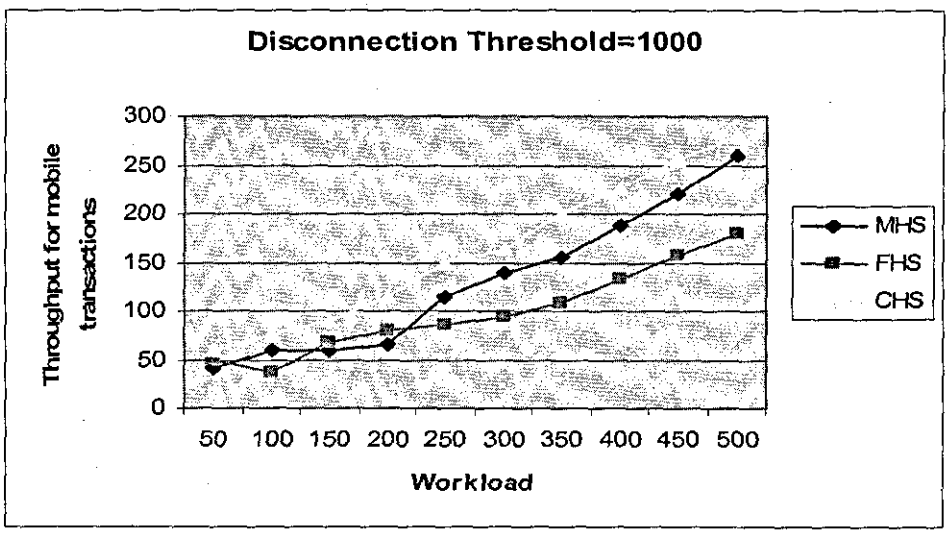


Figure3. 19: Throughput for mobile transactions (BW<= 1000)

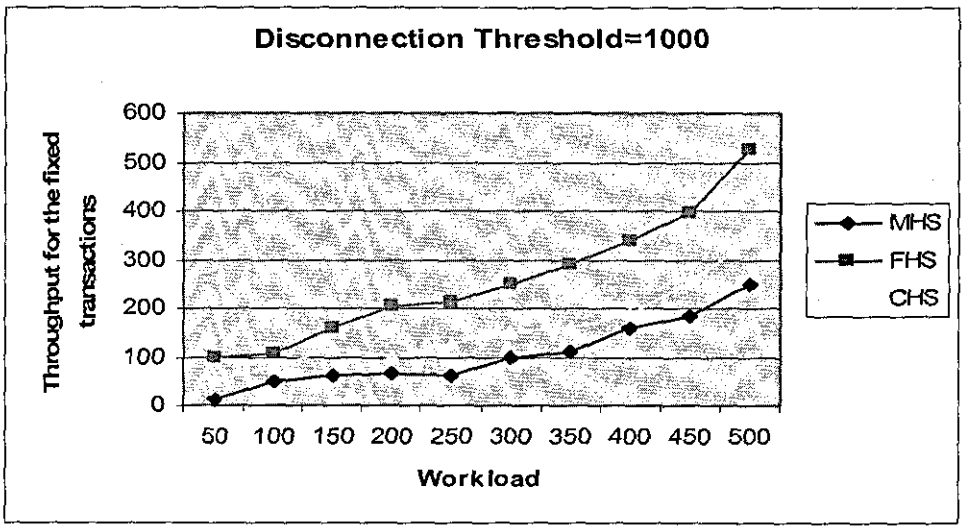


Figure3. 20: Throughput for fixed transactions (BW<= 1000)

CHAPTER 4

CONCURRENCY PROBLEM FOR MIXTURES OF TRANSACTIONS

4.1 Introduction

Each CC scheme enforces a specific serialization order. For example, the serialization order may be based on the start execution time, the completion time (i.e., the certification time), or a dynamically derived order, such as the data item access time. Based on this order, each CC scheme uses certain rules to decide whether or not a certain sequence of execution interleaving among transactions will satisfy the serialization order or requirement. However, serialization order can also be dynamically readjusted to resolve conflicts and reduce the number of transactions needed to be aborted or put into a wait state. We first consider several schemes without readjusting the serialization order, and then examine in Section different schemes that enlarge the eligible subset of serializable transactions by readjusting the serialization order.

4.1.1 Without readjusting serialization order

A TSO scheme assigns a time stamp to each transaction before it is executed. This time stamp usually is the start time of the transaction. If, during the course of execution, a transaction cannot be certified based on that time stamp ordering, it will be aborted.

The pure OCC scheme only checks to see whether a transaction can be certified at the end of its execution. Thus, the certification based on a weak lock

implementation effectively tries to certify transactions based on the completion order. Even though the broadcast OCC scheme tries to immediately abort transactions that are conflicting with the newly certified transaction, the serialization order is still based on completion times. However, the certification can also be based on time stamps, and the serialization order would be based on the time stamp order. As transactions may not complete in an order according to their start times, a serialization order based on start times may lead to more aborts than that based on completion times.

A 2PL scheme tries to preserve a serialization order based on the data item access order. (An SGT scheme is another example.) Here we assume no ordering among the compatible lock requests. If transaction X accesses a data item after transaction Y with an incompatible lock request, its serialization order must come after transaction Y. Even when the lock modes are compatible, if there are already intervening incompatible lock requests by other transactions, the serialization order of transaction X must also come after transaction Y. However, if all the locks held are released at the commit time, the serialization order of a 2PL scheme will also be the same as that of the completion times of the transactions. .

4.1.2 Dynamically readjusting serialization order

Dynamic time stamp allocation [87], and time stamp interval allocation [88] schemes have been proposed to have the certification time stamps dynamically derived and re-ordered either at data item access time or at the certification time. These schemes are designed to address the read-write conflict issue for transactions with a mixed read and write behaviour. The serialization order based on the completion times is more restricted than necessary, and a dynamically derived serialization order may avoid this type of abort. The time stamp interval approach can be based either on TSO [7, 71], or on the OCC certification approach. However, as pointed out in [69], the time stamp intervals derived by the TSO-based approach are limited since the serialization order is determined as soon as the conflicts occur. There are different ways to explore the concept of time stamp interval based on the certification- oriented approach. The basic idea is that each version or value of a data item is only valid for a certain period of time, i.e., between two consecutive updates. The transaction which has read a particular

version of a data item can only be certified with a time stamp in the valid interval of the data. If multiple data items are read, an intersection of their valid intervals, which may be null, has to be taken. Additionally, the data items updated by a transaction cannot be read by a certified transaction with a later time stamp. In [88], the timestamp or interval of timestamps was dynamically derived by maintaining a limited time stamp history of accessing transactions for each data item currently being accessed. At the transaction certification time, for each accessed data item the time stamp of the accessed version is compared with the time stamp history of the data item to determine its valid interval. This provides the information to re-order transactions at the certification time and to derive a back-shifted time stamp for certification in order to eliminate most unnecessary aborts due to read-write conflicts. This is referred to as certification based on Time Stamp History (TSH).

4.2 Mixed system model

The purpose of this subsection is to define a mixed system model depicted in Figure 4.1. We assume that the database system consists of two major components: the transaction manager (TM) and the data manager (DM). The TM maintains a transaction table to record the execution status of both mobile and fixed host transactions in the system. The TM includes two components: scheduler and data manager. The scheduler is responsible for concurrency control. When the scheduler receives an operation, it determines whether the operation should be processed, blocked, or rejected. If an operation is rejected, the corresponding transaction will be restarted. The scheduler maintains an access-status table to detect any possible data conflicts. For example, if 2PL is adopted for concurrency control, the scheduler may maintain a lock table to record the locking status of any data items accessed by all executing transactions (and the collection of the blocked transactions due to lock conflicts). The same table can also be used for the optimistic methods. All data access requests issued by an operation are handled by the DM, which retrieves the required data item. Transactions involved in this system consist of a sequence of read and writes operations, and end with a commit or an abort operation. Transactions are considered to be atomic processes. That is, they translate a database from a

consistent state into another consistent state. There are two types of transactions in this environment: fixed or wired transactions (FT) and mobile transactions (MT).

A fixed transaction is submitted directly to a database on the same host while the mobile transaction is submitted by a mobile device. Like in [52], a MT is a mobile transaction which is issued by a mobile host. The participation of a MH introduces dimensions inherent to mobility such as: movement, disconnections and variations on the quality of communication. As we will see in the following, the supporting TMs have to adapt their functionalities to deal with these dimensions. In the scope of this work we focus on systems with a client-server architectures where clients are MHs or FHs interacting with databases by invoking transactions.

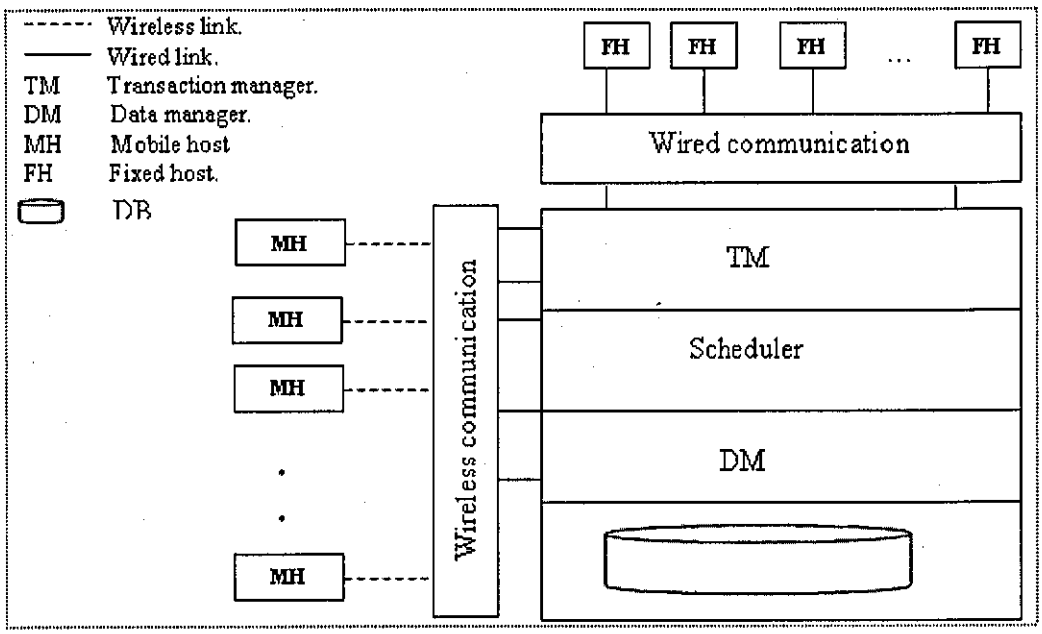


Figure4. 1: Mixed system model

In the next subsection we motivate our work in a mixed transaction environment, studying the blocking effect of the two-phase locking protocol which is the basic concurrency control technique used by most commercial database management systems. This study will estimate the delay of two-phase locking in the presence of bandwidth variability [89].

4.3 2PL-Locking Case Study

4.3.1 Blocking Delay

Bandwidth variability and handoff of wireless communications may cause mobile transaction operations to be delayed. A shared data item that is locked by a mobile transaction could hinder operations of other fixed or mobile host transactions from being executed. The delay of mobile transaction operations caused by the bandwidth variability can greatly affect the performance of any adopted concurrency control protocol. In locking-based concurrency control protocols, contention and blocking increase as the number of conflicting active transactions increase. For example, with the two-phase locking protocol, if a mobile transaction operation holds a lock on a data item, it remains locked until all mobile transaction operations are delivered. This delivery will suffer from transmission delay caused by bandwidth variability. As a result, the contention and conflicts increase and the performance of fixed host transactions will degrade dramatically. Although many excellent concurrency control protocols have been proposed for mobile database systems, most of these protocols ignore the effect of mobile transaction scheduling on the performance of fixed host and mobile transaction execution on each other. The unpredictable propagation delay of mobile transaction delivery imposes a serious overhead on the execution performance of both mobile and fixed host transactions. The mobility of clients in a mobile computing system also greatly affects the distribution of workload in the communication network. Disconnection between clients and stations is common. The poor quality of service provided by a mobile network seriously increases the overheads in resolving the data conflicts and affects the performance of existing concurrency control protocols which do not consider the characteristics of the mobile computing environment.

4.3.2 Bandwidth variability

Bandwidth variability occurs as the MH changes location. The ability to change location while retaining network connection is the key motivation for mobile computing. As mobile computers move, signal strength to the device varies, which can cause a loss of data or variations in bandwidth. This increases the time between mobile transaction's operations arriving at the fixed host. As a

consequence, the number of blocked fixed host transactions will be increased. Bandwidth variability occurs for two main reasons:

1. Different traffic loads: because the bandwidth is divided among the mobile users sharing a cell.

2. Handoff: Due to a change in the physical location, an MH can switch its supporting MSS when moving to a different cell. This leads to the need for a hand-off procedure to enable the new MSS involved to support and maintain the connection with the MH.

4.3.3 Effect of bandwidth variability

Consider the Figure 4.2 which represents a shared data item that may be accessed by fixed or mobile transactions. For each data item D_i there is a queue which contains an operations come from both types of transactions. Assume that a transaction becomes an active transaction if its first operation is being scheduled and sent to the specified queue based on the required data item. In the presence of bandwidth variability, there is a high probability that the time between arrivals of mobile transaction operations will be of a variable length. This will result in an increase in the number of active transactions in the system which, in turn, increases the contention and wait time for both mobile and fixed host transactions. The wait time of a new active transaction at a data item will be the sum of the wait times that transactions ahead in the queue will experience in obtaining the locks on data items they have yet to acquire, plus the total residence time at the I/O and CPU that will accumulate in processing these data items, together with a portion of the processing time for the current data item.

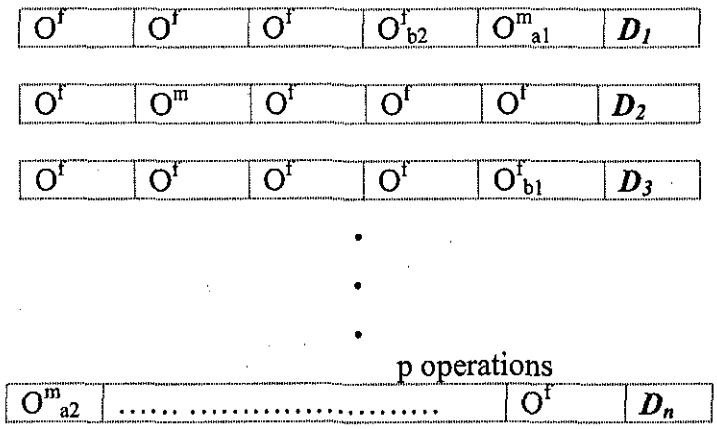


Figure4. 2: Mixed transactions interleaving operations

In Figure 4.2, transaction T_b has to wait for the mobile transaction T_a on data item D_1, and the mobile transaction operation O^m_{a2} on a data item D_n sees p other mobile or fixed operations ahead of it. So, T_b remains blocked until all operations ahead of it release their locks (in Figure 4.2, we assume, for simplicity, that T_b has two operations only). In the case of low bandwidth variability, there are q operations instead of p that O^m_{a2} see ahead of it, where $q < p$. So the blocking for mobile transactions is less in the case of a low bandwidth variability environment. In general, the blocking amount of transaction T_a is a function of the queue length ahead of its last operation.

In [68] the delay due to blocking is estimated analytically by using a queuing network model in its equilibrium state as follows. All I/O is grouped as a single server. There is one lock server for each lockable entity in the database and its service time for a transaction is the total time the entity must stay locked by the transaction. This includes the transaction's residence time at the CPU and I/O for the current lock together with the delays and service times for the remaining other locks it has yet to acquire. Each arriving transaction will eventually circulate between the aggregate of lock servers, the I/O and the CPU, a number of times equal to the total number of locks it requires. Thus, in equilibrium, an external arrival rate of λ gives rises to transaction arrival rate of $\lambda' = k\lambda$ at the lock servers, where k is the mean number of data items that transactions request. Also, since each of the N data items is equally likely to be requested, the arrival rate at any particular lock server will be $k\lambda/N$. The indistinguishability of data items implies that the behaviour of any lock server will yield the same delay. The wait time of a new arrival at a lock server will be the sum of the wait times that transactions ahead in the queue will experience in obtaining the locks on data items they have yet to acquire, plus the total residence time at the i/o and CPU that will accumulate in processing these data items together with a portion of the processing time for the current data item.

For the first operation of the mobile transaction the cost will be $(j*(T + D))+T$ where j is the number of locks to be acquired after the current one, T is the average processing time for any transaction operation at the database server. The

second T represents the processing time required for the data item currently being queued for. The $j*(T+D)$ accounts for the fact that the data item to be locked on the current queue will have to remain locked until the transaction acquires the remaining j locks (requiring j delays) and serving those j operations (requiring j multiples of T). In the case of granting a lock to the mobile transaction's operation, it has to remain locked until the mobile transaction acquires its remaining locks. Let J be the average number of locks transactions require after the current one and $q(i)$ the probability that the latest transaction operation sees i operations ahead of it. Then, the delay experienced by either fixed or mobile transactions, where the last delivered operation sees n operations ahead of it, becomes the average queue length multiplied by the time needed for each operation in the queue to get the lock on the data item as follows, where Q is the average queue length:

$$
\begin{aligned}
D &= q(0)*0 + q(1)*(J*(T+D)+T) \\
&\quad + q(2)*2*(J*(T+D)+T) \\
&\quad + q(3)*3*(J*(T+D)+T) + \ldots + q(n)*n*(J*(T+D)+T) \\
&= (q(0)*0 + q(1)*1 + q(3)*3 + \ldots + q(n)*n)*(J*(T+D)+T) \\
&= \left[\sum_{i=0}^{n} q(i)*i\right]*(J*(T+D)+T) \\
D &= Q*(J*(T+D)+T)) \qquad \textit{Equ 1}
\end{aligned}
$$

In the presence of bandwidth variability, there is a high probability for the time between arrivals of mobile transaction operations to be of variable length. This results in an increasing number of active transactions in the system. So, the average queue length in the presence of bandwidth variability will become Q+ Q_{inc}. Based on the assumption of uniform data access, each arriving mobile or fixed host transaction has to visit different queues based on the data items being accessed by each such transaction, and the increment in queue length Q_{inc} results from the increment in the number of active transactions arriving at the system during the delivery of the next transaction operation. Assume that λ_a is the arrival rate for the newly active transactions from both types of transactions and k is the mean number of data items that transactions request (equal for both type of transactions). Also, since each of the N data items is equally likely to be

requested, the increment in arrival rate at any particular data item's queue will be $\frac{k * \lambda_a}{N}$. In the case of delay due to the bandwidth variability, Q_{inc} replaces Q in Equ 1. So, $D_{bv} = Q_{inc} * (J * (T + D_{bv}) + T))$. For a Poisson arrival rate and exponential service time as in [89] the delay due to the bandwidth variability become

$$D_{bv} = \frac{\lambda_a * k * (J * (T + D_{bv}) + T) / N}{1 - \lambda_a * k * (J * (T + D_{bv}) + T) / N} * (J*(T+D_{bv}) + T)) \qquad Equ2$$

Equ. 2 is simply a quadratic equation for D_{bv} with two roots. The increment in the delay due to the bandwidth variability will be the smaller of the two roots which is

$$D_{bv} = \frac{-(T(J+1)(1+2J) - N/\lambda_a k - \sqrt{((T(J+1)(1+2J) - N/\lambda_a k)^2 - 4*(J^2+J)*T^2(J+1)^2}}{2*(J^2+J)}$$

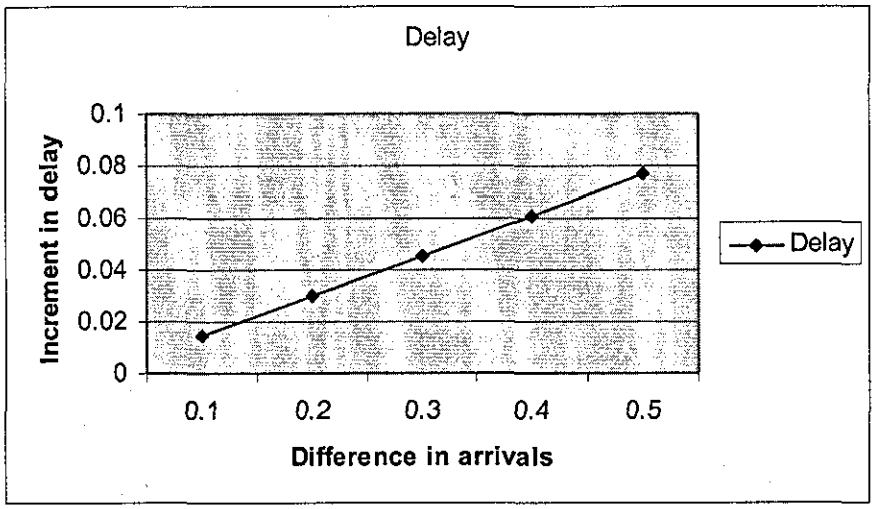


Figure4. 3: Delay due to bandwidth variability

As we can see in Figure 4.3, when the difference between arrivals increases, the delay for both mobile and fixed active transactions will increase which will affect the performance of the both kinds of transactions and consume more wireless resources in the case of the mobile transaction. From Figure 4.3, we can draw the following conclusion: the 2PL protocol is no longer appropriate for a mixture of mobile and fixed host transactions and this should lead us to think about adapting two phase locking or switching to other concurrency control paradigms which may be more suitable in such environments.

CHAPTER 5

CONCURRENCY CONTROL APPROACHES

In this chapter, we propose the two concurrency control approaches for transaction mixtures which were motivated in the previous chapter in subsection 4.3. Two approaches 'Lock-Mix' and 'OCC-Mix' are introduced. The former is a lock-based approach which combines the OCC and 2PL protocols. The latter is an optimistic approach which combines the optimistic and the timestamping protocols. The main objective of our approaches is to overcome the limitations of wireless environments by avoiding restarts and blockings of mobile transactions, caused by fixed or other mobile transactions, while still providing an opportunity for fixed transactions to finish their execution. The remainder of this chapter is organized in the following fashion. In Section 5.1, we describe our Lock-Mix approach and show how it deals with mixtures of concurrent mobile and fixed transactions. This will be done as follows. Sections 5.1.1 and 5.1.2 highlight the problems of locking and optimistic approaches, respectively, in mixed transaction environments. Qualitative comparisons between these approaches in mixed environments are presented in Section 5.1.3. Section 5.1.4 describes our Lock-Mix approach in detail. After that, our OCC-Mix approach is given in Section 5.2. Section 5.2.1.1 discusses the difference between the time interval and fixed timestamp method. Section 5.2.1.2 explains two basic methods used in optimistic concurrency control in the validation phase. A combination of the optimistic and timestamp interval concurrency control approaches forms our OCC-Mix approach for fixed and mobile transaction mixtures. The details are presented in section 5.2.3.

5.1 Lock-Mix approach

The Lock-Mix approach over transaction mixtures aims to overcome the problems of the 2-PL and OCC protocols if each was to be the sole concurrency control protocol. These problems derive from the nature of conflict resolution being used by these protocols.

5.1.1 Problems with a locking approach

The two-phase locking (2PL) mechanism introduced in [85] is now accepted as the standard solution to the concurrency control problem in traditional DBMSs. It depends on there being well-formed transactions, which do not lock, again, data items that have been locked earlier in the transaction, and whose execution is divided into a growing phase, in which locks are only acquired, and a shrinking phase, in which locks are only released. During the shrinking phase, a transaction is prohibited from acquiring locks. If, during the growing phase, a transaction attempts to acquire a lock that has already been acquired by another transaction, it is forced to wait until the lock is released. In mixed transaction environments, locking has been found to constrain concurrency and to add an unnecessary overhead. The following are the disadvantages of the 2PL protocol.

- Lock maintenance represents an unnecessary overhead for read-only transactions, which do not affect the integrity of the database.
- Releasing of locks is not permitted until the end of the transaction execution. Although not required, it is always done in practice to avoid cascaded aborts. It decreases concurrency.
- Most of the time it is unnecessary to use the locking that 2PL uses, to guarantee consistency, since most transactions do not overlap. Locking may be necessary only in worst cases.
- The performance of 2PL may severely degrade our mixed system, because the significant increase in the total number of concurrent transactions results in a high lock contention level and hence a high lock conflict probability. Communication delay that leads to longer duration of held locks makes the lock conflict probability even higher.

