Transaction Replication in Mobile Environments

By

Lau Wai Kwong

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Philosophy

(Computer Science & Engineering)

at the

CHINESE UNIVERSITY OF HONG KONG

JUNE 1996

Abstract

Transaction management is recognized to be a fundamental need in commercial distributed databases. It is expected that in the near future, distributed systems will be extended to support mobile computations over a network with fixed and mobile hosts. The restrictions imposed by the wireless channels and the mobility of mobile hosts make the traditional ways of handling transactions inadequate. However, it is found that only a few schemes which bear in mind the concept of transactions have been developed in mobile computing environments [10, 22, 27, 32].

We present a scheme for transaction management in mobile computing environments, with the use of the *transaction replication scheme* as the replication control scheme in the fixed network [14]. Each mobile host has a limited cache space to store some of the data objects in the database. Transactions submitted at the mobile hosts consisting solely of read operations are executed locally there, with the use of the cached data to reduce the execution time. Before the transactions can start to execute, data cached there have to be updated in order to ensure serializability. This is achieved by the broadcasting of data updated periodically to the mobile hosts. For those transactions consisting of both read and write operations, they are sent to the fixed hosts for execution. Results of the transactions are broadcasted back to the mobile hosts periodically.

The proposed scheme has been implemented and analyzed with the use of simulation programs, and shows a good performance. It is found that with the use of the transaction replication scheme, transactions submitted at the mobile hosts with writes can achieve a high commit ratio; and the response time of transactions executed at the mobile hosts is

kept low. Besides, it is verified through simulation that given a wireless channel that is unreliable, collecting data requested and sending them in batches to the fixed hosts provide a better performance than sending each request message individually. The scheme is also compared with a lock-based scheme which handles transactions by sending the operations of transactions to the fixed hosts for execution. The lock-based scheme is found to be inferior to our scheme in most cases and it outperforms our scheme only when the size of database is large and the transactions submitted at the mobile hosts contain read operations only.

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Wai-chee Fu , for her support and encouragement to my research. She was just so patient to my silliness and always respected my opinions. Her comments on the contents and organization of the thesis improved the quality greatly.

Special thanks also go to Prof. Tiko Kameda, Prof. M. H. Wong and Prof. C. Lu for serving on my thesis committee. Prof. Tiko Kameda carefully read the thesis, and provided many helpful suggestions for improvement. Its really my pleasure to have them revising my thesis.

I would like to thank my colleagues in the graduate school for without their inexhaustive support and encouragement I could not possibly make it. They gave me so much help whenever I came across any kind of problems. Among them, I would particularly like to thank Cothan Hwang Hoi Yee for his helpful comments to my graphics assignments; Johnson Chong Chiu Fai, for his expertise in using Matlab; Alice Ng Ching Ting, for solving all my problems about the transaction replication scheme; Phyllis Leung Wing Man, for her helpful tips in writing CSIM programs; Charles Cheng Sheung Hing and Derek Lam Ming Fan, for their excellent work in the simulation programs; Edward Tam Nai Biu, for always providing us with interesting magazines; Jeff Cheung Hon Kai, for telling us so much about his incredible experience.

The staff of the department provided excellent support. They maintained the machines in good conditions and installed more powerful machines. These are all helpful in conducting this research.

Last (but not least) grateful thanks to my family, who have put up with the noise from my hitting of the keyboard at midnight for so long. I am very grateful for their support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis presents a scheme that handles transactions submitted at the mobile hosts in a mobile computing environment. The work is motivated by the following reasons:

- The rapid expanding technology in wireless communication and the portability of computers make mobile computing possible. Yet the limitations of a mobile environment require the rethink of the ways of transaction management in mobile computing environments.

- Studies have been carried out in supporting database management over the mobile computing environments. However, they all have some shortcomings. Some of them put their focus on the caching of data at the mobile hosts, with little consideration about transactions; Or there are many details missing in their proposals, and there is no performance analysis.

Mobile hosts refer to portable computing devices that can move about while retaining network connection. Equipment like PCs, portables, laptops, notebooks, or palmtops can all be connected as mobile hosts. As is stated in [25], mobile hosts are characterized by their size, weight and the kind of power supply. In order for the mobile hosts to move around easily, they should be small in size and light in weight. Besides, since most of the mobile hosts are battery power-supplied, reducing power consumption should be one of the requirements in the design of mobile hosts. All these characteristics introduce new challenges to the computer science society.

Mobile hosts should communicate with other larger and more powerful systems so that they can access the data and services provided there. To achieve this, each mobile host has a wireless interface augmented to it. Messages can then be sent in the air by modulating radio waves or pulsing infrared light.

The resulting computing system is therefore known as the *mobile computing system* as it contains hosts that can move about while retaining network connection. It is a distributed system consisting of a set of powerful fixed hosts and a larger number of mobile hosts.

It is expected that mobile computing will be used in applications which are information service oriented and mail-enabled [19]. With the mobility nature, users carrying mobile hosts can access online information through the computer terminals. The information may be location-dependent, such as locating the nearest cinema. One goal of mobile computing is to allow access to databases at anytime from anywhere within a geographical area.

Most of the applications mentioned above require the use of transactions or transaction-like services. However, owing to the inherent properties of the mobile environments, transaction management in mobile systems is different from that for traditional distributed systems. For example, owing to the frequent disconnections of mobile hosts, data may have to be downloaded before a mobile host is going to be disconnected so as to allow some transactions to execute during the period of disconnection. This requires the redesign of algorithms for transaction management.

1.1 Limitations of the Mobile Computing Environments

Recent advances in technology have made a step forward in mobile computing. However, there are still lots of technical challenges to be tackled. These challenges are the consequences of the characteristics that are unique to the mobile computing environments [12]. They include:

- Wireless communication. Mobile hosts communicate with other parties through the wireless channels. However, the wireless channels are inferior to the wired channels in many dimensions. This generates a lot of problems, such as narrow bandwidth, frequent disconnection, security problems, etc.

- Mobility. Mobility of mobile hosts results in the volatility of information. Such a characteristic introduces many challenging problems in data consistency.

- Portability. For a mobile host to be portable, it should be small and light. However, these requirements are also the constraints in the power of mobile hosts.

Wireless Communication

Currently, a number of wireless networks are being used. They differ in the coverage area as well as the available bandwidth. Cellular network is used mainly for the transfer of voice data with hand-held phones. It cannot support a large number of users and the available bandwidth is very low.

Wireless LAN is the traditional LAN with wireless interface added to communicate with the mobile hosts. It is designed to be used in local environments and its coverage area is smaller than that of the cellular network.

Wide-area wireless network covers a wide area with low bandwidth channels. However, its scale with a large number of users is unknown.

Among the wireless networks mentioned, wireless LAN provides a data rate of about 1-to-2 Mbps, which is much larger than that of wide-area wireless network. However, this value is still an order of magnitude less than what is available in fixed networks [3, 20]. The narrow wireless bandwidth becomes a major performance bottleneck for the mobile computing systems [12, 19, 30].

It is also believed that the frequency of network failure in wireless communication is much higher than that of the wired communication [3]. This is due to the fact that the mobile hosts are being used in much harsher conditions than the fixed hosts. They may be used in places with a lot of background noise, or may be dropped accidentally. There

is a choice of whether to spend efforts on preventing disconnections or to allow the mobile hosts to work under disconnections.

Mobility

Mobility of mobile hosts brings a lot of problems to the design of mobile computing systems. Problems arising from this issue can be classified into location management and configuration management [20, 19].

As a mobile host moves, its network address changes. This introduces problems in updating and querying this kind of data. Location management can also be viewed as a data management problem [19].

Many distributed computing algorithms depend on the logical structures (grid, tree) being imposed on the network. However, as the mobile hosts move, the system configuration changes also. Besides, system resources have to be rearranged in order to cope with the dynamic changes of system configuration. These kinds of problems are known as the configuration management problems.

Portability

Computers are said to be portable if they are light, small, operational even under adverse situations, and requiring minimal power consumption. All these properties must be considered in the design of mobile hosts.

Battery used by the mobile hosts contributes a large portion of the weight. However, the life of battery used by mobile hosts currently is very limited. It is foreseen that the battery capacity will not increase much in the near future. With the demand of faster processors, the power constraint is further worsened.

With the size constraint on the mobile hosts, smaller user interfaces are required. Buttons, mouse, or even windowing environments may have to be sacrificed in order to save space.

With the limited physical size and power supply, the possible storage space for a mobile host becomes limited. The large power consumption of nonvolatile storage further reduces the amount of storage space allowed.

1.2 Applications of Transaction Replication in Mobile Environments

The first question that should be asked when a research is to be carried out is how it would be used. One application of implementing transaction management in mobile environments is providing information services in the air. Users carrying mobile hosts can connect to the network and query for the required online information through their terminals. The information ranges from electronic mails and news to yellow pages extended with online information, such as the movies showing in the cinemas [19]. This is also one of the core applications in mobile computing being predicted.

Transaction management in mobile environments can also find its application in those systems where the update rate to the database is not high. As an example, consider an airline database system. It is believed that the update rate to the database is maintained at a moderate level. At present, seat reservation can only be done through the hosts dedicated to it. If the system is implemented in the mobile environments, customers can try to make their seat reservations through their own mobile hosts. They may be able to examine the schedules and seat plan in order to make their purchase decisions.

1.3 Motivation for Transaction Replication in Mobile Environments

A scheme for replicated distributed database management, known as the *transaction replication scheme*, is proposed which is aimed at reducing the communication overhead in transaction management of distributed databases [14]. In this scheme, data are divided into two types: shared-private data and public data. Each shared-private data is owned by a particular host and can only be updated by transactions submitted at that host, transactions submitted from other hosts can only read the data. Public data can be read and modified by transactions submitted from every host.

Transactions are divided into two types, known as the local transactions and public transactions. Local transactions access only the shared-private data owned by the submission hosts, whereas public transactions may read public or shared-private data, and write only public data. The scheme reduces communication overhead in transaction execution by transaction replication. Public transactions are collected and broadcasted together with the latest committed values of the shared-private data to other hosts periodically. They are then executed without the need to communicate with other hosts before they tentatively commit.

It is this broadcasting nature of the transaction replication scheme that motivates the study of the scheme in the mobile computing environments. Issues about the efficient access to data by a large number of mobile users have been discussed extensively in [20, 30]. There are two fundamental methods: either by data broadcasting or on demand. For data broadcasting, it refers to the act of broadcasting data through the wireless channels periodically and requiring the mobile hosts to listen to it. Whereas the latter one requires the mobile hosts to uplink the requests and the fixed hosts respond by sending the required data back.

It is believed that wireless broadcasting may be the better way to propagate data objects that are requested frequently to the large population of mobile users [18, 20]. Broadcasting can increase the number of queries that can be handled per unit time under a fixed wireless bandwidth. Besides, the cost of sending a batch of data is lower than sending the same number of data individually. Therefore a scheme with a broadcasting nature is more appropriate to be used in the mobile computing environments. It is believed that the transaction replication scheme can be applied with only minor changes needed.

1.4 Major Simulation Results

In this thesis, a scheme is designed and simulation is carried out to measure its performance. It is found from the simulation of our scheme that the performance of transactions submitted at the mobile hosts is indeed quite good.

For those transactions with write operations, it is found that the commit ratio depends largely on the reliability of the wireless channels. Besides, it is found that with the use of transaction replication scheme, conflicts among data will have little effect on the performance of these transactions.

Moreover, simulation of our scheme shows that collecting the requests for missed data and sending them in batches can greatly improve the commit ratio of transactions executed at the mobile hosts under unreliable wireless channels, though it may increase the response time of transactions. Also, if the access pattern of mobile hosts is known in advance, the cache hit ratio as well as the response time of transactions can be improved. It is verified also through simulation that updating the cached data periodically provides a better performance of transactions than removing the invalid data from the cache periodically.

Concerning about the difference in performance between our scheme and a lock-based scheme, it is found that under most circumstances, our scheme outperforms the lock-based scheme. It is only when the database size is very large and the conflict rate is very low that the performance of the lock-based scheme becomes comparable with ours. Also, if the transactions from mobile hosts contain read operations only, then the lock-based scheme can have a better performance than ours in a system with large database.

1.5 Roadmap to the Thesis

The remainder of the thesis is organized as follows. Chapter 2 summarizes previous and related work on mobile computing. Chapter 3 presents the system model and assumptions under which the scheme is proposed. Chapter 4 describes the scheme for transaction management in mobile computing systems, and gives a correctness proof. Chapter 5 outlines the simulation created to conduct the research presented. Chapter 6 gives the simulation results on the proposed scheme and the comparison of it with a lock-based scheme. Chapter 7 presents the conclusions and suggests the work to be done for future research.

Chapter 2

Previous and Related Research

Various proposals have been made in the past several years concerning different problems of mobile computing. In this chapter, summaries of some previous related work in several areas of mobile computing are given. In particular, proposals concerning the database and file systems issues are discussed. The summaries are grouped according to the problems they investigated. Each summary or group of summaries is cited at an appropriate point and is followed by a short discussion.

2.1 File Systems

Replication can be used to improve the availability of data in the file systems. However, with the introduction of mobile clients, the scheme for file replica management has to be redesigned under the new mobile environment. Besides, disconnections are very frequent in the mobile environments. File systems should be designed in order to support disconnected operations. Proposals discussed below take into account some of these issues.

2.1.1 Management of Replicated Files

Tait and Duchamp [36] proposed an algorithm for managing replicas of a replicated file system with a new file system service interface to reduce latency induced by the mobility of mobile clients. The algorithm is proposed with the assumption that file access consists of sequential write-read sharing, with little concurrent sharing.

Latency is reduced by the use of the primary-secondary replication scheme in which a mobile client communicates with its primary replica only. The primary replica then communicates asynchronously with the secondary replicas. The primary replica is chosen to be the one that is nearest to the mobile client. When the mobile client moves around, it chooses some nearby replica to become its new primary replica and possibly drops another replica.

Mobile clients read files from their corresponding primary replicas. Updated files by the mobile clients are periodically 'pickup' by the primary replicas. They are then further propagated to the secondary replicas. A mobile client can purge the updated files only after enough secondary replicas have recorded the updates. In some cases, a mobile client can require for the pickup as a result of cache full.

A new read interface, known as the loose read, is defined. The traditional read, now called strict read, tries to access all the servers and possibly some of the clients if they are known to have strict read the same file before, to retrieve the latest copy. On the contrary, loose read accesses the copy with lowest cost, but with no guarantee to the value returned. Each call of loose read results in the following steps being performed:

1. The client's cache is checked.

2. If no copy of the file is found, the primary server is checked.

3. If none of the above contains the copy, then the secondary servers are checked.

In order to make strict read as efficient as loose read, the currency token is introduced. A mobile client that is shown to be the only writer of the file can obtain the currency token in response to the strict read of the file. A mobile client that succeeded in obtaining the currency token is guaranteed to find the file with the same sequence of steps followed by loose read.

2.1.2 Disconnected Operations

Satyanarayanan and Kistler proposed the Coda file system to support disconnected operations [24, 34]. Similar to the proposal made in [36], Coda is designed for applications with low degree of write sharing and that are not highly concurrent. The system under which Coda is designed consists of a large number of clients connected to the servers through a high bandwidth network. These clients may disconnect from the network owing to the failure of the network or the detachment of portable clients.

In Coda, two mechanisms are employed in order to achieve high availability of data. They are server replication and disconnected operations. For both of these mechanisms, optimistic replica control strategy is applied in order to provide the highest possible availability of data.

Data are replicated at more than one of the servers in Coda. The cost of replication is kept low by caching data at the clients. However, data cached at the clients have to be updated periodically with the replicas at the servers in order to make them useful.

When a client is disconnected from the network for whatever reasons, all the file system requests are serviced by the contents of its cache. In order to support disconnected operations, three states (hoarding, emulation and reintegration) are defined in which the cache manager of the client, called Venus, operates. Venus operates in the hoarding state if the client is connected to the network and relies on the server replication for data. In this state, Venus tries to store useful data for use during disconnection. Besides, it tries to balance the needs of disconnected and connected operations. This is done by periodically comparing the priorities of different data objects which is calculated by the reference history and user profiles.

Upon disconnection, Venus enters and remains in the emulation state until reconnection. During this state, Venus acts as the pseudo-server to take over many actions performed by the servers. Cache management during this state is similar to the hoarding state. One of the differences is that updated data are assigned infinite priority so as to prevent them from being purged before reconnection. Each update activity is recorded in

the log so that it can be replayed when the client reconnects to the network. However, log space may eventually be used up. In this case, no further update activity is allowed before reconnection.

When the client reconnects to the network physically, Venus changes its state to reintegration. In this state, changes made during emulation are propagated back to the servers and the cache of client is updated in order to reflect the current server state. During this period, all update activities are suspended until reintegration completes. Venus then changes back to the hoarding state.

If reintegration fails owing to write/write conflicts of files, the log records are written to a replay file and is repaired by the users later. Log records in the cache are then purged in order that subsequent references will be refetched from the servers.

Huston and Honeyman proposed another file system similar to Coda, and is known as AFS [17]. AFS differs from Coda in that Coda has the additional support of server replication to achieve high availability. Also, for Coda, the log is propagated to all the accessible servers and is executed there; whereas for AFS, log records are executed at the clients.

The proposals discussed above deal with the problem of consistency for a single file but do not provide consistency across a group of files. Ahamad and Smith [2] proposed another correctness criterion for detecting consistency among a group of files, which is appropriate for systems supporting disconnected operations. The technique is based on the causal ordering among operations executed and is known as the causal consistency.

For two operations o_1 and o_2 executed on a set of objects, o_1 is said to causally precedes o_2 if one of the following conditions holds:

- o_1 and o_2 are executed in the same process and o_1 is executed before o_2;
- o_2 reads an object written by the o_1;
- if there exists another operation o_3 such that o_1 causally precedes o_3 and o_3 causally precedes o_2, then o_1 causally precedes o_2;

If o_1 and o_2 are not causally ordered, then they are said to be concurrent.

The cached copies of two files f_1 and f_2 are said to be causal consistent either if the operations that created the two cached copies are concurrent or there does not exist another write operation that have created another version for either f_1 or f_2 and the version is not present in the cached copy. The set of objects are said to be causal consistent if each pair of them is causal consistent. It is shown that causal consistency is weaker than one-copy serializability used commonly in the database community.

All the proposals discussed above take into account the problem of file consistency. The correctness criteria employed by the proposals, causal consistency in [2] or one-copy UNIX serializability in [36], for example, are all weaker than one-copy serializability. Besides, they have no concept of transactions. Also, most of the proposals above make the assumption of low concurrent sharing of file accesses. For the Coda clients, they are connected to the servers through high bandwidth channels, which contradicts to the assumptions made to the wireless networks.

2.2 Database Management

Several problems have been studied about the database management issue. They include transaction management, caching and cache invalidation, data replication, etc. Some of the proposals concerning the problems stated above are discussed.

2.2.1 Data Replication Schemes

Data replication is used to reduce the communication cost as well as to increase the availability of data in a distributed system. However, static data replication schemes cannot be applied in the mobile environments owing to the mobile nature of mobile hosts. It is believed that dynamic replication schemes are more suitable to be applied in the mobile computing systems. A preliminary work about this issue has been reported in [5]. Another proposal can be found in [16].

The proposal made by Badrinath and Imielinski in [5] discusses various replication schemes under a system with mobile servers and clients. The mobile computing system consists of a number of wireless mobile terminals. These terminals serve to be either servers or clients and are connected to the base stations through the wireless channels. The base stations, in turn, are connected to the location servers by the wired network. Location servers correspond to Mobile Switching Offices and are connected among themselves by the fixed network. There are about 60-100 base stations under a location server.

Six different schemes have been given. For the first three schemes, the server may replicate a copy of the data at the mobile client, the location server of the client or the location server of the mobile server. Writes on the data are then propagated to the corresponding copy. The first case incurs a search cost in locating the mobile client. For the latter cases, writes are on static copies, therefore no search cost is required. However, reading cost by the mobile clients becomes larger.

The remaining three schemes require caching and invalidation. Similar to the first three schemes, the copy can be cached at the location server of the mobile server or client, or at the client itself. These schemes require invalidation messages to be sent. If the cached copy is invalid, the mobile server should then be located for a valid copy.

The schemes are evaluated based on the read and write patterns and the search cost for locating the servers and clients. It is shown in the paper that mobility introduces a search cost which may make it no longer worth placing a replica of some data item to the mobile clients unless the read activity of the mobile clients is high enough to compensate for the search cost.

The proposal provided a preliminary work in this area. It shows how mobility affects the problem of data replication. However, in this proposal the system model is over-simplified and no dynamic replication scheme is provided.

Huang, Sistla and Wolfson [16] proposed several static and dynamic data allocation methods to the mobile hosts. The primary goal of the proposal is to minimize the communication cost in accessing data objects.

The study is based on a system with a single fixed host and a single mobile host. The

allocation method is to decide whether to replicate a particular data in the mobile host or not. Cost analysis is carried out for each of the methods to compare their performance under different situations. The expected cost, average expected cost and the cost in the worst case are calculated. Throughout the analysis of the methods, the cost of read by the fixed host and the cost of write issued by the mobile host are ignored since their costs are fixed for all the methods.

Two schemes for data allocation, known as the one-copy and two-copies schemes, are presented. In the one-copy scheme only the fixed host has the data; whereas in the two-copies scheme, both the fixed host as well as the mobile host have that particular data.

Based on these two schemes, the static and dynamic data allocation methods are presented. Static allocation method refers to the use of the same data allocation scheme throughout the whole execution time. That is, either the one-copy or the two-copies scheme is used throughout the execution time. In contrast, dynamic data allocation method changes the data allocation schemes continuously throughout the execution according to the past k requests. If it is shown that the number of read requests is more than the number of write requests, then the two-copies scheme is applied. Otherwise, the one-copy scheme should be used. The read/write ratio is checked each time a request is received. The dynamic data allocation method is known as the sliding-window algorithm with window size equals k.

The analysis is performed on two models. In the first model, known as the connection cost model, cost is calculated in terms of connection time. For the second model, known as the message cost model, cost is calculated by counting the number of messages passed, including both the data messages and the control messages. In both of the models, it is shown that both the average expected cost and the worst case cost of the dynamic method are better than that of the static methods. However, there is a tradeoff in selecting the value of k since the average expected cost of the dynamic method decreases as the value of k increases, but its cost of the worst case increases with the value of k.

The paper has proposed data allocation methods that are quite close to the optimum.

However, only a single fixed host and a single mobile host is considered throughout the study. It is mentioned in [13] that the solution cannot be extended to a cell with many mobile hosts. The overhead of the algorithm in maintaining the window of requests is not shown. It is believed that the cost may increase sharply as the number of data in the database increases. Besides, the paper has no consideration of transactions.

2.2.2 Cache Invalidation and Query Processing

Several different caching algorithms have been proposed recently for the mobile computing environments [6, 7, 23, 39]. Barbará and Imieliński proposed in [6, 7] three different cache invalidation strategies, known as Broadcasting Timestamps (TS), Amnesic Terminals (AT), and Signatures (SIG). These strategies make use of the periodic broadcast of invalidation reports by the servers in cache invalidation. A stateless server, which refers to a server with no information about the number of mobile clients that are currently in its cell and the data items that a particular client has cached, is used. In these methods, queries submitted at a mobile client are collected and not handled until the next invalidation report is received.

The first two methods are similar. For the TS method, the invalidation report includes the identifiers as well as the timestamps of the latest change for the data items that have changed in the past w seconds. Each mobile client keeps a variable to hold the time when the last invalidation report is received. After a mobile client receives an invalidation report, it checks to see if the last invalidation report is received more than w seconds before. If so, the whole cache is purged; otherwise each data cached is checked against those included in the invalidation report. The data is either purged from the cache if the timestamp of the cached data is smaller than the one in the invalidation report, or the timestamp of the cached data is updated to the time when the report is broadcasted.

For the AT method, the invalidation report includes the identifiers of the data objects that are updated after the past invalidation report is broadcasted. If a mobile client misses the previous invalidation report, the whole cache is purged after the current report

is received. Otherwise, data in the cache are checked individually. Data is purged if it is included in the invalidation report.

For the SIG method, the invalidation report is formed by means of combined signatures that are used for comparing the copies of a file. Signatures are checksums computed over the values of data objects. An invalidation report is formed with a set of combined signatures of data objects. It can then be used to invalidate up to f data objects. If more than f data are updated after the previous report is broadcasted, there may be a high probability of false invalidation, i.e. a data is believed to be invalid although it is valid.

It is found that if the mobile clients disconnect frequently, the signature strategy outperforms the others. However, if the clients rarely disconnect, the AT method is the best one.

Jing, Bukhres, Elmagarmid and Alonso proposed another invalidation-report based algorithm in [23]. In this algorithm, the invalidation report consists of a set of binary bit sequences. Each of the sequences consists of a set of binary bits and a timestamp. Each bit represents a data object in the database. A bit is set to '1' if the corresponding data object has been updated since the time specified by the timestamp of the sequence.

The invalidation report is organized as a hierarchical structure, with the highest rank sequence consisting of a number of binary bits equal the number of data objects in the database. The one following this sequence should have half the bits of the highest one, and so on. Half of the bits in any sequence can be set to 1 to indicate the data that are last updated. The kth bit in the ith sequence corresponds to the kth 1 bit in the $i+1$th sequence. If the highest sequence is reached, then the data represented by the kth 1 bit has been updated.