To best explain these problems in the mixed transaction systems, consider the following example.

Example 5.1: Consider the interleaving of operations showed in Table 5.1 below. Assume that transactions T_1, T_3 and T_4 are mobile transactions and the rest of the transactions are fixed.

Table5. 1: Operations interleaving for example 5.1

Time						
τ_6	τ_5	τ_4	τ_3	τ_2	τ_1	Data Item
					R_1	D_1
					R_3	D_2
					R_4	D_3
		W_4	R_3	R_1	W_2	D_4
			R_2	W_6	R_5	D_5
					W_5	D_6
					W_1	D_7
					W_3	D_8
					W_4	D_9
					R_1	D_{10}
					W_3	D_{11}
R_5	W_{12}	R_9	W_8	R_{10}	W_7	D_{12}
					W_4	D_{13}

In 2PL protocols (see Figure 5.1), transaction scheduling order is determined purely by the order in which transactions acquire locks. Once transaction T_5 locks the data item D_6, transaction T_5 is not able to unlock the data item D_6 until all T_5 operations get their own lock on their data items. Transaction T_6 is waiting for T_5 and transaction T_2 is waiting for T_6 on the same data item D_5. If we assume that $R_5(D_{12})$ is the last operation of transaction T_5, then $R_5(D_5)$ and $W_5(D_6)$ can not free these data items until $R_5(D_{12})$ get its lock to obey the 2PL rules. Also, transactions T_1, T_3, and T_4 will be delayed until T_2 frees data item D_4, which may occur after transactions T_6, T_{12}, T_9, T_8, T_{10}, and T_7 have finished. This will result in a cascading delay caused by the early blocking of the 2PL protocol.

time τ_1					
T_1	T_3	T_5	T_4	T_7	T_2
Read_Lock (D_1);	Read_Lock (D_2);	Read_Lock (D_5);	Read_Lock (D_3);	Write_Lock (D_{12});	Write_Lock (D_4);
Write_Lock (D_7);	Write_Lock (D_8);	Write_Lock (D_6);	Write_Lock (D_9);		
Read_Lock(D_{10});	Write_Lock D_{11});		Write_Lock (D_{13});		

time τ_2		
T_1	T_2	T_{10}
Read_Lock (D_4);	Write_Lock (D_5);	Read_Lock (D_{12});

time τ_3		
T_2	T_3	T_8
Read_Lock (D_5);	Read_Lock (D_4);	Write_Lock (D_{12});

time τ_4	
T_4	T_9
Write_Lock (D_4);	Read_Lock (D_{12});

time τ_5	time τ_6
T_{12}	T_5
Write_Lock (D_{12});	Read_Lock (D_{12});

Figure5. 1: 2PL Interleaving Example

5.1.2 Problems with an OCC approach

The goal of Optimistic Concurrency Control (OCC) proposed in [7] is to avoid these problems of 2PL. OCC requires each transaction to consist of three phases: a read phase, a validation phase, and a write phase. During the read phase, all writes take place on local copies of the records to be written. Then, if it can be established during the validation phase that the changes the transaction made will not violate serializability with respect to all committed transactions, the local copies are made global. Only then, in the write phase, do these copies become accessible to other transactions. There are two properties of OCC that distinguish it from other approaches. First, synchronization is accomplished entirely by restarts, never blocking. Second, the decision to restart or not is made after the transaction has finished executing. To best explain how an OCC protocol can poorly effect the transaction processing in our environment, consider the following example.

Example 5.2: Consider the same transactions as in Example 4.1 with the interleaving as shown below in Figure 5.2. In optimistic concurrency control, transactions are allowed to execute unhindered until they reach their commit point, at which time they are validated. Thus, the execution of a transaction consists of three phases: read, validation, and write phase. The key component among these is the validation phase where a transaction's destiny is decided. In OCC, the scheduling order is determined by the arriving order of transactions at the validation phase. If we assume that transaction T_2 reaches its validation phase first, all active transactions which conflict with T_2 should be restarted.

TID				
T_1	$R_1(D_1)$ $W_1(D_7)$ $R_1(D_{10})$		$R_1(D_4)$	
T_2	$W_2(D_4)$	$R_2(D_5), V_2$		
T_3	$R_3(D_2)$ $W_3(D_8)$ $W_3(D_{11})$		$R_3(D_4)$	
T_4	$R_4(D_3)$ $W_4(D_9)$ $W_4(D_{13})$			$W_4(D_4)$
T_5	$R_5(D_5)$ $W_5(D_6)$			$R_5(D_{12})$
T_6		$W_6(D_5)$		
T_7	$W_7(D_{12})$			
T_8		$W_8(D_{12})$		
T_9	$R_9(D_{12})$			
T_{10}		$R_{10}(D_{12})$		
T_{12}	$W_{12}(D_{12})$			

Figure5. 2: OCC Interleaving Example

5.1.3 Qualitative comparisons

From the traditional viewpoint, various concurrency control algorithms basically differ in two aspects: the time when they detect conflicts and the way that they resolve conflicts. The locking and optimistic approaches in their basic form represent the two extremes in terms of these two aspects. Locking detects conflicts as soon as they occur and resolves them using blocking. An optimistic scheme detects conflicts only at transaction commit time and resolves them using restarts. In mixed transaction environments, the way these approaches are used in resolving data conflict for such transaction mixtures makes for further differences between concurrency control mechanisms. The impact of these further differences on performance is the major theme in the performance study of concurrency control in mixture transaction systems here. With respect to the impact of conflict resolution methods, the effect of blocking and restart should be considered in the

context of the available amount of system resources. Generally, a blocking-based conflict resolution policy conserves resources, whilst a restart-based policy wastes more resources. Previous performance studies on conventional database systems have shown that locking algorithms, that resolve data conflicts by blocking transactions, outperform restart-oriented algorithms, in environments where physical resources are limited. Also, it has been shown that, if resource utilizations are low enough so that a large amount of wasted resources can be tolerated, and there are a large number of transactions available to execute, then a restart-oriented algorithm that allows a higher degree of concurrent execution is a better choice.

In this study, we investigate the effect of blocking and restart in the context of mixed transaction environments. The timing of conflict detection and resolution also has a major impact on performance. With an optimistic algorithm using backward validation, delayed conflict resolution results in the wastage of more resources than the locking protocol, since a transaction can end up being restarted after having completed most of its execution. With forward validation, however, this problem is eliminated, since any transaction that reaches the validation phase is guaranteed to commit, and transactions involved in any nonserializable execution restart early in their read phase. Also the delayed conflict resolution of an optimistic approach helps better decisions to be made in conflict resolution, since more information about conflicting transactions is available at this later stage. On the other hand, the immediate conflict resolution policy of locking schemes may lead to useless restarts and blocking in mixed systems due to their lack of information on conflicting transactions at the time of conflict resolution.

5.1.4 Approach details

As mentioned in Section 5.1.3, a conventional 2PL scheme tends to suffer from a cascade of blocking. On the other hand, an OCC scheme may suffer from wasting of resources due to transaction aborts and restarts. In OCC, transactions become more vulnerable (i.e., more likely to be involved in conflicts and be marked for abort) as they make more progress toward completion, since more data items are accessed. As a result, longer transactions would incur higher abort probabilities in OCC. In a mixture database environment the abortion of mobile transactions by a

fixed host transaction will waste a valuable wireless resource to save a reliable wired resource. Furthermore, the cost of aborting a nearly completed mobile transaction by another mobile transaction is certainly higher than that of aborting a newly-started mobile transaction.

Lock-Mix approach based on [65] addresses this issue by combining the advantages of both OCC and 2PL. The transaction execution of both types, mobile and fixed, is divided into a non-blocking phase and a blocking phase. At the start of the execution, a fixed or mobile transaction is in the non-blocking phase where it simply obtains a fixed lock synchronously for each data item access. In this phase neither fixed nor mobile transaction will block other fixed or mobile transactions. After a mobile or fixed transaction has accessed a predefined number of data items, it tries to enter the blocking phase to prevent other transactions from aborting it at a late stage of its execution. It will try to convert all the fixed locks, on the already accessed data items, into mobile locks as in the certification process of a conventional OCC. If successful, the transaction will switch to the blocking phase.

We have two parameters, μ and η, associated with fixed and mobile transactions respectively, which are used as metrics for determining the switching point from non-blocking to blocking phase. They represent predefined values which dictate after how many operations with their own fixed locks on their data items, can a switch to a blocking phase occur. Fixed host transactions, can finish in the first stage of their executions. For a mobile transaction, after it decides to enter the blocking phase, it then obtains a mobile lock on each subsequent data item access and waits for locks obtained by other mobile transactions which have already converted to the blocking phase, if held under an incompatible mode. This prevents mobile transactions from aborting at a late stage of their life, and also reduces the average holding time of a mobile lock, thus reducing the blocking effect caused by fixed host transactions under the pure 2PL protocol.

```
Lock-Request (Ti, Op.ID, Mode, Dk)
Begin
  If (Ti.type = mobile and Op.ID ≥ η ) or (Ti.type = fixed and Op.ID ≥ μ ) Then
// check the number of operation already granted locks on their data items to
switch into blocking phase.
     Execute mobile lock request (Ti, Mode, Dk)     // switch into blocking phase.
Else
     Execute fixed lock request (Ti, Mode, Dk)       // stay in non-blocking phase.
End
```

Figure5. 3: Lock request for mobile and fixed transactions

So, each data item can be locked in different lock modes such as shared and exclusive modes. Shared locks on the same data item are compatible, whilst an exclusive lock is incompatible with a shared lock or another exclusive lock on the same data item. Similarly, in the proposed scheme, the CC manager maintains a lock table where each data item can be locked in either shared or exclusive mode. Furthermore, two lock types, mobile and fixed, are used in different ways based on the type of requesting transaction and the order of operations inside the transaction as we can see in Figure 5.3. The lock type compatibility matrix is given in Table 5.2, and the pseudo-code for the concurrency control algorithm is given in Figures 5.3, 5.4, 5.5 and 5.6. As shown in Table 5.2, the mobile lock request is superior to the incompatible fixed lock requests denoted by "S", in the sense that if the requested data item is currently held by fixed locks, the incompatible fixed locks are released to grant the mobile lock request and the fixed lock holders are marked for abort. Fixed lock requests are always compatible with other fixed locks. A fixed lock request in shared mode and a mobile lock held in shared mode are regarded as compatible. Otherwise, a fixed lock request is not compatible with a mobile lock, and the requester has to be put into a wait state.

Table5. 2: Compatibility Matrix for Lock-Mix

Holder / Requeste		Fixed		Mobile	
		Read	Write	Read	Write
Fixed	Read	Y	Y	Y	N
	Write	Y	Y	N	N
Mobile	Read	Y	N,S	Y	N
	Write	N,S	N,S	N	N

Execute fixed lock request (T_i, Mode, D_k)

Begin

If no holder on requested data item **Then**

Return success //grant the lock request

Else If all holders with compatible locks **Then**

Return success //grant lock request

Else wait for the lock

End

Figure5. 4: Execute fixed lock request

```
Execute mobile lock request (Ti, Mode, Dk)
Begin
If no holder on requested data item Then
  Return success                                  // grant the lock request
     Else If all holders with compatible locks Then
          Return success                          // grant the lock request
          Else If all holders with lock mode that can be superseded Then
               Mark the current lock holders for abort and
                    Return success  // grant the lock request
               Else wait for the lock
End
```

Figure5. 5: Execute mobile lock request

```
Execute unlock request (Ti, Mode, Dk)
Begin
If the unlock request is on a mobile lock type Then
     If lock wait queue is not empty Then
          If mobile lock waiting Then
                    Grant the next mobile lock request and subsequent
               compatible mobile and fixed lock requests
          Else if no mobile lock waiting Then
                  Grant all pending compatible fixed lock requests
               Else //   no fixed lock waiting
                    Grant all pending fixed lock requests
End
```

Figure5. 6: Execute unlock request

As we see from the figures above that describe the Lock-Mix protocol, the execution of a fixed transaction is scheduled as in the conventional 2PL protocol until the commit time. So, the possibility that a fixed transaction aborts or blocks a mobile transaction is highly reduced. Furthermore, a mobile transaction may be blocked by a fixed host transaction if the number of the fixed transaction executed operations exceeds μ. A mobile transaction, which aborts as a result of converting a fixed transaction from a non-blocking to a blocking stage, will be in its early stages. To explain why this approach is better than the 2PL and OCC approaches for transaction mixtures, considers the following example.

Example 5.3: Consider the same transactions as in Example 5.1 with the same operation interleaving. Figure 5.7 below shows the lock request for each transaction under the Lock-Mix approach. In the Lock-Mix protocol, the fixed lock is used to simulate the OCC protocol and the mobile lock is used to simulate blocking resulting from conflicts of mobile and other mobile or fixed transactions. Recall, that the two parameters η and μ, associated with fixed and mobile transactions respectively, are used as a metrics for determining the switching point from a non-blocking to a blocking phase. Assume that $\eta = 4$, $\mu = 6$ the Lock-Mix protocol can be illustrated as follows:

time τ_1					
T_1	T_3	T_5	T_4	T_7	T_2
F_R_lock (D_1);	F_R_lock (D_2);	F_R_lock (D_5);	F_R_lock (D_3);	F_W_lock (D_{12});	F_W_lock (D_4);
F_W_lock (D_7);	F_W_lock (D_8);	F_W_lock (D_6);	F_W_lock (D_9);		
F_R_lock (D_{10});	F_W_lock (D_{11});		F_W_lock (D_{13});		

All transactions (fixed and mobile) are granted their fixed lock request because Op.ID < 4 for mobile transactions and Op.ID<6 for fixed transactions.

time τ_2		
T_1	T_2	T_{10}
M_R_lock (D_4);	F_W_lock (D_5);	F_R_lock (D_{12});

T_1: grant the lock on D_4

T_2: marked for abort

time τ_3		
T_2	T_3	T_8
F_R_lock (D_5);	M_R_lock (D_4);	F_W_lock (D_{12});

T_3: grant the lock on D_4

time τ_4	
T_4	T_9
M_W_lock (D_4);	F_R_lock (D_{12});

T_4: wait for T_1 and T_2 on D_4

time τ_5	time τ_6
T_{12}	T_5
F_W_lock (D_{12});	F_R_lock (D_{12});

Figure5. 7: Lock-Mix Interleaving Example

Both fixed and mobile transactions start their execution by requesting a fixed read and write lock. After the mobile transaction executes its fourth operation, it will switch to the blocking phase and can't be restarted by the fixed transaction or other mobile transactions. The same is true for the fixed transaction after executing its sixth operation. In Figure 5.7 above, transactions T_1, T_3 and T_4 are willing to lock the data item D_4, which is already locked by transaction T_2. Since R_1 (D_4) is the 4^{th} operation of T_1, it will acquire a mobile lock on D_4. Because the mobile lock requested by T_1 supersedes the fixed lock granted to transaction T_2 on D_4, T_2 will be marked for abort and T_1 granted a mobile lock on D_4. So, T_1 is switched into a blocking stage and can not be marked for abort by other fixed or mobile transactions. When transaction T_3 tries to get the lock on D_4 it succeeds because it's compatible with R_1 (D_4). Transaction T_4 tries to get a mobile lock on D_4 in an exclusive mode. So, it has to wait for unlock (D_4) by transaction T_1 and T_3. Since the remaining number of operations for the transactions to switch to their blocking stage is less than transaction length, (this depends on the η , μ values) we expect a reasonable reduction in the average blocking time for all mobile transactions. Thus, transactions T_1 and T_3 don't have to wait for transaction T_2 which would have to wait for T_5, T_6, T_7, T_9, T_8, T_{12} and T_{10} in order to unlock the data item D_4 according to the 2PL protocol. On the other hand, other transactions still have the opportunity to finish their execution without being restarted by other fixed or mobile transactions that enter their validation phase according to the pure OCC protocol.

5.2 OCC-Mix approach

Flexibility of a timestamp interval over a fixed timestamp method along with the validation process of the OCC protocol, make the OCC-Mix approach a promising candidate for scheduling mobile and fixed transaction mixtures where the main part of such mixtures suffers from wireless constraints.

5.2.1 Time interval versus fixed timestamp

In a fixed timestamp method, timestamps are chosen for transactions when they begin. Whenever a transaction makes a request that would create a conflict between itself and another transaction, the timestamps of the two transactions are

compared. If the order of the timestamps is the same as the serialization order required by the conflict, the request is allowed; otherwise the requesting transaction is aborted and restarted with a new timestamp. Thus, the transaction serialization order is essentially fixed in advance, which has the potential to cause many unnecessary aborts. Using the Time Intervals method [87] each transaction has two timestamps. These timestamps can be thought of as the upper and lower bounds of an interval of timestamp time in which the transaction must appear in the serialization order. Time intervals are partially ordered, with the relations '<'applying only to intervals that are disjoint (note that non-disjoint intervals can always be truncated in such a way as to impose either ordering on them). Every transaction's initial interval spans the entire allowable timestamp range, representing the fact that there is no restriction on its place in the serialization order until it encounters conflicts with other transactions. When a conflict is encountered, the time intervals of the transactions involved are compared. If the intervals are disjoint, then their relative ordering has already been established; in this situation the algorithm is exactly the same as for the tied timestamp case. On the other hand, if the intervals overlap, then they can certainly be truncated so as to effect the desired ordering, after which the request can be granted. In the limiting case, an interval may be shrunk down to a single point, which is then no different in its interpretation than a fixed timestamp.

5.2.2 Forward versus backward validation

In optimistic concurrency control, transactions are allowed to execute unhindered until they reach their commit point, at which time they are validated. Thus, the execution of a transaction consists of three phases: a read phase, a validation phase, and a write phase. During the read phase, all writes take place on local copies of the records to be written. Then, if it can be established during the validation phase that the changes the transaction made will not violate serializability with respect to all committed transactions, the local copies are made global. Only then, in the write phase, do these copies become accessible to other transactions. There are two properties of OCC that distinguish it from other approaches. First, synchronization is accomplished entirely by restarts, never blocking. Second, the decision to restart or not is made after the transaction has

finished executing. The key component among these is the validation phase where a transaction's destiny is decided. Validation comes in several flavours, but every validation scheme is based on the following principle to ensure serializability: "If a transaction T_i is serialized before transaction T_j, the write of T_i should not affect the read phase of T_j". Generally most validation processes can be carried out basically in either of the following two ways [4].

5.2.2.1 Forward Validation

In this scheme, validation of a transaction is done against currently running transactions. This process is based on the assumption that the validating transaction is ahead of every concurrently running transaction still in read phase in the serialization order. Thus the detection of data conflicts is carried out by comparing the write set of the validating transaction and the read set of active transactions. That is, if an active transaction, T_i has read an object that has been concurrently written by the validating transaction, the value of the object used by T_i is not consistent. Such data conflicts can be resolved by restarting either the validating transaction or the conflicting transactions in the read phase. Optimistic algorithms based on this validation process are studied in [4, 71].

Let T_a (a = 1, 2,..., n, a $\neq$ v) be the conflicting transactions in their read phase. Then the forward validation can be described by the procedure in Figure 5.8.

```
Validate (Tv);
Begin
  valid := true;
    For each Tc (c = 1, 2,..., n)
     Begin
        If WS (Tc) ∩ RS (Tv) ≠ [] Then
        valid := false;
        If not valid Then exit loop;
     End
        If valid Then
         Commit WS (Tv) to database
       Else
        restart (Tv);
End
```

Figure5. 8: Forward validation

5.2.2.2 Backward Validation

In this scheme, the validation process is carried out against (recently) committed transactions. Data conflicts are detected by comparing the read set of the validating transaction and the write set of committed transactions, since it is obvious that committed transactions precede the validating transaction in serialization order. Such data conflicts should be resolved to ensure serializability. The only way to do this is to restart the validating transaction. The classical optimistic algorithm in [4] is based on this validation process. Let T_v be the validating transaction and T_c ($c = 1, 2,...., n, c \neq v$) be the transactions recently committed with respect to T_v, i.e., those transactions that commit between the time when T_v starts executing and the time at which T_v enters the validation phase. Let RS (T) and WS (T) denote the read set and write set of transaction T, respectively. Then the backward validation operation can be described by the procedure in Figure 5.9.

```
Validate (Tv);
Begin
  Valid := true;
    For each Ta (a = 1, 2,..., n)
      Begin
       If RS (Ta) ∩ WS (Tv) ≠ [] Then
       valid := false;
      End
       If valid Then
         commit WS (Tv) to database
       Else
         conflict resolution (Tv);
End
```

Figure5. 9: Backward validation

5.2.3 Approach details

Our OCC-Mix is an optimistic protocol based on the dynamic adjustment of serialization order. As in [69], OCC-Mix uses the notion of timestamp intervals to record and represent serialization orders induced by concurrency dynamics. Timestamps are associated with both transactions and data items. Each data item has a read and a write timestamp, where the read and the write timestamps are the

timestamps of last committed transactions that have read or written to the data item, respectively. For each transaction, OCC-Mix associates with each active transaction a timestamp interval expressed as a [lower bound (lb), upper bound (ub)] pair. The timestamp interval denotes the validity interval of a transaction. The timestamp intervals are also used to denote serialization order between transactions. For example, if T_i (with timestamp interval $[lb_i,ub_i]$) is serialized before T_j (with timestamp interval $[lb_j ,ub_j]$), denoted $T_i \rightarrow T_j$, then the following relation must hold: $ub_i < lb_j$.

5.2.3.1 Adjustment of timestamp interval

Each transaction at the start of execution is assigned a timestamp interval of [0, ∞], i.e., the entire timestamp space. As the transaction proceeds through its lifetime in the system, its timestamp interval is adjusted to reflect serialization dependencies as they are induced. Serialization dependencies may need to be modified either when a transaction is accessing data items in its read phase, or when it belongs to the conflict set of a different validating transaction.

5.2.3.1.1 Adjustment at the read phase

In this phase, the timestamp interval is adjusted with regard to the read and write timestamps of the data item read or updated. In the process of adjusting, the timestamp interval may 'shut out', i.e., become empty. In such a case, the transaction cannot be successfully serialized and needs to be restarted. Note that this is one of the major differences between conventional protocols and protocols based on dynamic adjustment of serialization order. In conventional OCC algorithms, restarts can only occur at validation times. In our case, however, transactions can restart at other times if a timestamp interval shut out is detected. The exact mechanics for these adjustments are shown in the procedures given in Figure 5.10. Note that, in the remainder sections, we use the notation $TI(T_i)$ to denote the timestamp interval of transaction T_i and $RTS(D_i)$ and $WTS(D_i)$ to denote the read and write timestamps respectively of data item D_i. As a transaction successfully validates, a final timestamp is assigned to it.

```
Read_Phase (Ta)
  Begin
          For each Di in RS (Ta)
            Begin
               Read (Di);
               TI (Ta) = TI (Ta) ⋂ [WTS (Di), ∞]
               If TI (Ta) = [] Then Restart (Ta);
            End
          For each Di in WS (Ta)
            Begin
               Pre-write (Di);
               TI (Ta) = TI (Ta) ⋂ [WTS (Di), ∞] ⋂ [RTS (Di), ∞]
               If TI (Ta) = [] Then Restart (Ta);
            End
  End
```

Figure5. 10: Adjustment of TI (T_a) at the read phase

5.2.3.1.2 Adjustment at the validation phase

In the case of being in the conflict set of a different validating transaction, the timestamp interval of the active transaction is modified to dynamically adjust the serialization order. The adjustment of the serialization order for both mobile and fixed transactions implemented with timestamp intervals creates a partial order between transactions based on conflicts and transaction type. Suppose we have a validating transaction T_v and an active transaction T_a. Let TS (T_v) be the final timestamp of the validating transaction T_v and TI (T_a) the timestamp interval of the active transaction T_a. Let TI (T_v) be the timestamp interval of the validating transaction and type (T_i) be the transaction type where type (T_i) $\in$ {mobile, fixed}. We assume here that there is no blind write. So, there are two possible types of conflicts which are resolved using adjustment of serialization order between T_v and T_a:

(1) *read-write* conflict which occur when the RS (T_v) $\cap$ WS (T_a) $\neq \varphi$ which can be resolved by forward adjustment.

(2) *write-read* conflict which occur when the RS (T_a) $\cap$ WS (T_v) $\neq \varphi$ and resolved by backward adjustment..

```
Iterate conflicting transaction (Ta, Tv)
Begin
    For all Di ∈ (RS (Tv) ∪ WS (Tv))
        Begin
            For each Ta ∈ conflicting set of the validating transaction
              Begin
                If  Di ∈ (RS (Tv) ∩ WS (Ta)) Then
                Forward adjustment (Ta, Tv);
                If  Di ∈ (RS (Ta) ∩ WS (Tv)) Then
                Backward adjustment (Ta, Tv);
              End
        End
End
```

Figure5. 11: Iterate Readset/Writeset of Validating Transaction

The adjustment of timestamp intervals (TI) in Figure 5.11 iterates through the read set (RS) and write set (WS) of the validating transaction (T_v). First we check that the validating transaction has read from committed transactions. This is done by checking data item's read timestamp (RTS) and write timestamp (WST). These values are fetched when the read/write operation to the current data item is made. Then, the algorithm iterates the set of active conflicting transactions. When access has been made to the same objects both in the validating transaction and in the active transaction, the temporal time interval of the active transaction is adjusted. Non- serializable execution is detected when the timestamp interval of an active transaction becomes empty. If the timestamp interval is empty the transaction is restarted.

5.4.1.2.1 Forward adjustment

A read-write conflict between T_v and T_a can be resolved by adjusting the timestamp interval of the active transaction forward (i.e. $T_v \rightarrow T_a$). Suppose that the validating transaction is a mobile transaction and the active transaction is a fixed transaction. In this case, there is no need for the timestamp interval of the mobile transaction to be reduced because the fixed transaction has a fair opportunity to continue execution without affecting the validating mobile transaction. If the validating transaction is a fixed host transaction which has a

conflict with a mobile transaction, the mobile transaction should obtain the advantage. This is achieved by reducing the timestamp interval of the validating fixed host transaction and selecting a new final timestamp earlier in the timestamp interval; see Section 5.4.2. Normally, the current time or the maximum value from the timestamp interval is selected, but now a different value is selected based on a predefined σ*-value*. As the σ*-value* increases, the opportunity for the active mobile transaction to commit increases at the expense of the fixed host transaction. When the σ*-value* = 2 this means that the validating fixed host transaction reduces its interval by a half, to the advantage of active mobile transactions.

Example 5.4: Let TI (T_1) = [20, 60], TS (T_1) = 60 and TI (T_2) = [10, 80). Let T_1.type = fixed and T_2.type=mobile. Assume we have a read-write conflict between transactions T_1 and T_2 and σ=2. We first make more room for the mobile transaction T_2 and then move the mobile transaction forward.

$$TS(T_1) = 20 + \left\lfloor \frac{60-20}{2} \right\rfloor = 40$$

$$TI(T_2) = [10, 80] \cap [40, \infty] = [40, 80]$$

The resulting selected value should be within the timestamp interval. This offers a greater chance for the mobile transaction to commit in its timestamp interval. If this resulting point cannot be selected, the validating transaction is restarted. This is wasted execution, but it is required to ensure the execution of the mobile transaction. This forward adjustment can be described by the procedure in Figure 5.12.

Forward Adjustment (T_a, T_v)

Begin

If T_v.type = = Fixed **Then**

If T_a.type = = Mobile **Then**

$$TS(T_V)' = \min(TI(T_V)) + \left\lfloor \frac{TS(T_V) - \min(TI(T_V))}{\sigma} \right\rfloor$$

If $TS(T_v)' > \max((TI(T_a))$ **Then**

Restart (T_v);

Else

$TI(T_a) = TI(T_a) \cap [TS(T_v)', \infty]$

Else T_a.type = =Fixed

$TI(T_a) = TI(T_a) \cap [TS(T_v), \infty]$

If $TI(T_a) = []$ **Then** Restart (T_a)

Else T_v.type = =Mobile

$TI(T_a) = TI(T_a) \cap [TS(T_v), \infty]$

If $TI(T_a) = []$ **Then** Restart (T_a);

End

Figure5. 12: Forward adjustment

Example 5.5: Forward adjustment

Let $R_i(x)$ and $W_i(x)$ denote a read and write operation, respectively, on the data item x by transaction i, and let v_i and c_i denote the validation and commit of transaction i, respectively. Consider three transactions T_1, T_2, and T_3:

T_1: $R_1(D_1)$ $W_1(D_3)$ $R_1(D_2)$ v_1

T_2: $W_2(D_2)$ $W_2(D_4)$ v_2.

T_3: ... $W_3(D_1)$ v_3

Now, suppose they execute as follows:

H =... $R_1(D_1)$ $W_3(D_1)$ v_3 $W_2(D_2)$ $W_1(D_3)$ $W_2(D_4)$ $R_1(D_2)$ v_1 v_2.

D_1	RTS	40
	WTS	20
D_2	RTS	50
	WTS	30
D_3	RTS	30
	WTS	65
D_4	RTS	40
	WTS	75

Operation		$TI(T_i)$
$R_1(D_1)$	$TI(T_1) = [0, \infty] \cap [20, \infty]$	$[20, \infty]$
$W_3(D_1)$	$TI(T_3) = [0, \infty] \cap [20, \infty] \cap [40, \infty]$	$[40, \infty]$
V_3	$TS(T_3)$= validation time=81	[40,81]
C_3	$TI(T_1) = [20, \infty] \cap [0,80]$ After backward adjustment of $TI(T_1)$	[20, 80]
$W_2(D_2)$	$TI(T_2) = [0, \infty] \cap [30, \infty] \cap [50, \infty]$	$[50, \infty]$
$W_1(D_3)$	$TI(T_1) = [20,80] \cap [30, \infty] \cap [65, \infty]$	[65,80]
$W_2(D_4)$	$TI(T_2) = [50, \infty] \cap [40, \infty] \cap [75, \infty]$	$[75, \infty]$
$R_1(D_2)$	$TI(T_1) = [65,80] \cap [30, \infty] \cap [50, \infty]$	[65,80]
V_1	$TS(T_1)$= validation time=100	

Figure5. 13: processing for example 5.5

Figure 5.13 shows how the timestamp intervals for transactions are adjusted with respect to the RST and WST of the data items accessed by these transactions. Let us illustrate the forward adjustment by taking the following scenarios:

(1) T_1 fixed and T_2 mobile (active: mobile, validating: fixed, read/write conflict).
(2) T_1 fixed and T_2 fixed (active: fixed, validating: fixed, read/write conflict).
(3) T_1 mobile and T_2 mobile (active: mobile, validating: mobile, read/write conflict).
(4) T_1 mobile and T_2 fixed (active: fixed, validating: mobile, read/write conflict).

Let the validation time of transaction T_1=100. Because 100 $\notin$ TI (T_1), we select max (TI (T_1)) to be the final commit timestamp. So TS (T_1) = 80. At the validation time of transaction T_1, we find that RS (T_1) $\cap$ WS (T_2) = {D2}. So, transaction T_2 is in the conflict active set of T_1 with a read-write conflict, which results in a forward adjustment of transaction T_2. In scenario (1), since the validation transaction is fixed and the active is mobile, we first ensure that the validation of the fixed transaction T_1 will not result in restarting the mobile transaction T_2 by checking that $TS(T_1)' < Max\ (TI\ (T_2))$.

$$TS(T_1)' = \min(TI(T_1)) + \left\lfloor \frac{TS(T_1) - \min(TI(T_1))}{\sigma} \right\rfloor$$

Assume $\sigma = 2$, then $TS(T_1)' = \min([80,\infty)) + \left\lfloor \frac{80-65}{2} \right\rfloor = 67$

Because the condition is satisfied (i.e. $67<\infty$), we first make more room for the mobile transaction T_2 and then move the transaction forward. So, the time interval of T_2 becomes TI (T_2) = $[75,\infty]$ $\cap$ $[67,\infty]$ = $[67,\infty]$ and transaction T_1 is successfully validated against the mobile transaction T_2 and commits with a final timestamp TS (T_1) =67. For the other scenarios (2), (3) and (4), the forward adjustment of TI (T_2) will be as follows: TI (T_2) = $[75,\infty]$ $\cap$ $[80,\infty]$ = $[80,\infty]$. And transaction T_1 is successfully validated and commits with a final timestamp TS (T_1) =80.

Example 5.6: Forward adjustment:

Consider three transactions T_1, T_2, and T_3: which are different from transactions in Example 4.5.

T_1: R_1 (D_1) W_1 (D_2) R_1 (D_3) v_1

T_2: R_2 (D_2) W_2 (D_4) v_2.

T_3: ... W_3 (D_1) v_3

Now, suppose that these transactions execute as follows:

H_2 =... R_1 (D_1) W_3 (D_1) v_3 R_2 (D_2) W_1 (D_2) W_2 (D_4) R_1 (D_3) v_2 v_1.

Figure 5.14 shows how the timestamp intervals for transactions have been adjusted with respect to the RST and WST of the data items accessed by these transactions.

Consider the same scenarios as in Example 5.5. Forward adjustment will work as follows:

In scenario (1), since the validation transaction is fixed and the active is mobile, we first ensure that the validation of the fixed transaction T_2 will not result in restarting the mobile transaction T_1.

$$TS(T_2)' = \min(TI(T_2)) + \left\lfloor \frac{TS(T_2) - \min(TI(T_2))}{\sigma} \right\rfloor$$

Assume $\sigma = 2$, then $TS(T_2)' = \min([65,100)) + \left\lfloor \frac{100-65}{2} \right\rfloor$=82.

Since $TS(T_2)'$ > Max (TI (T_1)), that is 82>max ([65, 80]), the validating fixed transaction T_2 will be restarted and give the mobile transaction T_1 the opportunity to continue its execution and commit. For the others scenarios (2), (3) and (4), forward adjustment of TI (T_1) will be as follows: TI (T_1) = [65, 80] $\cap$ [100, ∞] = [], since

TI (T_1) is empty, Transaction T_1 will be restarted and T_2 will commit with a final timestamp TS (T_2) =100.

D_1	RTS	40
	WTS	20
D_2	RTS	50
	WTS	30
D_3	RTS	30
	WTS	65
D_4	RTS	40
	WTS	65

Operation		$TI(T_i)$
$R_1(D_1)$	$TI(T_1) = [0, \infty] \cap [20, \infty]$	$[20, \infty]$
$W_3(D_1)$	$TI(T_3) = [0, \infty] \cap [20, \infty] \cap [40, \infty]$	$[40, \infty]$
V_3	$TS(T_3)$= validation time=81	[40,81]
C_3	$TI(T_1) = [20, \infty] \cap [0,80]$ After backward adjustment of $TI(T_1)$	[20, 80]
$R_2(D_2)$	$TI(T_2) = [0, \infty] \cap [30, \infty] \cap [50, \infty]$	$[50, \infty]$
$W_1(D_2)$	$TI(T_1) = [20,80] \cap [30, \infty] \cap [50, \infty]$	[50,80]
$W_2(D_4)$	$TI(T_2) = [50, \infty] \cap [40, \infty] \cap [65, \infty]$	$[65, \infty]$
$R_1(D_3)$	$TI(T_1) = [65,80] \cap [30, \infty] \cap [50, \infty]$	[65,80]
V_2	$TS(T_1)$= validation time=100	

Figure5. 14: processing for example 5.6

5.4.1.2.2 Backward adjustment

A write-read conflict between T_v and T_a can be resolved by adjusting the timestamp interval of the active transaction backward, (i.e. $T_a \rightarrow T_v$). If the validating is a mobile transaction and the active conflicting is a fixed transaction, backward adjustment is appropriate in the sense that it gives the fixed transaction its opportunity to continue its execution without affecting the validating mobile transaction. If the validating transaction is fixed and the active conflicting transaction is mobile, then backward adjustment is done if the active transaction is not aborted in backward adjustment. Otherwise, the validating transaction is restarted. This is wasted execution, but it is required to ensure the execution of the

mobile transaction. In backward adjustment, we cannot move the validating transaction to the future to obtain more space for the mobile transaction. We can only check if the timestamp interval of the mobile transaction would become empty. In forward ordering we can move the final timestamp backward if there is space in the timestamp interval of the validating transaction. Again we check if the timestamp interval of the mobile transaction would shut out. We have chosen to abort the validating transaction when the timestamp interval of the mobile transaction shuts out. Thus, this protocol favours the mobile transaction that uses scarce and expensive resources. Backward adjustment can be described by the procedure in Figure 5.15.

```
Backward Adjustment (Ta, Tv)
 Begin
    If (Tv.type = = Fixed) Then
        If (Ta.type = = Mobile) Then
            If (TS (Tv) – 1 < Min (TI (Ta)) Then
                    Restart (Tv);
            Else
                    TI (Ta) = TI (Ta) ∩ [0, TS (Tv)-1]
        Else Ta.type = = Fixed
                    TI (Ta) = TI (Ta) ∩ [0, TS (Tv)-1]
                    If TI (Ta) = [] Then Restart (Ta)
     Else Tv.type = =Mobile
                    TI (Ta) = TI (Ta) ∩ [0, TS (Tv)-1]
                    If TI (Ta) = [] Then Restart (Ta)
 End
```

Figure5. 15: Backward Adjustment

Example 5.7: backward adjustment:
Consider three transactions T_1, T_2, and T_3:

T_1:R_1 (D_1) W_1 (D_3) W_1 (D_2) v_1

T_2:R_2 (D_2) W_2 (D_4) ... v_2

T_3: ...W_3 (D_1) v_3

Now, suppose they execute as follows:

H =... R_1 (D_1) W_3 (D_1) v_3 R_2 (D_2) W_1 (D_3) W_2 (D_4) W_1 (D_2) v_1 ... v_2.

Figure 5.16 shows how the timestamp intervals for transactions have been adjusted with respect to the RST and WST of the data items accessed by these transactions.

We will consider two cases (a) and (b) based on the WTS (D_4) for the following scenarios:

(1) T_1 fixed and T_2 mobile (active: mobile, validating: fixed, write/read conflict)

(2) T_2 fixed and T_1 mobile (active: fixed, validating: mobile, write/read conflict)

Case (a)				Case (b)		
D_1	RTS	40		D_1	RTS	40
	WTS	20			WTS	20
D_2	RTS	50		D_2	RTS	50
	WTS	30			WTS	30
D_3	RTS	30		D_3	RTS	30
	WTS	65			WTS	65
D_4	RTS	40		D_4	RTS	40
	WTS	65			WTS	75

Operation		$TI(T_i)$
$R_1(D_1)$	$TI(T_1)=[0,\infty] \cap [20,\infty]$	$[20,\infty]$
$W_3(D_1)$	$TI(T_3)=[0,\infty] \cap [20,\infty] \cap [40,\infty]$	$[40,\infty]$
V_3	$TS(T_3)$= validation time=71	[40,81]
C_3	$TI(T_1)=[40,\infty] \cap [0,70]$ After backward adjustment of $TI(T_1)$	[40, 70]
$R_2(D_2)$	$TI(T_2)=[0,\infty] \cap [30,\infty]$	$[30,\infty]$
$W_2(D_3)$	$TI(T_1)=[40,70] \cap [30,\infty] \cap [65,\infty]$	[65,70]
$W_2(D_4)$	$TI(T_2)=[30,\infty] \cap [40,\infty] \cap [65,\infty]$	$[65,\infty]$
$W_2(D_4)$	$TI(T_2)=[30,\infty] \cap [40,\infty] \cap [75,\infty]$	$[75,\infty]$
$W_1(D_2)$	$TI(T_1)=[65,70] \cap [30,\infty] \cap [50,\infty]$	[65,70]
V_1	$TS(T_1)$= validation time=90	

Figure5. 16: processing for example 5.7

Case (a): WTS (D_4) = 65:

In scenario (1), the timestamp interval of transaction T_2 after executing W_2 (D_4) will be set to $[65,\infty]$. For transaction T_1, because the validation time does not

belong to TI (T_1) we chose max (TI (T_1)) as a final commit timestamp (which is 70). At the validation time of T_1, transaction T_2 is in the active conflict set of T_1 with write-read conflict WS (T_1) $\cap$ RS (T_2) = {D2}. So, TI(T_2) will be backward adjusted as follows: we first make sure that the validation of the fixed transaction still gives an opportunity for the mobile transaction to be validated (i.e.(T_1)-1<min(TI(T_2)); then do the adjustment TI (T_2) =[65, ∞] $\cap$ [0,69]= [65,69]. So, T_1 is successfully validated and T_2 still has the opportunity to be validated in the time interval TI (T_2) = [65, 69].

In scenario (2), the validating transaction T_1 is mobile. So, we do the backward adjustment for the fixed transaction T_2 directly as follow: TI (T_2) = [65, ∞] $\cap$ [0, 69] = [65, 69]. Then, we check if the interval is shut out. If so the fixed transaction will be restarted.

Case (b): WTS (D_4) = 75:

In scenario (1), the timestamp interval of transaction T_2 after executing W_2 (D_4) will be [75, ∞]. For transaction T_1, because the validation time=90 does not belong to the TI (T_1) we choose max (TI (T_1)) as the final commit timestamp (which is 70). At the validation time of T_1, transaction T_2 is in the active conflict set of the transaction T_1 with write-read conflict WS(T_1) $\cap$ RS(T_2)={D_2}. So, TI(T_2) will be backward adjusted as follows: we first make sure that the validation of the fixed transaction still gives an opportunity for the mobile transaction to be validated (i.e. TS(T_1)-1<min(TI(T_2)); then do the adjustment; since 70-1<75 and the validating transaction T_1 will result in restarting the mobile transaction T_2, we chose to restart T_1. As a result, T_2 still has an opportunity to complete its execution and it may successfully validate.

In scenario (2), since the validating transaction T_1 is mobile, T_1 is validated first. Then, we do the backward adjustment for the fixed transaction T_2 as follows. TI (T_2) = [75, ∞] $\cap$ [0, 69] = []. Since the timestamp interval of T_2 is shut out, the fixed transaction will be restarted. In general, if the validating and the active conflict transactions are not backward adjusted before, the active conflicting transaction will be never restarted by the validating transaction in such a case. This is true for all possible scenarios for both mobile and fixed transactions.

5.2.3.2 Final timestamp selection

In the OCC-Mix protocol, we should select the final (commit) timestamp TS (T_v) in such a way that room is left for backward adjustment. In our validation algorithm Figure 5.17, we set TS (T_v) as the validation time if it belongs to the time interval of T_v or the maximum value from the time interval otherwise. To justify our choice consider the following example:

Example 5.8: Let RTS(x) and WTS(x) be initialized to 100, and let transactions T_1, T_2 and history H, where T_1.type= T_2.type = mobile, be as follows:

T_1: $R_1(x)$ $W_1(x)$ v_1 c_1.

T_2: $R_2(x)$ v_2 c_2

$H = R_1(x)$ $R_2(x)$ $W_1(x)$ v_1.

Consider the two cases (a) and (b):

Case a: TS (T_v) = max (TI (T_v))

As before, transactions T_1 and T_2 are forward adjusted to $[100,\infty)$. Transaction T_1 starts the validation at time 1000, and the final (commit) timestamp is selected to be TS (T_1) = validation_time =1000. Because we have one write-read conflict between the validating transaction T_1 and the active transaction T_2, the timestamp interval of the active transaction must be adjusted: $TI(T_2)=[100,\infty) \cap [0,999]$ =[100,999]. Thus, the timestamp interval is not empty, and we have avoided unnecessary restart. Both transactions commit successfully. History *H* is acyclic, that is, serializable. Therefore the selection TS (T_v) = max (TI (T_v)) avoids the unnecessary restart problem.

Case b: TS (T_v) = min (TI (T_v))

Transaction T_1 executes $R_1(x)$ which causes the timestamp interval of the transaction to be forward adjusted to $[100,\infty)$. Then, T_2 executes a read operation on the same object, which causes the timestamp interval of the transaction to be forward adjusted similarly, e.g. to $[100,\infty)$. T_1 then executes $W_1(x)$ which causes the timestamp interval of the transaction to be forward adjusted to $[100,\infty)$. T_1 starts the validation, and the final (commit) timestamp is selected to be TS (T_1) =min ($[100,\infty)$) =100. Because we have one write-read conflict between the validating transaction T_1 and the active transaction T_2, the timestamp interval of

the active transaction must be adjusted. Thus $TI(T_2)=[100, \infty) \cap [0,99] =[]$. The timestamp interval is shut out, and must be restarted. However this restart is unnecessary, because history H is acyclic and so serializable. Taking the minimum as the commit timestamp ($TS(T_1)$) was not a good choice here.

```
Select final time stamp (Tv)
  Begin
    If  (validation_time) ∈ TI (Tv) Then
          TS (Tv) = validation time;
    Else
          TS (Tv) = max (TI (Tv));
  End
```

Figure5. 17: Select commit timestamp

We have also used a deferred dynamic adjustment of serialization order. In the deferred dynamic adjustment of serialization order all adjustments of timestamp intervals are done to temporal variables (i.e. the timestamp intervals of all conflicting active transactions are adjusted after the validating transaction is guaranteed to commit). If a validating transaction is aborted no adjustments are done. Adjustment of the conflicting transaction would be unnecessary since no conflict is present in the history after abortion of the validating transaction. Unnecessary adjustments may later cause more restarts.

Finally, in Figure 5.18 current read timestamps and write timestamps of accessed data items are updated and changes to the database are committed.

```
Update Data Item Timestamps (RS (Tv) ,WS (Tv))
  Begin
     For all Di ∈ (RS (Tv) ∪ WS (Tv))
        Begin
          If (Di ∈ RS (Tv)) Then
          RTS (Di) = max (RTS (Di)), TS (Tv));
          If (Di ∈ WS (Tv)) Then
          WTS (Di) = max (WTS (Di)), TS (Tv));
        End
          Commits (Tv) to database;
  End
```

Figure5. 18: Update Data Item Timestamps

A backward and forward with deferred adjustment algorithm previously described, creates an order between conflicting transaction timestamp intervals. A final (commit) timestamp is selected from the remaining timestamp interval of the validating transaction. Therefore, the final timestamps of the transactions create a partial order between transactions.

CHAPTER 6

PERFORMANCE EVALUATION

We have constructed and conducted a series of simulation experiments to evaluate and compare the performance of our concurrency control approaches on mixed transactions environments. In the following sections, we discuss the simulation model, the workload being used, performance metrics, and finally present our research findings in these experiments.

6.1 Simulation model

A simulator has been written to imitate our architecture indicated by Figure 4.2. The simulator models both fixed and mobile parts. Stationary units are classified as either fixed hosts or base stations. Fixed hosts are database servers connected to the existing wired network. Such a server is accessible by two types of users - users who use the wired reliable network and the users who are not capable of connecting directly to these information servers. A base station is equipped with a wireless interface and works as a coordinator and communication interface between the mobile units and the stationary unit. Based on this, there are two types of transactions, mobile transactions submitted by the mobile clients, and fixed transactions submitted by wired client. Both are processed at the database server. Mobile and fixed host transactions consist of both read and write operations. Each data object has an equal chance of being accessed by any transaction's operation. To simulate this, two generators are implemented. The fixed host generator is responsible for the generation of the fixed host transactions. The second generator is for mobile transactions with specific attributes that imitate both the mobile device and the wireless environment discussed before.

During execution time, a transaction may need user interaction to input data. Many studies have pointed out that the cost of communication setup is very expensive [91]. To avoid re-establishing communication each time the transaction needs the user's interaction, we assume that the communication is kept during the execution of the mobile transaction. Transactions generated, both fixed and mobile, are lined up in the CPU queue according to the first-in, first-out scheduling discipline. When the CPU is available, the transaction at the front of the CPU queue is submitted for processing. To read a data object, transactions need to line up in the disk queue for data access. Transactions will repeat these steps until all operations are processed. In this system, different data structures have been used for different approaches. For example, a data object table and a transaction table are maintained. The data object table keeps a read timestamp and a write timestamp for each data object in the database and the transaction table maintains the read set, the write set and the timestamp interval of each transaction.