The algorithm proposed has the advantage of small invalidation report size that is independent of the number of data objects to be invalidated. Besides, the algorithm can lessen the adverse effect caused by false invalidation, which is likely to happen in the SIG method mentioned above.

The caching algorithms mentioned above make use of the cached data to answer queries

collected. However, the proposals do not provide any way of handling the updates generated by the mobile clients and they have not included the concept of transactions. Besides, they do not consider the distributed database protocol to be used in the fixed network. Our scheme tries to deal with these problems and its performance is studied.

2.2.3 Transaction Management in Mobile Environments

Pitoura and Bhargava [31, 32, 33] proposed a scheme to deal with the problem of data consistency with transaction support in mobile distributed environments. In this scheme, some of the fixed hosts are augmented with wireless interfaces to communicate with the mobile hosts and are known as the *Mobile Support Stations* (MSS). The coverage of a MSS's signal is known as its *cell.* Besides, the database is assumed to be distributed among the fixed hosts and the mobile hosts. It is partitioned into a set of clusters. Data stored in the same or neighbouring hosts form a cluster; whereas data stored in disconnected or remote hosts are considered to be in different clusters. Based on this definition, data contained in a MSS and in the mobile hosts residing in the MSS's cell form a cluster, while those residing in another MSS as well as the mobile hosts in its cell form another cluster. Given that the disconnection of mobile hosts is predictable, the configuration of clusters is dynamic in the sense that clusters can be created or merged upon a disconnection or connection of a mobile host.

In order to deal with the problem of data inaccessibility caused by network contention or disconnections, two new types of transactions, known as the weak and strict transactions, are defined. Weak transactions consist of operations accessing data copies that belong to the same cluster. These data are usually not strictly consistent and the operations are known as the weak read and weak write operations. Strict transactions consist of traditional read and write operations, and strict data consistency can be guaranteed. Each transaction submitted from the mobile or fixed host has to be decomposed into a number of weak and strict subtransactions.

A weak read operation on data x reads a locally available copy of x, with the value of

x written by the last weak or strict write operation in the cluster. A weak write operation on x writes a local copy of x. A weak transaction has to commit locally at the cluster and globally after cluster merging. Data updated by a weak transaction become permanent only when it is committed globally. For example, a weak transaction may be executed at a mobile host that is disconnected from the network. The operations are therefore served by the data available locally in the mobile host. When the mobile host reconnects to the network later (cluster merging), the transaction has to commit globally in order to have its updated data become permanent.

For a strict read operation, it reads the value that is written by the last strict write operation. This strict write operation is required to write several copies of the data.

It can be observed that strict data consistency can be maintained for data located in the same cluster, while different degrees of inconsistency are allowed for data in different clusters.

M-degree consistency is defined to bound the divergence of data versions created by weak transactions from those created by strict transactions. The degree M can be the number of weak writes at each cluster, the number of weak transactions allowed at each cluster, etc.

The scheme allows transactions to be executed locally at the mobile hosts under adverse network conditions by defining a new correctness criterion on the transaction execution. However, some details about the scheme are not mentioned. For example, the algorithm for the global commit of weak transactions upon cluster merging has not been given, and no clear definition for the formation of clusters are provided. Besides, it is believed that a high abort rate of weak transactions upon cluster merging may result owing to the data conflicts. However, no performance analysis has been given concerning this issue.

Jing, Bukhres and Elmagarmid proposed a lock-based scheme for transaction management in mobile computing environments [22]. The scheme allows the read lock and unlock of a data object to be executed at different copies, hence reduces the message cost over the fixed network. Data are replicated in order to reduce the message cost further. An

optimistic read-one-write-all concurrency control approach is used, in which read locks are granted immediately while write locks are deferred until commitment.

It is assumed that a mobile host will move away from a cell only when the results of all the operations submitted from it are received, and that only one transaction may be initialized by a mobile host at any time.

Before the method can be applied, several problems have to be solved. First of all, it is observed in 2-phase locking that a lock has to be held by a transaction until no new lock request is required by it. However, if the read locking and unlocking can be executed at different fixed hosts, this requirement is not guaranteed. This is solved by comparing the version number of the data copies in the local fixed host with the one obtained at lock time. If the two numbers agree, then the serialization order can still be maintained.

Besides, it must be able for an update transaction to determine that a data is not locked by another transaction for reading. This can be done by sending two rounds of messages to all the fixed hosts in order to obtain the information about the status of the data copies as well as granting the write locks. In the first round of message exchange, the transaction coordinator collects the lock/unlock information from all the fixed hosts. If it is shown that each read lock is matched with an unlock for the data to be written, write locks are set and updates are performed in the second round of message exchange. In order to keep the message cost to the minimum, these messages are merged in the 2 phase commit protocol.

The enforcement of write operations is guaranteed by the introduction of a new lock mode, known as the write intend lock (W_INTEND). A W_INTEND lock is not compatible with a write lock or a W_INTEND. Besides, a granted W_INTEND is not compatible with a requested read lock but a requested W_INTEND is compatible with a granted read lock. In the first round of message exchange, W_INTEND is set for a copy if no conflicting lock is granted there; otherwise it is blocked. In the second phase, a W_INTEND is changed to a write lock if the coordinator decides to commit the transaction. In this way, updates can be performed even with the existence of pending read locks.

In order to solve the problem of frequent disconnection of mobile hosts, timeout is

used to abort a transaction submitted by a mobile host that is disconnected for a time-out period. The timeout parameter can be specified by users in order to support their applications effectively even during a period of disconnection.

The primary goal of the proposal is to reduce the message cost in the fixed network. However, the commit ratio of transactions is not considered. It is believed that a high abort rate of transactions may result owing to the high ratio of data conflict. Besides, the utilization of the wireless channels is not studied. Our scheme tries to provide results about these and shows that data conflict has little effect on the performance of transactions.

Chapter 3

System Model and Assumptions

In this chapter, a system model under which the scheme is proposed is given. It is then followed by the correctness criteria as well as the assumptions made on the system.

3.1 System Architecture

One of the major components in the mobile computing system is the mobile hosts. In our system architecture, mobile hosts known as the *walkstations* are supported [19]. These mobile hosts have their own resources and do significant amount of processing. They have their own disks with a limited amount of space (*cache space*) which can cache a portion of the database. The cached data can then be accessed and updated. However, a mobile host may use the resources of a fixed host occasionally owing to its inferiority to the fixed host. In order for the mobile hosts to be able to communicate with the rest of the network, some special purpose machines, known as the *mobile support stations* (MSS) [21], are required. A MSS can be thought of as a fixed host augmented with a wireless interface to communicate with the mobile hosts. It is assumed that MSSs have the same functionalities as fixed hosts.

The *cellular networks with frequency reuse* network topology is used [29], in which the large service area is divided into smaller areas, known as the *cells*. Each cell is covered by the signal of a MSS and is serviced by it. Communications between the MSS and a mobile host is possible only if the mobile host is located physically within the cell and identified itself with it. The mobile host is then said to be *local* to that particular cell.

The cells are allowed to overlap, but a mobile host can be local to at most one MSS at any point of time. A mobile host that is not local to any MSS is said to be *disconnected.* A mobile host may disconnect from the network *voluntarily*, when it physically detaches from the network; or *involuntarily*, when network failure occurs [32].

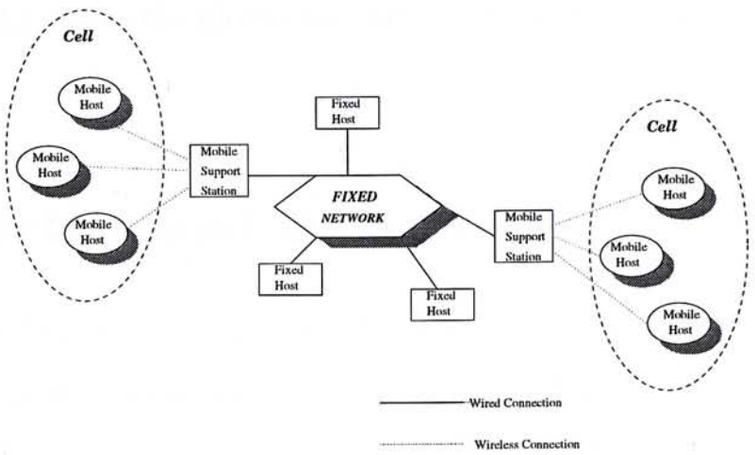


Figure 3.1: The Mobile Computing System Model

Communications to and from a mobile host in a cell should pass through and be controlled by the MSS [29]. A message sent from a mobile host MC_1 to another mobile host MC_2 will first be received by the local MSS of MC_1. This MSS then forwards the message to the local MSS of MC_2 which forwards it to MC_2 over the wireless network. This may incur a large overhead in searching the current MSS that MC_2 is local to owing to the mobility nature of mobile hosts [4].

Different access methods are used for messages sent to and from the mobile hosts through the wireless channels. Broadcasting of messages from the MSSs to the mobile hosts uses the Time Division Multiplexing (TDM) scheme, in which each channel user owns the entire bandwidth for a short period of time periodically [37]. Messages sent by a mobile host to its MSS uses the CSMA/CD method [37]. It is mentioned in [1] that TDM is the general method for multiplexing from the MSS to the mobile hosts. The papers [29, 40] describe that versions of CSMA methods are the most popular multiple access methods in mobile computing systems.

The whole picture of a mobile computing system therefore consists of a set of reliable and powerful fixed hosts, with some of them act as the mobile support stations, and a

larger number of mobile hosts. The set of fixed hosts together with the large bandwidth communication paths among them form the *fixed* network. The fixed network has a number of low-bandwidth *wireless* networks connected to it. Each of these wireless networks consists of a MSS and a number of mobile hosts that are physically located within the MSS's cell. Figure 3.1 shows the global architecture to support mobile wireless computing [19].

3.2 Transaction and Data Model

The database is a collection of n named data objects. It contains information about one particular enterprise. It is assumed that the database is fully replicated in all the fixed hosts, although the assumption can be relaxed. Each of these fixed hosts is said to be holding a *copy* of the data.

In our system, user jobs are carried out in the form of *transactions.* A transaction is a program unit that accesses and possibly updates various data objects in the database [26]. It consists of a set of read and write operations, as well as either a *commit* or an *abort* operation at the end to indicate whether the transaction completes its execution successfully. Each data accessed by a transaction is read exactly once by the transaction and written at most once if the data is updated. When a transaction is first sent to a host, the transaction is said to originate from (be submitted at) this host.

Transaction Replication Scheme (TRS) [14] is applied as the replicated database protocol for the fixed network. As is defined in [14], data objects can be of either one of the following types:

- *shared-private* data owned by fixed host f. Only transactions submitted at f can perform writes, and these transactions access only shared-private data at f.
- *public* data. Transactions submitted at any host can perform reads and writes.

In order to handle two different types of data in the database, two types of transactions are identified:

- *local* transaction. Transaction submitted at a fixed host f accessing shared-private data that are owned by f.

- *public* transaction. Non-local transaction that may read public and/or shared-private data, and that can write only public data.

In TRS, the set of clock values are divided into a number of fixed intervals of $[t_1, t_2)$, where $t_2 - t_1$ is a constant value equal to δ. Each of these intervals is called a *TRS period.* Public transactions submitted at a fixed host during a TRS period are collected. In contrast, local transactions submitted at a fixed host are executed there immediately. Their latest committed values, together with the public transactions collected, are broadcasted at the end of the TRS period. When a fixed host receives all the broadcasting messages from other fixed hosts, it starts executing the public transactions.

An unique timestamp is assigned to each public transaction before it starts execution. The timestamp can either be the value of a global system clock or the value of a global logical counter that is incremented after a new timestamp has been assigned. Suppose that a transaction T_i has been assigned a timestamp t_i at host a. If a new transaction T_j is submitted at host a, then its timestamp, say t_j, should be assigned such that $t_i < t_j$.

The timestamps determine the serialization order (see section 3.3) among public transactions. TRS ensures that the result of transaction execution is equivalent to a serial schedule where local transactions are executed immediately before the public transactions submitted in the same period. This requires a fixed host to remember the values of the shared-private data it owns until the set of public transactions is received. However, before the set of public transactions arrives, some local transactions may have assigned new values to some of the shared-private data. Therefore it is required for each fixed host to keep *several values* of the shared-private data. Each of these data values is tagged with a *version number.* The version number measures how up-to-date each value is. A write operation will give a data value with version number greater than the latest written value of the same data object by one. A data value, together with the version number tagged with it, forms a *version* for a data object. A version of a shared-private data can

be discarded only when the set of public transactions requiring this version is *completed*, i.e. either committed or aborted.

3.3 One-copy Serializability

In a one-copy database, it is expected that the concurrent execution of transactions to be *equivalent* to a serial execution of those transactions. Such an execution is called *serializable.* In order to determine if the concurrent executions of transactions are serializable, the serialization graph (SG) is defined [8]. A serialization graph is a directed graph whose nodes are the transactions and an edge between two nodes exists if some of the operations between the two transactions involved conflict. Given T_i and T_j be two transactions. An edge $T_i \rightarrow T_j$ exists if one of T_i's operations precedes and conflicts with one of T_j's operations in the execution of the transactions.

This correctness criterion is extended to a replicated database as the *one-copy serializability* which requires the interleaved execution of transactions in a replicated database to be equivalent to a serial execution of the transactions in a one-copy database.

In a replicated database, a data object and its copies are called *logical* data object and *physical* data objects respectively [14]. Operations of a transaction submitted specify the accesses of logical data objects, and are known as the *logical operations.* These logical operations, upon execution, are translated by some translation function τ into a set of *physical operations*, i.e. operations accessing physical data objects.

Transaction execution in a replicated database system can be modeled by a *replicated history.* As is mentioned in [8], a replicated history H over a set of transactions $T = \{T_0, \cdots, T_n\}$ is a partially ordered set $L = (\sum(T), <)$ where

1. $H = \tau(\bigcup_{i=0}^{n} T_i)$, where τ is the translation function for T_i;

2. there exist two dummy transactions T_b and T_f such that T_b is translated into a set of physical write operations for each copy of each data object and T_f is translated into a set of physical read operations for each copy of each data object. Operations

in T_b precede all other physical operations and operations in T_f are preceded by all other physical operations;

3. for each T_i and any two operations p_i and q_i in T_i, if $p_i <_i q_i$, then every operation in $\tau(p_i)$ is related by $<$ to every operation in $\tau(q_i)$;

4. all pairs of conflicting physical operations are related by $<$, where two physical operations conflict if they operate on the same physical copy of a data object and at least one of them is a write;

A committed transaction T_j *reads-x-from* another transaction T_i in a replicated history if there exists some copy x_a such that

1. $w_i[x_a]$ is a physical write operation in T_i and $r_j[x_a]$ is a physical read operation in T_j;

2. $w_i[x_a] < r_j[x_a]$;

3. there is no physical write operation $w_k[x_a]$ by T_k such that $w_i[x_a] < w_k[x_a] < r_j[x_a]$.

A history H is one-copy serial if H consists only of logical operations and for any two transactions T_i, T_j that appear in H, either all operations in T_i appear before all operations in T_j or vice versa. A history H_1 is equivalent to another history H_2 if both H_1 and H_2 have the same reads-from relationships. A replicated history is one-copy serializable if it is equivalent to a one-copy serial history over the same set of transactions.

In order to determine if a replicated history is one-copy serializable, a replicated data serialization graph (RDSG) [8] is used. A RDSG for a replicated history H is a SG with enough edges added such that the following conditions hold.

Suppose that T_i, T_j and T_k are transactions, for all data items x,

1. if T_i and T_k write x, then either a path exists from T_i to T_k or vice versa;

2. if T_j reads-x-from T_i, T_k writes some copy of x, and a path exists from T_i to T_k, then a path exists from T_j to T_k.

A graph satisfying condition (1)/(2) is said to induce a write/read order for H. If H has an acyclic RDSG, then H is one-copy serializable [8].

Consider an example as shown in [8]. In this example, a database with data objects x and y with copies x_a, x_b, y_c and y_d is given. Figure 3.2 shows a history H with three transactions T_0, T_1 and T_2. Note that $x_a\sqcap$ denotes the failure of the copy x_a.

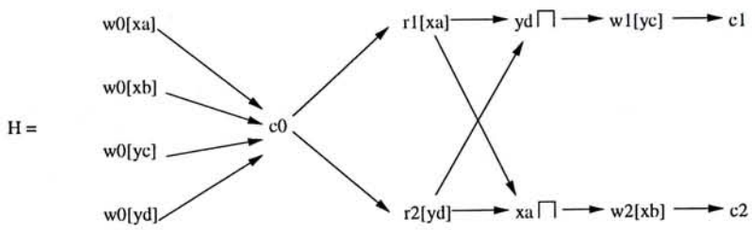


Figure 3.2: A History for the Execution of Transactions

It is found that H is not one-copy serializable. However, the serialization graph of the history as shown in Figure 3.3 does not contain any cycle.

Figure 3.3: The Serialization Graph for H

Figure 3.4 shows the RDSG of H. Since T_1 reads-x-from T_0, T_2 writes x, and $T_0 \rightarrow T_2$, therefore $T_1 \rightarrow T_2$ appears in the graph. It is found that the graph has a cycle, which means that the history is not one-copy serializable.

Figure 3.4: The RDSG for H

3.4 Assumptions

Before going on to present our scheme, the following assumptions are made for the data and network model:

- assumptions about the network:

- each fixed host and mobile host has an unique ID;
- a single channel exists between the mobile hosts and their local MSS through which messages sent are received in the *first in first out* order;
- the underlying network protocol does not handle message loss in the wireless channels; the scheme proposed should therefore take into account the problems incurred by it;
- all the fixed hosts act as mobile support stations and the terms MSS and fixed host are used interchangeably;
- no bound is placed on the time interval between a mobile host disconnects from the network and reconnects to it and on the time delay in sending a message through the wireless channels;
- the fixed network is synchronized to within δ, where δ is sufficiently small. That is, the difference of clock reading between any two non-faulty fixed hosts' clocks is smaller than δ. It is assumed also that $\delta << $ TRS period;

• assumptions about data and transactions:

- mobile hosts do not have any shared-private data of their own;
- each public transaction pre-declares supersets of its *readset* and *writeset*, which are supersets of data objects that the transaction reads and writes respectively;
- transactions submitted at the mobile hosts are not interactive;
- at most one version of a data object can be cached in the mobile hosts at any point of time;
- the ratio of R/W transactions (transactions containing write operations) submitted at the mobile hosts is low;

Note that some proposals have made the assumption that the underlying network protocol handles message loss in the wireless channels. For example, the proposal [22] assumed the network handles message loss by retransmission, or the proposal does not work.

Chapter 4

Transaction Replication in a Mobile Environment

In this chapter, a scheme for transaction management in a mobile computing environment is proposed. We focus on the basic scheme first. The extension of the scheme to deal with the problem of partition failure in the fixed network is discussed later.

Transactions can be submitted at either the fixed hosts or mobile hosts. However, since the mobile hosts do not have any shared-private data of their own, no local transaction is submitted there.

Local transactions and public transactions submitted at the fixed hosts during a TRS period $[t_1, t_2)$ are executed at different times. Local transactions submitted at a fixed host s are executed there immediately according to some local concurrency control mechanism. At the end of the TRS period, their latest committed values, together with the batch of public transactions accumulated locally at s are broadcasted to all the other fixed hosts. This batch of public transactions is called the *local batch* of s at t_2. The set of all public transactions collected in $[t_1, t_2)$ is called the *global batch* at t_2.

For each fixed host s, s *implicitly broadcasts* at t_2 the latest committed version for each shared-private data owned by it. If a data object is not updated during the period $[t_1, t_2)$, then its current committed value is implicitly broadcasted, though no version is physically broadcasted.

After the global batch is received, the new versions of the shared-private data sent from other fixed hosts, if any, are written to the local copies. Public transactions in the

global batch are then executed according to the timestamp protocol. A public transaction, if it has to read some shared-private data, should read the version that is broadcasted at t_2. This requires each fixed host to keep multiple versions of the shared-private data it owns. After the execution of the global batch is completed, this version of shared-private data is discarded.

For the public transactions submitted at the mobile hosts, they are further divided into two types, known as the *read-only* public transactions and *R/W* public transactions. Read-only public transactions consist of read operations only, while R/W public transactions contain read and write operations. For the read-only public transactions submitted at the mobile hosts, since they involve no change to the database, they are collected in batches and executed locally at the mobile hosts in order to reduce the response time. In order for these transactions to be serialized with other transactions, the cached data are required to be *updated* before the transactions are executed. The method for cache update and execution of read-only transactions are described in section 4.1.

On the other hand, owing to data conflicts, it is believed that poor performance may result if the R/W transactions are executed at the mobile hosts. They are therefore sent to the fixed network for execution under TRS, where data conflicts will have little effect. Results of transactions, either committed or aborted, are broadcasted back to the mobile hosts. The method for transaction execution at the fixed network and the broadcasting of results are described in section 4.2.

A correctness proof of the scheme is given in section 4.3. The extension of the scheme to support partitioning failures in the fixed network is described in section 4.4.

4.1 Read-only Public Transactions

In order for the read-only public transactions to be executed locally at the mobile hosts, some of the data objects should be cached in the mobile hosts in advance. This is achieved by the broadcasting of data from the MSSs to the mobile hosts. However, data cached may become invalid some time later and some mechanism is needed to inform the mobile

hosts about the changes of the cached data.

4.1.1 Data Broadcasting

In order to execute public transactions locally at the mobile hosts, both the shared-private and public data should be broadcasted to the mobile hosts periodically. Broadcasting the latest committed values of shared-private data at the end of each TRS period as is used in the fixed network no longer works here. A new mechanism has to be designed.

Each fixed host keeps multiple versions of each shared-private data it owns. In order to have the transactions executed at the mobile hosts serializable with other transactions, a correct version of the shared-private data should be broadcasted to the mobile hosts. Assume by now that a global batch is collected in each TRS period. The case that a global batch is absent for a particular period is discussed later.

Suppose that two global batches T_g and T_{g+1} are collected in two consecutive TRS periods $[t_1, t_2)$ and $[t_2, t_3)$ respectively. A transaction in T_g writes a public data p_i and a shared-private data s_i is broadcasted with T_{g+1} to other fixed hosts. Our scheme ensures that the read-only transactions executed at the mobile hosts, accessing the public data versions created by T_g, can be serialized immediately before T_{g+1}. Based on this scheme, if a read-only transaction executed at a mobile host requires to access the data objects p_i and s_i, and it reads the latest committed value of p_i in T_g, it should read the version of s_i broadcasted with T_{g+1}.

There can be another choice of choosing the version of s_i broadcasted with T_g such that the serializability of transactions execution can be maintained. However, this version of s_i may be discarded when the global batch T_g completes its execution.

In order to have such a serial order, data are broadcasted to the mobile hosts when a global batch completes its execution. A variable, known as **MSS_time**, is defined to keep track of the completion of global batches.

Definition 4.1 A variable **MSS_time** is kept in each fixed host. It's value shows the broadcast time of the previous global batch that has completed its execution.

Suppose that a global batch broadcast at t_i completes its execution (either committed or aborted) during the period $[t_m, t_{m+1})$, the value of MSS_time should then be changed by each fixed host to t_i by the time the execution of the global batch completes.

Public data can be broadcasted to the mobile hosts once a global batch completes its execution. However, problem may occur if the global batch completes its execution before the next global batch is received.

Consider two global batches T_g and T_{g+1} collected in two consecutive TRS periods $[t_1, t_2)$ and $[t_2, t_3)$ respectively. If T_g completes its execution before T_{g+1} is received, and data are broadcasted immediately when T_g completes its execution, then the set of data broadcasted will not contain the versions of shared-private data included in the message sent with T_{g+1}. This may result in a non-serializable history.

To solve the problem, data are required to be broadcasted to the mobile hosts at the end of the TRS period when the value of MSS_time is changed and the global batch following the latest completed global batch is received. If such a global batch is not received by the end of the period, data are broadcasted when the global batch is received later.

It is assumed in the above that a global batch is collected during each TRS period. Now we show how the assumption can be relaxed. Consider the case of three consecutive TRS periods $[t_1, t_2)$, $[t_2, t_3)$, $[t_3, t_4)$, with global batches collected at t_2 and t_4 only. It is shown below the problem and the way it is solved.

By the definition of MSS_time, its value should not be changed after the execution of the global batch at t_2 completes and before the execution of the global batch at t_4 completes. The versions of shared-private data created in $[t_3, t_4)$ are therefore not broadcasted to the mobile hosts. On the other hand, during the execution of transactions at the mobile hosts, some of the transactions may require the version of some shared-private data created in $[t_2, t_3)$. However, this version may be discarded when the global batch at t_4 starts to execute. This may result in a non-serializable history, especially when the shared-private data is missed in the cache (see section 4.1.3 for the way to handle cache miss). This is solved by adding a special public transaction ρ in each period. ρ is

assumed to take infinitely short time to complete. With ρ is included, a global batch will be collected during each TRS period and the value of MSS_time will be changed accordingly. The versions of shared-private data created in each TRS period will therefore be broadcasted to the mobile hosts eventually.

Definition 4.2 An **update notification** U for TRS period $[t_i, t_{i+1})$, which contains all the data to be broadcasted to the mobile hosts, is defined. It contains two timestamps t_c and t_p, with the value of t_c equals the current value of MSS_time and the value of t_p equals that of t_c in the previous update notification. The following data versions are also included:

$\{[i, value] | value$ is the latest committed value of shared-private data i between t_{p+1} and $t_{c+1} \vee$ $value$ is the latest committed value of public data i in the period $[t_i, t_{i+1})\}$

where $[t_p, t_{p+1})$ and $[t_c, t_{c+1})$ are TRS periods.

An update notification is broadcasted by each MSS to the mobile hosts at the end of the TRS period during which the value of MSS_time is changed, i.e. there exists at least one global batch that has completed its execution in that period. If the global batch at t_{c+1} is not received by the end of the period, the broadcasting is delayed until the global batch is received by all the MSSs.

Note that the update notification contains all the data that are updated after the previous update notification is broadcasted. This can be used to update the cache of the mobile hosts.

4.1.2 Cache Update

When a mobile host receives an update notification broadcasted from its local MSS, its cache has to be updated before transactions can be executed there. This is done by comparing the cache contents against the update notification, and changing the values of the cached data that exist in the update notification.

In order to keep track of the update notification received by each mobile host, a variable known as **Mobile_time** is defined.

Definition 4.3 A variable **Mobile_time** is kept in each mobile host. Its value is changed to that of the timestamp t_c each time an update notification is received. The variable is used in the update of data objects cached in the mobile host when the next update notification arrives.

When a mobile host receives an update notification, the value of Mobile_time is checked against the timestamps of the update notification (t_c and t_p). The relationship between the values of Mobile_time and the timestamps determines the fate of the cached data.

If the value of t_c is smaller than or equal to that of Mobile_time, the mobile host should have received the same update notification before. This may occur if a mobile host moves to a cell that is connected through a congested wireless channel. An update notification broadcasted through this channel may arrive at the mobile hosts later than the one broadcasted in other cells. Since there is no bound on the message delay through the wireless channels, an update notification may reach a mobile host after the mobile host has received an update notification with the same timestamp from other cells, i.e. $t_c \leq$ Mobile_time.

Since the cached data of the mobile host has been updated with the same set of data versions, the mobile host ignores this message.

Otherwise, i.e. $t_c >$ Mobile_time, the fate of the cached data is determined by the value of t_p in the update notification. If the value of t_p is larger than that of Mobile_time, the mobile host should have been disconnected from the network when the previous update notification was broadcasted. Some of the data in the cache may be invalid at the time the update notification arrives. Therefore all the data in the cache are purged. The cache is then replaced with data objects that appear both in the readset of any of the read-only public transactions collected and the update notification.

On the other hand, if the value of t_p equals that of Mobile_time, the mobile host should have received the previous update notification. In this case, each data in the cache is checked to see if it is updated by the transactions executed at the fixed hosts after the previous update notification was sent. The victims are then purged. Those data that exist

both in the update notification and the readset of any of the read-only public transactions collected are cached, possibly with the removal of data that are least recently used (LRU).

Update of Mobile Computer's Cache

(Let U be the update notification received)

```
Case One:
(t_c ≤ Mobile_time)
   // do nothing
Case Two:
(t_c > Mobile_time) && (t_p > Mobile_time) {
   drop entire cache;
   for each data object j ∈ U {
      if ((j ∈ readset of a transaction) && (cache not full))
         cache j;
   }
}
Case Three:
(t_c > Mobile_time) && (t_p == Mobile_time) { // received previous notification
   for each data object j ∈ cache {
      if (j ∈ U)
         purge j;
   }
   for each data object j ∈ U {
      if (j ∈ readset of a transaction) {
         if (cache not full)
            cache j;
         else
            purge the least recently used data and replace with j;
      }
   }
}
```

Figure 4.1: Update of Mobile Computer's Cache

Note that given the value of $t_c >$ Mobile_time, the value of t_p should not be smaller than that of Mobile_time.

Figure 4.1 shows the algorithm executed by each of the mobile hosts to update its cache upon receipt of the update notification.

After the cache is updated, the value of Mobile_time is changed to that of t_c. The

mobile host can then execute the read-only transactions using the cached data. The scheme for the execution of read-only public transactions at the mobile hosts is discussed in section 4.1.4.

4.1.3 Cache Miss

Owing to the limited cache size of the mobile hosts, a transaction may fail to find the data it requires locally in the cache (cache miss). It is believed that sending the requests for the missed data by each mobile host and requiring the fixed hosts to make replies to each of the requests individually may not be suitable for the mobile computing environments with unreliable wireless channels. Besides, it may be a waste of the wireless bandwidth.

After an update notification has been received by the mobile hosts and their cache contents are updated, each of them checks for the read-sets of the read-only public transactions collected against the cached data.

Definition 4.4 The set of data names appears in anyone of the read-sets of the public transactions collected but absent in the cache of a mobile host is known as the **miss-set** of the mobile host.

A message including the miss-set and Mobile_time of the mobile host is sent to the local MSS after the cache update.

A MSS, after broadcasting the update notification, waits for a pre-defined period of time, known as the **collection period**, to collect the miss-sets from the mobile hosts. Mobile_time of each request message is checked to see if its value agrees with that of MSS_time. If so, the mobile host that sent the request message should have received the latest update notification broadcasted by the MSS, and there does not exist any global batch completed after the broadcasting of the update notification. Data names in the message are then stored by the MSS. If there does not exist a global batch which has completed its execution during the collection period (MSS_time is not changed after the broadcasting of update notification), versions of data stored are broadcasted to the

mobile hosts together with its MSS_time at the end of the collection period. The set of data versions broadcasted are selected according to the followings:

- If it is a shared-private data, then the version selected is the one that is accessed by the public transactions being executed in the fixed network; otherwise
- it is a public data and the version written by the latest committed global batch is selected.

A mobile host, upon receiving the reply from the MSS about the miss-set, checks the value of Mobile_time against that of MSS_time in the reply message. If two values agree, data versions in the message are cached.

There may exist the case that the reply of miss-set does not contain all the data required by a mobile host. This happens if the miss-set sent by a mobile host is interfered by some noise in the wireless channel and cannot be received by the MSS. In this case, the mobile host sends a request message including the data name of the data missed as well as Mobile_time to the local MSS when a transaction has to read it. When the MSS receives the request message, it checks to see if the value of MSS_time and that of Mobile_time in the request message agree. If so, a data version that is chosen by the same method above is sent back to the mobile host.

4.1.4 Execution of Read-only Public Transactions

Read-only public transactions submitted at a mobile host are collected. They are then executed in batch after an update notification is received by the mobile host and the cached data are updated.

For each transaction, those read operations that involve accesses to data objects found in cache are executed first. Others that have to access data missed in the cache are forced to wait until the reply from the MSS concerning the miss-set is received.

If a mobile host fails to receive the reply of miss-set for whatever reasons, in order not to abuse the narrow wireless bandwidth, transactions that are not committed have to

abort. Otherwise, the transactions continue to execute and any further miss of data will cause an individual request message being sent to the MSS for the data version. Section 4.1.3 has discussed how cache miss is handled.

```
send the miss-set to the MSS;
for each transaction tr_x ∈ PU_i {
    for every READ(j) operation ∈ tr_x {
      if ((j in cache) && (no update notification received))
        read j;
      else if (a new update notification received) {
        abort tr_x;
        break; }
    }
    if (all READ(j) processed)
      commit tr_x;
    else if (tr_x not aborted) {
      wait for reply from MSS about the miss-set;
      if (reply received)
        for every READ(j) operation not processed {
          if ((j in cache) && (no update notification received))
            read j;
          else if (a new update notification received)
            break;
          else { // cache miss
            send request to MSS for j;
            if (reply received)
              read j;
            else // timeout in waiting for reply
              break;
          }
        }
      if (all READ(j) processed)
        commit tr_x;
      else
        abort tr_x;
    }
}
```

Figure 4.2: Execution of Read-only Public Transactions at M_i

If a new update notification is received while the transactions are executing, in order to maintain the serialization order, all those transactions should be aborted.

Consider a mobile host M_i. Let PU_i be the batch of read-only public transactions

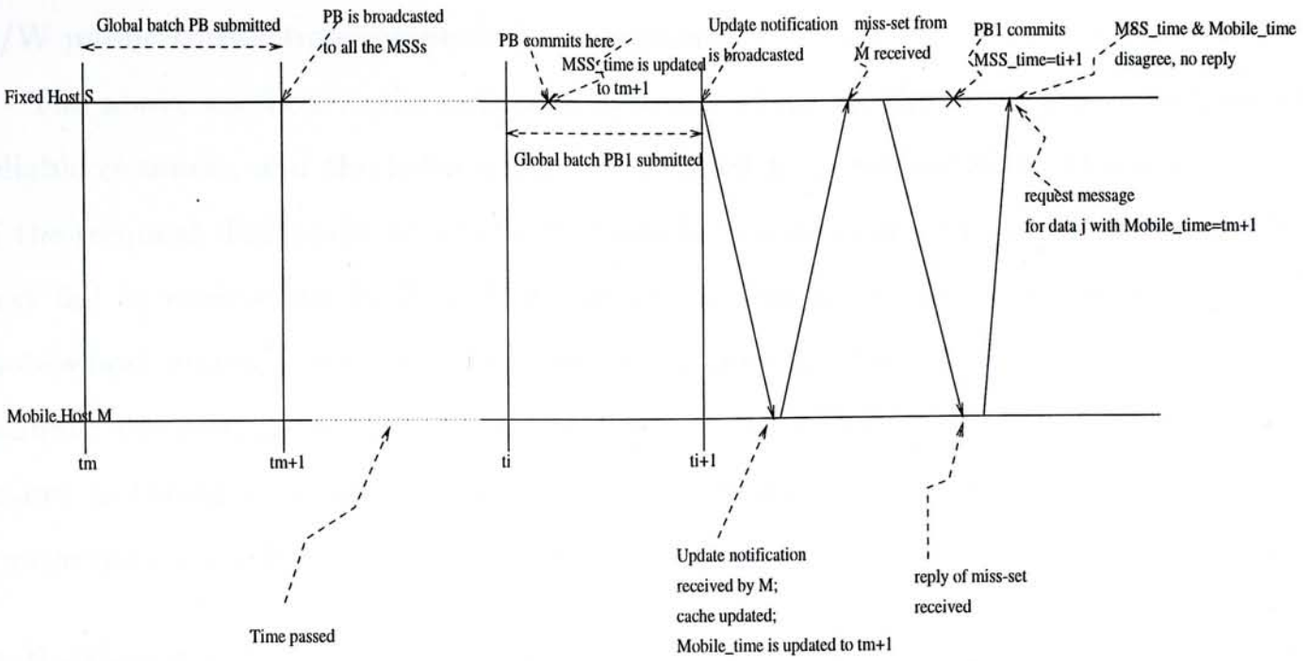


Figure 4.3: An Execution of the Proposed Scheme

collected at M_i. Figure 4.2 shows the algorithm for the execution of read-only public transactions in M_i after the cache is updated. Note that although the algorithm shows a serial execution of the transactions, they are in fact executed concurrently.

Figure 4.3 shows a scenario of the scheme where the mobile host M fails to get the value of the missed data j from the MSS owing to the commitment of a newer global batch PB_1. In the figure, vertical direction represents the wireless channel, and horizontal direction represents time, with later time positioned at the right of the earlier one. Note that the TRS periods are not drawn to scale.

4.2 R/W Public Transactions

R/W public transactions submitted at the mobile hosts are executed in the fixed hosts. When a R/W public transaction is submitted at a mobile host, it is sent to the local MSS s immediately and is included in the local batch of s. The local batch is then broadcasted to other fixed hosts at the end of the current TRS period and is executed there. Result of the R/W public transaction, either commit or abort, is broadcasted with the update notification by each MSS at the end of the period when the global batch containing the

R/W public transaction completes its execution.

The above method works well with systems where all the hosts are connected with reliable channels, and the hosts seldom disconnect from the network. However, because of the frequent disconnection of mobile hosts in mobile computing environments, a MSS may fail to receive the R/W public transaction sent by a particular mobile host, or a mobile host may fail to receive the result of a transaction that it sent previously. In order to make the method suitable for the mobile computing environments, a set of variables, known as **latest_committed**, are defined to keep track of the results of the R/W public transactions received by the mobile hosts.

Definition 4.5 A variable $latest\_committed_i$ is defined for each mobile host i. It keeps the largest timestamp among those R/W public transactions sent by i that have their results received by i. Each $latest\_committed_i$ is maintained by and replicated among all the MSSs.

For each mobile host i, each MSS keeps a list of R/W public transactions sent by i. At the end of each TRS period, the set of R/W public transactions received by each MSS are broadcasted to all the other MSSs in the local batches. The value for anyone of $latest\_committed_i$ is broadcasted also if its value is changed. The value of $latest\_committed_i$ is then written to the copies in other MSSs if its value is greater than the copy there. When a global batch completes its execution during a TRS period, data have to be broadcasted to the mobile hosts at the end of that period. Each MSS finds out all the R/W public transactions in the lists with timestamps greater than the corresponding values of $latest\_committed_i$ and are either committed or aborted. Results of the transactions, together with their timestamps, are broadcasted to the mobile hosts with the update notification.

When a mobile host i receives an update notification, R/W public transactions submitted by it with results in the update notification have their commit/abort realized. However, there may be some transactions with timestamps smaller than some of the

committed/aborted transactions but do not have their results included in the update notification. This occurs if the transactions cannot be received by the MSSs for whatever reasons. In this case the transactions are considered to be aborted.

After the checking is completed, the mobile host i sends an acknowledgement to the local MSS which includes the largest timestamp of those transactions that are realized to be either committed or aborted. The value of the timestamp will then be written to the copy of the variable $latest\_committed_i$ in the MSS if its value is greater than that of $latest\_committed_i$.

4.3 Correctness Argument

To prove that the proposed scheme is correct, it is necessary to prove that all histories representing executions that could be produced by it are one-copy serializable. We try to simplify the proof to the case with one MSS and a mobile host. The proof can be extended to the case with multiple mobile hosts and MSSs easily.
Formally, a transaction can be defined with the following notations:

T_i: it refers to a transaction i.

x: a data object named x. Two distinct copies, known as x_o and x_c, are defined. x_o is the copy residing in the MSS whereas x_c is the cached copy in the mobile host.

$r_i[x]$: a read operation by transaction i accessing data object x.

$w_i[x]$: a write operation by transaction i accessing data object x.

c_i: the commit operation for transaction i.

Two notations for global batches are defined:

GB_i: a global batch at i.

cc_i: the global commit operation for the global batch GB_i.

Definition 4.6 A special transaction, known as the **cache transaction**, is defined to represent the process of sending data from the MSS to the mobile host, such as the broadcasting of update notification. For each data object x sent from a MSS to a mobile host, the cache transaction contains a read that reads the value of the replica of x at the MSS immediately after the value is committed, and writes the value of x both in the MSS and the mobile host. It may also contains a set of purge operation p to purge the copy y_c of some data objects y from the cache. □

Augmentation 4.1 The definition of read-from relation is augmented since a data object can be purged from the mobile host. A read-only transaction T_j executed at a mobile host is said to *read-x-from* a transaction T_i if the following conditions hold:

1. $w_i[x_c] < r_j[x_c]$.
2. there does not exist another transaction T_k such that $w_i[x_c] < w_k[x_c] < r_j[x_c]$.
3. there does not exist a cache transaction T_m with a purge operation p_m such that $w_i[x_c] < p_m[x_c] < r_j[x_c]$.

Since only the cache transaction will write the copy x_c for some data x, a read-only transaction executed at the mobile host must read-x-from a cache transaction. A cache transaction T_n, in turn, *reads-x-from* a transaction T_m if the following conditions hold:

1. $w_m[x_o] < r_n[x_o]$.
2. $c_m < r_n[x_o]$; or if T_m is a public transaction included in a global batch GB_a, then $cc_a < r_n[x_o]$.
3. there does not exist another transaction T_p such that $c_m < c_p < r_n[x_o]$; or if T_m is a public transaction included in a global batch GB_a, then there does not exist another transaction T_p included in another global batch GB_b $(a \neq b)$ such that $cc_a < cc_b < r_n[x_o]$. □

Augmentation 4.2 The following edges are added to SG(H) to form a RDSG for H:

Given T_j be a read-only transaction submitted at a mobile host, and it reads-x-from a transaction T_i. Then for each transaction (excluding cache transaction) T_k that writes x and $T_i \rightarrow T_k$, an edge $T_j \rightarrow T_k$ is added. □

In the next sub-section, we show that the resulting graph is a RDSG for H.

4.3.1 Correctness Proof

A MC history H is a replicated history that models an execution of the transactions in the scheme. Every MC history H has the following properties:

Property 1: If T_i writes x, then H contains either $w_i[x_o]$ or both $w_i[x_o]$ and $w_i[x_c]$.

Property 2: If T_i reads x, then H contains either $r_i[x_o]$ or $r_i[x_c]$ exclusively.

Property 1 says that a transaction writes at least the copy residing in the MSS. Property 2 says that either one of the copies of a data can be read by a transaction.

Let GB_m and GB_n be two consecutive global batches and T_i be a cache transaction that caches the values of some data written by the public transactions in GB_m to the mobile host. Next we describe the properties of T_i.

Property 3: $cc_m < r_i[x_o] < cc_n$.

Property 4: If T_j is a public transaction in GB_n that reads (writes) x, then $w_i[x_o] < r_j[x_o]$ ($w_i[x_o] < w_j[x_o]$).

Property 5: If a local transaction T_l writes a data y such that its value is read by some public transaction in GB_n, then $c_l < r_i[y_o]$; and for every local transaction T_m that reads (writes) y and $T_l \rightarrow T_m$, $w_i[y_o] < r_m[y_o]$ ($w_i[y_o] < w_m[y_o]$).

Property 3 & 4 say that the cache transaction should read the values of data written from the public transactions just committed and should be serialized before the next

global batch. Property 5 shows that serialized order of the cache transaction and local transactions.

To prove the correctness of the scheme, it is required to prove that every MC history H has an acyclic RDSG. We will first show that the resulting graph with the above edges added is a RDSG of H and then show that the RDSG is acyclic [8].

Lemma 4.1 Let H be a MC history. SG(H) with edges added induces a write order for H.

Proof: From property 1, it is shown that each of the transactions containing write operations writes at least the copy x_o for each of the data x being written. Therefore for any two transactions that write a data x, there must be a path between them. The resulting graph hence induces a write order for H. □

Lemma 4.2 Let H be a MC history. SG(H) with edges added induces a read order for H.

Proof: Suppose T_j reads-x-from T_i, T_k writes x ($i \neq k$ and $j \neq k$), and there is a path from T_i to T_k. We need to prove there is a path from T_j to T_k.

Case 1: T_j reads x_o.

T_i, T_j and T_k may either be public (local) transactions or the cache transactions. We will show that any combination of these transactions induces a read order.

Suppose that T_i, T_j are public (local) transactions, and T_k is a cache transaction. Since T_k writes the copy x_o, T_j and T_k conflict. Either $r_j[x_o] < w_k[x_o]$ or $w_k[x_o] < r_j[x_o]$. In the former case, a path exists from T_j to T_k and we are done. For the latter case, T_j should read-x-from T_k or some transaction T_m such that a path exists from T_i to T_m, which is a contradiction.

The same argument can be applied to other combinations of T_i, T_j and T_k.

Case 2: T_j reads x_c.

T_j should be a read-only transaction at the mobile host. T_i should be a cache transaction

and T_k can be either a public (local) transaction or a cache transaction. If T_k is a public (local) transaction, then with the edge being added to SG(H) by Augmentation 4.2, we have a path from T_j to T_k. If T_k is a cache transaction, since T_k contains $w_k[x_c]$, either $r_j[x_c] < w_k[x_c]$ or $w_k[x_c] < r_j[x_c]$. In the former case, a path exists from T_j to T_k and we are done. If $w_k[x_c] < r_j[x_c]$, then T_j must read-x-from T_k or some transaction T_m such that there is a path from T_i to T_m, which is a contradiction. □

We have proved that every MC history has a RDSG. It remains to prove that the RDSG for every MC history H is acyclic.

Lemma 4.3 Let H be a MC history. RDSG of H is acyclic.

Proof: We first show that the inclusion of the cache transaction will not affect the ordering of transactions executed under TRS. We then show that the inclusion of read-only transactions create no cycle. From properties 3 & 4, it is shown that a cache transaction, accessing the latest committed values of a global batch GB_i, can be serialized after GB_i and immediately before the global batch following GB_i. Besides, it is shown in property 5 that the cache transaction can be serialized before the local transactions after the one whose value is broadcasted to the mobile host. A serial order of these transactions therefore looks like:

$$\cdots T_{l_1} \rightarrow T_{l_2} \rightarrow \cdots T_{l_m} \rightarrow T_c \rightarrow T_{p_1} \cdots \rightarrow T_{p_n} \rightarrow T_{l_{m+1}} \cdots$$

where T_{l_i}, T_{p_i}, and T_c are local, public and cache transactions respectively.

We then show that the inclusion of read-only transactions in the mobile hosts will not create any cycle. Assume that a serial order of transactions as shown below is obtained without read-only transactions (note that '$\rightarrow$' refers to an edge):

$$\cdots T_{a_1} \rightarrow T_{a_2} \rightarrow \cdots T_{a_{o-1}} \rightarrow T_{a_o} \cdots$$

Suppose, by way of contradiction, a cycle exists in the RDSG with read-only transactions included. The cycle may look like:

$$T_{a_1} \rightarrow T_{c_1} \rightarrow T_{c_2} \rightarrow \cdots T_{c_m} \rightarrow \cdots T_{a_n} \rightarrow T_{b_1} \rightarrow T_{a_1}$$

where T_{c_i} may be T_{a_j} for some j or a read-only transaction submitted at a mobile host.

The cycle should contain some read-only transactions submitted at the mobile host. Assume that T_{b_1} is such a transaction [1]. T_{b_1} should have read-x-from T_{a_n} for some data x, where T_{a_n} is a cache transaction. Note that there may be some other read-only transactions included the cycle, but since their existence will not affect the arguments shown below, we focus on T_{b_1} here.

On the other hand, an edge $T_{b_1} \rightarrow T_{a_1}$ appears if either one of the following cases holds:

1. some physical operation in T_{b_1} conflicts with some physical operation in T_{a_1}. Since T_{b_1} is a read-only transaction, T_{a_1} cannot be a read-only transaction. T_{a_1} must have therefore written some data y. Besides, since T_{b_1} reads the copy y_c, T_{a_1} should have written the copy y_c, that is to say, T_{a_1} is a cache transaction.

2. the edge is added according to Augmentation 4.2. This means that T_{a_1} is a local or public transaction.

However, the above serial ordering of operations will prevent case 1 to occur. It remains to prove that case 2 will leads to contradiction.

It is shown in Augmentation 4.2 that the edge $T_{b_1} \rightarrow T_{a_1}$ is added if T_{a_1} is either a local or public transaction that has written some data y, and there exists a transaction T_{a_u} such that T_{b_1} reads-y-from T_{a_u} and $T_{a_u} \rightarrow T_{a_1}$. From the serial order of $T_{a_1} \rightarrow \cdots T_{a_n}$, we get

$$T_{a_u} \rightarrow T_{a_1} \rightarrow T_{c_1} \rightarrow T_{c_2} \rightarrow \cdots T_{a_n}$$

[1] Since a read-only transaction from a mobile host will not have an edge to another read-only transaction from mobile host, such a transaction will be isolated from other read-only transactions.

However, if T_{a_1} is a local or public transaction that has written y, then there must be a cache transaction T_{a_c} (T_{a_c} may be T_{a_n}) such that a value of y is written to the copy y_c. Therefore T_{b_1} should read-y-from some cache transaction between and including T_{a_c} and T_{a_n}, which is a contradiction. □

Theorem 4.1 The proposed scheme is correct.

Proof: By Lemma 4.1 and 4.2, SG(H) is a RDSG of H. By Lemma 4.3, SG(H) is acyclic. Therefore H is 1SR. □

4.4 Extension to Support Partition Failures

So far we are assuming that the fixed hosts can communicate with each other in the fixed network, and no partition failure occurs. However, it is not the case in the real world. With the introduction of partition failures, some modifications to the scheme proposed in the previous sections have to be made. This section describes how the scheme is modified to cope with this problem.

In TRS, partition failure is handled by means of the *virtual partition protocol* [11]. In this protocol, each transaction is executed under a *view*, which is a subset of the set of all the fixed hosts. Each fixed host has a view V, showing the set of fixed hosts that it is able to communicate with. Each view has a unique view-id.

For the execution of read-only transactions, problem occurs when a mobile host moves from one partition to another. In order to solve the problem, the view-id where the MSS currently has is also broadcasted in the update notification. On the other hand, each mobile host keeps the view-id of the MSS that is received in the previous update notification. On receipt of a new update notification, each mobile host first checks whether the view-id in the update notification agrees with that kept by it. If so, the same steps as mentioned in cache update are carried out. Otherwise, the mobile host should have moved from one partition to another, and the cached data should all be purged. The steps

as is mentioned in case two of the cache update are carried out. The value of view-id kept by mobile host is then changed to the one in the update notification.