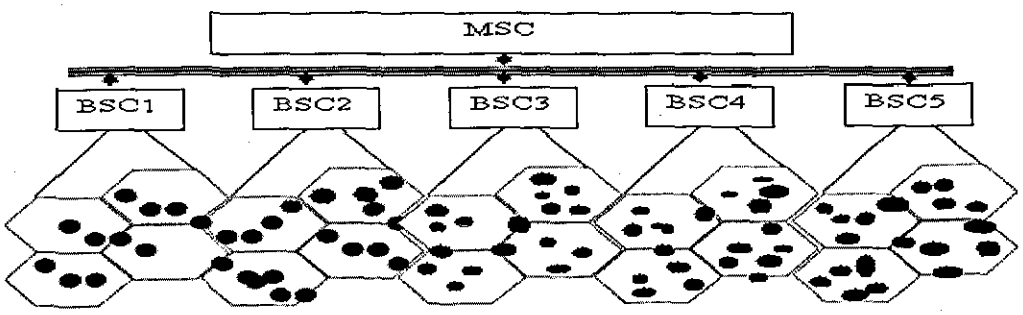


Figure6. 1: Wireless part of mixed transaction environment

When the mobile unit is within the cell of a base station, the base station will provide a communication channel available in this cell. Due to movement of a mobile user, when a mobile user enters a new cell, the new base station should provide an idle channel to the mobile unit to sustain its communication. This process is called a `handoff'. If there is no idle channel in the new cell to provide for the user, disconnection will occur. The disconnection of an active mobile transaction wastes system resources, since the transaction should be rolled back and restarted later.

In general, the wireless part consists of a number of cells, each of them controlled by a mobile support station (MSS) and a group of MSSs is governed by

one base station controller (BSC). A group of BSCs is governed by one mobile switching centre (MSC) which has a connection to the fixed host server as in Figure 6.1. The mobility of each mobile host is modelled by estimating the number of visited base stations during transaction execution. The visited base station for each mobile host is randomly selected when the handoff occur. This will affect the number of users in each cell during the simulation, and this affects the available bandwidth experienced by the mobile host as a consequence. These factors (i.e. number of users in a cell and the handoff process) are simulated through changing the time between arrivals of mobile transaction operations. The power consumption is measured by monitoring the three basic energy-metric factors: (i) *transmission* power required to send a message (i.e. operation), (ii) *reception* power required receiving or listening to a message, and (iii) *idle* power required to stay in at the active state (awake) during transaction execution. Therefore, the power consumed by any mobile host during each execution is given by the summations of three terms as follows:

$$\text{Power consumption } (MH_i) = \sum NOpS * \beta + \sum NDR * \alpha + \sum \omega * Idel\_time$$

where *NOpS* is the number of operations sent by Ti, *NDR* is the number of data received by Ti, and β and α are fixed costs associated with each operation and data item at the time of sending of the operation and receiving the data item, respectively; ω is the *idle* power required to stay in at the active state (awake) during transaction execution.

6.2 Parameter setting

Table 6.1 gives the parameters that control system resources. The parameters, *CPUTime* and *DiskTime* capture the CPU and disk processing times per data item. Table 6.1 also summarizes the key parameters that characterize system workload and transactions. The numbers of data objects accessed by a transaction are determined randomly from the database.

Table6. 1: Summary of workload used for baseline experiments

System parameters	
Reported value per test session	means of 10 times replication of each
Database size	500 data items
CPUTime	2 time units /operation
DiskTime	5 time units/data item
CPU queuing discipline	first-in, first-out
Send communication cost	15 time units/operation
Receive communication cost	5 time units/ control message
Transaction parameters	
Multiprogramming level	10-100 concurrent transactions of both
Mobile transaction Percentages	20%, 50%, 80%
Fixed host transaction length	3 - 15
TBA of fixed transaction operations	2 – 5 uniformly distributed
Probability of disconnection	0.1, 0.2, 0.3
Mobility values	1, 2, 3, 4, and 5 BS/transaction
Power of the mobile host	200 – 600 ms
Mobile transaction length	3 – 15 operations
Fixed transaction *WriteProb*	0.5
Mobile transaction *WriteProb*	0.5

6.3 Performance metrics

In our mixed environment, different performance measures are used to demonstrate the effectiveness of the protocols. For example, the restart ratio which measures the amount of restarts experienced by a mobile and fixed transaction before it can commit. The power consumption ratio (PCR) for mobile transactions indicates the amount of resource spent resulting from the blocking and restarting overheads of transactions. Reducing these overheads not only saves resources, but also helps to soothe resource and data contention. Mobile transaction rollback frequency and the fixed transaction rollback frequency help to analyze the source of transaction restarts. The frequencies can reflect the proportion of data conflict between mobile and fixed transactions and that among mobile transactions as the utilization of mobile transactions varies. The last performance measure specifically for the OCC-Mix approach is the

average adjustment made per transaction. This metric can help to understand the effectiveness of the OCC-Mix approach at different mobile transaction percentages. The formulas of the main performance measures are listed below.

$$\text{Power consumption ratio (PCR)} = \frac{P_{initial} - P_{final}}{P_{initial}}$$

$$\text{Restart ratio} = \frac{N_{restart}}{N_{committed}}$$

$$\text{Fixed transaction rollback frequency} = \frac{N_{fixed,restart}}{N_{committed}}$$

$$\text{Mobile transaction rollback frequency} = \frac{N_{mobile,restart}}{N_{committed}}$$

$$\text{Average adjustment} = \frac{N_a}{N_{commited} + N_{restart}}$$

Where

$N_{restart}$: number of restarted transaction of both types.

$N_{committed}$: number of committed transactions of both types.

$N_{mobile, restart}$: number of committed mobile transactions which caused other mobile transactions to be restarted.

$N_{fixed, restart}$: number of committed mobile transactions which caused other fixed transactions to be restarted.

N_a: total number of adjustments made.

6.4 Experiments and results

6.4.1 Experiment 1: Impact of mobility

6.4.1.1 Power consumption ratio

Figure 6.2 gives the power consumption rate as a function of the mean mobility value when 20% of transactions present in the system are mobile transactions. Since the number of base stations crossed by a mobile transaction increases as

mobility increases, the delay between mobile transaction operations increases and the average number of active transactions increase. The Lock-Mix approach is affected by the blocking overhead of mobile transactions caused by other fixed or mobile transactions which are switch into their blocking stage even if this mobile transaction has to wait for a few lock requests. The OCC protocol has the worst performance for all mobile transaction percentages. This is due to the large number of restarted mobile transactions by the fixed host transaction. Despite that, the performance of the OCC protocol becomes close to the Lock-Mix approach as the percentage of mobile transactions increases. This is not due to an improvement of the OCC protocol, but, rather, a degrading of the performance of the Lock-Mix approach. For the OCC-Mix approach, it works well by eliminating blocking overhead. The PCR reduction problem under this approach still exists because of restarting of mobile transactions. The improvement made by OCC-Mix over Lock-Mix in reducing the PCR is degraded when the percentage of mobile transactions increases as a consequence of increasing conflict between the mobile transactions themselves. So, the best improvement for the OCC-Mix can be gained when the fixed and mobile percentages are equal in Figure 6.3. In Figure 6.4, the Lock-Mix approach become closer to the OCC-Mix approach when the majority of transactions present in the system are mobile. As part of the overall improvement, the PCR is increased for all approaches with increasing the percentage of mobile transactions.

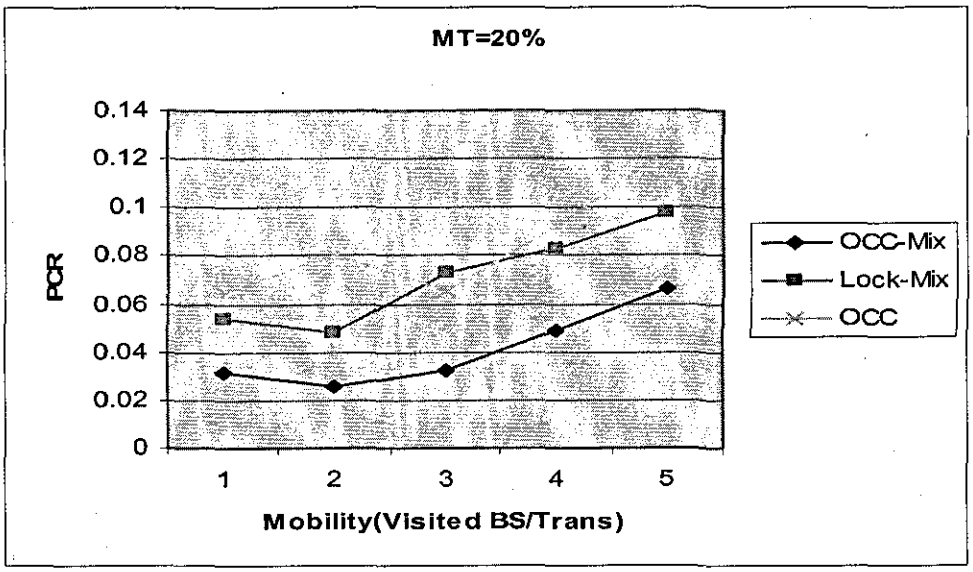


Figure6. 2: Power consumption ratio (MT = 20%)

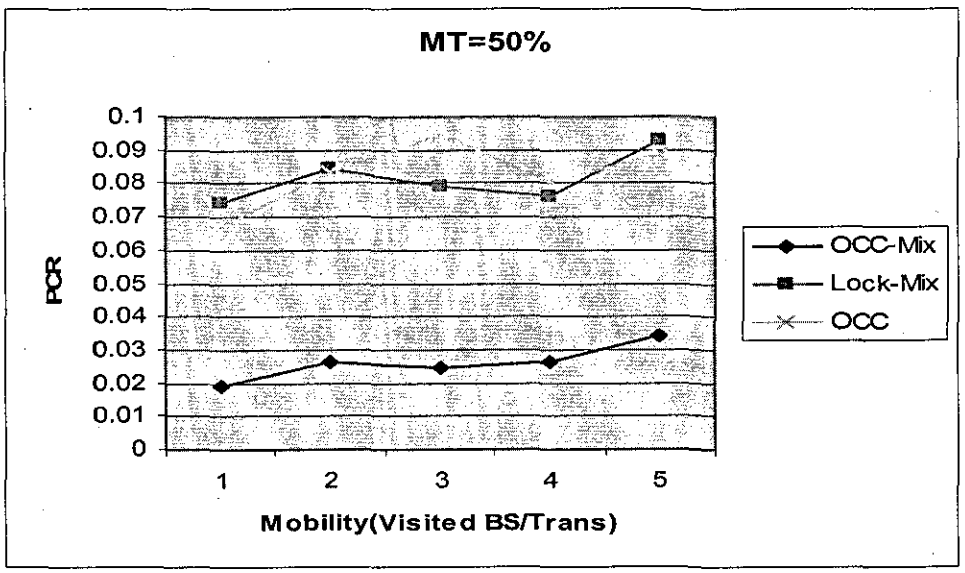


Figure6. 3: Power consumption ratio (MT = 50%)

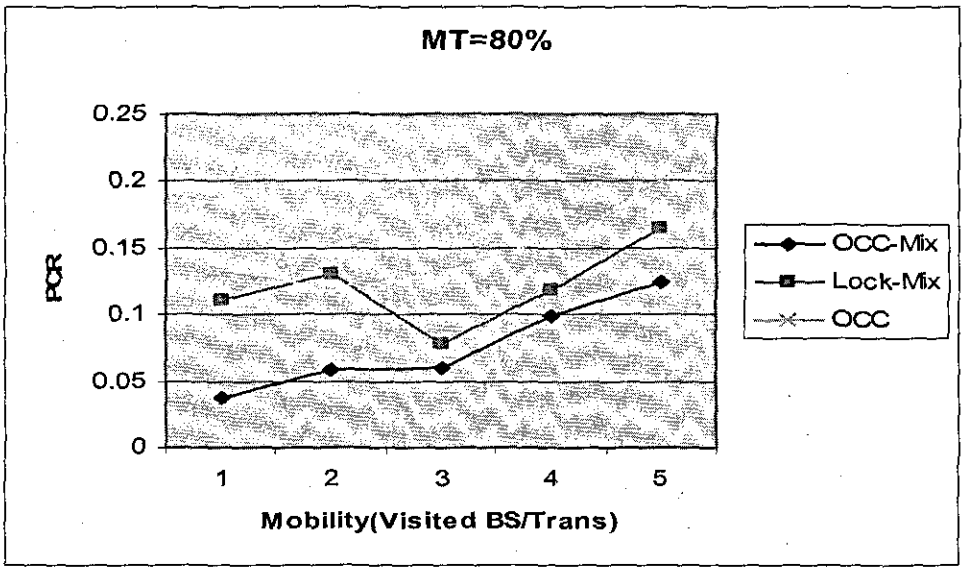


Figure6. 4: Power consumption ratio (MT = 80%)

6.4.1.2 Disconnection

Figures 6.5, 6.6, and 6.7 shows the disconnection of mobile hosts for all protocols, when the mobile transaction percentages are 20%, 50% and 80% respectively, as a function of client mobility. To examine the impact of mobility for a mobile user, we vary the amount of mobility, which represents the expected residence time that user stay in a cell or the (average)expected number of cells that mobile users may cross per time unit. In a fast mobility environment, a transaction is less likely to complete in one cell, so the frequency of handoffs may increase. This also increases the disconnection of mobile transactions. For all protocols, disconnection increases as mobility increases. But, different disconnection ratios can be observed for different protocols.

In the Lock-Mix approach, when the percentage of mobile transactions increases as in Figures 6.6 and 6.7, the effect of blocking increases. This is due to an increasing number of active transactions, both mobile and fixed, which enter their blocking phase. This will lengthen the time for mobile transactions to finish their execution and, as a result of that, they will suffer from disconnection more than other approaches. For the OCC-Mix approach, the performance decreases as the percentages of mobile transactions increases. This is due to increasing probability for mobile transactions to be in the conflict set of a validating mobile transaction. So, for a mobile transaction, there is an increase in the probability of being

disconnected as it may experience a cell with no available channels to offer. For the OCC protocol, the difference in its performance doesn't change significantly an increase of mobile transaction percentages. So, the differences between these protocols are not significant in Figure 6.7, when the percentage of mobile transaction is 80%.

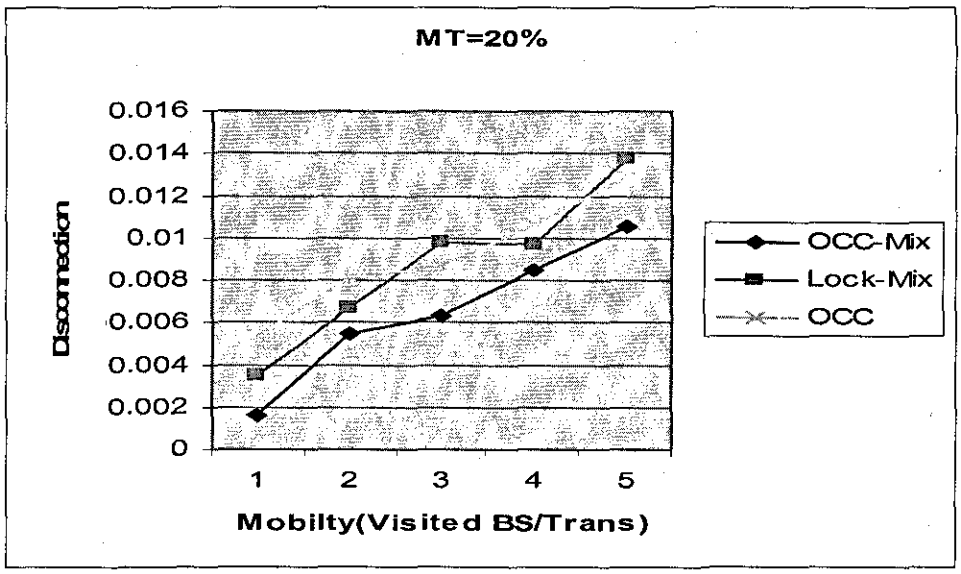


Figure6. 5: Mobile transaction aborts due to disconnection (MT = 20%)

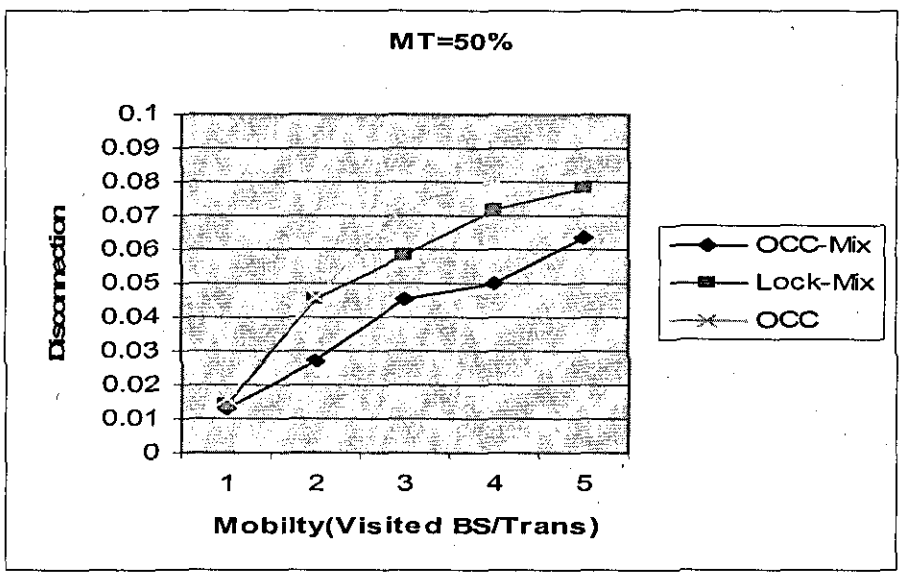


Figure6. 6: Mobile transaction aborts due to disconnection (MT = 50%)

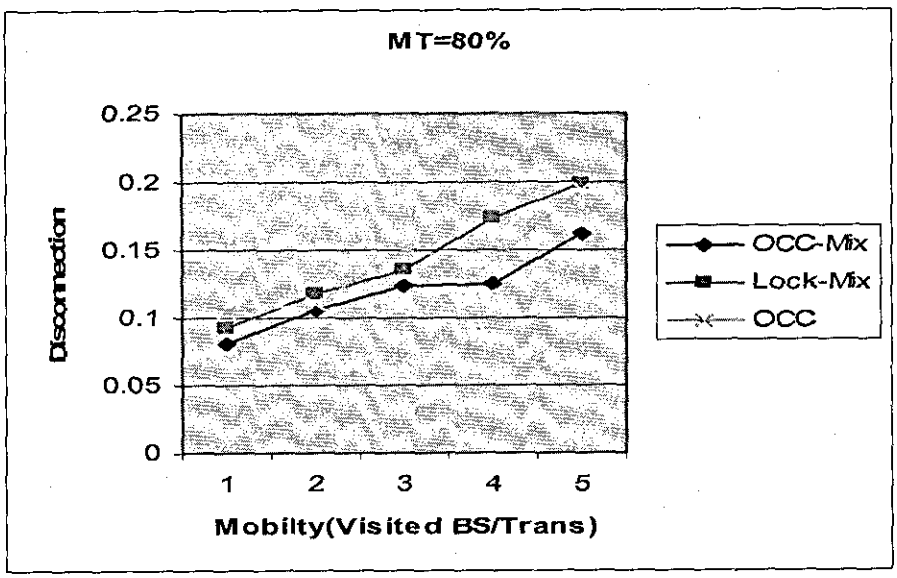


Figure6. 7: Mobile transaction aborts due to disconnection (MT = 80%)

6.4.1.3 Throughput

Figure 6.8 give the throughput of mobile transactions. When the percentage of mobile transactions is 20%, there is no significant degradation in throughput. The reason for this is that when the mobility of users increases, the probability of disconnection increases, so that the degree of data contention will decrease. As a consequence, the degradation of transaction throughput is not significant. Figure 6.9 shows that OCC-Mix has the best throughput when there are an equal number of both mobile and fixed transactions. The Lock-Mix approach has a better transaction throughput than the other two protocols when the majority of transactions present in the system are mobile; see Figure 6.10. In Figure 6.10 when the mobile transactions dominate, the Lock-Mix approach still outperforms other protocols. This means that it is better to block the mobile transaction when it has few remaining operations, like in the Lock-Mix approach, instead of restarting the mobile transaction from scratch.

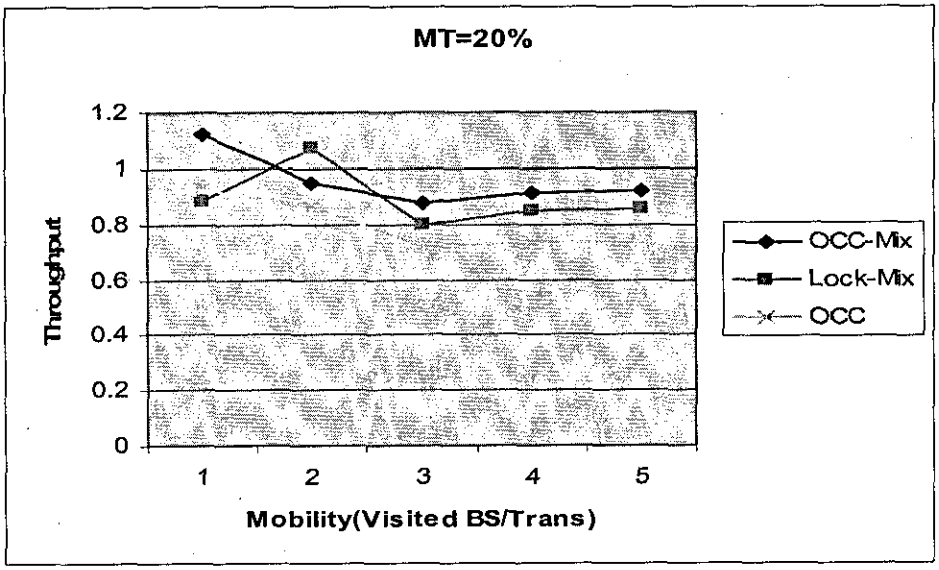


Figure6. 8: Throughput of mobile transactions (MT = 20 %)

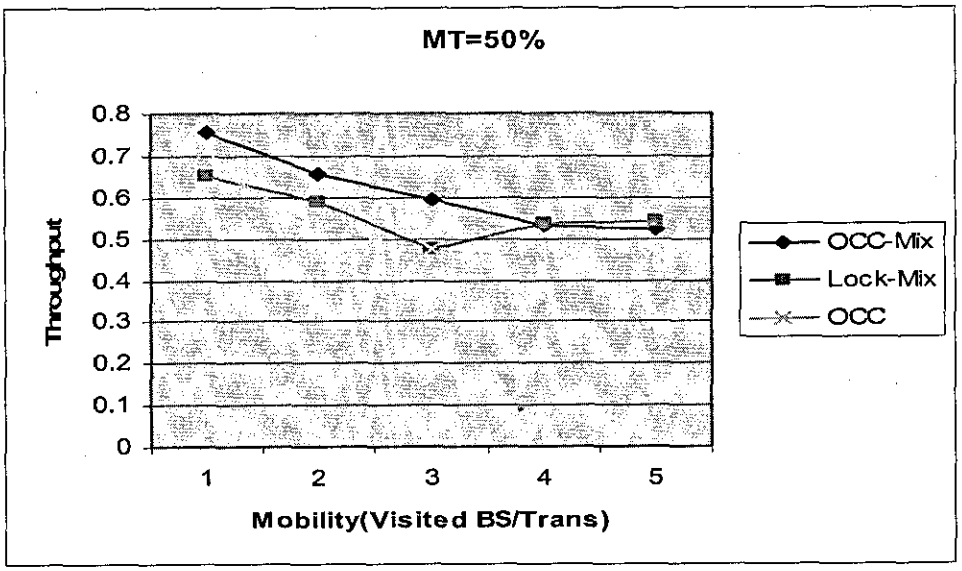


Figure6. 9: Throughput of mobile transactions (MT = 50 %)

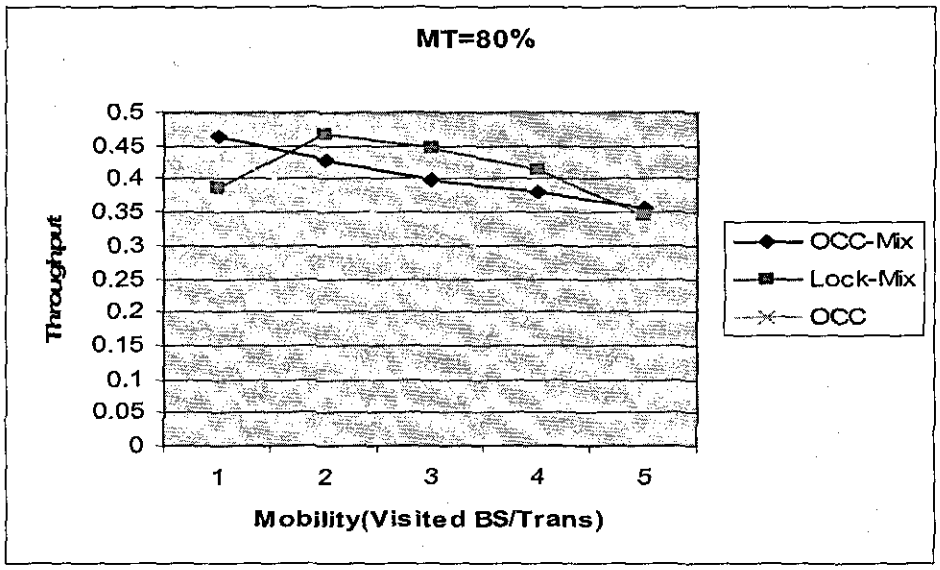


Figure6. 10: Throughput of mobile transactions (MT = 80 %)

6.4.1.4 Restart ratio

Figure 6.11 and 6.12 gives the restart ratio when 20% of transactions present in the system are mobile, for mobile and fixed transactions, respectively. As the mobility increases, the restart ratio increases slightly. In the OCC-Mix approach, most of the data conflicts are among mobile and fixed transactions. The restart ratios under the OCC-Mix and Lock-Mix approaches are similar when mobility is low. As mobility increases, the difference between OCC-Mix and Lock-Mix becomes notable. Figure 6.12 shows the opposite behaviour of these approaches for fixed transactions. From these figures we can see the Lock-Mix improvement of decreasing mobile transaction restarts while sustaining a comparable result with other protocols in term of fixed transaction restart. Figure 6.13 and 6.14 gives the restart ratio when the utilization of mobile transactions is 50 percent. Note that the restart ratio become notable at low mobility value. This is due to the presence of mobile transactions. The restart ratio of both mobile and fixed transactions increases as mobility increases, until it is reaches a peak for the Lock-Mix approach when the mobility is 4. That is, there is an increasing delay between mobile transaction operations as the number of mobile transactions increases. When the mobility is greater than 3, the number of simultaneously active transactions increases in the system and this increases the blocking effect of the Lock-Mix approach. Even so, the Lock-Mix approach performs better than other

protocols because of its flexible blocking nature of mobile transaction. When the mobility is greater than 4, restart ratio decreases under all protocols. This is due to an increasing number of disconnected mobile clients, which decreases data contention in the system and this leads the Lock-Mix approach to be slightly better than OCC-Mix when the mobility is greater than 4.

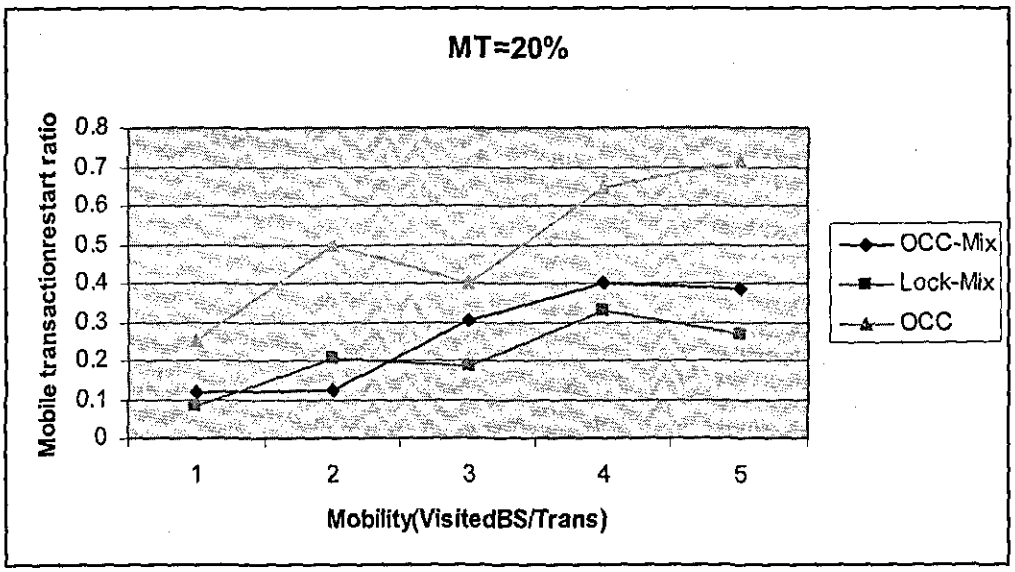


Figure6. 11: Restart ratio for mobile transactions (MT = 20 %)

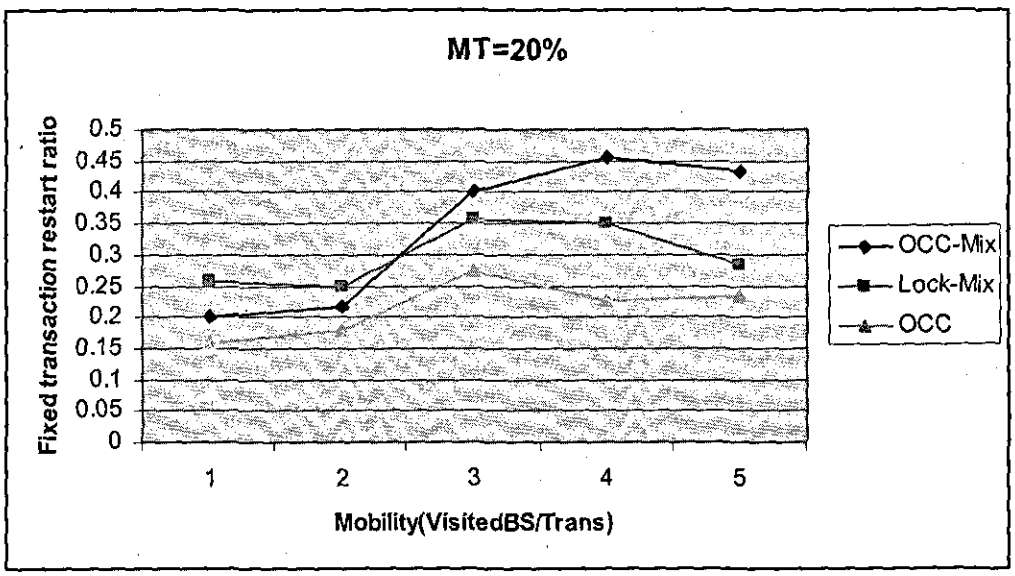


Figure6. 12: Restart ratio for fixed transactions (MT = 20 %)

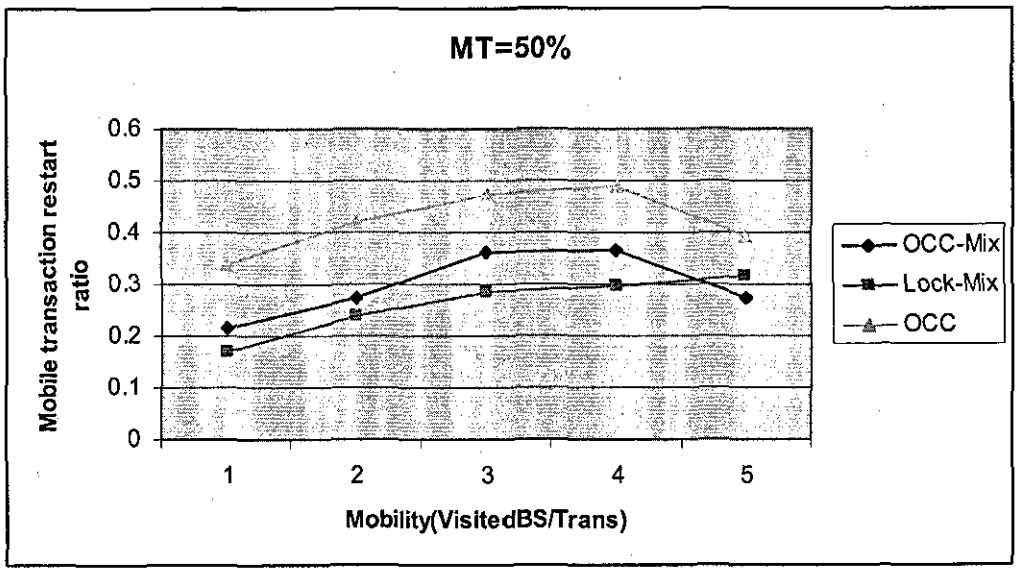


Figure6. 13: Restart ratio for mobile transactions (MT = 50 %)

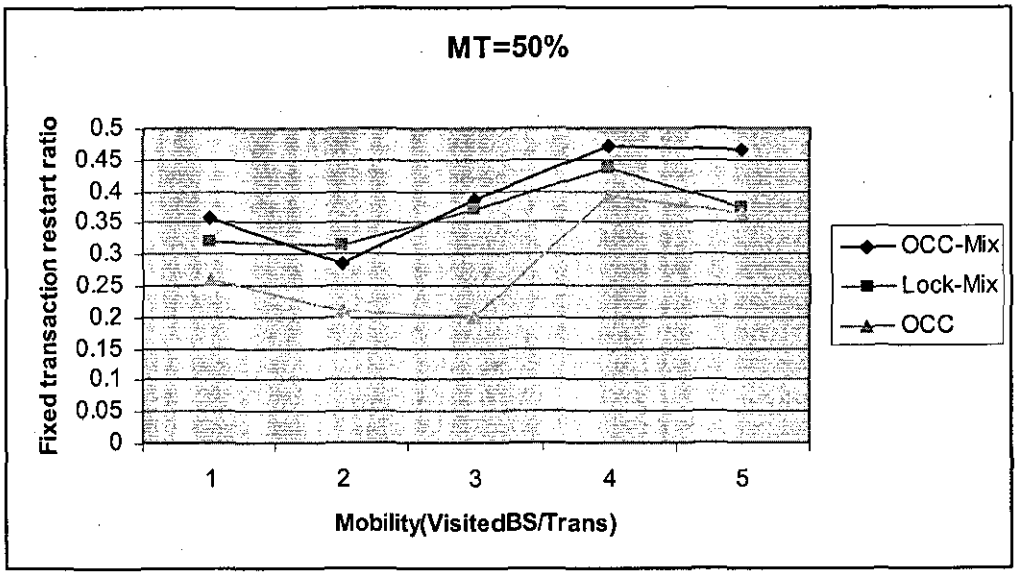


Figure6. 14: Restart ratio for fixed transactions (MT = 50 %)

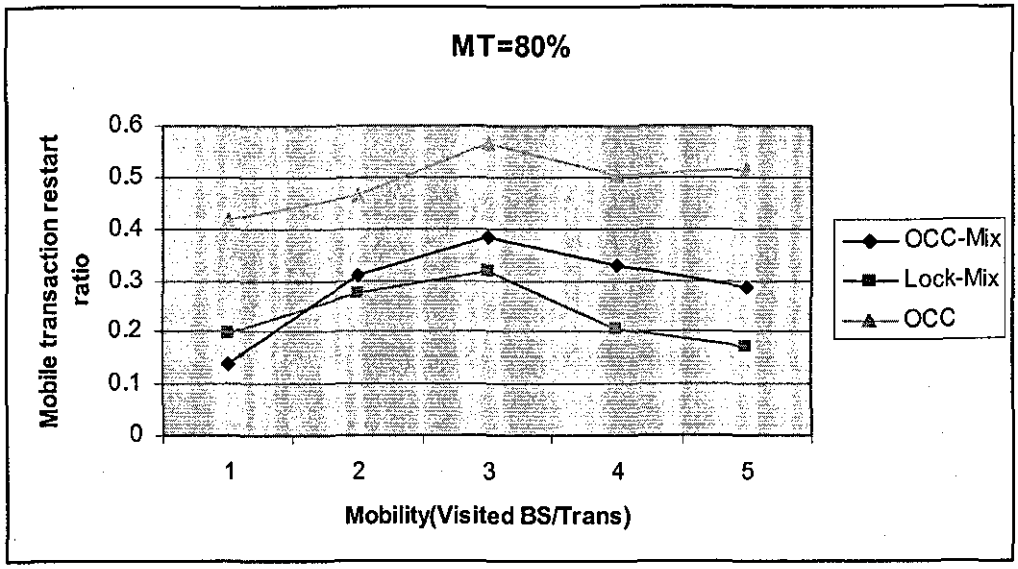


Figure6. 15: Restart ratio for mobile transactions (MT = 80 %)

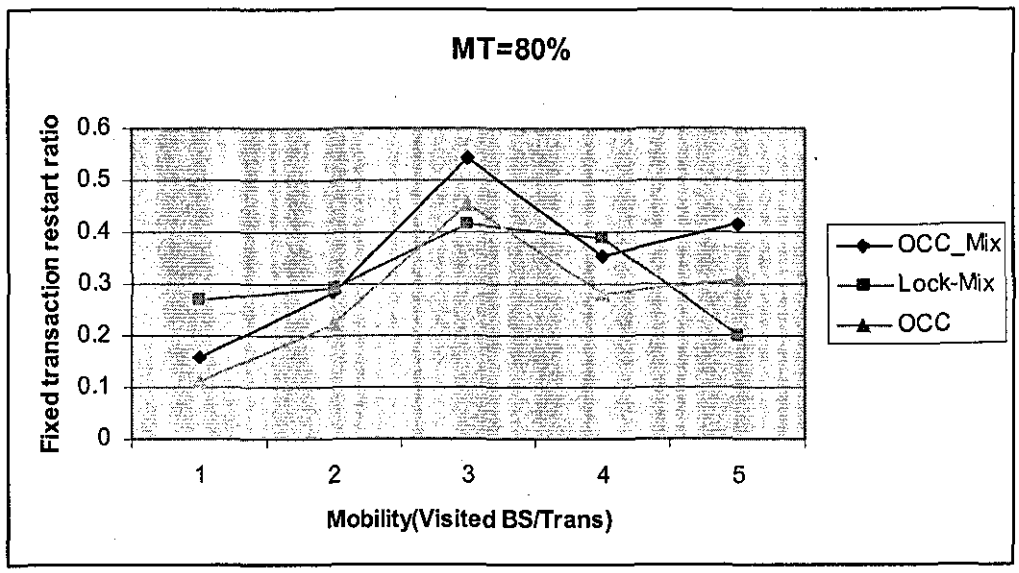


Figure6. 16: Restart ratio for fixed transactions (MT = 80 %)

Figure 6.15 gives the restart ratio when the system utilization of mobile transactions is 80 percent. In this experiment, due to the domination of mobile transactions, the improvement made by the OCC-Mix and Lock-Mix approaches is less than in Figure 6.13 where the utilization of mobile and fixed transactions are equally likely in the system. This domination of mobile transactions leads to the situation where most of the data conflicts are among mobile transaction

themselves. The restart ratio of the OCC-Mix approach reaches a peak when the mobility is around 3. For the OCC and Lock-Mix approaches they reach peaks when the mobility is also 3. As mobility increases, the disconnection of mobile hosts increases and the restart ratio decreases as a side effect of this situation. This can also be deduced Figure 6.14 and Figure 6.16 for the restart rate of fixed host transactions, where OCC improves in this experiment as the possibility for restarted mobile transactions coming from other committed mobile transactions increases. In particular, such a reasonable reduction of the mobile restart ratio saves much wireless resource from processing unnecessarily restarted mobile transactions, so that other mobile transactions have more opportunity to complete their execution while moving.

6.4.2 Experiments 2: Impact of transaction length

6.4.2.1 Restart ratio

In Figure 6.17, we vary the transaction length from 3 operations to 15 operations. The restart rate of all protocols increases when the length of the mobile transactions increases. The length is defined as the number of operations in a transaction. In order to access more data objects, a mobile transaction may have to span over a larger number of cells. As a result, it is more likely for it to be affected by the congestion at the visited base stations. In addition, the prolonged execution time also increases the chances of having data conflicts with mobile and fixed transactions because more transactions will be executed concurrently with the mobile transaction. Because a mobile transaction may suffer from high data conflict under OCC and OCC-Mix, or have to wait more if the Lock-Mix approach is being applied, their probability of being restarted increases and so does their PCR. On the other hand, OCC-Mix out performs other protocols especially when transactions are longer than 9 operations. Figures 6.17, 5.18 and 5.19 show the restart ratio of all protocols when the mobile transaction percentages are 20%, 50% and 80% respectively.

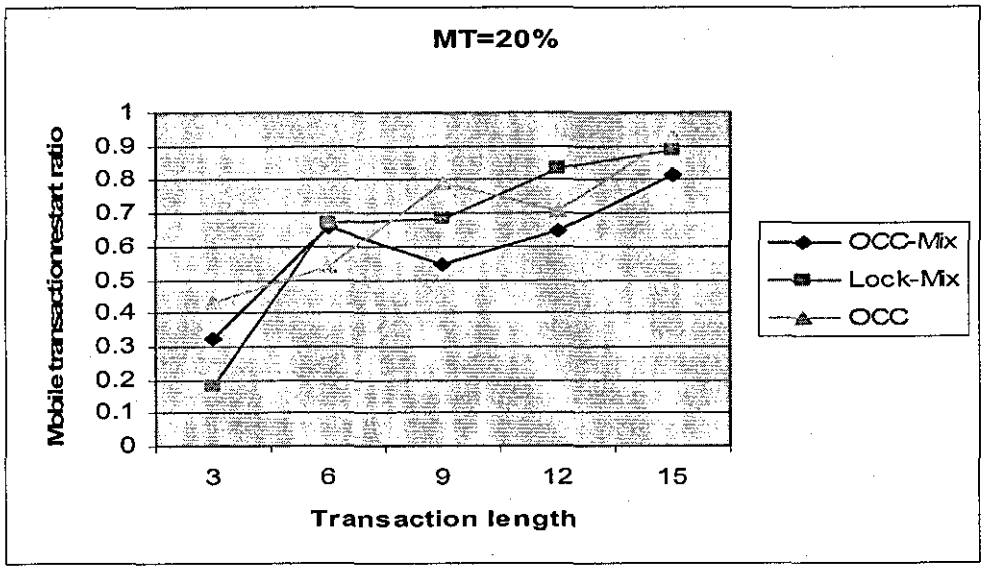


Figure6. 17: Restart ratio for mobile transactions (MT = 20 %), TL

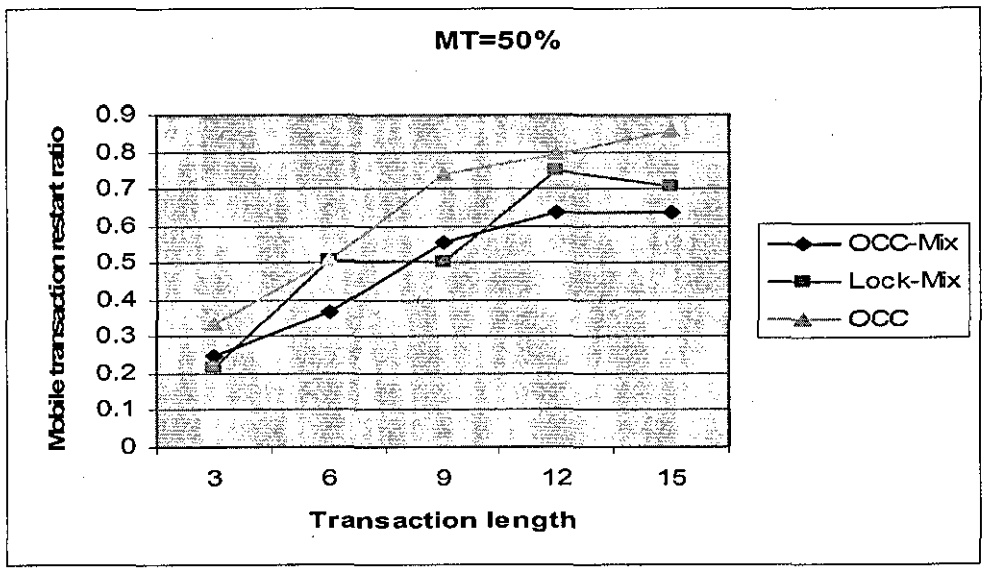


Figure6. 18: Restart ratio for mobile transactions (MT = 50 %), TL

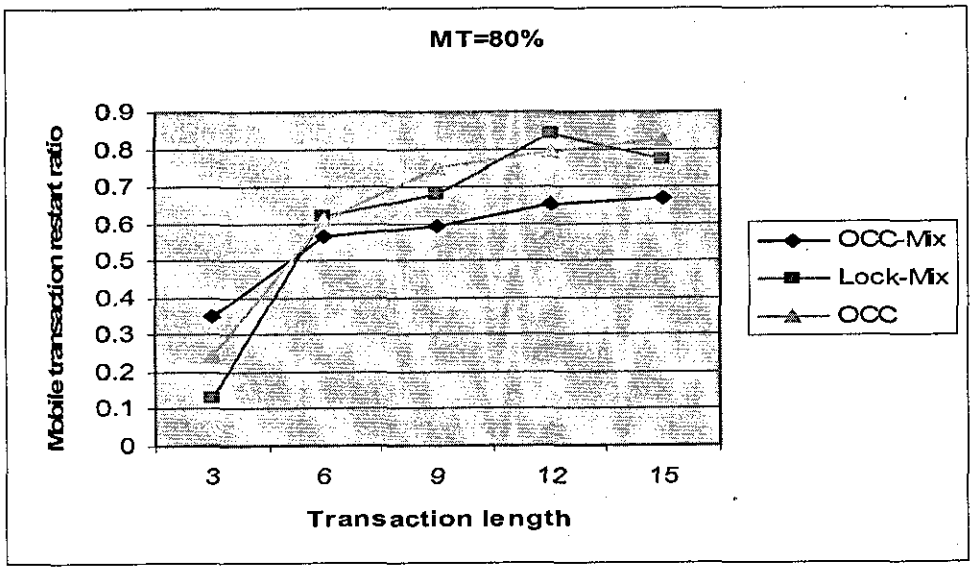


Figure6. 19: Restart ratio for mobile transactions (MT = 80 %), TL

6.4.2.2 Disconnection

Figure 6.20 shows mobile transaction aborts due to disconnection of mobile hosts for these protocols for different transaction lengths. We can see that the disconnection grows when transaction length increases. This is because a transaction is less likely to complete in one cell, so the frequency of handoffs increases and disconnection increases. This figure resembles the effect of mobility on aborts due to disconnection in Figure 6.7. Further to that, the increment of transaction length increases the probability of data conflicts between the concurrent transactions being executed. So, the abort rate will increase as the percentage of mobile transaction increases, as we see in Figure 6.21 and Figure 6.22, when the mobile transaction percentages are 50% and 80% respectively.