Besides, the request messages for missed data should also include the values of the view-id kept by the mobile hosts. Each time a request message is received, the MSS first checks for the value of view-id in the request message with that kept by it. Disagreement of the two values will result in the ignorance of the request. Otherwise, reply with the view-id of MSS included is sent back to the mobile host.

Partition failure will also affect the broadcasting of R/W public transactions sent by a mobile host to all the other fixed hosts. In order to solve the problem, it is required for the MSSs to broadcast the set of R/W public transactions received as well as their results to other MSSs upon view update.

Chapter 5

Design and Implementation of the Simulation

In order to experiment with the scheme proposed, a simulation is designed and constructed. The simulation is a program written in the CSIM [35] language. The simulation gathers statistics during a run, and presents a summary upon completion. These statistics provide much information about the performance of the scheme under different system settings which can be used to analyze and compare the performance of various designs, and also validate the correctness of the scheme.

The simulation does not provide a fine-grained model of the transaction replication scheme, since we are primarily interested in the behaviour and performance of the transactions submitted at the mobile hosts. A detailed performance study about the transaction replication scheme can be found in [28].

5.1 CSIM Language

CSIM is a simulation language which is copyrighted by Microelectronics and Computer Technology Corporation. CSIM is a library of routines, which can be used to create process-oriented discrete-event simulation models with the use of C or C++ programs. A CSIM program models a system as a set of CSIM processes which interact with one another by using the CSIM structures, such as requesting service at facilities, waiting for events, etc.

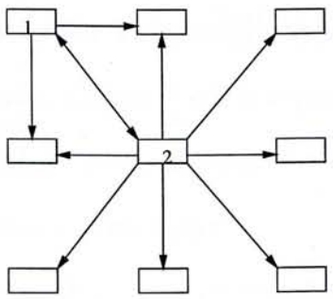


Figure 5.1: Possible Moves for Mobile Hosts in Cell 1 & 2

Each simulation model maintains simulated time. It can be used to produce estimates of time and performance. CSIM provides structures that can be used to collect simulation statistics during the execution of a model.

5.2 Simulation Components

5.2.1 Fixed Network

The fixed network consists of a fixed number of MSSs. Each MSS is assigned an unique ID number before the simulation. It is assumed that each MSS has infinite resources and that the fixed network is highly reliable, hence no failure happens there. It is assumed also that the fixed network is perfectly synchronous. For the wired channels, it is assumed that the time delay in sending and receiving messages is negligible.

Concerning about the global batches of public transactions, it is assumed that they can be broadcasted and executed to completion within 2 TRS periods. Since the fixed network is assumed to be highly reliable, all the global batches can be committed globally.

5.2.2 Mobile Host

The number of mobile hosts in the system is assigned before a particular run of the simulation. Each mobile host has an unique number assigned as its ID. A mobile host can move from one cell to another periodically. A predefined pattern of movement is used to govern the movement of the mobile hosts over the fixed number of cells in the system.

Figure 5.1 shows the possible moves for the mobile hosts in a particular cell. A mobile host can move to anyone of the cells allowed with equal probability. The period of time between successive moves of a mobile host across the cells is chosen randomly, with an exponential distribution, and a fixed mean time.

A mobile host can be powered off during a particular run. The period of time for it to remain connected before it is to be powered off is chosen randomly with an exponential distribution over a fixed mean time. It is assumed that the mobile hosts that are powered off will reconnect to the system after a short period of time, which is randomly assigned over a fixed mean time with exponential distribution. Besides, a mobile host may fail to send or receive a message through the wireless channels owing to the background noise.

5.2.3 Wireless Channel

Each mobile host communicates with the local MSS through the wireless channel. In order to simplify the simulation, it is assumed that CSMA/CD is used as the access method for the broadcasting of messages from the MSSs to the mobile hosts, and that no collision occurs during the transmission of messages. The resulting access method to be used for the transmission of messages through the wireless channels is the half-duplex CSMA/CD method, with no collision occurs. With these assumptions, the messages sent through a wireless channel are simply queued up.

It is mentioned in [15] that the time cost in sending and receiving a message should be calculated as:

$$\text{Transmit\_Delay} + \frac{\text{Message\_Size}}{\text{Bandwidth}} + \text{CPU}$$

However, with a small-sized wireless cell [20] and narrow wireless bandwidth, the factor $\frac{\text{Message\_Size}}{\text{Bandwidth}}$ becomes dominant. Therefore, it is assumed that the time cost for sending and receiving a message through the wireless channel is computed by:

$$\frac{\text{Message\_Size}}{\text{Bandwidth}}$$

Unreliable property of a wireless channel is simulated by using a variable with value that is uniformly distributed from 0 to 1. Each time when a mobile host has to send/receive a message through the wireless channel, a value of this variable is generated and is checked against a predefined threshold with value between 0 and 1. If the value of this variable is greater than the threshold, the channel is said to be noisy at that time and the message is lost. Otherwise, the message is sent/received through the wireless channel successfully.

5.2.4 Database and Transactions

Data in the database are represented by distinct numbers. Updated data versions are broadcasted to the mobile hosts periodically. In order to examine the effect of different caching strategies on the performance of the scheme, a simple caching method derived from [13] is used in one of the experiments. The method assumes that the data objects accessed by the transactions submitted at the mobile hosts is based on a 80/20 access model. That is, 80% of the accesses is directed to 20% of the data objects. The frequently accessed data objects are known as the *popular* data objects and the rest are called *unpopular* data objects.

Transactions are submitted at a host with an exponential distribution. Each transaction consists of a number of operations in a predefined range and the set of data read by a transaction should be the superset of the data written by it. It is assumed that the execution time of local transactions in a fixed host is negligible. For the public transactions submitted at both the fixed hosts and mobile hosts, shared-private and public data are accessed with equal probability.

Since it is assumed that a mobile host that powered off will soon reconnect to the system, transactions that do not start to execute when the mobile host is going to power off will be kept until the mobile host reconnects to the system. For those transactions that are executing when the mobile host is about to power off, the mobile host waits for them to finish before shutting down. However, no message will be sent and received by the mobile host in this period.

5.3 A Lock-based Scheme

In order to have concrete and meaningful simulation results, a simple scheme which makes use of locking in handling transactions executed in the mobile computing environments [9] is used for comparison with our scheme. The scheme is proposed with the assumption that the wireless channels are reliable and a message sent by one party through the wireless channel should be received by another party, given that both of them are connected to the network.

As is in our scheme, data are divided into shared-private data and public data. Each data object is replicated among all the fixed hosts. However, the transaction replication scheme is not applied here. Instead, the read-one-write-all approach [8] is applied to manage the replicated data.

Two-phase locking [26] is implemented as the concurrency control scheme. A read operation from a transaction of a mobile host will cause the data copy in the local MSS of the mobile host being locked with shared-mode. Data read by the transaction will then be cached in the mobile host during the execution of the transaction, and purged upon completion. For a write operation on data x, all the data copies of x in the fixed hosts have to be locked exclusively.

Transactions in a mobile host are executed serially. If there is a transaction being processed, any newly arrived transactions are forced to wait until the transaction finished execution. The scheme makes use of an operation-oriented approach in processing transactions submitted at the mobile hosts. Operations of a transaction submitted at a mobile host are sent to the local MSS for execution one-by-one. When an operation is sent to the MSS, the mobile host has to wait for the reply before processing the next operation. If the reply shows that the lock for the data cannot be acquired, the transaction has to abort.

A status indicator will be sent to the MSS after all the operations of the transaction are processed or if the transaction has to abort owing to the failure in acquiring a lock. The MSS can then commit or abort the transaction accordingly.

When a mobile host is about to disconnect from the network, it sends a disconnection indicator to the MSS. On the other hand, when the MSS receives the disconnection indicator, it checks and releases all the locks being held by the mobile host.

Timeout is used by the fixed hosts in handling deadlock as well as the missing of messages owing to the disconnection of mobile hosts.

5.4 Graphing

Data collected by the simulation programs have to be represented in the form of graphs before they can be analyzed. C-shell utilities and Matlab [1] are used to postprocess and generate graphs in Encapsulated Postscript format.

Results and graphs of the simulation are provided in Chapter 6.

[1] Copyrighted by the MathWorks, Inc., 1984 - 1994

Chapter 6

Results and Analysis

This chapter presents the results of various simulation experiments carried out to examine the performance of the proposed scheme. The chief objective of the experiments is to observe the performance of the transactions submitted at the mobile hosts as well as the utilization of the wireless channels under different system settings.

Two sets of experiments are carried out. Performance of the proposed scheme is studied first. It is then followed by the comparison with the lock-based scheme mentioned in chapter 5.

6.1 Results Dissection

For the experiments discussed below, several statistics are shown. The meaning as well as the computation for these statistical data are described below:

1. Utilization of wireless channels. This shows the amount of time that the wireless channels are occupied with messages. It is calculated by:

$$\frac{\text{busy time}}{\text{total elapsed time}}$$

2. Transaction throughput. This gives the number of transactions submitted at the mobile hosts that can be committed in a unit time.

3. Response time of read-only public transactions. This gives the mean execution time of read-only public transactions executed at the mobile hosts. Execution time of a

transaction refers to the period of time between the submission of the transaction and its commitment. Note that only the committed transactions are considered.

4. Commit ratio of read-only public transactions. This refers to the fraction of read-only public transactions submitted at the mobile hosts that can be committed during the simulation. It is calculated by:

$$\frac{\text{number of committed transactions}}{\text{number of committed transactions} + \text{number of aborted transactions}}$$

5. Cache hit ratio. This refers to the number of read accesses that can be gained locally in the cache (cache hit) of the mobile hosts after update notifications are received and before the corresponding replies of miss-sets are received. It is calculated by:

$$\frac{\text{number of cache hit}}{\text{number of cache hit} + \text{number of cache miss}}$$

6. Response time of R/W public transactions. Similar to the case for read-only public transactions.

7. Commit ratio of R/W public Transactions. Similar to the case for read-only public transactions.

6.2 Performance of the Scheme

In this section, the performance of the scheme under various system settings are studied. The set of parameters and their values are described first. It is then followed by the experiments and analysis.

6.2.1 Parameters Setting

In the simulation program, the parameters are set with reference to [13, 38]. Each of the transactions, both local and public, submitted at a fixed host has number of operations in the range between 8 and 12. Local transactions are submitted at a fixed host with mean time equals 10 seconds.

Transactions submitted at a mobile host have number of operations ranges from 4 to 8. Each of the mobile hosts can cache up to 30 data versions.

There are 9 fixed hosts in the system. Each fixed host owns 20 shared-private data objects. The database has 150 public data. The system therefore has altogether 330 data objects.

Parameter	**Description**	**Value**
Parameters for Fixed Network		
NUM_SERVER	number of fixed hosts in the system	9
LOC_ARR_TIME	inter-arrival time of local transactions	10 seconds
F_MIN_OPER	minimum number of operations in a transaction	8
F_MAX_OPER	maximum number of operations in a transaction	12
Parameters for Data Objects		
F_PDATA	number of public data in the system	150
F_SDATA	number of shared-private data for each fixed host	20
Parameters for Mobile Hosts		
CACHE_SIZE	cache size	30 data
MAX_OPER	maximum number of operations in a transaction	8
MIN_OPER	minimum number of operations in a transaction	4
Parameters related to Message and Time		
ID_SIZE	size of data id	10 bytes
DATA_SIZE	size of a data version	1K bytes
TRAN_SIZE	size of a R/W transaction	100 bytes
COMMIT_MESG	commit/abort of R/W public transactions	100 bytes
ACK_MESSAGE	acknowledgement of transactions results received	100 bytes
MESG_HEADER	fix-sized header for each message	30 bytes
BANDWIDTH	size of each wireless channel	1×10^6 bps
DATA_IO	I/O time for mobile hosts	0.035 seconds
DATA_CPU	CPU time for mobile hosts	0.01 seconds
POWER_OFF	mean time for a mobile host to power off	1500.0 seconds
DIS_DURATION	mean time for a mobile host to remain powered off	100.0 seconds
W_TIMEOUT	timeout waiting a message in the wireless channel	1.5 seconds
SIMTIME	simulation time	12000 seconds

Table 6.1: Parameters Setting

Each copy of a data object occupies 1K bytes; whereas the data ID for each data object is of size 10 bytes. The size of a transaction is 100 bytes. Each message broadcasted by a MSS concerning the commit/abort of R/W public transactions submitted at the mobile

Variable Parameters	Description
TRS_PERIOD	duration of the TRS period
MOBILE_NO	number of mobile hosts in the system
ARR_TIME	inter-arrival time of public transactions at a mobile host
COLLECTION_PERIOD	collection period for the miss-sets
CONNECT_PROB	probability that a wireless channel is reliable
PUB_ARR_TIME	inter-arrival time of public transactions at a fixed host
L_BATCH_TIME	lower bound for the execution time of global batches
U_BATCH_TIME	upper bound for the execution time of global batches
RW_PROB	percentage of R/W transactions submitted at a mobile host
HAND_OFF	mean time for a mobile host to leave the cell

Table 6.2: Variable Parameters

hosts is sized 100 bytes; and each of the acknowledgements sent by a mobile host for the commit/abort of these transactions has size equals 100 bytes also.

It is assumed that the messages sent through the wireless channels consist of a fix-sized header of 30 bytes. All the messages mentioned above are sent through the wireless channels with bandwidth sized 1×10^6 bps. A mobile host may timeout waiting for a message sent through the wireless channel and the timeout period is set to 1.5 seconds.

Each I/O access for a mobile host uses 0.035 seconds and each cpu access uses 0.01 seconds. The mean time for a mobile host to remain connected before power off is 1500 seconds. When a mobile host disconnects from the network, it will reconnect to the system after a period of time that is exponentially distributed over a mean of 100 seconds. Table 6.1 shows the list of parameters together with their values.

Table 6.2 shows the set of parameters with their values to be varied for each run of the simulation. These parameters include the duration of the TRS period, number of mobile hosts in the system, etc. The execution time for the global batches is also varied. The parameters L_BATCH_TIME and U_BATCH_TIME give the lower and upper bounds for the execution time of a global batch. Values of L_BATCH_TIME and U_BATCH_TIME are fractions to the variable TRS_PERIOD.

6.2.2 Experiments and Results

In this section, results of the experiments carried out are presented. For each of the experiments, the values of the variable parameters are shown, and a short description is given at the end to analyse the results.

Experiment 1 Variation of TRS Period

This experiment is to study the effect of the length of TRS period on the performance of transactions submitted at the mobile hosts. The values of the variable parameters are shown in Table 6.3.

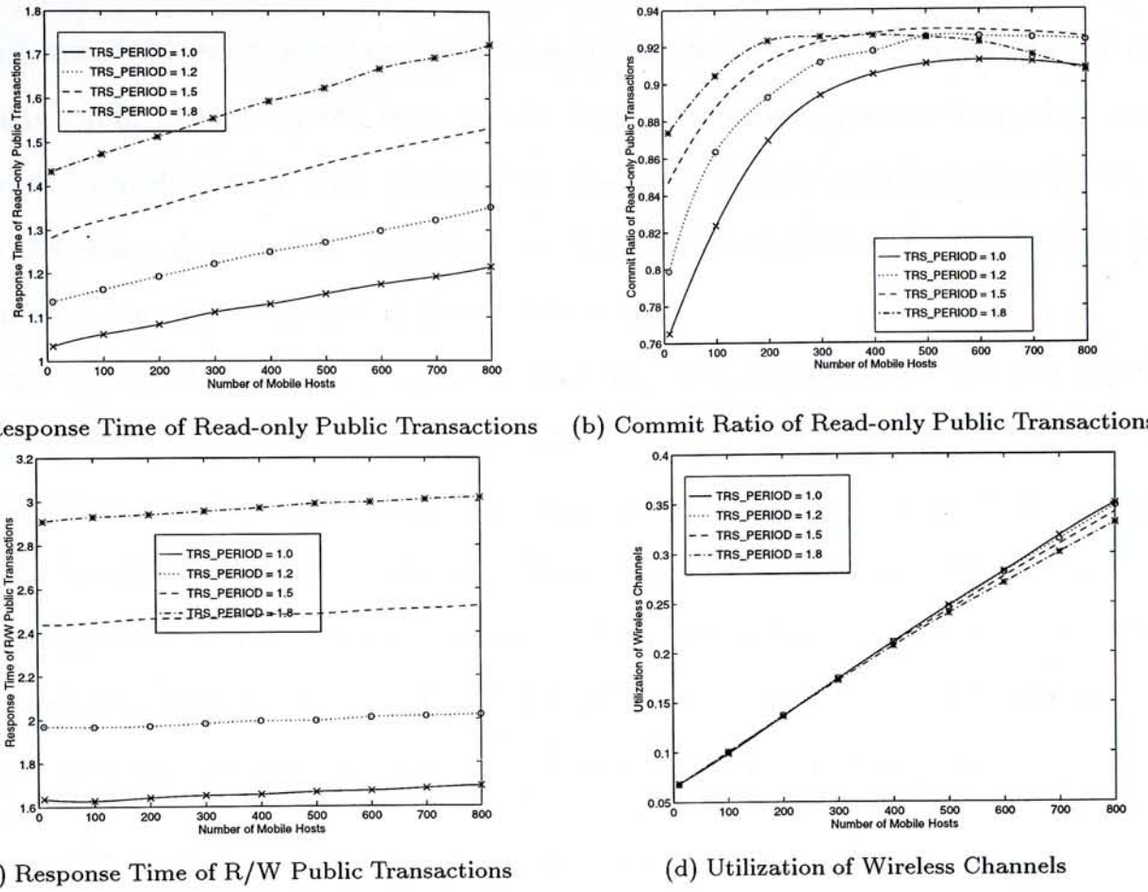
It is assumed that a global batch can be committed globally within the period of one TRS period. Given the value of TRS_PERIOD equals 1.5 seconds, a global batch can be committed globally in the period between 1.2 (L_BATCH_TIME $\times$ TRS_PERIOD) and 1.5 seconds (U_BATCH_TIME $\times$ TRS_PERIOD).

Parameter	Value
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
CONNECT_PROB	0.95
PUB_ARR_TIME	5.0 seconds
L_BATCH_TIME	0.8
U_BATCH_TIME	1
RW_PROB	0.1
HAND_OFF	1500.0 seconds

Table 6.3: Variable Parameters Setting (experiment 1)

Figure 6.1 shows the results of the experiment with TRS_PERIOD having values of 1.0, 1.2, 1.5 & 1.8 seconds respectively. The system has number of mobile hosts varies from 10 to 800.

It is found in Figures 6.1 (a) & (c) that the response time of transactions submitted at the mobile hosts increases as the length of TRS period increases. With a longer TRS period, transactions submitted at a mobile host have to wait for a longer time before they



(a) Response Time of Read-only Public Transactions

(b) Commit Ratio of Read-only Public Transactions

(c) Response Time of R/W Public Transactions

(d) Utilization of Wireless Channels

Figure 6.1: Variation of TRS Period

can be executed. This contributes to the gaps among the curves. Besides, it is found in Figure 6.1 (a) that the response time of the read-only transactions for all the cases are increasing steadily. This is owing to the increased number of mobile hosts in the system, which increases the number and size of messages being sent through the wireless channels.

Furthermore, it is found in Figure 6.1 (b) that the commit ratio of the read-only public transactions for the system with the length of TRS period equals 1.0 seconds is maintained at a lower level than the others. It is because with such a short TRS period, some requests for the missed data may arrive at the MSSs after a newer global batch is committed globally, which results in the increase in abort ratio. Also, it is found that the commit ratio for the system with the length of TRS period equals 1.8 seconds starts to drop when the number of mobile hosts is about 400. It is because with a longer TRS period, more transactions are collected by each mobile host. This results in a larger set

of data missed and a longer time is required to send and receive this set of data. The execution time of the read-only transactions hence increases. When the number of mobile hosts reaches 400, the execution time of the transactions becomes so long that some of them cannot complete their execution when the next update notification arrives. This results in the sharp decrease in commit ratio. Under this system settings, the best length for TRS period should be around 1.2 to 1.5 seconds.

It should be noted that the upward slopes at the beginning of the curves in Figure 6.1 (b) are caused by the higher probability of missing the update notifications by the mobile hosts. With fewer mobile hosts in the system, the total number of messages being sent through the wireless channels decreases. However, the number of update notifications being broadcasted is more or less the same for different number of mobile hosts. Given a fixed message loss probability (CONNECT_PROB), the probability of losing an update notification by a mobile host increases with fewer number of mobile hosts.

Experiment 2 Variation of Reliability of Wireless Channels

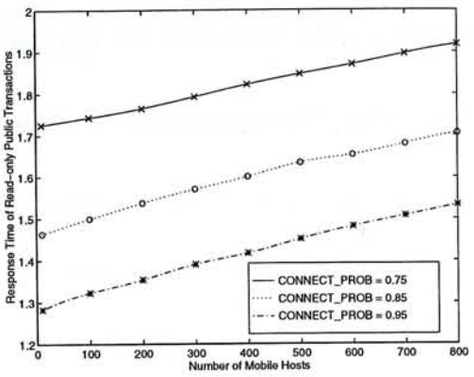
Parameter	**Value**
TRS_PERIOD	1.5 seconds
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
PUB_ARR_TIME	5.0 seconds
L_BATCH_TIME	0.8
U_BATCH_TIME	1
RW_PROB	0.1
HAND_OFF	1500.0 seconds

Table 6.4: Variable Parameters Setting (experiment 2)

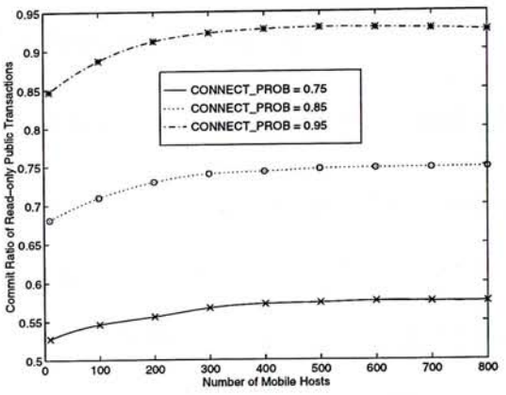
This experiment is to investigate the performance of the scheme with wireless channels interfered with different levels of noise. The values of the variable parameters are shown in Table 6.4.

Figure 6.2 shows the curves for the systems with values of CONNECT_PROB equal 0.75, 0.85, and 0.95 respectively. Each curve shows the performance of the system with

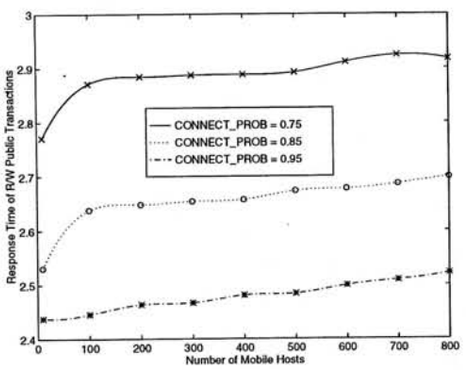
number of mobile hosts varies from 10 to 800.



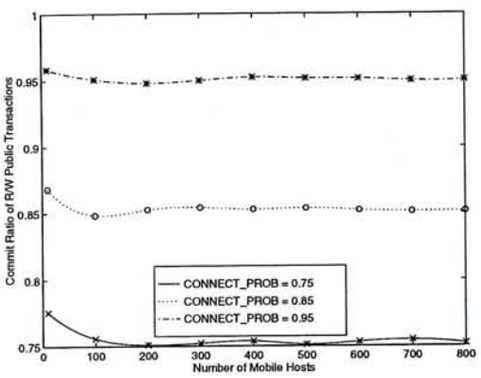
(a) Response Time of Read-only Public Transactions



(b) Commit Ratio of Read-only Public Transactions



(c) Response Time of R/W Public Transactions



(d) Commit Ratio of R/W Public Transactions

Figure 6.2: Variation of Reliability of the System

It is found that as the reliability of wireless channels decreases, the response time of public transactions submitted at the mobile hosts increases and their commit ratio decreases. It is because as the wireless channels become more noisy, a message sent through a wireless channel is more probable of being lost. If a mobile host cannot receive an update notification, read-only public transactions collected there cannot be executed. Besides, results of R/W public transactions are lost. The mobile host will then have to wait for the next update notification before it can have the results of the R/W public transactions realized and the read-only public transactions executed.

Also, as the wireless channels become more unreliable, more messages sent by mobile hosts will be lost and more requests will timeout waiting for their replies. Therefore the commit ratio of transactions is lower for the system with wireless channels that are more

unreliable. Note that the commit ratio of the R/W public transactions is very close to the value of CONNECT_PROB. This shows that the commit ratio of R/W public transactions depends largely on whether the transactions can be sent to the MSSs.