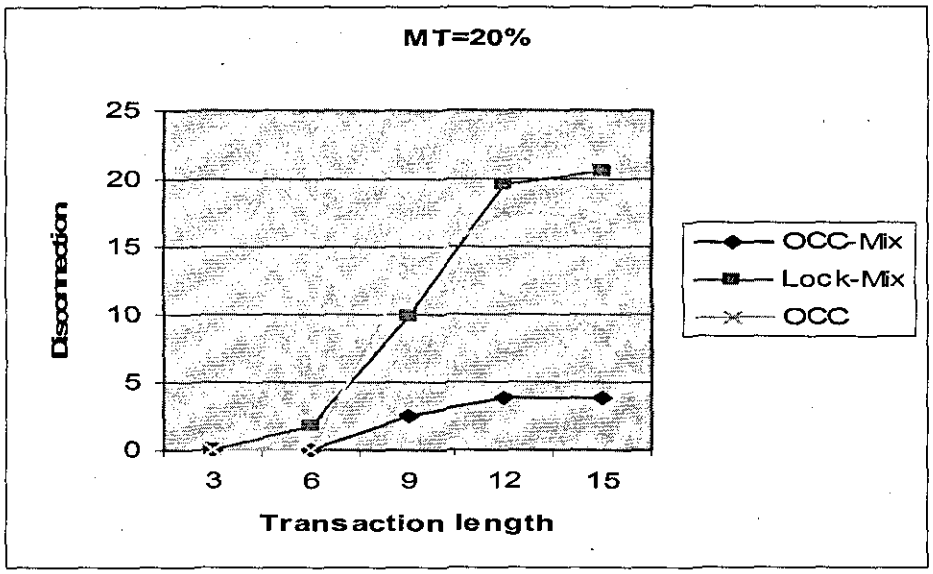


Figure6. 20: Mobile transaction aborts due to disconnection (MT = 20%)

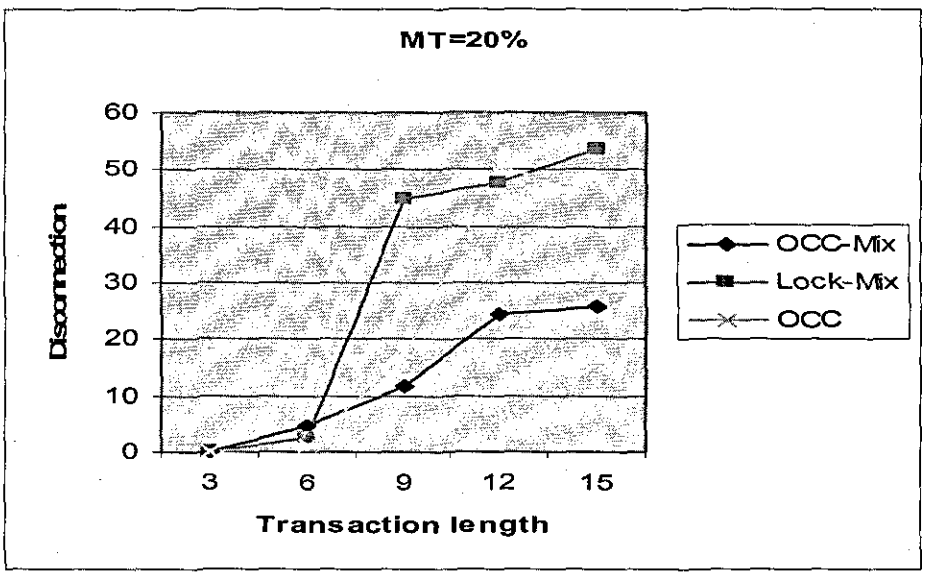


Figure6. 21: Mobile transaction aborts due to disconnection (MT = 50%)

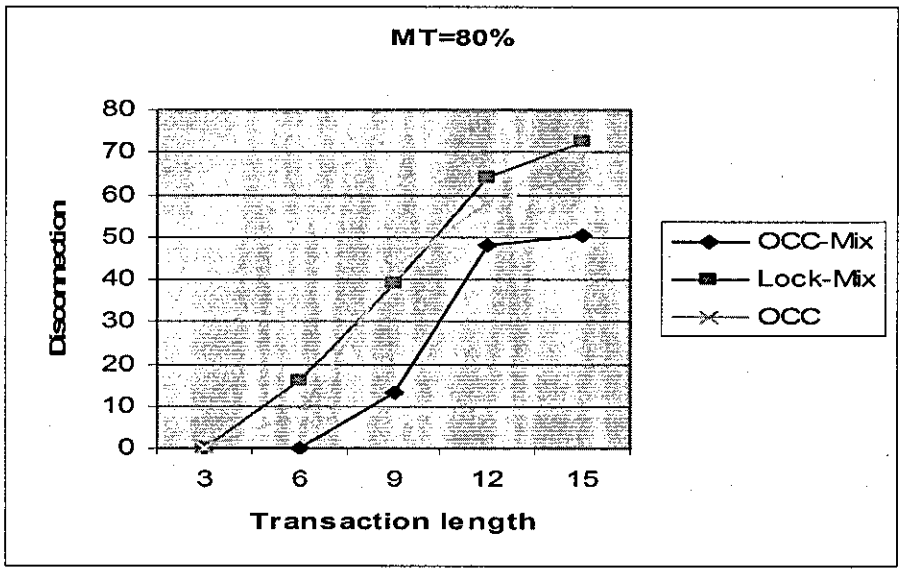


Figure6. 22: Mobile transaction aborts due to disconnection (MT = 80%)

6.4.3 Experiments 3: Impact of data contention

6.4.3.1 Restart ratio

Different degrees of data contention may affect the system performance when these protocols are applied. In these experiments, we simulate transaction executions for different proportions of read and write operations in a mobile transaction. Figure 6.23 shows the restart rate for mobile transactions as the write probability varies when 20% transactions present in the system are mobile. It can be seen that there is no difference between the performances of the protocols at both ends of the write probability. When all operations are read, there is no data conflict and no adjustment is required. It makes no difference which OCC protocol is employed. On the other hand, when all operations are writing, the performance of the OCC-Mix approach is the same as that of the other two protocols because it is impossible to adjust the serialization order between the conflicting transactions since all data conflicts are not adjustable. The OCC-Mix approach resembles the OCC protocol when all operations are either read or write. Therefore, the OCC-Mix approach functions more effectively when transactions have a mix of both read and write operations. For the Lock-Mix approach, the increase of mobile transaction restarts come as a result of restarting those transactions at the non-blocking phase of their executions. Figures 6.24 and 6.25

give the restart rate when the mobile transactions percentages are 50% percent and 80%, respectively. Since there are both read and write operations in mobile transactions, when there is write-read conflict between a mobile transaction and a fixed transaction, the fixed transaction restart can be avoided by backward adjusting the fixed transaction. On the other hand, when all operations in fixed transactions are writes, the situation becomes similar to that in Figure 6.23. All data conflicts are serious and no adjustment can be made. For the Lock-Mix protocol, when all transactions are read only transactions, it has the similar performance to other protocols. But, when the restart ratio increases highly under these protocols and when the write probability of transaction operations increases, there is a significant difference in the performance compared to other protocols as we can see from all the figures.

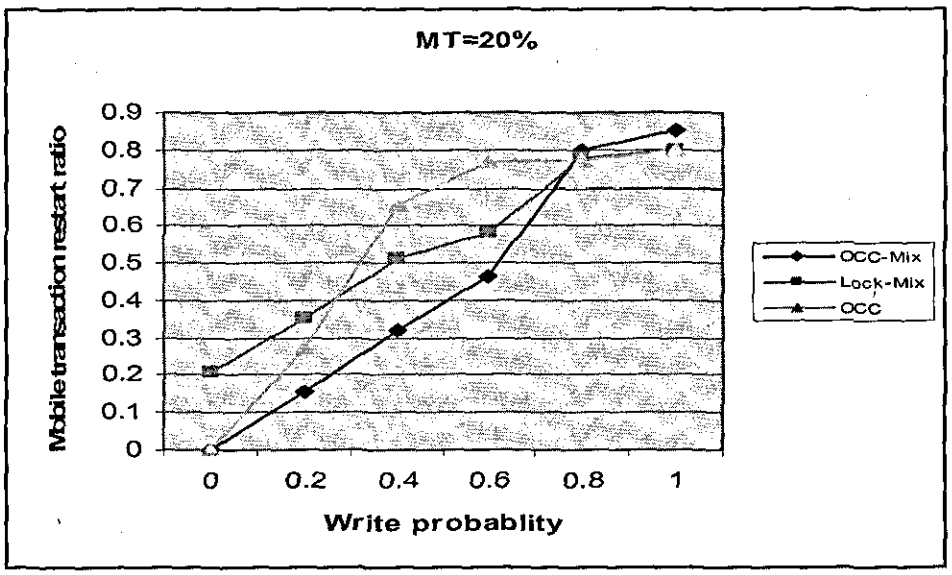


Figure6. 23: Restart ratio for mobile transactions (MT = 20 %)

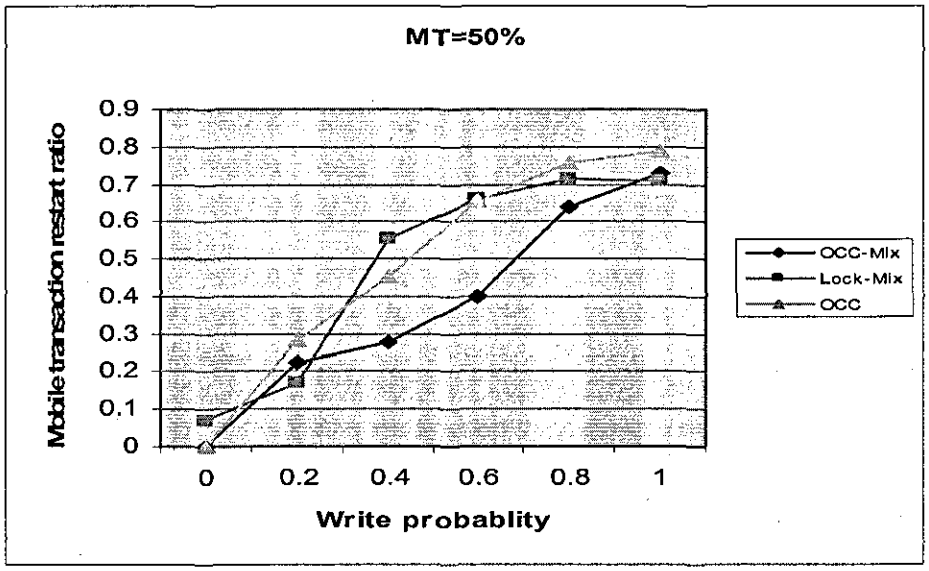


Figure6. 24: Restart ratio for mobile transactions (MT = 50 %)

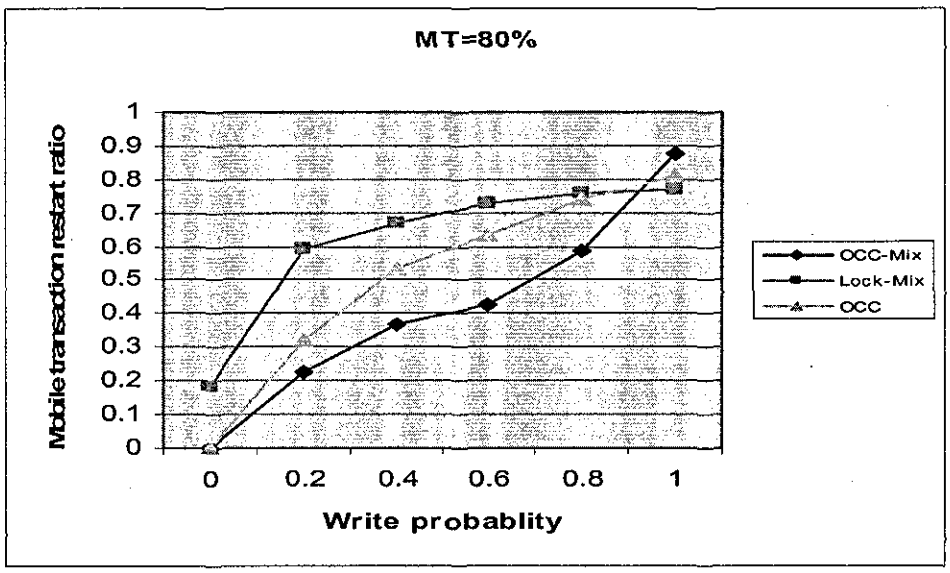


Figure6. 25: Restart ratio for mobile transactions (MT = 80 %)

6.4.3.2 Adjustment rate

One of the overheads of the OCC-Mix approach is the adjustment of the serialization order for those transactions that are in conflict with the validating transaction. Figure 6.26 give the adjustment rate for different mobile transaction percentages. This ratio, in fact, measures the effectiveness of the OCC-Mix

approach. When the ratio is zero, it means that no dynamic adjustment is made in the case of 100 percent read-only operations. The adjustment ratios increase when the conflicting transactions contain a mixture of read and write operations in the system.

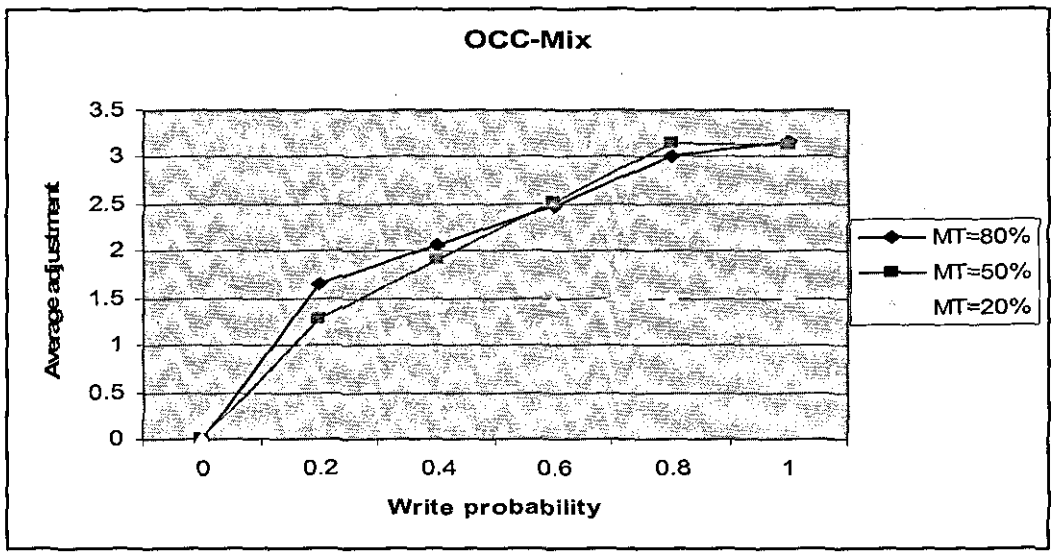


Figure6. 26: Adjustment Rate

6.4.4 Experiments 4: Impact of workload

6.4.4.1 Power consumption ratio

Figure 6.27 gives the power consumption ratio PCR as a function of the workload, when 20% of transactions present in the system are mobile. As the workload increases, the data access time by the mobile transactions increases. When the workload is low, the system is not congested and the PCR of the protocols represent the actual power needed for mobile transactions to complete execution. When the workload is greater than 65, the system begins to saturate and the consumption rate increases accordingly. As shown in Figure 6.27, the difference between Lock-Mix and OCC is not significant. This means that the time needed for restarted mobile transactions is close to the time that mobile transactions spend in waiting to get locks on their data items. In fact, the improvement made by OCC-Mix can not reflect the effectiveness of this protocol over the Lock Mix approach because the system saturation in this case is due to resource contention

instead of data contention. Whereas, the aim of the OCC-Mix approach is to save wireless resources by decreasing mobile transaction restarts and using non-blocking schemes, which are mainly motivated by data contention.

Figure 6.28 gives the PCR of the protocols when the system utilization of mobile transactions is 50 percent. In this experiment, the presence of mobile transactions intensifies resource contention. When the workload is low, the PCR of the OCC-Mix and OCC protocols is low. The system becomes saturated earlier when the majority of transactions are mobile this due to saturation of fixed and mobile resources by mobile transactions which need more time to complete their execution. The number of restarts of mobile transactions before they get a chance to commit under the OCC protocol, is more than under the OCC-Mix approach due to multiple restarts caused by using the OCC protocol. For the Lock-Mix approach, blocking increases the number of active transactions which also increase the possibility of restarting mobile transactions due to an increasing number of fixed transactions that switch to their blocking phase. So the PCR increases as a result of that, as shown in Figure 6.28. Nevertheless, the performance of the OCC-Mix approach is still superior to the other protocols across the whole range of workload. Figure 6.29 gives the PCR when the mobile transactions increase further to 80 percent. In this case, mobile transactions dominate. When the workload is low, the OCC-Mix approach can still function well. For a higher workload, the improvement made by the OCC-Mix approach becomes more notable. As the workload increases, the system begins to saturate and the advantages gained from using the OCC-Mix protocol decrease. On the whole, the performance of the OCC-Mix approach in decreasing PCR is consistently better than that of the other two protocols across different workloads and different amounts of domination of mobile transactions.

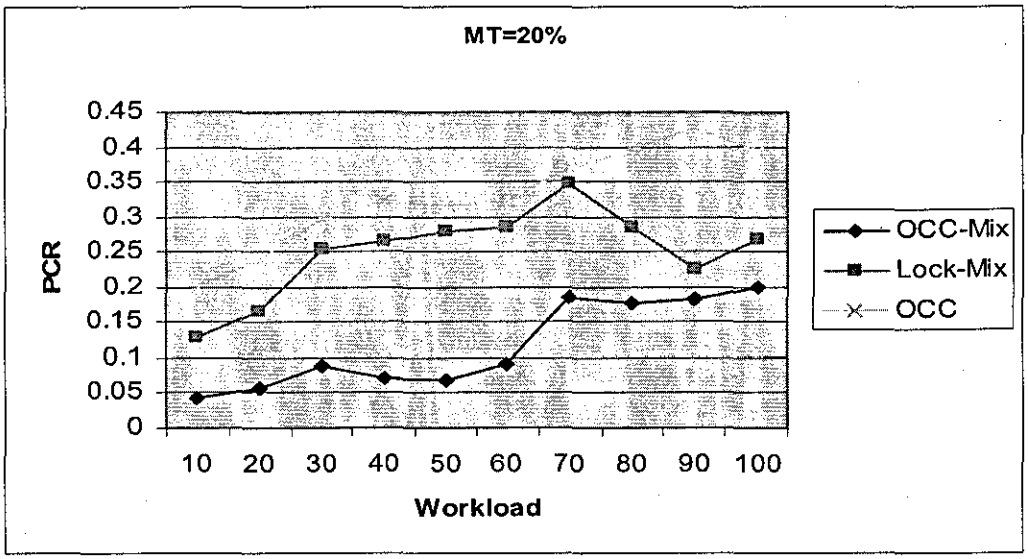


Figure6. 27: Power consumption ratio (MT = 20%)

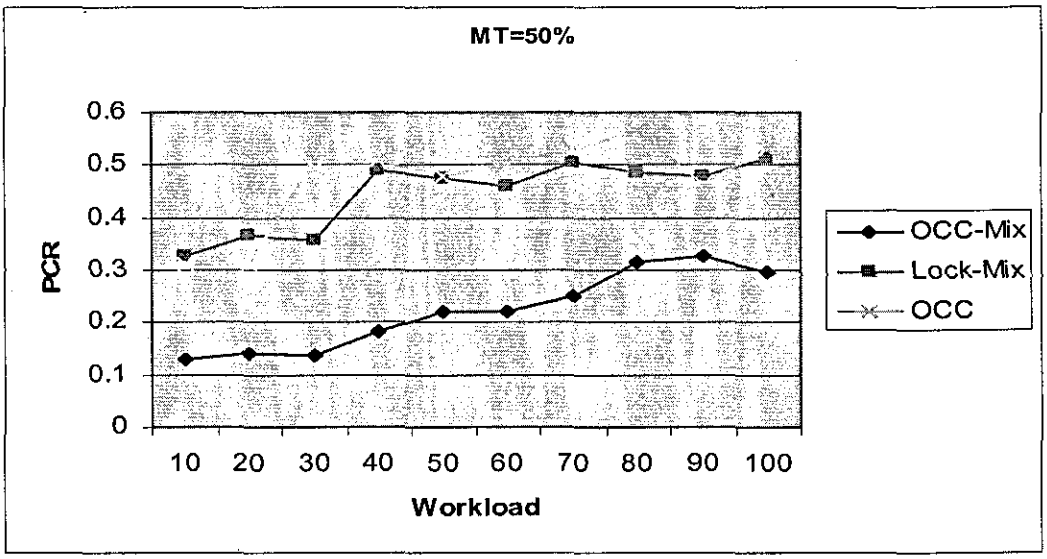


Figure6. 28: Power consumption ratio (MT = 50%)

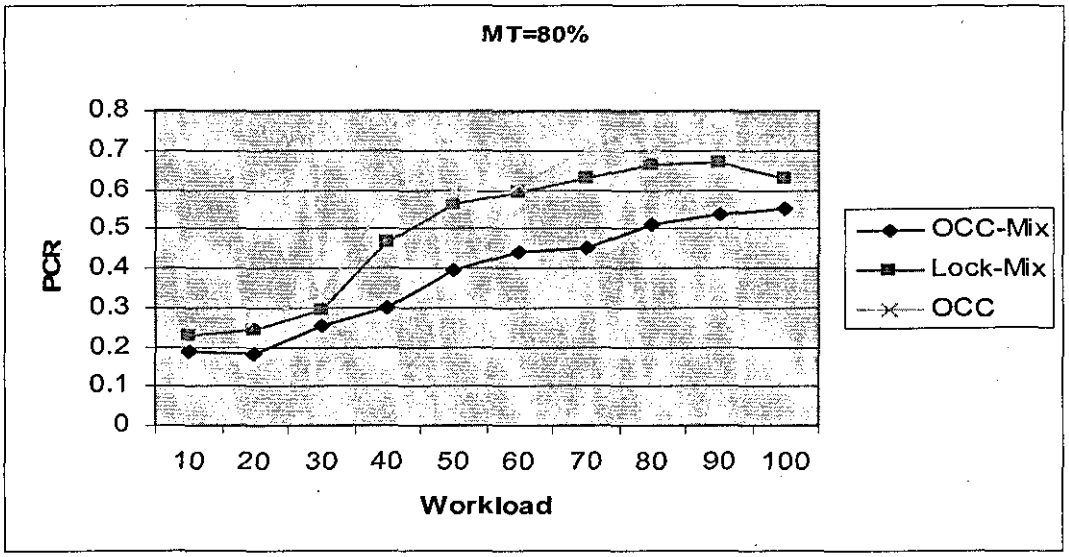


Figure6. 29: Power consumption ratio (MT = 20%)

6.4.4.4 Rollback frequency

To have a deeper understanding of the data conflict between fixed and mobile transactions, two performance measures, called fixed transaction rollback frequency and mobile transaction rollback frequency, are collected. The frequencies indicate the amount of data conflict between fixed and mobile transactions and among mobile transactions, respectively. The latter frequency also represents the fraction of committed mobile transactions which have roll-backed other mobile transactions.

Figure 6.30 and Figure 6.31 gives the mobile transaction rollback frequency when 20% of transactions present in the system are fixed. Most of data conflicts are among fixed transactions. As the workload increases, data conflict intensifies and it is more likely for a transaction to rollback others in order to be committed. When the system begins to saturate, frequency decreases as the number of committed transactions decreases. For a transaction being restarted, there must be one other transaction to roll it back. Figure 6.32 and Figure 6.33 give the frequencies when the utilization of fixed transactions is 50 percent. In Figure 6.32,

it can be observed that the amount of data conflicts between mobile and fixed transactions increases as the number of fixed transactions increases until the system begins to saturate. In these two figures, it can also be observed, for the OCC-Mix approach, that it is more likely for a mobile transaction to rollback a fixed transaction than for a fixed transaction to rollback another fixed transaction, though the utilization of mobile transactions is equal to that of fixed transactions. Since the OCC-Mix approach makes more room for mobile transactions, any fixed transaction that has data conflicts with a mobile transaction and whose timestamp interval shuts out, will be roll backed by the mobile transaction when it commits. A further increase of the utilization of mobile transactions exacerbates the situation.

Figure 6.33 and Figure 6.34 give the frequencies when the utilization of fixed transactions is 80 percent. When the workload is low, under the Lock-Mix approach data conflict between mobile and fixed transactions increases. As the workload increases, data conflicts among fixed transactions increase. On the whole, we can observe that the OCC-Mix approach can effectively help to reduce the number of restarts, whether they are due to data conflicts between mobile and fixed transactions or among mobile transactions. Moreover, even the OCC-Mix approach negatively affects the number of committed fixed transactions, as the restart ratio of fixed transaction increases when the percentage of mobile transaction present in the system increases. This ratio is acceptable since the wasted resources are concentrated in the fixed part of the network which can be tolerated especially when the performance gain in the scarce wireless resources is high as is shown by the restart ratio of mobile transactions.

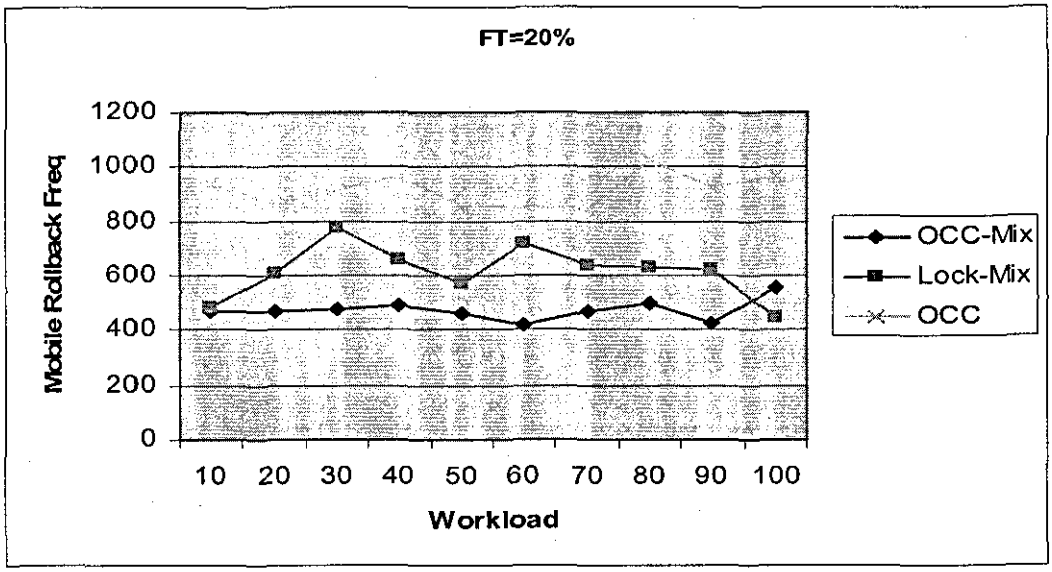


Figure6. 30: Mobile transaction rollback frequency (FT = 20%)

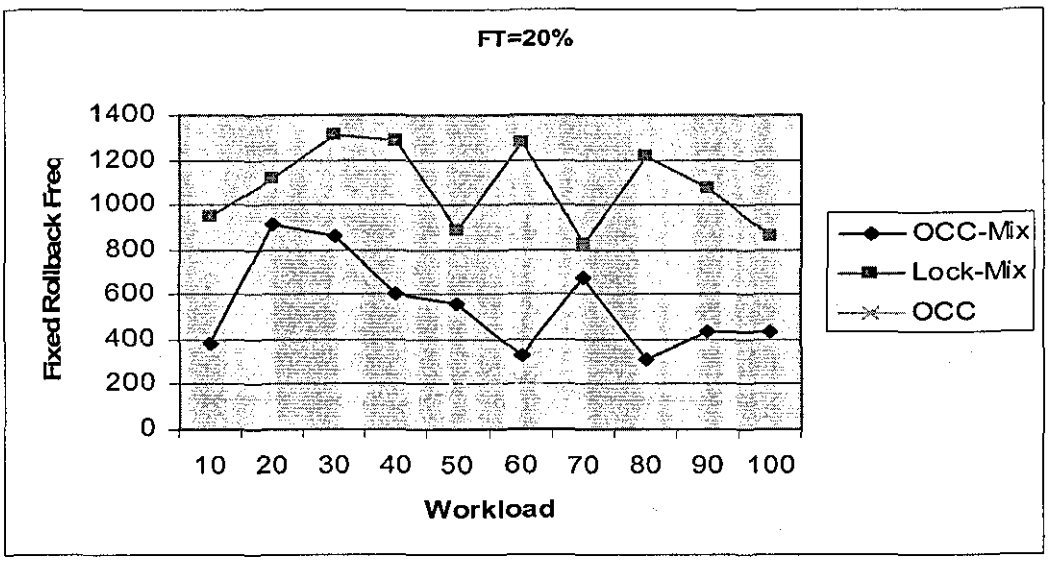


Figure6. 31: Fixed transaction rollback frequency (FT = 20%)

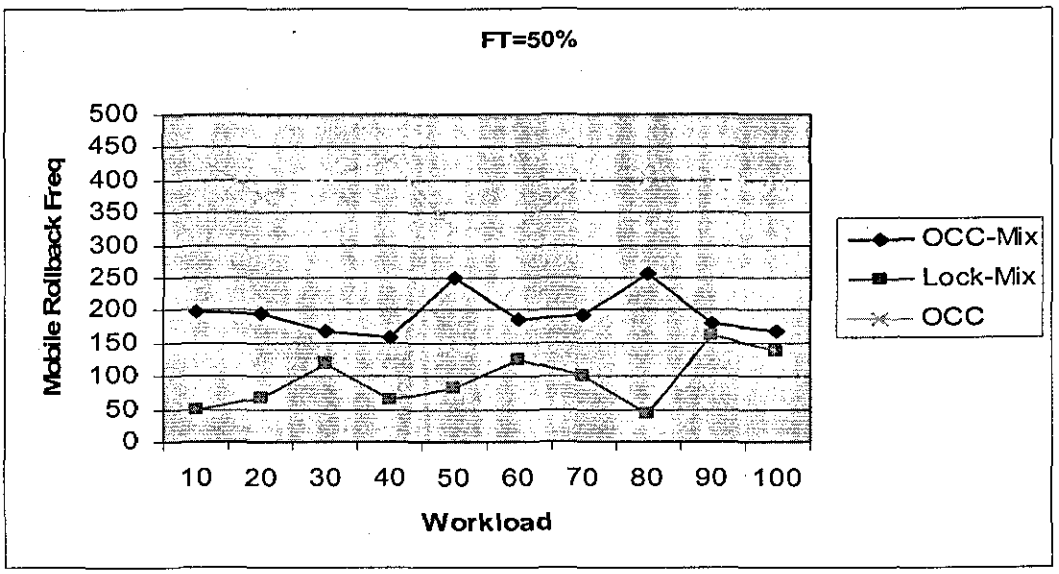


Figure6. 32: Mobile transactions rollback frequency (FT = 50%)

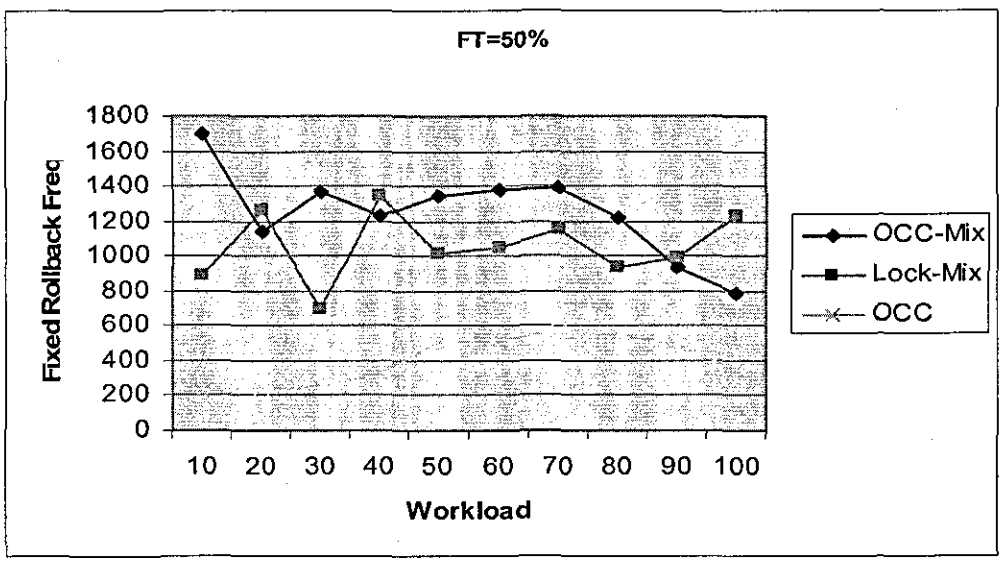


Figure6. 33: Fixed transactions rollback frequency (FT = 50%)

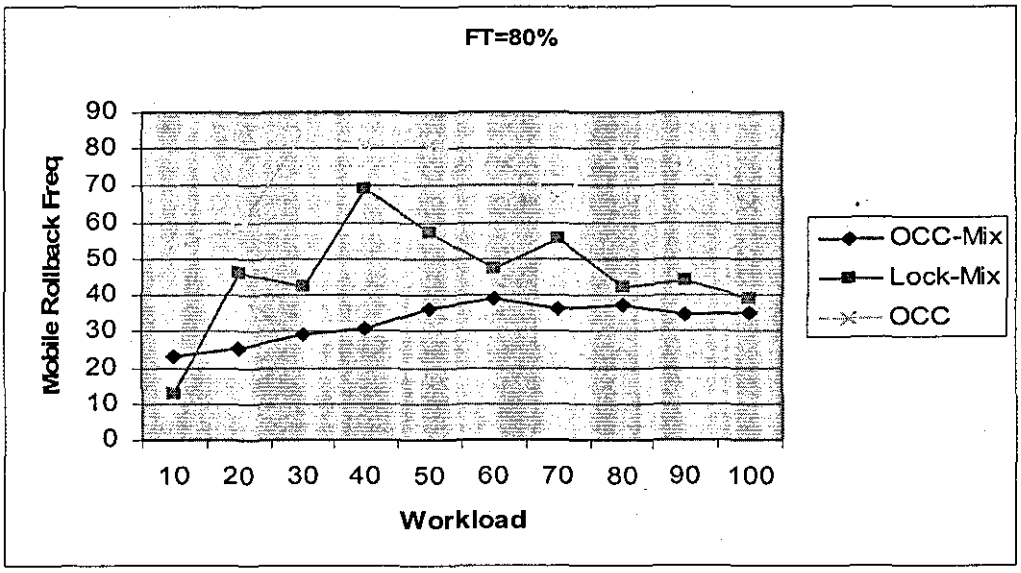


Figure6. 34: Mobile transactions rollback frequency (FT = 80%)

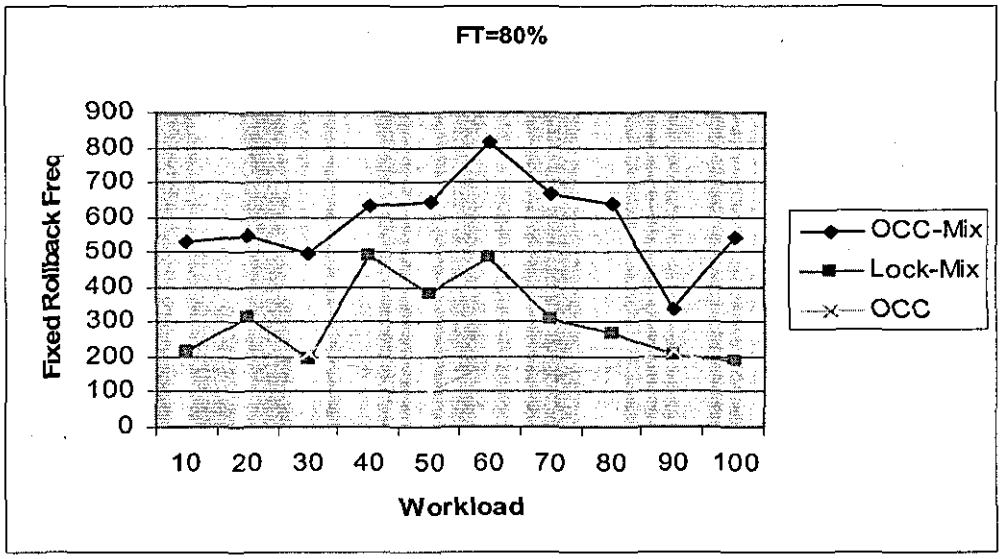


Figure6. 35: Fixed transactions rollback frequency (FT = 80%)

5.4.5 Experiments 5: Impact of μ-value and η -value

As the μ-value decreases, switching to the blocking stage, for fixed transactions, occurs earlier. This will increase the restart effect caused by the fixed transactions

on other mobile transactions especially when the μ-value becomes less than the η-value. Smaller values for both the μ-value and η-value will increase the effect of blocking resulting from the interactions between mobile and other mobile or fixed transactions and vice-versa. This can be seen by comparing Figure 6.36 and 6.37 which show the PCR of mobile transactions under different workloads for different values of μ and η-values. The PCR is used as it can reflect the effects of restarting and the blocking overhead for both the μ-value and the η-value. On the other hand, a reduction in the PCR for mobile transactions also has an inverse trend in terms of the restart rate and blocking overhead on the fixed transactions. As the figures indicate, a choice of μ-value =0.5*η-value leads to the best value of PCR with a reasonable number of restarted fixed transactions. Larger values of η-value lead to more restarts of mobile transactions where smaller values of η-value increase the effect of blocking and increase the PCR. By balancing the impact of these two effects, this choice of η-value leads to a reasonable reduction in the PCR compared to the restart rate for fixed transactions as we can see from Figures 6.38 and 6.39.

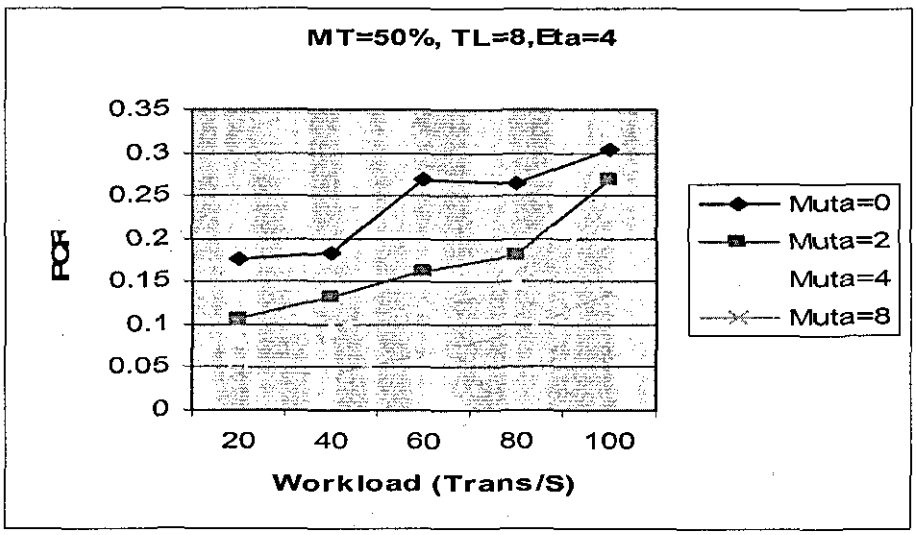


Figure6. 36: PCR at different μ- values

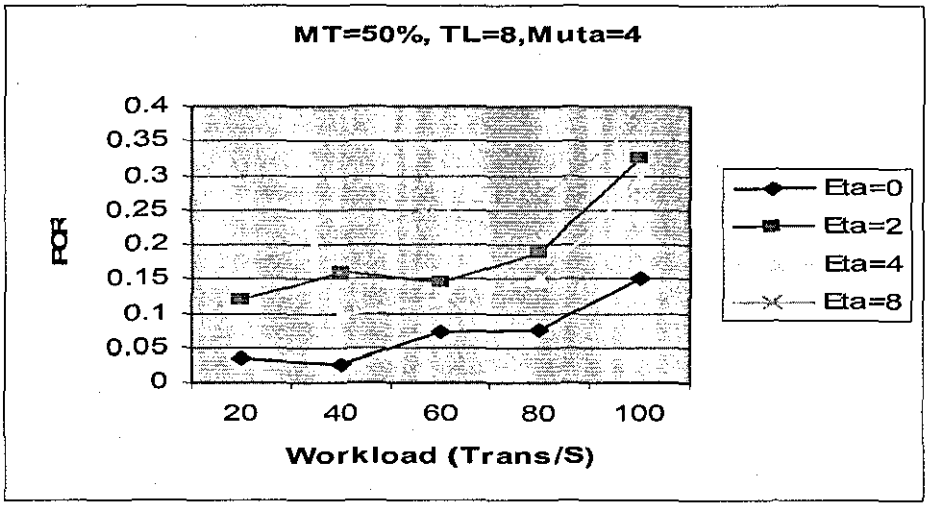


Figure6. 37: PCR at different η - values

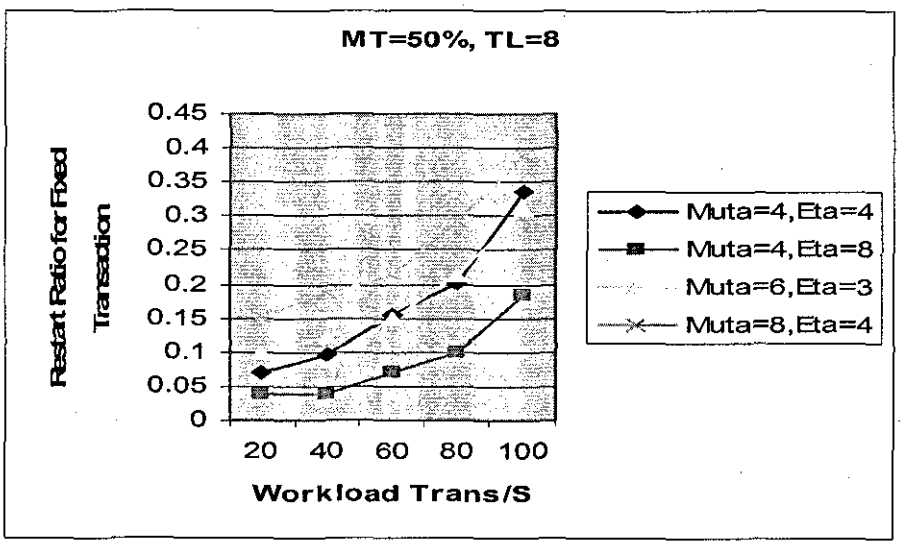


Figure6. 38: Fixed transaction restart ratio at different η and μ -values

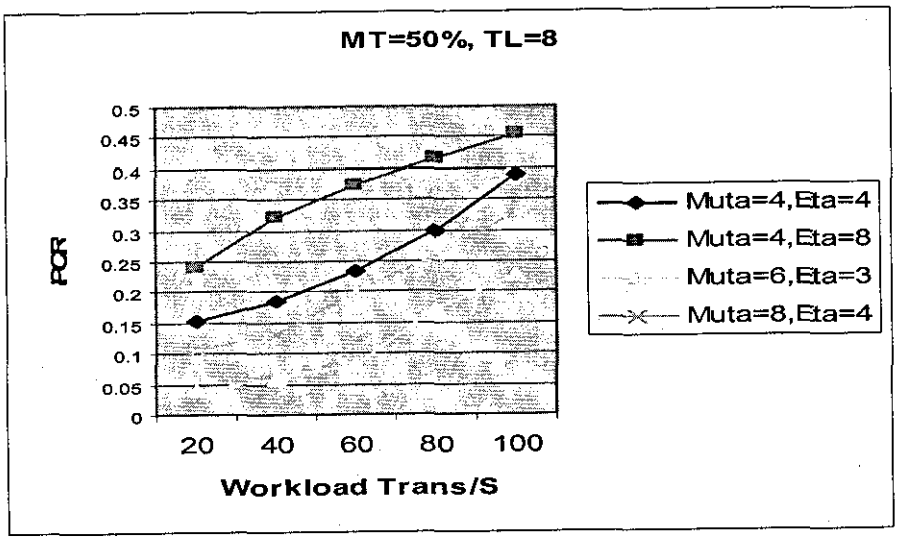


Figure6. 39: PCR at different η and μ -values

6.4.6 Experiments 6: Impact of σ -value

The σ -value can affect highly the nature of interactions between fixed and mobile transactions. A larger σ -value means little opportunity for fixed transactions to commit in their time intervals. Figure 6.40 shows the fixed rollback frequency for different values of σ. This measure determines the number of fixed transactions aborted because they are in conflict with other mobile transactions. As we can see in Figure 6.40 the number of fixed transactions roll backed by other mobile transactions increases as the σ -value increases. A choice of σ -value =2 leads to the smallest value of fixed rollback frequency. Smaller σ -value lead to more restarts in mobile transactions as we can see in the Figure 6.41 for mobile rollback frequency. This means that the adjustments of an active mobile transaction that is in conflict with a validating fixed transaction whose timestamp interval is bigger, will shorten the timestamp intervals of other mobile and fixed transactions which may be in conflict in the future. As we shown in Figure 6.40 and 6.41, intermediate values of σ lead to the best reduction of both fixed and mobile rollback frequency.

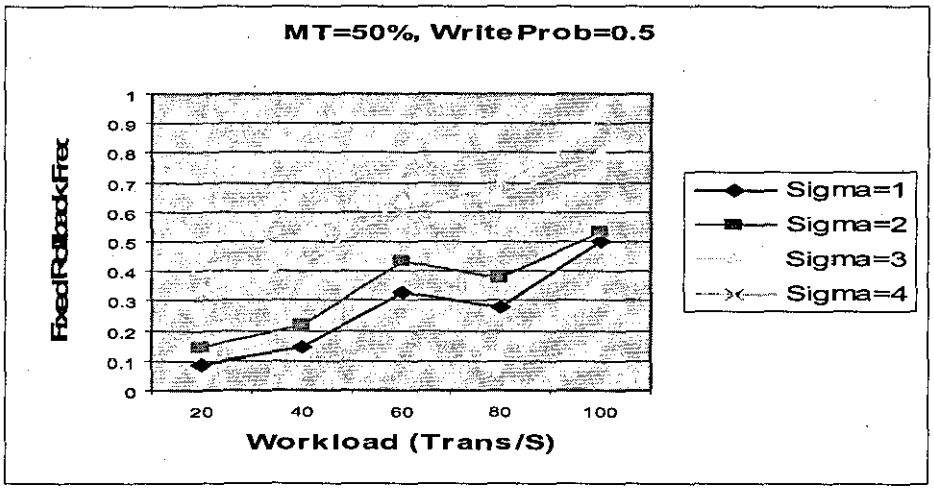


Figure6. 40: Fixed rollback frequency at different σ -value

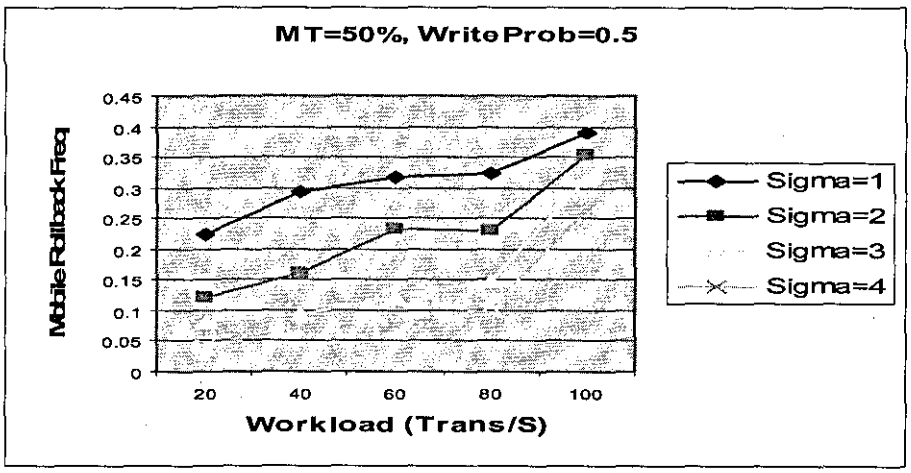


Figure6. 41: Mobile rollback frequency at different σ -value

CHAPTER 7

CONCLUSIONS AND FUTURE WORKS

Recent studies on concurrency control in mobile database systems have reported that optimistic approaches outperform locking protocols in reducing the resource consumption in wireless environments. However, most work based on the optimistic approaches experiences the problem of multiple transaction restarts. This problem is detrimental to wireless resources in mobile computing environments, because transaction restarts can significantly increase the system workload and intensify resource and data contention. In mixed transaction environments, the presence of fixed transactions exacerbates the problem where the fixed and mobile transactions are equally likely to be restarted. In such environments different types of transactions use resources with different characteristics whilst sharing the same data access. Mobile transactions use a very scarce wireless resource whereas fixed transactions use the reliable fixed host resources. Since the resources used by the mobile transaction are more expensive, reducing the number of mobile transaction restarts is very important in mixed transaction environments. In order to save resources whilst maintaining reasonable performance for fixed transaction execution, mobile transaction abort caused by fixed transactions and multiple restart caused by other mobile transactions, should be avoided. Therefore, when there are data conflicts between mobile and fixed transactions, mobile transactions should be given more room than fixed transactions. As a result, mobile transactions would be less likely to be aborted in mixed transaction environments.