Experiment 3 Variation of Inter-arrival Time of Public Transactions at Fixed Hosts

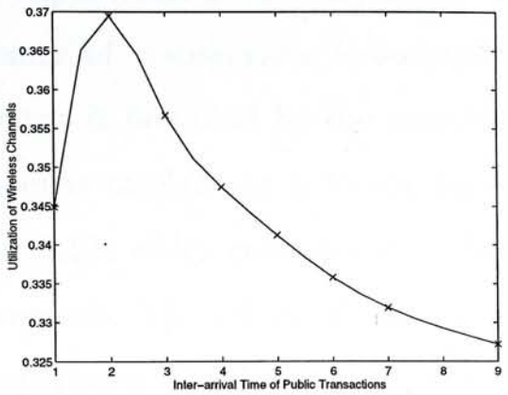
This experiment is to examine the effect of the number of public transactions submitted at the fixed hosts on the performance of the scheme. In other words, it is to examine the effect of the amount of data updated in each global batch on the system's performance. The inter-arrival time of public transactions, PUB_ARR_TIME, is given the values 1, 2, 3, 4, 5, 6, 7 & 9 seconds. The values of other variable parameters are shown in Table 6.5.

Parameter	**Value**
TRS_PERIOD	1.5 seconds
MOBILE_NO	800
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
CONNECT_PROB	0.95
L_BATCH_TIME	0.8
U_BATCH_TIME	1.0
RW_PROB	0.1
HAND_OFF	1500.0 seconds

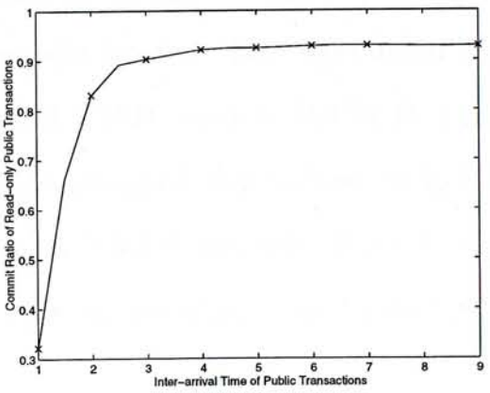
Table 6.5: Variable Parameters Setting (experiment 3)

Figure 6.3 shows the results of the experiment with number of mobile hosts equals 800 for all the cases.

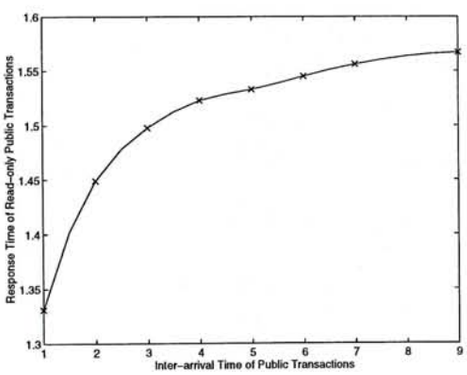
It is found from the results that the performance of transactions deteriorates greatly if the amount of data updated in each global batch is very large. As shown in the graphs of Figure 6.3, the commit ratio of read-only public transactions for the case with the value of PUB_ARR_TIME equals 1 second falls below 0.4. Besides, the response time of R/W public transactions for the same case is greater than the others for about 0.2 seconds. It is due to the larger sets of data copies being included in the update notifications, which greatly increases the message cost in sending them to the mobile hosts. Response time of R/W public transactions therefore increases.



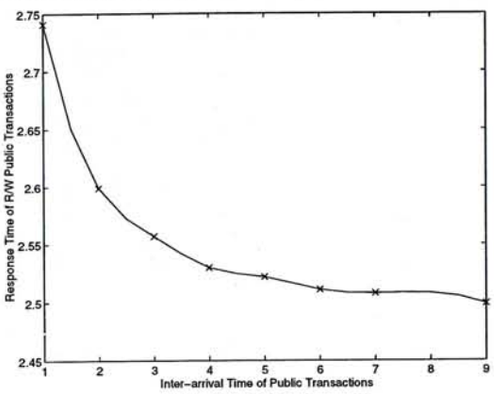
(a) Utilization of the Wireless Channels



(b) Commit Ratio of Read-only Public Transactions



(c) Response Time of Read-only Public Transactions



(d) Response Time of R/W Public Transactions

Figure 6.3: Variation of Inter-arrival Time of Public Transactions at Fixed Hosts

Besides, if the message delay of sending the update notifications is so large as in the case with the value of PUB_ARR_TIME equals 1 second, no mobile host will be able to send the miss-set to the MSSs by the end of the collection periods, hence no reply will be received. This can be verified from Figure 6.3(a) in which the utilization of wireless channels drops for the case with the value of PUB_ARR_TIME equals 1 second. According to the scheme proposed, read-only transactions requiring data missed have to abort if no reply of miss-set is received.

However, a larger amount of data broadcasted can reduce the response time of read-only transactions owing to the increased cache hit ratio. Under this system setting, the optimal value for PUB_ARR_TIME should be around 4 to 5 seconds.

Experiment 4 Variation of Execution Time for Global Batches

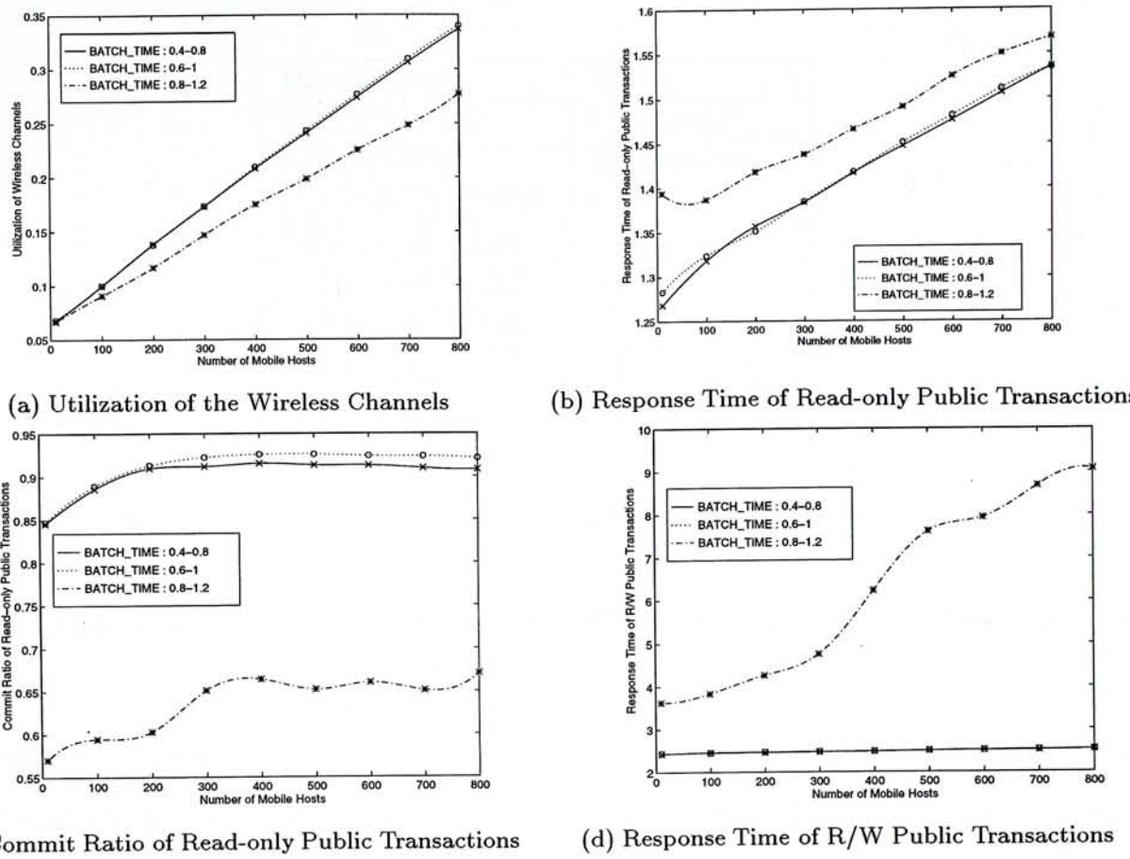
This experiment is to analyse the effect of execution time of global batches on the performance of transactions submitted at the mobile hosts. The execution time of a global batch is bounded by the variables L_BATCH_TIME and U_BATCH_TIME. The experiment is carried out with the pair of variables assigned the values (0.4, 0.8), (0.6, 1) & (0.8, 1.2), which correspond to the time periods 0.6-1.2 seconds, 0.9-1.5 seconds, & 1.2-1.8 seconds. The values of other variable parameters are shown in Table 6.6. Results of the experiment are shown in Figure 6.4.

Parameter	**Value**
TRS_PERIOD	1.5 seconds
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
CONNECT_PROB	0.95
PUB_ARR_TIME	5.0 seconds
RW_PROB	0.1
HAND_OFF	1500.0 seconds

Table 6.6: Variable Parameters Setting (experiment 4)

As shown in Figure 6.4, performance of the scheme is comparable for the cases with the pair of variables having values (0.4, 0.8) and (0.6, 1). However, if the execution time of a global batch is allowed to exceed the duration of a TRS period, the results worsen sharply. It can be found in Figure 6.4 (b), (c) & (d) that with the pair of variables having values (0.8, 1.2), the commit ratio of the read-only transactions as well as the response time of the transactions submitted at the mobile hosts are worse than that of the other two cases to large extent.

It is because with such a long execution time for the global batches, the update notifications may not be sent at the end of each TRS period. This results in the sharp increase of response time for the R/W public transactions. Besides, a larger number of read-only transactions are collected by each mobile host before they can be executed. The need for the wireless bandwidth increases, hence more transactions may timeout waiting for replies of their requests. This will both increase the abort rate as well as the response



(a) Utilization of the Wireless Channels

(b) Response Time of Read-only Public Transactions

(c) Commit Ratio of Read-only Public Transactions

(d) Response Time of R/W Public Transactions

Figure 6.4: Variation of Execution Time for Global Batches

time of read-only transactions. Another reason for the sharp decrease in commit ratio of read-only transactions is that a global batch may now commit at the very beginning of a TRS period, which causes more requests by the read-only transactions for the missed data to be rejected by the MSSs.

Experiment 5 Variation of the Length of Collection Period

This experiment is to study the effect of the length of collection period on the performance of the read-only public transactions executed at the mobile hosts. The variable parameter COLLECTION_PERIOD is given the values 0.3, 0.4, & 0.5 seconds. A special case with the value of COLLECTION_PERIOD equals 0 second is also included. In this case, the mobile hosts send the requests for missed data on demand after the update notifications are received. The process of sending the miss-sets is ignored in this case.

Values of other variable parameters are shown in Table 6.7.

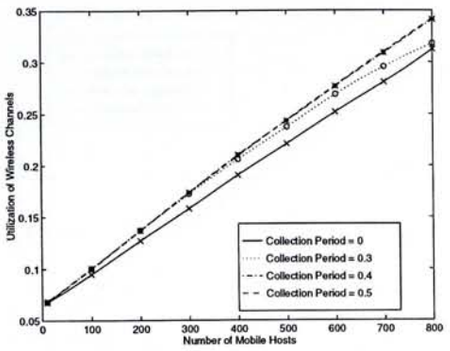
Parameter	**Value**
TRS_PERIOD	1.5 seconds
ARR_TIME	15.0 seconds
CONNECT_PROB	0.95
PUB_ARR_TIME	5.0 seconds
L_BATCH_TIME	0.8
U_BATCH_TIME	1
RW_PROB	0.1
HAND_OFF	1500.0 seconds

Table 6.7: Variable Parameters Setting (experiment 5)

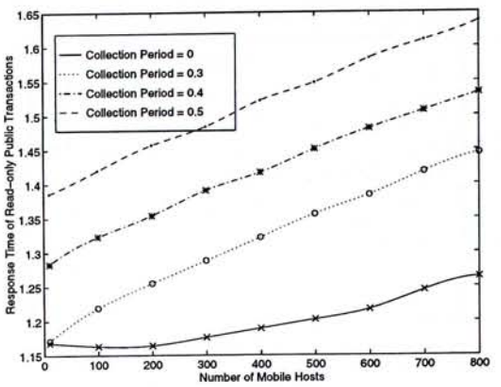
Results of the experiment with number of mobile hosts varies from 10 to 800 are shown in Figure 6.5.

As shown in Figure 6.5, response time of read-only transactions decreases as the duration of collection period decreases. However, if the length of collection period drops beyond 0.4 seconds, the commit ratio starts to deteriorate. It can be observed from Figure 6.5(c) that the commit ratio for the curve with the value of COLLECTION_PERIOD equals 0.3 seconds starts to drop when the number of mobile hosts reaches 200. This is owing to the fact that as the number of mobile hosts increases, the number of R/W public transactions submitted at the mobile hosts increases, which in turn increases the size of the update notifications. Eventually, the collection period is not long enough for the MSSs to collect even one single miss-set. As a result, no reply will be sent and the transactions have to abort.

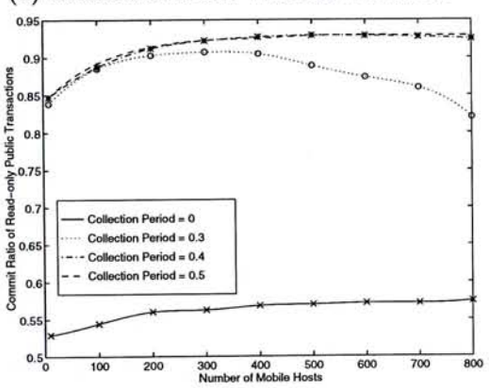
Moreover, it is found that for the case with the value of COLLECTION_PERIOD equals 0 second, the commit ratio of read-only transactions maintains at a very low level. It is due to the unreliable wireless channels which cause some of the messages sent through them to be lost. Since no miss-set is sent by the mobile hosts in this case, more messages have to be sent for the missed data. Nevertheless, missing anyone of these messages or its reply will cause the transaction in question to abort. Therefore more transactions are aborted when compared with the other two cases.



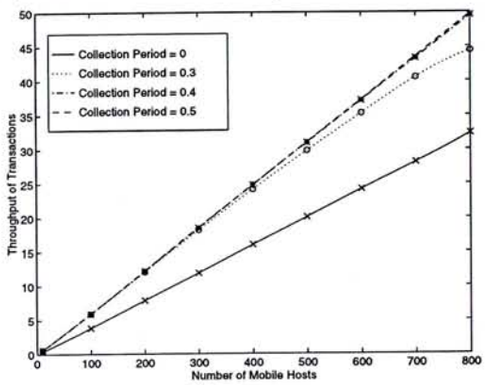
(a) Utilization of the Wireless Channels



(b) Response Time of Read-only Public Transactions



(c) Commit Ratio of Read-only Public Transactions

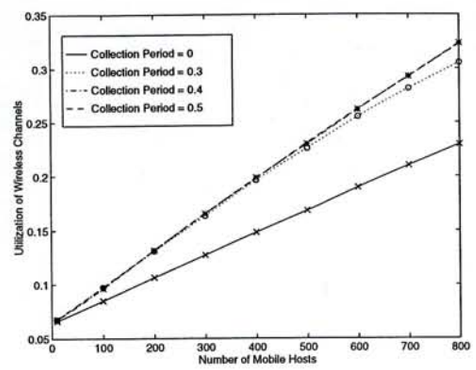


(d) Transactions Throughput

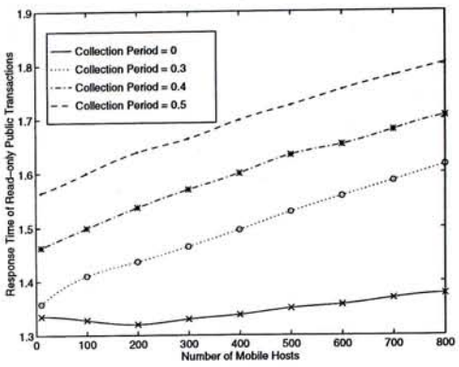
Figure 6.5: Variation of the Length of Collection Period (CONNECT_PROB = 0.95)

In order to verify the above arguments, the same experiment with wireless channels tuned to different levels of reliability is carried out. Two runs of the experiment, one with more unreliable wireless channels (CONNECT_PROB = 0.85) and the other with reliable wireless channels (CONNECT_PROB = 1.0), are carried out. Results of the experiments are shown in Figure 6.6 & 6.7.

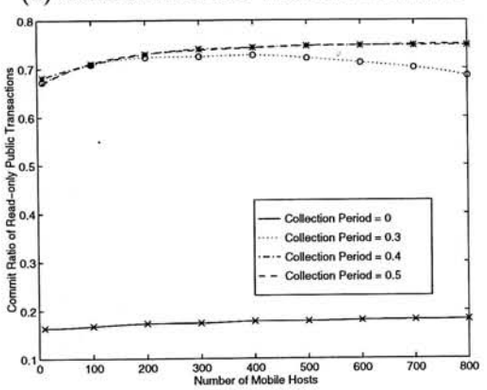
As shown in Figure 6.6, when the value of CONNECT_PROB equals 0.85, the commit ratio of read-only transactions for the case with the value of COLLECTION_PERIOD equals 0 second drops below 0.2; whereas for the rest of the cases, a commit ratio of about 0.7 can still be maintained. However, as shown in Figure 6.7, if the value of CONNECT_PROB equals 1.0, i.e. totally reliable wireless channels, the commit ratio of the read-only transactions for the case with the value of COLLECTION_PERIOD equals 0 second makes a sharp increase to more than 90%. This shows that in a system with



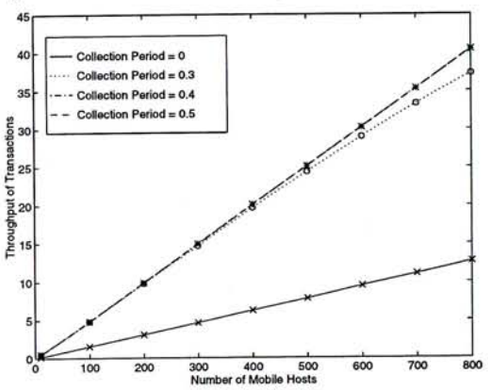
(a) Utilization of the Wireless Channels



(b) Response Time of Read-only Public Transactions



(c) Commit Ratio of Read-only Public Transactions



(d) Transactions Throughput

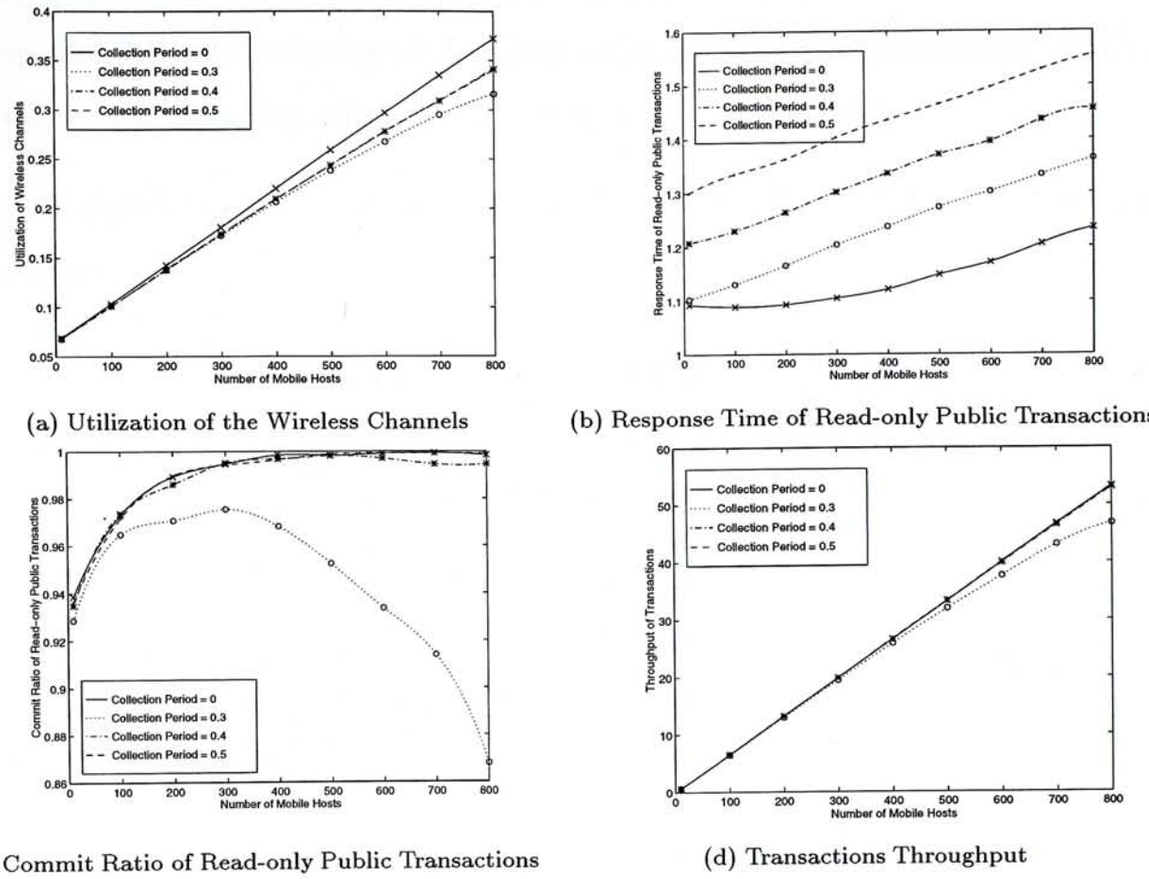
Figure 6.6: Variation of the Length of Collection Period (CONNECT_PROB = 0.85)

unreliable channels, grouping and sending the missed data in batches can greatly improve the commit ratio of read-only transactions, though the response time will be increased a little bit.

Experiment 6 Performance of the Scheme with Different Transaction Access Patterns

It is shown in the previous experiments that the set of data broadcasted in the update notifications may affect the performance of the scheme. In this experiment, the 80/20 access model is applied to the transactions submitted at the mobile hosts in order to observe the difference in the performance of the scheme. Different caching schemes are employed to cope with the change in transaction access model.

In this experiment, the first 30 numbered public data (20%) are selected as the popular public data; while the first 4 numbered shared-private data in each fixed host are chosen to



(a) Utilization of the Wireless Channels

(b) Response Time of Read-only Public Transactions

(c) Commit Ratio of Read-only Public Transactions

(d) Transactions Throughput

Figure 6.7: Variation of the Length of Collection Period (CONNECT_PROB = 1.0)

be the popular shared-private data. There are altogether 66 popular data in the database. Three different variations of the scheme are compared:

- *has_popular.* In this method, transactions submitted at the mobile hosts follow the 80/20 access model. For those data that are updated and have to be included in the update notifications, only the popular data objects will have their data values included. The unpopular data objects will only have their data IDs included for invalidation.

- *in_between.* In this method, transactions submitted at the mobile hosts follow the 80/20 access model, and all the data objects in the update notifications will have their data values included.

- *no_popular.* This is the method used in the previous experiments, with transactions from mobile hosts accessing all the data objects with equal probability, and update notifications including all the data values of the updated data.

Table 6.8 shows the values of the variable parameters. Figure 6.8 shows the results of the experiment.

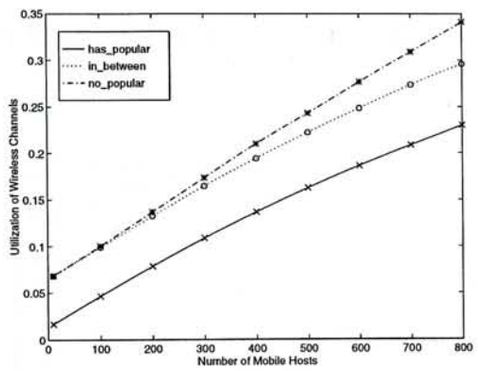
Parameter	**Value**
TRS_PERIOD	1.5 seconds
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
CONNECT_PROB	0.95
PUB_ARR_TIME	5.0 seconds
L_BATCH_TIME	0.8
U_BATCH_TIME	1
RW_PROB	0.1
HAND_OFF	1500.0 seconds

Table 6.8: Variable Parameters Setting (experiment 6)

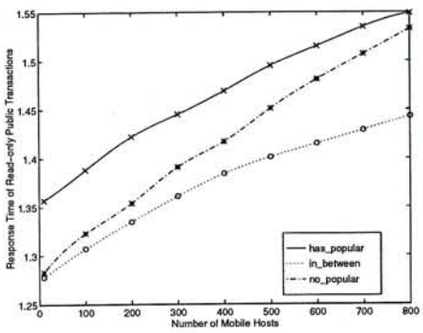
In Figure 6.8, it can be observed that the method *has_popular* outperforms the other two methods in the utilization of the wireless channels and the response time of the R/W public transactions. This can be explained by the smaller update notifications being broadcasted. With only the values of updated popular data objects being included, the size of update notifications for the method *has_popular* should be smaller than that of the others. This in turn reduces the time in broadcasting the update notifications.

However, as shown in Figure 6.8 (b), the smaller size of update notifications for the method *has_popular* results in the worst response time of read-only transactions among the three methods. It is because at each mobile host, there will be some transactions submitted during the broadcasting of update notifications. These transactions can start execution once the update notifications arrive, hence they can have very short response time. However, with small-sized update notifications as is in the method *has_popular*, only few transactions can enjoy such benefit. For the other two methods, with similar-sized update notifications, they can have more read-only transactions with short response time

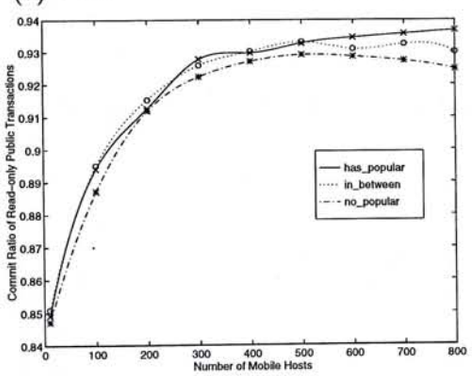
owing to the reason stated above. With more mobile hosts included in the system, more data are updated during each period. This further increases the difference in the size of update notifications between the methods *in_between* and *has_popular*. Hence the overall response time of read-only transactions for the two methods diverge.



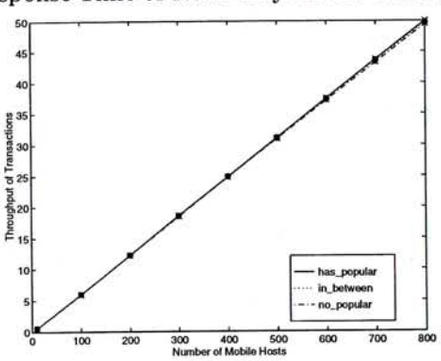
(a) Utilization of the Wireless Channels



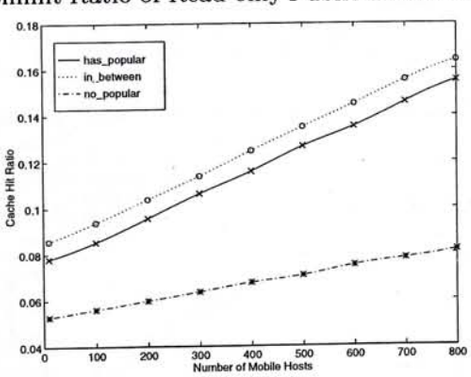
(b) Response Time of Read-only Public Transactions



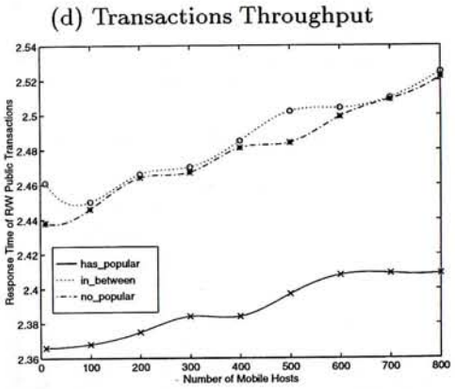
(c) Commit Ratio of Read-only Public Transactions



(d) Transactions Throughput



(e) Cache Hit Ratio

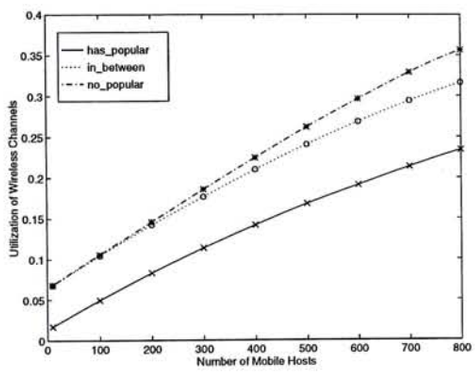


(f) Response Time of R/W Public Transactions

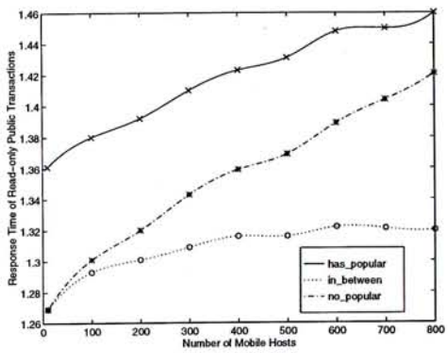
Figure 6.8: Performance of the Scheme with Different Access Patterns (RW_PROB = 0.1)

On the other hand, it is found that the response time of read-only transactions for the method *no_popular* converges with that of the method *has_popular* as the number of

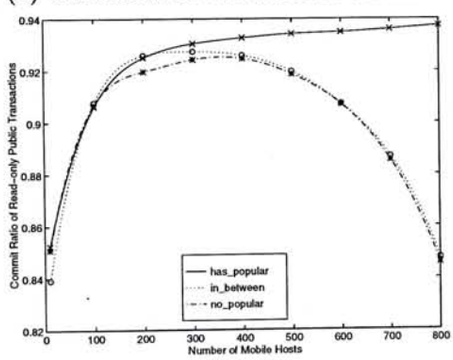
mobile hosts increases. It is because of the low cache hit ratio of the method *no_popular*. As shown in Figure 6.8 (e), the cache hit ratio for the method *no_popular* increases slower as compared with the other two methods, and maintains at a low level. As a result, more data are missed and more messages have to be sent. This is reflected from Figure 6.8 (a). The response time of the read-only public transactions for this method hence increases more than that of the others with the increase in the number of mobile hosts.



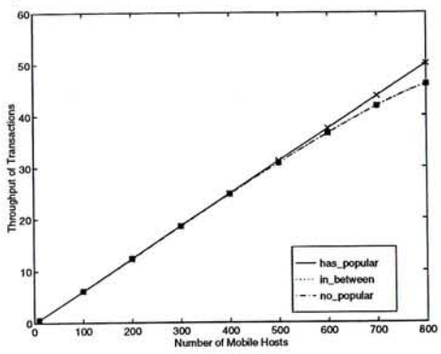
(a) Utilization of the Wireless Channels



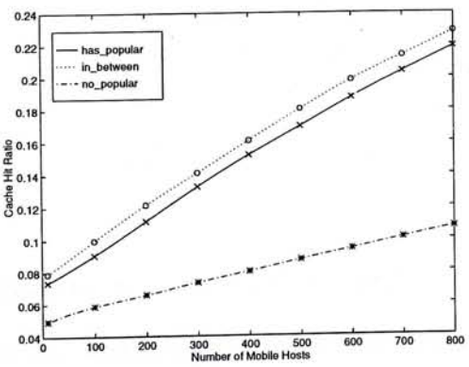
(b) Response Time of Read-only Public Transactions



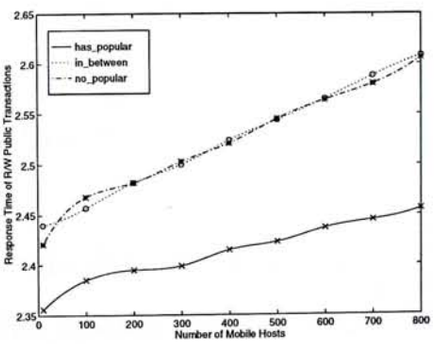
(c) Commit Ratio of Read-only Public Transactions



(d) Transactions Throughput



(e) Cache Hit Ratio



(f) Response Time of R/W Public Transactions

Figure 6.9: Performance of the Scheme with Different Access Patterns (RW_PROB = 0.2)

In order to observe how the methods perform with a larger set of updated data, the value of RW_PROB is changed from 0.1 to 0.2 and the same experiment as above is performed. Results of the experiment are shown in Figure 6.9.

It can be found in Figure 6.9 that the commit ratio of read-only transactions for the methods *in_between* and *no_popular* start to drop when the system has more than 300 mobile hosts. This shows that in a system with high update rate, the method *has_popular* can achieve higher commit ratio of read-only public transactions, but with the tradeoff of higher transaction response time.

Experiment 7 Cache Update Vs Cache Invalidation

This experiment is to determine whether the data values of updated data should be included in the update notifications. Transactions submitted at the mobile hosts follow the 80/20 access model here. Three models are examined in order to observe their difference in behaviour.

Parameter	**Value**
TRS_PERIOD	1.5 seconds
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
CONNECT_PROB	0.95
PUB_ARR_TIME	5.0 seconds
L_BATCH_TIME	0.8
U_BATCH_TIME	1
RW_PROB	0.1
HAND_OFF	1500.0 seconds

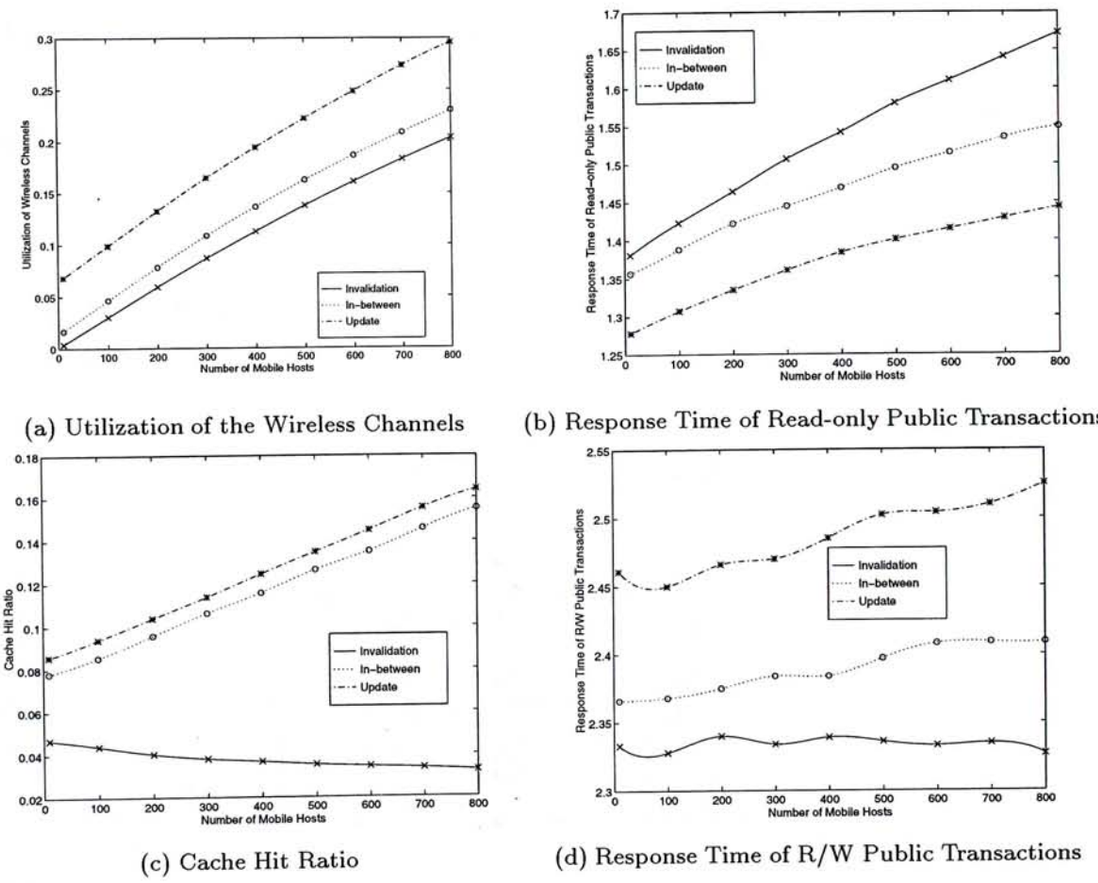
Table 6.9: Variable Parameters Setting (experiment 7)

- *Invalidation.* The update notifications contain no data values of the updated data objects. Only their data IDs are included. The update notifications now serve for invalidation instead of update.
- *In-between.* The update notifications contain data values of the updated popular data objects. Updated unpopular data objects will only have their data IDs included

in the update notifications. This is in fact the method *has_popular* used in the previous experiment.

- *Update.* The update notifications contain data values of all updated data objects. Data cached can then be updated. This is the method *in_between* used in the previous experiment.

Table 6.9 shows the values of the variable parameters used in this experiment. Results of the experiment are shown in Figure 6.10.



(a) Utilization of the Wireless Channels

(b) Response Time of Read-only Public Transactions

(c) Cache Hit Ratio

(d) Response Time of R/W Public Transactions

Figure 6.10: Cache Update Vs Cache Invalidation

It can be observed from the results that the response time of read-only public transactions is the best for the case with the largest set of data values included in the update notifications. From Figure 6.10 (b), it is found that the response time of read-only transactions for the method *Update* is smaller than that of the method *In_between*, which in

turn is smaller than that of the method *Invalidation.* It is owing to the gain in response time by the transactions submitted during the broadcasting of update notifications as mentioned in the previous experiment. On the other hand, it can be observed that the response time of read-only transactions for the method *Invalidation* diverges from the other two. This can be explained by the low cache hit ratio of the method which results in more cache misses, hence more data have to be requested from the MSSs.

Besides, it can be seen from Figure 6.10 (d) that the response time of the R/W public transactions for the two methods *Invalidation* and *In_between* are quite close. This can be explained by the small difference in the size of update notifications broadcasted for the two methods as shown in Figure 6.10 (a).

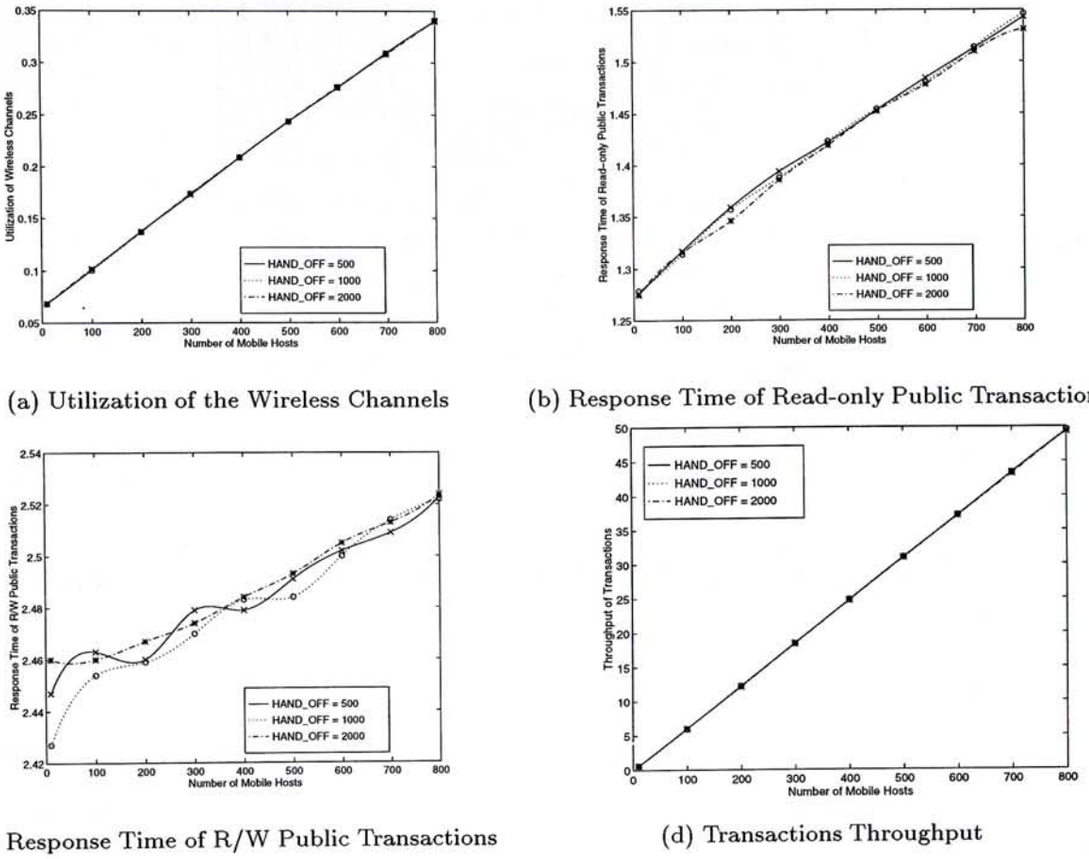
Experiment 8 Variation of HAND_OFF

This experiment is to study the performance of the scheme with different frequencies of mobile hosts moving from one cell to another. The values of the variable parameters are shown in Table 6.10.

Parameter	**Value**
TRS_PERIOD	1.5 seconds
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
CONNECT_PROB	0.95
PUB_ARR_TIME	5.0 seconds
L_BATCH_TIME	0.8
U_BATCH_TIME	1
RW_PROB	0.1

Table 6.10: Variable Parameters Setting (experiment 8)

Results of the experiment with values of HAND_OFF being 500, 1000 & 2000 seconds are shown in Figure 6.11. It is found that the curves are very close for all the cases. This shows that given the movement of mobile hosts is not vigorous, it has little influence on the performance of the scheme.



(a) Utilization of the Wireless Channels

(b) Response Time of Read-only Public Transactions

(c) Response Time of R/W Public Transactions

(d) Transactions Throughput

Figure 6.11: Variation of HAND_OFF

Experiment 9 Variation of Clock Synchronization among Fixed Hosts

This experiment is to study the effect of clock synchronization among the fixed hosts on the performance of the scheme. It is cited in [14] that the clocks of fixed hosts can be synchronized to within a few milliseconds of one another. The scheme is tested with the maximum difference between any two clocks of fixed hosts to be within 0, 3, 5, 7 & 9 seconds.

Table 6.11 shows the values of the variable parameters used in the experiment. Figure 6.12 shows the results of the experiment.

It is found that the difference in performance of the scheme between a perfectly synchronized system and a system with the difference between any two fixed hosts' clocks to be about a few milliseconds is very small.

Parameter	Value
TRS_PERIOD	1.5 seconds
ARR_TIME	15.0 seconds
COLLECTION_PERIOD	0.4 seconds
CONNECT_PROB	0.95
PUB_ARR_TIME	5.0 seconds
L_BATCH_TIME	0.8
U_BATCH_TIME	1
RW_PROB	0.1
HAND_OFF	1500.0 seconds

Table 6.11: Variable Parameters Setting (experiment 9)

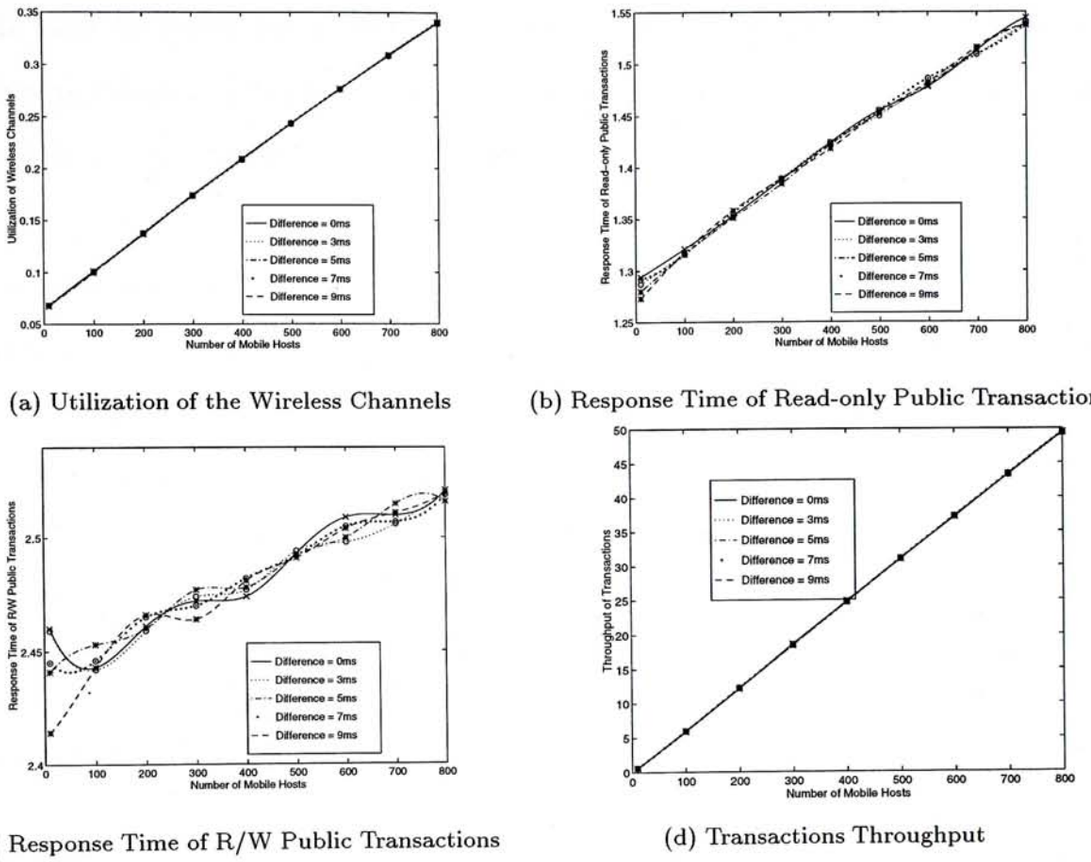
6.3 Comparison with the Lock-based Scheme

In this section, performance of the proposed scheme is compared with that of the lock-based scheme as mentioned in chapter 5. The objective of the experiments is to investigate whether read-only transactions should be executed at the mobile hosts and whether locking is a suitable method to be used in the mobile computing environments.

In order to have the two schemes comparable, the following assumptions are made to the system:

- all data objects in the database are public data objects;
- the wireless channel is reliable, hence all the messages sent through it should be received by the recipients;
- the mobile hosts are not allowed to move from one cell to another;

It is shown in the previous section that the movement of mobile hosts from one cell to another has little effect on the performance of our scheme. For the lock-based scheme, it is believed that the movement of mobile hosts will increase the message cost in releasing locks. Since the cost of sending messages through the wired channels is assumed to be negligible, mobility of mobile hosts will not affect the performance of the scheme. In order to simplify the simulation, it is not implemented here.



(a) Utilization of the Wireless Channels

(b) Response Time of Read-only Public Transactions

(c) Response Time of R/W Public Transactions

(d) Transactions Throughput

Figure 6.12: Variation of Clock Synchronization among Fixed Hosts

For all the experiments in this section, the database is assumed to be partitioned into popular and unpopular data, and 80% of the data accesses by the transactions submitted at the mobile hosts will be on the popular data.

6.3.1 Parameters Setting

There are 8 fixed hosts in the system. It is assumed that the mobile hosts are evenly distributed among the cells. Transactions submitted at a fixed host have number of operations ranges from 8 to 12, and they access the data in the database with equal probability.

Each message sent to and from a mobile host has a fix-sized header of 30 bytes. This includes the identity of the mobile host, and other information. For the lock-based scheme, this includes also the operation being sent to the fixed host.

The mean time for a mobile host to power off is about 1500 seconds. The duration for the mobile host to remain powered off is exponentially distributed over 100 seconds.

For the lock-based scheme, a read operation executed at the fixed host takes 0.01 seconds, while a write operation takes 0.02 seconds.

Table 6.12 shows the list of parameters together with their values that are common to both of the schemes. Table 6.13 shows the list of parameters used in our scheme, which are equivalent to the ones stated in the previous section. Table 6.14 shows the list of parameters used in the lock-based scheme only. The set of parameters with their values to be varied in the experiments followed are shown in Table 6.15.

Parameter	**Description**	**Value**
Parameters for Fixed Network		
NUM_SERVER	number of fixed hosts in the system	8
F_MIN_OPER	minimum number of operations in a transaction	8
F_MAX_OPER	maximum number of operations in a transaction	12
Parameters related to Message and Time		
ID_SIZE	size of data id	10 bytes
DATA_SIZE	size of a data version	1K bytes
MESG_HEADER	fix-sized header for each message	30 bytes
BANDWIDTH	size of each wireless channel	1×10^6 bps
SER_TM	CPU time for mobile hosts	0.01 seconds
POWER_OFF	mean time for a mobile host to power off	1500.0 seconds
DIS_DURATION	mean time for a mobile host to remain powered off	100.0 seconds

Table 6.12: Values for Parameters Used in Both Schemes

6.3.2 Experiments and Results

In this section, results of a number of experiments carried out are shown. For each of the experiments, utilization of the wireless channels is shown. Besides, the response time, commit ratio and throughput of transactions submitted at the mobile hosts are given. Note that for our scheme, only the commit ratio of read-only public transactions are given, as the R/W public transactions will have commit ratio equals 1 under a system with reliable wireless channels.

Parameter	Description	Value
CACHE_SIZE	cache size	100 data
TRS_PERIOD	length of the TRS period	1.5 seconds
COLLECTION_PERIOD	collection period for the miss-sets	0.4 seconds
DATA_IO	I/O time for mobile hosts	0.035 seconds
W_TIMEOUT	timeout waiting messages in wireless channels	1.5 seconds
L_BATCH_TIME	lower-bound for the execution of global batches	0.8
U_BATCH_TIME	upper-bound for the execution of global batches	1.0
COMMIT_MESG	commit/abort of R/W public transactions	100 bytes
ACK_MESSAGE	acknowledgement of transactions results received	100 bytes
TRAN_SIZE	size of a R/W transaction	100 bytes

Table 6.13: Values for Parameters Used in Our Scheme

Parameter	Description	Value
FH_READ_TIME	time for reading a data object in fixed host	0.01 seconds
FH_WRITE_TIME	time for writing a data object in fixed host	0.02 seconds
OPER_TIMEOUT	timeout in acquiring a lock	50.0 seconds

Table 6.14: Values for Parameters Used in the Lock-based Scheme

Experiment 10 Variation of Database Size

This experiment is to study the effect of database size on the performance of the two schemes. The database is varied with size equals 1500, 3000, 4500, 5000 and 6000 data objects. Table 6.16 shows the values of the variable parameters. Results of the experiment are shown in Figure 6.13.

It is found that the performance of the lock-based scheme is inferior to ours and their difference is very large. Besides, it is found that the performance of our scheme is indifferent to the variation of database size, whereas the lock-based scheme performs better with a larger database. It is because with a larger database size, the number of data conflicts reduces, which improves both the response time and the commit ratio of transactions.