In this study, two concurrency control protocols, called OCC-Mix and Lock-Mix, are proposed to alleviate the problem in mixed transactions environment.

The idea is to avoid wasting of scarce and expensive resources of mobile environments by: (1) avoiding multiple transaction restarts of mobile transactions by lengthens its timestamp interval during backward and forward adjustments and dynamically adjusting the serialization order of the conflicting transactions with respect to the validating transaction and (2) maintaining bigger timestamp intervals for mobile transactions giving them a better chance to commit.

Under the OCC-Mix approach, these are achieved by exploiting the semantics of the read and write operations in transactions, in addition to the transaction type, such that serializability can be preserved without restarting the conflicting transactions of the validating transaction. Only those transactions with strict conflicts with the validating transaction have to be restarted. In cases of non-strict conflicts, we only need to adjust the serialization order of those conflicting transactions with respect to the validating transaction. As a result, restarts can be decreased. In addition to allowing those non-strict conflicting fixed and mobile transactions to have an opportunity to complete their executions, resources can be saved from being utilized by restarted transactions such that other ongoing transactions will not be affected.

In the Lock-Mix approach, transaction execution can be divided into a non-blocking phase, where transactions wait for locks but do not block other transactions, and a blocking phase as in conventional locking. Data accessed during the non-blocking phase can lead to transaction aborts as in the OCC scheme. Since transactions in the proposed scheme explicitly wait for locks during the non-blocking phase, the abort probability is reduced by avoiding accessing data items currently under validation. Furthermore, except for deadlocks, no data accesses during the blocking phase can lead to an abort of a transaction. By properly choosing the switching values η and μ, the protocol can strike a balance between the effects of transaction abort and lock wait. We show that this approach can lead to better performance at different parameters settings.

A series of simulation experiments has been carried out to investigate the performance of the OCC-Mix approach. and results are reported in chapter 5. It has been found that the proposed protocols outperform the traditional optimistic concurrency protocols for a wide range of workload parameters. The first order of

improvement can be observed in decreasing the number of mobile transaction restarts leading to a significant saving of resources. The second order of improvement can be observed in the reduction of power consumption rate, crucial to mobile computing environments. Comparing the proposed approaches, OCC-Mix and Lock-Mix, the OCC-Mix approach shows an improved performance over Lock-Mix because there is no blocking effect. Also, the policy of delayed resolution of data conflict results in a greater chance of completing transactions in OCC-Mix when compared to the Lock-Mix approach. From the experimental results, we can conclude that OCC-Mix is of comparable performances with Lock-Mix when the degree of data contention is low. However, when the degree of data contention increases, the performance of Lock-Mix starts degrading, and the OCC-Mix outperforms it in such a situation. In addition, the time to complete a transaction using Lock-Mix is longer than that for OCC-Mix, implying that the expensive wireless resources are held longer by a mobile transaction under the Lock-Mix approach. No matter which concurrency protocol is employed, some transactions may still be disconnected when handoff occurs. We have also noted that the waste in fixed host resources is acceptable in comparison with saved wireless resources under both the OCC-Mix and Lock-Mix approaches. Therefore, these approaches are promising candidates for mixed transaction environments.

In the first part of this thesis two types of problems have been examined: mobile environment modelling and mobile transaction processing. The research in this part represents a substantial effort, though the result is preliminary. The performance of proposed transaction scheduling approaches is evaluated by the means of simulation. The simulation experiments performed in this research investigate the performance of three transaction execution strategies under the assumption that network disconnection occurs. The experiments are conducted from several perspectives by adjusting model parameters, such as, mobility timer and disconnection ratio. The simulation results show that if there is little or no network disconnection, the fixed host strategy has the shortest response time and the highest system throughput. This is not surprising because the system operates at or near the traditional fixed network environment. However, as network connectivity deteriorates, the mobile host strategy produces better system

throughput and response time than does the fixed host. In general, the combined host strategy produces better performance in terms of overall response time, elapsed processing time of a mobile host and total number of transactions completed by the system. Generally, by viewing a mobile host and a fixed host as two relatively independent computing stations; the combined host strategy can support autonomous operations with better system performance.

Our research work can be extended in several directions. First, alternative approaches to evaluating the performance of real-time database systems other than using simulation can be explored. These approaches include the measurement from actual running system and the use of analytical methods which may provide a concrete experience with a real system and an improved understanding of the functional requirements and operational behaviours of mixture transaction systems. Also, it is interesting to develop an analytical method for evaluating the performance of concurrency control algorithms for mixtures transactions systems. More over, we hope to analytically determine the best values of the μ and η for Lock-Mix and the best value of σ for the OCC-Mix approach.

Another direction for future work is to consider the issue of fixed and mobile interactions at various execution strategies for mobile transaction in connection with concurrency control. One limitation of our current work is that we have considered only the issue of relaxation atomicity to study and compare the different execution strategies by using 2-PL protocol. i.e., communication effect point view. However, mixture transaction systems have many others resources shared by mobile and fixed transactions resulting from scheduling of data access operations from both fixed and mobile transactions.

Bibliography

[1] H. Garcia-Molina, J. Ullman and J. Widom: Database Systems: The Complete Book, Prentice Hall, 2001.

[2] A. K. Elmagarmid: Database Transaction Models for Advanced Applications, Morgan Kaufmann, 1992.

[3] J. Gray and A. Reuter: Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, 1993.

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

[5] R. Elmasri and S. B. Navathe: Fundamentals of Database Systems, Addison Wesley, 2000.

[6] H. T. Kung and J. T. Robinson: On Optimistic Methods for Concurrency Control, ACM Transactions on Database Systems (TODS), 6(2), 1981, pp213-226.

[7] T. Härder: Observations on optimistic concurrency control schemes, Information Systems, 9(2), 1984, pp 111-120.

[8] G. H. Coulouris, J. Dollimore and T. Kindberg: Distributed Systems: Concepts and Design, Pearson Education, 2001.

[9] Y. Zhang, Y. Kambayashi, X. Jia, Y. Yang and C. Sun: On Interactions Between Coexisting Traditional and Cooperative Transactions, International Journal of Cooperative Information Systems (IJCIS), 8(2-3), 1999, pp 87-110.

[10] J. Jing, A. Helal and A. K. Elmagarmid: Client-Server Computing in Mobile Environments, ACM Computing Surveys, 31(2), 1999, pp 117-157.

[11] M. T. Özsu and P. Valduriez: Principles of Distributed Database Systems, 1999

[12] K. Ramamritham and P. Chrysanthis: Advances in Concurrency Control and Transaction Processing, IEEE Computer Society Press, 1996.

[13] R. A. Dirckze and L. Gruenwald: A pre-serialization transaction management technique for mobile multidatabases, Mobile Networks and Applications (MONET), 5(4), 2000, pp 311-321.

[14] G. H. Forman and J. Zahorjan. The Challenges of Mobile Computing. IEEE Computer Volume: 27(4), pp. 38-47, April 1994.

[15] R. Alonso and H. F. Korth. Database System Issues in Nomadic Computing. In Proceedings of the 1993 ACM SIGMOD International Conference on Management

of Data, pp. 388 – 392, 1993.

[16] E. Pitoura, B. Bhargava, Maintaining consistency of data in mobile computing environments,in: Proceedings of 15th International Conference on Distributed Computing Systems, June,1995 (Extended version to appear in IEEE Transactions on Knowledge and Data Engineering, 1999).

[17] Y. Breitbart, H. Garcia-Molina and A. Silberschtz. Overview of Multidatabase Transaction Management. VLDB, 1(2): 181-239, 1992.

[18] S.K. Madria, B. Bhargava, E. Pitoura, V. Kumar, Data organization issues in location dependent query processing in mobile computing environment, in: Proceedings of 4th East-European Symposium on Advances in Databases and Information Systems (in co-operation with ACM-SIGMOD), Prague, Czech Republic, 2000.

[19] O. Wolfson, X. Bo, C. Sam, L. Jiang, Moving objects databases: issues and solutions, in:Proceedings of SSDBM, 1998, pp. 111-122.

[20] J. Gray: The Transaction Concept: Virtues and Limitations, Very Large Data Bases, 1981, pp 144-154.

[21] J. E. B. Moss: Nested transactions: an approach to reliable distributed computing, Massachusetts Institute of Technology, 1985.

[22] G. Weikum: Principles and Realization Strategies of Multilevel Transaction Management, ACM Transactions on Database Systems, 16(1), 1991, pp 132-180.

[23] H. Garcia-Molina and K. Salem: Sagas, ACM SIGMOD International Conference on Management of Data, 1987, pp 249-259.

[24] C. Pu, G. E. Kaiser and N. C. Hutchinson: Split-Transactions for Open-Ended Activities., Very Large Data Bases (VLDB), 1988, pp 26-37.

[25] P. K. Chrysanthis: Transaction Processing in Mobile Computing Environment, IEEE Workshop on Advances in Parallel and Distributed Systems, 1993, pp 77-83.

[26] G. D. Walborn and P. K. Chrysanthis: Transaction Processing in PROMOTION, ACM Symposium on Applied Computing (SAC), 1999, pp 389- 398.

[27] A. Yendluri , Wen.C and Chih.F " Improving Concurrency Control in Mobile Databases" Lecture Notes in Computer Science, Volume 2973 , P 642-655 , February 12, 2004 .

[28] E. Pitoura and B. K. Bhargava: Data Consistency in Intermittently Connected Distributed Systems, IEEE Transactions on Knowledge and Data Engineering (TKDE), 11(6), 1999, pp 896-915.

[29] S. K. Madria and B. K. Bhargava: A Transaction Model for Mobile Computing, International Database Engineering and Application Symposium (IDEAS), 1998, pp 92-102.

[30] S. K. Madria and B. K. Bhargava: A Transaction Model to Improve Data Availability in Mobile Computing, Distributed and Parallel Databases, 10(2), 2001, pp 127-160.

[31] M. H. Dunham, A. Helal and S. Balakrishnan: A Mobile Transaction Model That Captures Both the Data and Movement Behavior., Mobile Networks and Applications (MONET), 2(2), 1997, pp 149-162.

[31] K.-I. Ku and Y.-S. Kim: Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems, Research Issues in Data Engineering (RIDE), 2000, pp 39-46.

[33] A. K. Elmagarmid, Y. Leu, W. Litwin and M. Rusinkiewicz: A Multidatabase Transaction Model for InterBase, International Conference on Very Large Data Bases, 1990, pp 507-518.

[34] M. Satyanarayanan, Mobile information access, IEEE Personal Communications 3 (1) (1996).

[35] Jian Chen, "Mobility Information and Mobile Transaction Processing", http://hdl.handle.net/1993/1220

[30] Filip Perich, Anupam Joshi, Timothy Finin, Yelena Yesha, "On Data Management in Pervasive Computing Environments," IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 5, pp. 621-634, May, 2004

[31] M.H. Dunham and V. Kumar, "Location dependent data and its management in mobile databases," in Int. DEXA Workshop on Mobility in Databases and Distributed Systems, Vienna, Austria, Aug. 1998.

[32] Huiping Cao , Shan Wang , Lingwei Li, Location dependent query in a mobile environment, Information Sciences—Informatics and Computer Science: An International Journal, v.154 n.1-2, p.71-83, August 2003

[33] Akar, M., & Mitra, U."Motion Constraint Based Handoff Protocol for Mobile Internet," IEEE Wireless Communications and Networking Conference (WCNC), March 2003] [80[(2003). Soft handoff algorithms for CDMA cellular networks. IEEE Transactions on Wireless Communications,2(6), 1259–1274.

[34] P. S. Yu and D. M. Dias. Analysis of hybrid concurrency control schemes for a high data contention environment. IEEE Trans. on Software Engineering, 18(2):118--129, Feb. 1992.

[35] R.Bayer, K.Elhardt, J.heigert, and A.Reiser. Dyanamic time stamp allocation for

transaction in database systems. In H.J.Schneider, editor, Distributed Data Bases, North-Holland, 1982

[36] [36] D. Barbara, T. Imielinski, Sleepers and workaholics: caching strategies in mobile environments, VLDB Journal (1995).

[37] K. Wu, P.S. Yu, M. Chen, Energy Efficient caching for wireless mobile computing, in: Proceedings of the 12th International Conference on Data Engineering, New Orleans, February, 1996.

[38] B.R. Badrinath, T. Imielinski, Replication and mobility, in: 2nd IEEE Workshop on the Management of Replicated Data, November, 1992, pp. 9-12.

[39] Wu Shiow-yang, Y. Change, An active replication scheme for mobile data management, in: IEEE Proceedings of 6th DASFAA, Taiwan, 1999.

[40] M. Faiz, A. Zaslavsky, Database replica management strategies in multidatabase systems with mobile hosts, in: 6th International Hong Kong Computer Society Database Workshop, 1995.

[41] Y. Huang, P. Sistla, O. Wolfson, Data replication for mobile computers, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1994.

[42] J. Gray, P. Helland, P. O'Neil, D. Shasha, The dangers of replication and a solution, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 1996, pp.173-182.

[44] D.B. Terry, D. Goldberg, D.A. Nichols, B.M. Oki, Continuous queries over append-only databases, in: Proceedings of the ACM-SIGMOD International Conference on Management of Data, June, 1992.

[45] S.K. Madria, M. Mohania, J. Roddick, A query processing model for mobile computing using concept hierarchies and summary databases, in: Proceedings of the 5th International Conference on Foundation for Data Organization (FODO'98), Japan, November, 1998

[46] T. Imielinksi, B.R. Badrinath, Querying in highly distributed environments, in: Proceedings of the 18th VLDB, August, 1992, pp. 41-52.

[47] Minsoo Lee , Sumi Helal, "HiCoMo: High Commit Mobile Transactions", Distributed and Parallel Databases, Kluwer Academic Publishers. Manufactured in The Netherlands, vol.11, 73–92, 2002

[48] Huang, Y., "Efficient Transaction Processing in Broadcast-based Asymmetric Communication Environment," Ph.D. Dissertation Proposal, 2001.

[49] Q. Lu and M. Satynarayanan, "Improving data consistency in mobile computing using isolation-only transactions,"in IEEE HotOS Topics Workshop, Orcas Island,

USA, May 1995.

[50] P. Serrano-Alvarado, C. Roncancio and M. E. Adiba: A Survey of Mobile Transactions, Distributed and Parallel Databases, 16(2), 2004, pp 193-230.

[51] B. Lim, A. R. Hurson, K. M. Kavi. Concurrent Data Access in a Mobile Heterogeneous System. Proceedings of the 32nd Annual Hawaii International Conf. on System Sciences. 1999 .

[52] Angelo Brayner and José A. Morais F., "Increasing Mobile Transaction Concurrency in Dynamically Configurable Environments", Proceedings of the 3rd. IEEE Workshop on Mobile Distributed Computing (MDC). 2005.

[53] M. Shapiro, A. I. T. Rowstron, and A.-M. Kermarrec, "Application-independent Reconciliation for Nomadic Applications," in ACM SIGOPS European Workshop 2000, 2000, pp.1–6.

[54] Can Turker and Gabriele Zini, "A Survey of Academic and Commercial Approaches to Transaction Support in Mobile Computing Environments", Swiss Federal Institute of Technology Zurich Institute of Information Systems, ETH Zentrum, Techniqual report #429,NOV 2003.

[55] Xiaoyan Hong , Mario Gerla , Guangyu Pei , Ching-Chuan Chiang, A group mobility model for ad hoc wireless networks, Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, p.53-60, August 20-20, 1999, Seattle, Washington, United States

[56] Popovici and G. Alonso, "Ad-hoc transactions for mobile services," in VLDB Workshop on Technologies for E-Services, Hong-Kong, China, August 2002.

[57] Sanjay Kumar Madria, Mukesh K. Mohania, Sourav S. Bhowmick, Bharat K. Bhargava: Mobile data and transaction management. Inf. Sci. 141(3-4): 279-309 (2002)

[58] N. Barghouti, G. Kaiser, Concurrency control in advanced database applications, ACMComputing Surveys 23 (3) (1991) 269-317.

[59] K. Ramamritham, P.K. Chrysanthis, A taxonomy of correctness criterion in database applications, Journal of Very Large Databases 4 (1) (1996).

[60] G.D. Walborn, P.K. Chrysanthis, Supporting semantics-based transaction processing in mobile database applications, in: Proceedings of 14th IEEE Symposium on Reliable Distributed Systems, September, 1995, pp. 31-40.

[61] E. Pitoura, B. Bhargava, Revising transaction concepts for mobile computing, in: Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications, December, 1994, pp. 164-168.

[62] A. Rasheed, A. Zaslavsky, Ensuring database availability in dynamically changing mobile computing environment, in: Proceedings of the 7th Australian Database Conference, Melbourne, Australia, 1996.

[63] E. Pitoura, B. Bhargava, Building Information Systems for Mobile Environments, in: Proceedings of 3rd International Conference on Information and Knowledge Management, 1994, pp. 371-378.

[64] M.R. Ebling, Evaluating and improving the effectiveness of caching for availability, Ph.D.Thesis, Department of Computer Science, Carnegie Mellon University, 1997.

[65] Philip S. Yu and Daniel M. Dias, "Performance Analysis of Concurrency Control Using Locking with Deferred Blocking", IEEE transactions on software engineering, vol. 19, no. 10, 1993

[66] Pitoura, E., "Supporting Read-Only Transactions in Wireless Broadcasting," Proc. of the DEXA98 International Workshop on Mobility in Databases and Distributed Systems, pp. 428-422, 1998.

[67] M. A. Viredaz, L. S.Brakmo, and W. R. Hamburgen, "Energy Management on Handheld Devices," Queue, vol. 1, no. 7, pp. 44–52, 2003.

[68] Carl S. Hartzman, "The Delay Due to Dynamic Two-Phase Locking", IEEE transactions on software engineering, vol. 15, no. 1, 1989

[69] S. H. Son, J. Lee, and Y. Lin. Realtime scheduling using dynamic adjustment of serialization order for real-time concurrency control. Real-Time Systems, 4(3):243--268, Sept. 1992

[70] Bernstein, Philip A. and Goodman, Nathan. "Concurrency Control in Distributed Database Systems". ACM Computing Surveys. Retrieved on September 21, 2005.

[71] Hien. N. Le. A transaction processing system for supporting mobile collaborative works, PhD thesis, Department of Computer and Information Science, Norwegian University of Science and Technology 2006

[72] Kung H. T., and J. T. Robinson, "On Optimistic Methods for Concurrency Control," ACM Transactions on Database Systems, 6(2): 213-226, June 1981.

[73] J.D. Noe and D.B. Wagner. Measured performance of time interval concurrency control techniques. In Proc. of Very Large Data Bases, pages 359--365, 1987.

[74] Pitoura, E. and Chrysanthis, P. K., "Exploiting Versions for Handling Updates in Broadcast Disks," Proc.of the 25th VLDB Conference, pp. 114-125, Scotland, 1999.

[75] Lee, V. C. S., Lam, Kwok-wa and Son, S. H., "Real-time Transaction Processing with Partial Validation at Mobile Clients," Proc. of the Seventh International Conference on Real-Time Computing Systems and Applications (RTCSA 2000), pp. 473-477, South Korea, December 2000.

[76] Pitoura, E., "Supporting Read-Only Transactions in Wireless Broadcasting," Proc. of the DEXA98 International Workshop on Mobility in Databases and Distributed Systems, pp. 428-433, 1998.

[77] Herman, G., Gopal, G., Lee, K. C. and Weinreb, A., "The Datacycle Architecture for Very HighThroughput Database Systems," *Proc. of the ACM SIGMOD Conference*, U.S.A., pp. 97-103, 1987.

[78] Wu, Simon, Lee, V. C. S. and Lam, Kwok-wa, "Broadcast Transaction Scheduling in Mobile Computing Environments," Proc. of the 3rd International Conference on Mobile Data Management, pp. 161-162, Singapore, January 2002.

[79] EE · Ho-Jin Choi, Byeong-Soo Jeong: "A Timestamp-Based Optimistic Concurrency Control for Handling Mobile Transactions". ICCSA (2) 2006: page796-805.

[80] Alqerem, A.; Hussak, W."Mixed Mobile and Fixed Transactions Scheduling in Mobile Computing environment" the 8th annual conference PGNET 2007

[81] Angelo Brayner, José Maria Monteiro: Temporal Serialization Graph Testing: An Approach to Control Concurrency in Broadcast Environments. SBBD 2000: 287-301

[82] Victor C. S. Lee, Kwok-Wa Lam, Sang Hyuk Son, Eddie Y. M. Chan: On Transaction Processing with Partial Validation and Timestamp Ordering in Mobile Broadcast Environments. IEEE Trans. Computers 51(10): 1196-1211 (2002)

[83] R. E. Gruber, "Optimism vs. Locking: A Study of Concurrency Control for Client-Server Object-Oriented Databases," MIT Laboratory for Computer Science, Cambridge, MA, Tech. Rep. MIT/LCS/TR-708, 1997.

[84] C. Boksenbaum et al. Concurrent certifications by intervals of timestamps in distributed database systems. IEEE Trans. on Software Engineering, SE-13(4):409--419, Apr. 1987.

[85] J. N. Gray et al. Granularity of locks and degrees of consistency in a shared data base. In G. M. Nijssen, editor, Proc. of IFIP TC-2 Working Conference on Modelling in Data Base Management Systems, pages 1--29. North-Holland, 1976.

[86] D. Agrawal, A. El Abbadi, and A. E. Lang. Performance characteristics of protocols with ordered shared locks. In Proc. of Int. Conf. on Data Engineering,

pages 592--601, 1991.

[87] R. Bayer et al. Dynamic timestamp allocation for transactions in database systems. In H.-J. Schneider, editor, Proc. of 2nd Int. Symp. On Distributed Data Bases, pages 9--21. North- Holland, 1982.

[88] P. S. Yu, H.-U. Heiss, and D. M. Dias. Modelling and analysis of a time-stamp history based certification protocol for concurrency control. IEEE Trans. on Knowledge and Data Engineering, 3(4):525--537, Dec. 1991.

[89] Alqerem, A.; Hussak, W."Concurrency control in presence of bandwidth variability" IADIS International Wireless Applications and Computing 2007apos; 07. 2ndVolume 3, Issue , 5-9 June 2007 Page(s): 2926 - 2928

[90] D. Tse and P. Viswanath, Fundamentals of Wireless Communication, Cambridge University Press, 2005.

Appendix: Publications

1. Alqerem, W.Hussak, "Concurrency Control for Mixed Mobile and Fixed Host Transaction in Mobile Database System" 2007 IEEE (ICSPC07) (accepted)

2. Alqerem, W.Hussak, "Modeling Data Scheduling for Mobile Transaction in Broadcasting Environments" Accepted for publication in IAJIT journal 2008 (accepted)

3. Alqerem, W.Hussak, "Evaluation of transaction execution strategies in mobile data base systems" accepted for publication in Advance in computer science and engineering journal ACSE 2008 (accepted)

4. Alqerem, W.Hussak, "Data Scheduling for Mobile Transaction in Broadcasting Environments (wireless computing conference IADIS 2007) (accepted)
5. Alqerem, W.Hussak, "Concurrency control for mobile transactions in presence of bandwidth variability 2007(wireless computing conference IADIS 2007) (accepted)

6. Alqerem, W.Hussak, "Concurrency Control for Moflex Transaction Model" ICIT 2005 (accepted)

7. Alqerem, W.Hussak, "Concurrency Control for location dependent transactions in mobile computing environment" 2006 IEEE (ICTTA 06) (accepted)

8. Alqerem, W.Hussak, "Mixed Mobile and Fixed Transactions Scheduling in Mobile Computing Environment" PGNET 2007 UK (accepted)

9. Alqerem, W.Hussak, "Lock-Mix Approach for Concurrent Mixture of Fixed and Mobile Transactions" Pervasive and Mobile Computing Journal 2008 (submitted)

10. Alqerem, W.Hussak, "OCC-Mix Approach for Concurrent Mixture of Fixed and Mobile Transactions" Pervasive and Mobile Computing Journal 2008 (submitted)

11. Alqerem, W.Hussak, "Comparative Study of Different Concurrency Protocols in Mixed Transaction Environments" 2008 IEEE (CITE) (submitted)