On the other hand, it is found that the lock-based scheme has a better utilization of the wireless channels than ours, and its value is kept low. This can be explained by the

Parameter	Description
PUB_ARR_TIME	inter-arrival time of public transactions in fixed host
NUM_DATA	number of data objects in the database
NUM_MOBILE	number of mobile hosts in the system
POPULARITY	percentage of popular data in the database
RW_PROB	percentage of R/W transactions submitted at mobile hosts
ARR_TM	inter-arrival time of public transactions at mobile hosts
MIN_OPER	minimum number of operations per transaction at mobile hosts
MAX_OPER	maximum number of operations per transaction at mobile hosts

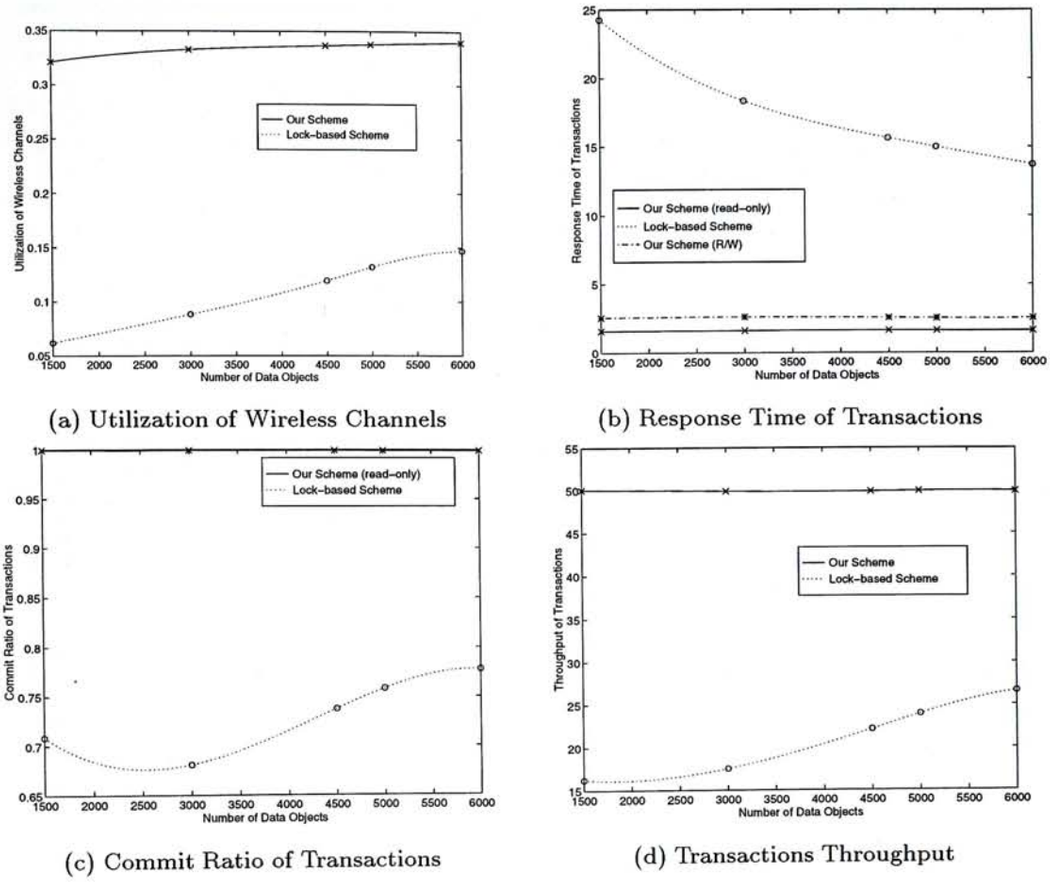
Table 6.15: Variable Parameters

Parameter	Value
PUB_ARR_TIME	5.0 seconds
NUM_MOBILE	800
POPULARITY	0.2
RW_PROB	0.1
ARR_TM	15.0 seconds
MIN_OPER	4
MAX_OPER	8

Table 6.16: Variable Parameters Setting (experiment 10)

large sets of data being broadcasted in the update notifications in our scheme and the low commit ratio of the lock-based scheme, both of which increase the difference in the number and size of messages being sent through the wireless channels.

Figure 6.14 gives the results of the experiment with the size of database equals 6000, and the number of mobile hosts varies from 100 to 800. It is found that the performance of the lock-based scheme is better than ours when the system has fewer than 200 mobile hosts. It is owing to the small number of public transactions submitted at the mobile hosts, which reduces the number of data conflicts. In order to further verify the effect of conflict ratio on the performance of the lock-based scheme, the same experiment with value of RW_PROB set to 0 is carried out. Results of the experiment are shown in Figure 6.15.



(a) Utilization of Wireless Channels

(b) Response Time of Transactions

(c) Commit Ratio of Transactions

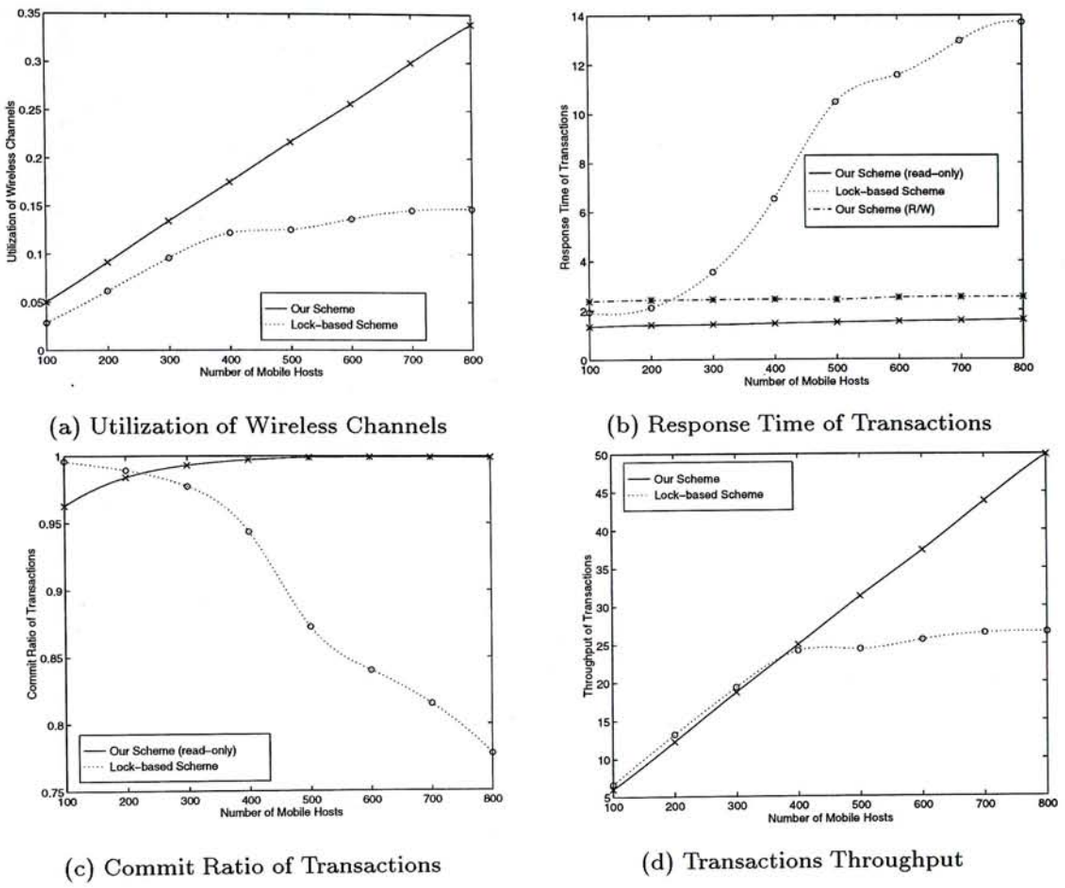
(d) Transactions Throughput

Figure 6.13: Variation of Database Size

In Figure 6.15, it is found that the performance of the lock-based scheme becomes better than ours, especially when the database size reaches 3000. Besides, as the size of database increases, the commit ratio of transactions for the lock-based scheme converges to 1. This shows that the conflict ratio affects the performance of the lock-based scheme greatly.

Experiment 11 Variation of Popularity Ratio

This experiment is to study the performance of the two schemes with different number of popular data in the database. The percentage of popular data in the database is varied from 0.2 to 1. The size of database is fixed to 3000 data objects. The values of other variable parameters are shown in Table 6.17. Figure 6.16 shows the results of the experiment.



(a) Utilization of Wireless Channels

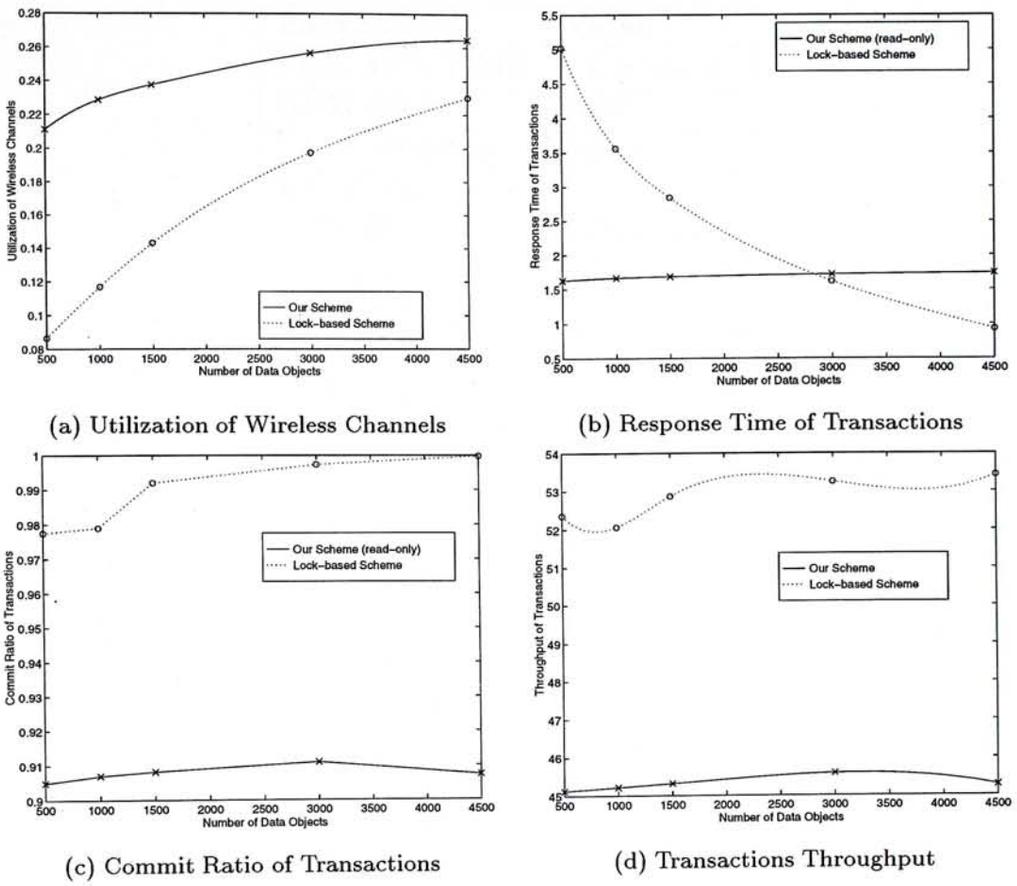
(b) Response Time of Transactions

(c) Commit Ratio of Transactions

(d) Transactions Throughput

Figure 6.14: Variation of Database Size (6000 data objects)

It is found that the increase in popularity ratio has a positive effect on the lock-based scheme, but a negative effect on ours. As shown in Figure 6.16 (c), the commit ratio of the read-only public transactions for our scheme starts to drop when the value of POPULARITY is about 0.9; whereas the commit ratio for the lock-based scheme keeps on rising until the value of POPULARITY reaches 0.8, where it starts to flatten. This can be explained by the increased size of update notifications being broadcasted in our scheme, which increases the message delay. As a result, the MSSs may fail to receive the miss-sets from the mobile hosts, and the read-only transactions requiring data missed have to abort. However, increasing popularity ratio of the database reduces the data conflicts for the lock-based scheme, which causes the improvement of its performance. This also reflects that under this system setting, having 80% of the data in the database as popular data is the optimal value of the lock-based scheme.



(a) Utilization of Wireless Channels

(b) Response Time of Transactions

(c) Commit Ratio of Transactions

(d) Transactions Throughput

Figure 6.15: Variation of Database Size with Read-only Transactions Only

Experiment 12 Variation of Inter-arrival Time of Transactions at Mobile Hosts

This experiment is to study the performance of the schemes with different work loads from the mobile hosts. The mean time for the submission of transactions at the mobile hosts is given the values 10, 15, 20, 30, 70 & 100 seconds. The values of other variable parameters are shown in Table 6.18. Results of the experiment are shown in Figure 6.17.

It is found in Figure 6.17 that the performance of the lock-based scheme improves as the inter-arrival time of transactions at the mobile hosts increases. This can be explained by the decreased number of data conflicts which improves both the commit ratio and the response time of transactions. On the other hand, it is found that as the inter-arrival time of transactions increases, the commit ratio of our scheme decreases. This is owing to the decreased number of R/W transactions submitted at the mobile hosts, which results in fewer data being updated, especially the popular data. More data are therefore missed

Parameter	Value
PUB_ARR_TIME	5.0 seconds
NUM_DATA	3000
NUM_MOBILE	800
RW_PROB	0.1
ARR_TM	15.0 seconds
MIN_OPER	4
MAX_OPER	8

Table 6.17: Variable Parameters Setting (experiment 11)

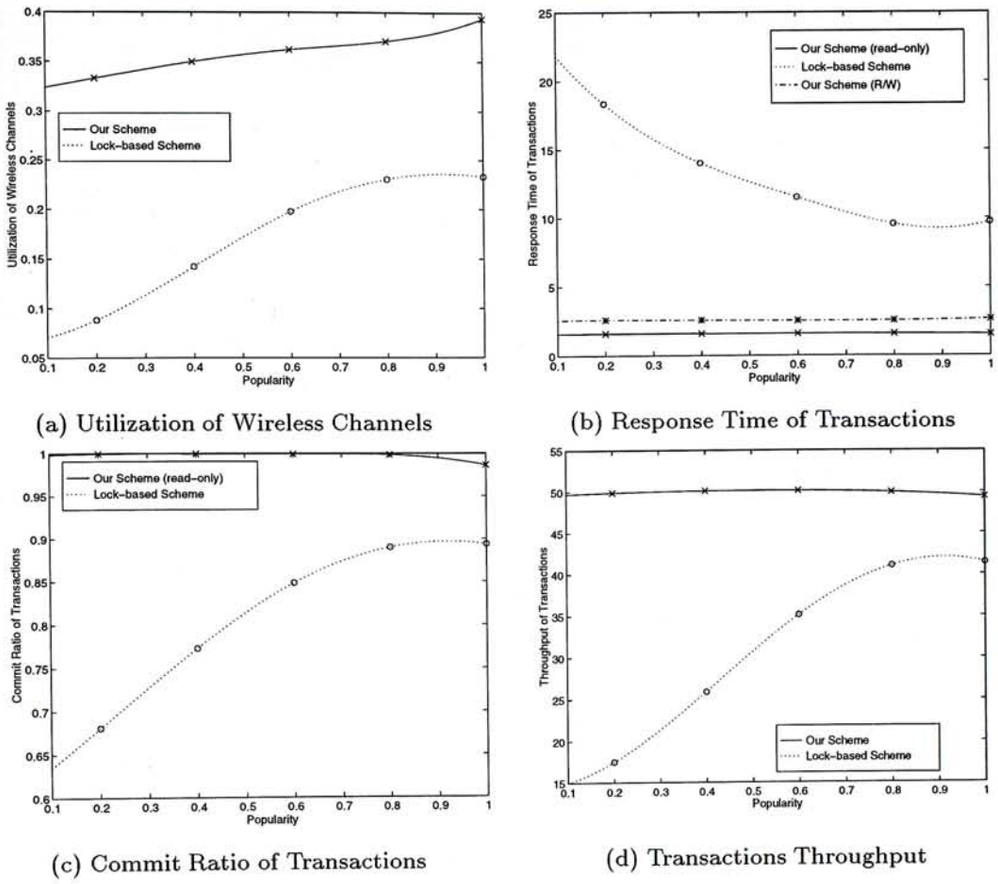
Parameter	Value
PUB_ARR_TIME	5.0 seconds
NUM_DATA	3000
NUM_MOBILE	800
POPULARITY	0.2
RW_PROB	0.1
MIN_OPER	4
MAX_OPER	8

Table 6.18: Variable Parameters Setting (experiment 12)

in the cache of the mobile hosts and more request messages have to be sent to the MSSs for the missed data, the commit ratio hence decreases.

It is shown in the previous experiments that data conflicts contributes greatly to the difference in performance between the two schemes. The same experiment is therefore carried out with the value of RW_PROB set to 0 in order to observe the difference in performance with fewer data conflicts for the lock-based scheme. Results of the experiment are shown in Figure 6.18.

It is found that as long as the inter-arrival time of transactions at the mobile hosts is smaller than 30 seconds, the lock-based scheme outperforms ours in handling the read-only transactions. This shows that given a system with large database size and all the transactions from the mobile hosts contain read operations only, the lock-based scheme can provide a good performance.



(a) Utilization of Wireless Channels

(b) Response Time of Transactions

(c) Commit Ratio of Transactions

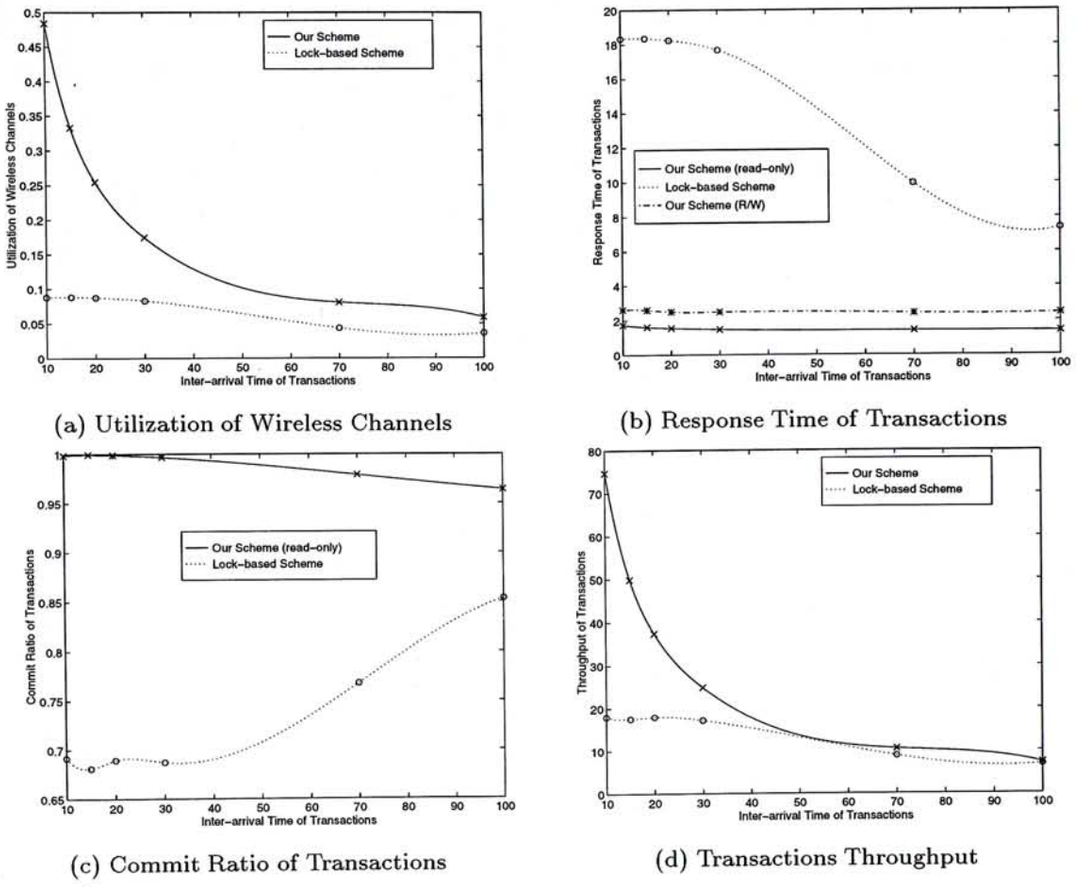
(d) Transactions Throughput

Figure 6.16: Variation of Popularity Ratio

However, as the inter-arrival time of transactions reaches 30 seconds, the response time of transactions for the lock-based scheme starts to rise. It is because with fewer transactions submitted at the mobile hosts, the probability that a particular data is being locked exclusively by some transaction from the fixed hosts increases, hence much time is required in waiting for the lock to be released by the transaction.

Experiment 13 Variation of Inter-arrival Time of Transactions at Fixed Hosts

This experiment is to study the effect of the number of transactions executed at the fixed hosts on the performance of transactions submitted at the mobile hosts. The mean inter-arrival time of transactions from the fixed hosts is given the values 5, 10, 15, 20, 30, 50, & 70 seconds. Table 6.19 gives the values of other variable parameters used. Results of the experiment are shown in Figure 6.19.



(a) Utilization of Wireless Channels

(b) Response Time of Transactions

(c) Commit Ratio of Transactions

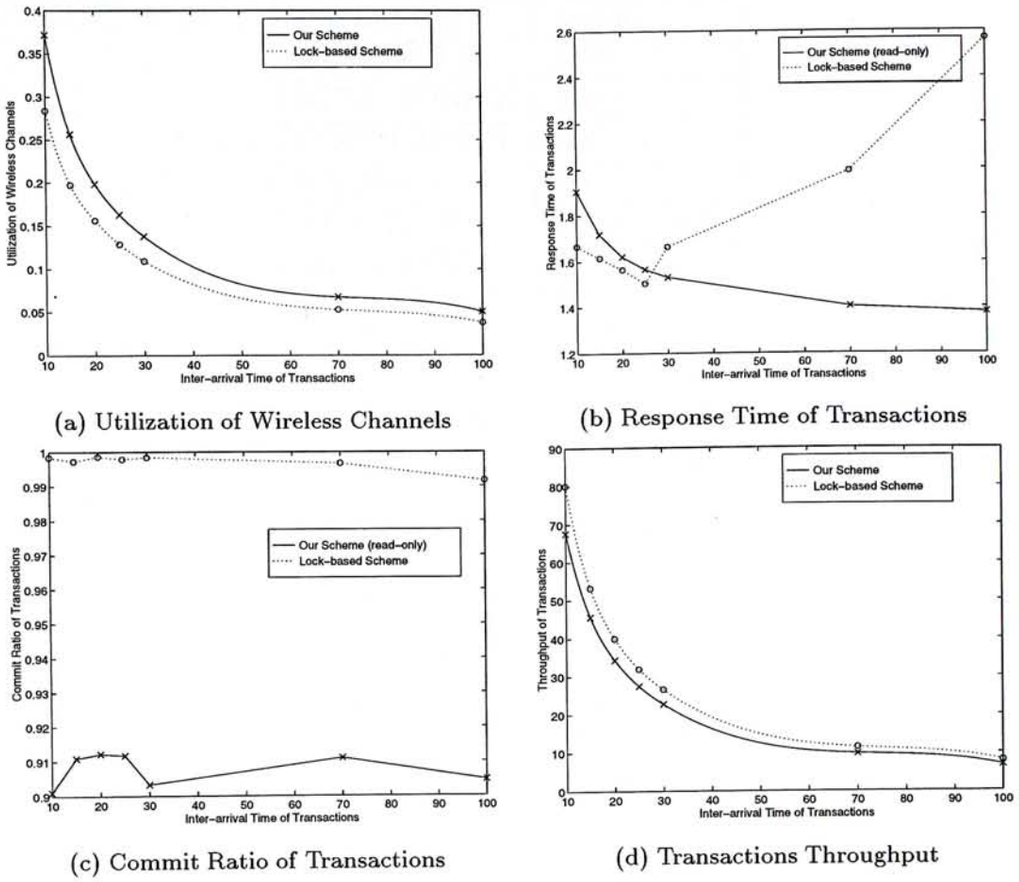
(d) Transactions Throughput

Figure 6.17: Variation of Inter-arrival Time of Transactions at Mobile Hosts

It is shown in Figure 6.19 that changing the inter-arrival time of transactions at the fixed hosts does not seem to have any effect on the performance of the two schemes. This is owing to the large number of transactions submitted at the mobile hosts which buried their effects.

Experiment 14 Variation of Number of Operations per Transaction at Mobile Hosts

This experiment is to study whether the size of transactions submitted at the mobile hosts will have any effect on the performance of the two schemes. The experiment is carried out with number of operations for transactions submitted at the mobile hosts having the ranges of 2-6, 4-8, 6-8 & 8-12. Values of other variable parameters are shown in Table 6.20. Results of the experiment are shown in Figure 6.20.



(a) Utilization of Wireless Channels

(b) Response Time of Transactions

(c) Commit Ratio of Transactions

(d) Transactions Throughput

Figure 6.18: Variation of Inter-arrival Time of Transactions at Mobile Hosts (read-only)

It is shown in Figure 6.20 that increasing the number of operations in the transactions submitted at the mobile hosts has an adverse effect on the performance of the lock-based scheme, while our scheme shows little change in performance. This can be explained by the increased number of data conflicts which worsens the performance of the transactions executed under the lock-based scheme. Besides, it is shown in Figure 6.20 (a) that the utilization of wireless channels for our scheme increases with the number of operations per transaction, while that of the lock-based scheme remains more or less constant with the the change in the number of operations. This can be explained by the increased number of data missed in the cache of the mobile hosts, which results in more request messages to be sent through the wireless channels for our scheme. Nevertheless, it is shown in Figure 6.20 (c) that the commit ratio of transactions for the lock-based scheme is decreasing. This results in fewer messages to be sent through the wireless channels, hence the utilization

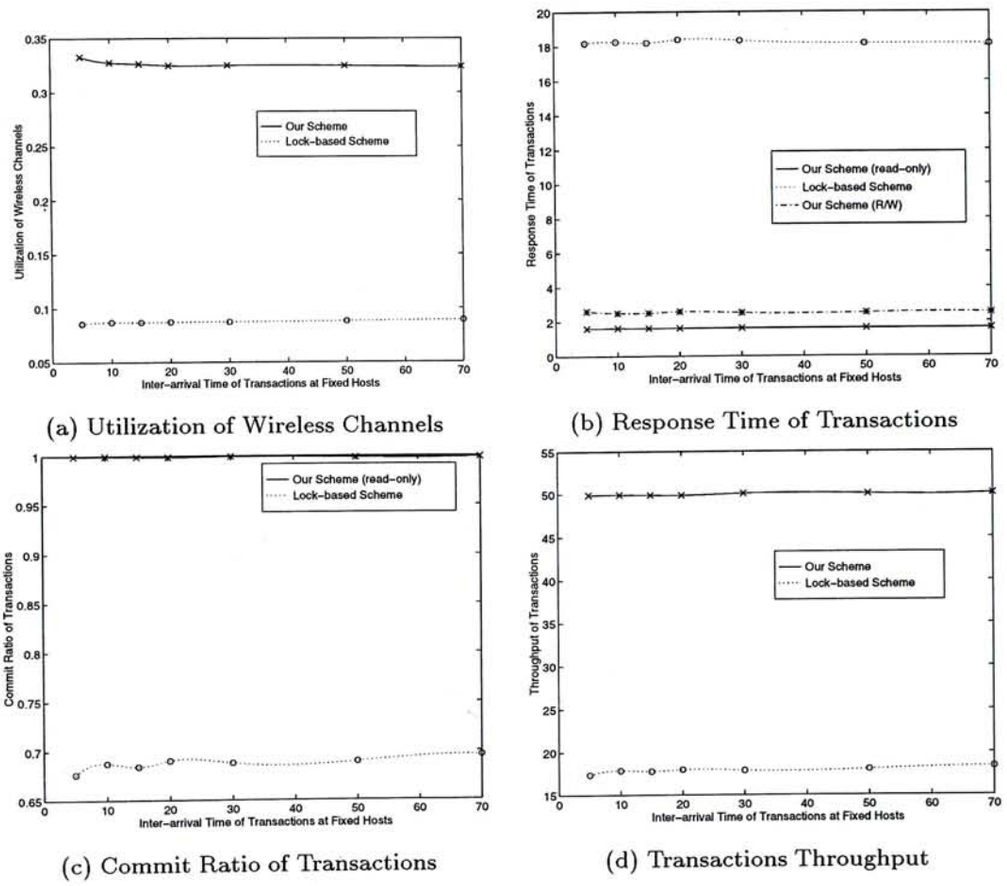
Parameter	**Value**
NUM_DATA	3000
NUM_MOBILE	800
POPULARITY	0.2
RW_PROB	0.1
ARR_TIME	15.0 seconds
MIN_OPER	4
MAX_OPER	8

Table 6.19: Variable Parameters Setting (experiment 13)

Parameter	**Value**
PUB_ARR_TIME	5.0 seconds
NUM_DATA	3000
NUM_MOBILE	800
POPULARITY	0.2
RW_PROB	0.1
ARR_TIME	15.0 seconds

Table 6.20: Variable Parameters Setting (experiment 14)

of wireless channels shows no increase.



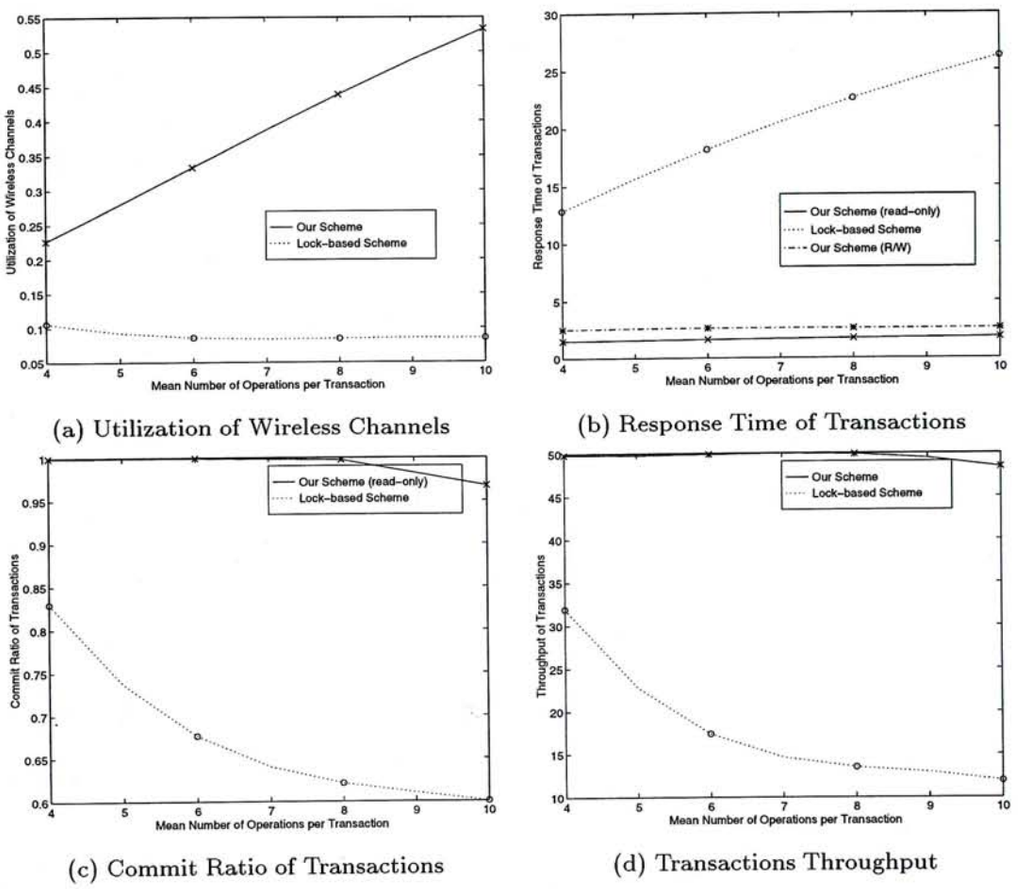
(a) Utilization of Wireless Channels

(b) Response Time of Transactions

(c) Commit Ratio of Transactions

(d) Transactions Throughput

Figure 6.19: Variation of Inter-arrival Time of Transactions at Fixed Hosts



(a) Utilization of Wireless Channels

(b) Response Time of Transactions

(c) Commit Ratio of Transactions

(d) Transactions Throughput

Figure 6.20: Variation of Number of Operations per Transaction

Chapter 7

Conclusions and Future Work

7.1 Conclusions

We have developed a scheme for transaction management in the mobile computing environments. We assume the underlying network protocol does not handle message loss and the scheme developed has to get along with it. The scheme classifies the transactions submitted at the mobile hosts into two types, with those consisting of read operations only being the read-only transactions and those containing reads and writes the R/W transactions. Read-only transactions will not update the database and can therefore be serviced by the data cached in the mobile hosts. R/W transactions, owing to the presence of writes, may conflict with other transactions. It is shown in [28] that TRS behaves well in conflict cases. R/W transactions are therefore sent to the fixed hosts for execution under TRS.

Data cached in the mobile hosts have to be updated periodically in order to have the transactions executed there serializable with others. An update notification containing the data updated by the latest committed global batch in the past TRS period is broadcasted to the mobile hosts at the end of the period. The cache of each mobile host is updated with the set of data received in the update notification and is then used by the transactions collected by the mobile host in execution. A request message is sent to the MSS if some transaction has required data absent in the cache. In order to save the wireless bandwidth, missed data are collected in batch and sent to the MSS by each mobile host.

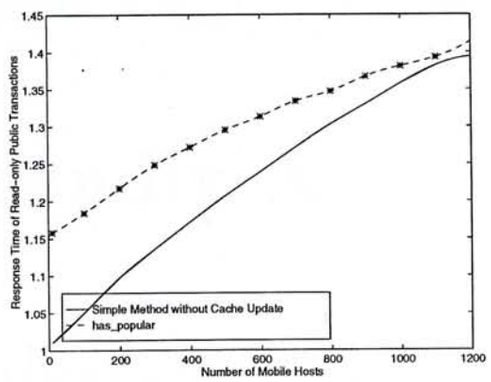
A simulation is conducted to study the behaviour and verify the correctness of the scheme. The study shown that the length of TRS period and the execution time of global batches affect the performance of the scheme greatly. While a long TRS period increases the response time of transactions, a TRS period which is shorter than the execution time of a global batch reduces the commit ratio of transactions. Besides, it is shown that collecting the missed data and sending them in batches can greatly improve the performance of the scheme.

When compared with the lock-based scheme, it is found that locking is feasible in mobile computing environments only when the transactions from mobile hosts contain read operations only and the size of database is large, in which case the number of data conflicts is the lowest. Otherwise, our scheme out-performs it in many dimensions.

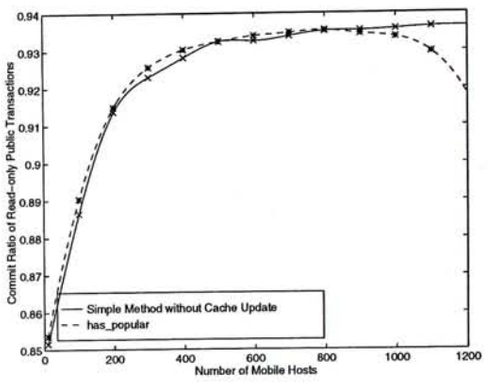
7.2 Future Work

It is found that our scheme suffers from the problem of low cache hit ratio, which increases the response time of transactions executed at the mobile hosts. Consider a simple method which does not require cache update or invalidation, and instead, each mobile host sends the readset of all the collected read-only transactions to its local MSS. Besides, a message similar to the update notification but without any information of updated data is broadcasted by each MSS at the end of the TRS period when a global batch completes its execution. All the cached data are purged when a mobile host receives this message. Figure 7.1 shows the performance of the method with the period of time that a MSS collects the readsets of read-only transactions equals 0.02 seconds. It is compared with the method *has_popular* as mentioned in Experiment 6 of Chapter 6, with the length of collection period equals 0.2 seconds.

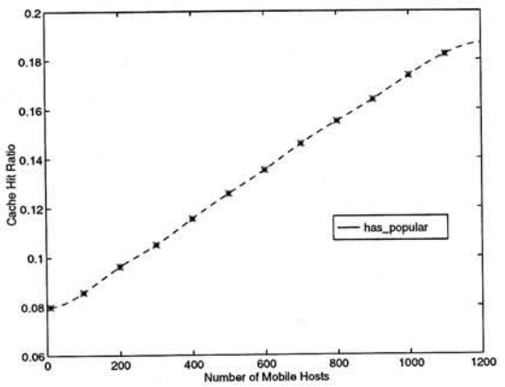
It is found that when the number of mobile hosts are small, sending a large number of data objects to the mobile hosts will result in a longer transaction response time than the case where cached data are simply purged. This is owing to some useless data versions being included in the update notifications in the method *has_popular*, which



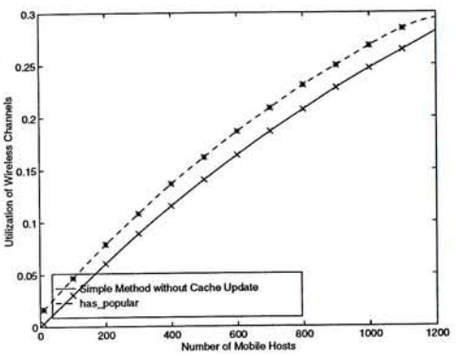
(a) Response Time of Read-only Public Transactions



(b) Commit Ratio of Read-only Public Transactions



(c) Cache Hit Ratio



(d) Utilization of Wireless Channels

Figure 7.1: Comparison with the Method without Cache Update and Invalidation

incurs some delays in the execution of transactions. However, as the number of mobile hosts increases, their difference reduced. This is because for the method *has_popular*, increasing the number of mobile hosts increases the chance that a particular data in the update notification will be used by the transaction of some mobile host. Besides, cached data will also serve to reduce the execution time of transactions. This results in a lower increase rate in transaction response time of the method *has_popular*.

It is believed that if the cache hit ratio is increased, performance of our scheme can be improved. This can be achieved if we have a prior knowledge of the access patterns of mobile hosts. If their access patterns are known, the data to be included in the update notifications can be adjusted to suit the requirements of the mobile hosts. The cache hit ratio hence increases, which in turn both increases the response time and commit ratio of transactions submitted at the mobile hosts. This problem deserves further study.

Appendix A

Implementation Details

In this chapter, some implementation details of the simulation program written are described. The logical model for the system and mobile hosts are illustrated in Figure A.1 & A.2.

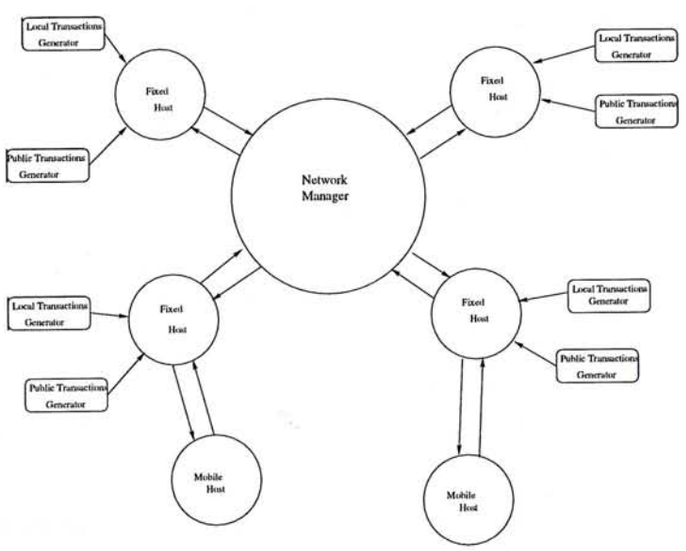


Figure A.1: Simulation Model

Two transaction generators are used to generate the local and public transactions for each fixed host. The transactions are then executed by the fixed hosts and their updated data are broadcasted to the mobile hosts periodically. The following CSIM processes simulate different activities performed by the fixed hosts:

- **fh_create_loc_tranx** generates the local transactions for each fixed host throughout the simulation time. The transactions generated are then appended at the end of

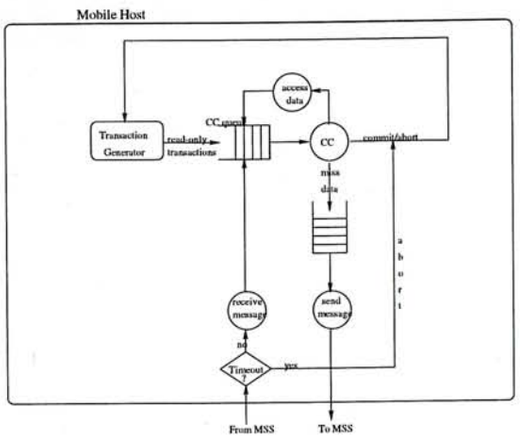


Figure A.2: Simulation Model of a Mobile Host

a queue for holding the local transactions of that particular fixed host. The data written by the transactions will then be collected and broadcasted to the mobile hosts afterwards.

- **fh_create_pub_tranx** generates the public transactions for each fixed host. The public transactions generated by the fixed hosts are then appended at the end of a single queue which acts as the global batches collected.

- **fh_exec_batch** executes the global batches generated by generating a random time period between the lower and upper bounds of the execution time allowed and advances the variable MSS_time after the given amount of time passed.

- **fh_collect_data** collects all the data to be included in the update notification at the end of the TRS period when MSS_time is advanced by searching through all the transaction queues.

- **fh_broadcast_data** broadcasts the set of data collected in fh_collect_data by each fixed host to the mobile hosts residing in its cell.

- **fh_broadcast_missset** collects the miss-sets from the mobile hosts in the MSS's cell and broadcasts the set of data missed at the end of the collection period.

For the mobile hosts, a transaction generator generates the public transactions for

each mobile host. The following CSIM processes simulate different activities performed by the mobile hosts.

- **mh_hand_off** governs the movement of a mobile host from one cell to another throughout the simulation time.
- **mh_create_pub_tranx** generates the public transactions for each mobile host. The read-only transactions are then appended at the end of a queue for holding the transactions at that particular mobile host, while R/W transactions are sent to the local MSS by the process mh_send_pub_tranx.
- **mh_send_pub_tranx** sends the R/W public transactions generated by each mobile host to the local MSS. They are then appended at the end of the queue for holding the global batches.
- **mh_caching** receives the update notifications broadcasted from the MSSs and updates the data cached at the mobile host.
- **mh_send_missed** collects the miss-set and sends it to the local MSS.
- **mh_receive_missed** receives the replies of miss-sets from the MSS and caches the data in the mobile host.
- **mh_get_pub_tran** & **mh_exec_pub_tran** execute the read-only transactions collected at the mobile host. If data is found to be missed in the cache after the reply of miss-sets is received, the process mh_cache_miss is called to request for the missed data in the local MSS.
- **mh_cache_miss** triggers another process mh_cache_upload to send the missed data ID to the MSS and waits for the reply. The process mh_exec_pub_tran is notified if the reply can be received within the timeout period.
- **mh_cache_upload** performs the sending of the missed data ID to the MSS and checks whether the variable MSS_time is advanced. If so, the process terminates; otherwise, the reply is received and it is cached in the mobile host.

Bibliography

[1] Norman Abramson. Multiple Access in Wireless Digital Networks. *Proceedings of the IEEE*, 82(9):1360–1369, September 1994.

[2] Mustaque Ahamad and Shawn Smith. Detecting Mutual Consistency of Shared Objects. In *Proceedings of the 1994 Workshop on Mobile Computing Systems and Applications*, December 1994.

[3] Rafael Alonso and Henry F. Korth. Database System Issues in Nomadic Computing. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, May 1993.

[4] B. R. Badrinath, Arup Acharya, and Tomasz Imieliński. Structuring Distributed Algorithm for Mobile Hosts. In *Proceedings of the 14th Intl. Conf. on Distributed Computing System*, June 1994.

[5] B. R. Badrinath and T. Imielinski. Replication and Mobility. In *Proceedings of the Second IEEE Workshop on the Management of Replicated Data*, November 1992.

[6] Daniel Barbará and Tomasz Imieliński. Sleepers and Workaholics: Caching Strategies in Mobile Environments. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 1994.

[7] Daniel Barbará and Tomasz Imieliński. Sleepers and Workaholics: Caching Strategies in Mobile Environments (extended version). *MOBIDATA: An Interactive journal of mobile computing*, 1(1), November 1994. Available through the WWW, http://rags.rutgers.edu/journal/cover.html.

[8] Philip Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Reading, Mass. : Addison-Wesley Pub. Co., 1987.

[9] Charles Sheung-hing Cheng and Derek Ming-fan Lam. Transaction processing in mobile computing system. Final Year Project Report, 1996.

[10] Panos K. Chrysanthis. Transaction Processing in Mobile Computing Environment. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 77–83, October 1993.

[11] A. El-Abbadi and S. Toueg. Maintaining Availability in Partitioned Replicated Databases. *ACM Transactions on Database Systems*, 14(2):264–290, June 1989.

[12] George H. Forman and John Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(6):38–47, April 1994.

[13] Ada Fu, John C.S. Lui, and M.H. Wong. Dynamic Policies in Selecting a Caching Set for a Distributed Mobile Computing Environment. Technical Report CS-TR-95-06, Dept. of Comp. Science, the Chinese University of Hong Kong, 1995.

[14] Ada Wai-chee Fu and David Wai-lok Cheung. A Transaction Replication Scheme for a Replicated Database with Node Autonomy. In *Proceedings of the 20th VLDB Conference*, pages 214–225, 1994.

[15] Jim Gray. The Cost of Messages. In *Proceedings of the 7th ACM Annual Symposium on Principles of Distributed Computing*, pages 1–7, 1988.

[16] Yixiu Huang, Prasad Sistla, and Ouri Wolfson. Data Replication for Mobile Computers. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 13–24, 1994.

[17] Larry B. Huston and Peter Honeyman. Disconnected Operation for AFS. In *Proceedings USENIX Symposium on Mobile and Location-Independent Computing*, pages 1–10, August 1993.

[18] Tomasz Imieliński and B. R. Badrinath. Data Managment for Mobile Computing. *SIGMOD Record*, 22(1):34–39, March 1993.

[19] Tomasz Imieliński and B. R. Badrinath. Mobile Wireless Computing: Solutions and Challenges in Data Management. Technical Report DCS-TR-296, Dept. of Comp. Science, Rutgers University, 1993.

[20] Tomasz Imieliński and B. R. Badrinath. Mobile Wireless Computing: Challenges in Data Management. *Communications of the ACM*, 37(10):19–28, October 1994.

[21] John Ioannidis and Gerald Q Maguire. The Design and Implementation of a Mobile Internetworking Architecture. In *Proceedings of the USENIX Winter 1993 Technical Conference*, January 1993.

[22] Jin Jing, Omran Bukhres, and Ahmed Elmagarmid. Distributed Lock Management for Mobile Transactions. Technical Report CSD-TR-94-073, Department of Computer Science, Purdue University, 1994.

[23] Jin Jing, Omran Bukhres, Ahmed Elmagarmid, and Rafael Alonso. Bit-Sequences: A New Cache Invalidation Method in Mobile Environments. Technical Report CSD-TR-94-074, Computer Sciences Department, Purdue University, 1995. Available through the WWW, http://www.cs.purdue.edu/homes/jing/bits.ps.

[24] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[25] Henning Koch, Lars Krombholz, and Oliver Theel. A Brief Introduction into the World of 'Mobile Computing'. Technical Report THD-BS-1993-03, Department of Computer Science, University of Darmstadt, 1993.

[26] Henry F. Korth and Abraham Silberschatz. *Database System Concepts.* McGraw-Hill Book Co., second edition, 1991.

[27] Vivek R. Narasayya. Distributed Transactions in a Mobile Computing System. Submitted as part of the requirement for CSE552, 1993.

[28] Ching-ting Ng. Performance Study of Protocols in Replicated Database. Master's thesis, Computer Science and Engineering Department, the Chinese University of Hong Kong, 1996.

[29] Kaveh Pahlavan and Allen H. Levesque. Wireless Data Communications. *Proceedings of the IEEE*, 82(9):1398–1430, September 1994.

[30] Evaggelia Pitoura and Bharat Bhargava. Dealing with Mobility: Issues and Research Challenges. Technical Report TR-93-070, Dept. of Comp. Sciences, Purdue University, 1993.

[31] Evaggelia Pitoura and Bharat Bhargava. Building Information Systems for Mobile Environments. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 371–378, November 1994.

[32] Evaggelia Pitoura and Bharat Bhargava. Maintaining Consistency of Data in Mobile Distributed Environments. Technical Report TR-94-025, Dept. of Comp. Science, Purdue University, 1994.

[33] Evaggelia Pitoura and Bharat Bhargava. Revising Transaction Concepts for Mobile Computing. Position Paper, 1994. Purdue University, Dept. of Comp. Science.

[34] M. Satyanarayanan, James J. Kistler, Lily B. Mummert, Maria R. Ebling, Puneet Kumar, and Qi Lu. Experience with Disconnected Operation in a Mobile Computing Environment. In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, August 1993.

[35] Herb Schwetman. *CSIM17 Users' Guide.* Mesquite Software, Inc.

[36] Carl D. Tait and Dan Duchamp. Service Interface and Replica Management Algorithm for Mobile File System Clients. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 190–197, 1991.

[37] Andrew S. Tanenbaum. *Computer Networks.* Prentice-Hall International, Inc., second edition, 1988.

[38] Kevin Wilkinson and Marie-Anne Neimat. Maintaining Consistency of Client-Cached Data. In *Proceedings of the 16th VLDB Conference*, pages 122–133, 1990.

[39] M. H. Wong and W. M. Leung. A Caching Policy to Support Read-only Transactions in a Mobile Computing Environment. Technical Report CS-TR-95-07, Computer Science and Engineering Department, the Chinese University of Hong Kong, 1995.

[40] William W. Wu, Edward F. Miller, Wilbur L. Pritchard, and Raymond L. Pickholtz. Mobile Satellite Communications. *Proceedings of the IEEE*, 82(9):1431–1448, September 1994.