Maintaining Cache Consistency in Mobile Computing Environments

By

Leung Wing Man

Submitted to Department of Computer Science & Engineering

in partial fulfillment of the requirements for the

degree of Master of Philosophy

at the

The Chinese University of Hong Kong

August 1996

香港中文大學
大學
圖書館系統
10 MAY 1997
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

Maintaining Cache Consistency

in

Mobile Computing Environments

By

Leung Wing Man

Submitted to the Department of Computer Science & Engineering

in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Abstract

Mobile computing is a new and challenging technology which will definitely revolutionize the way we use computers in the next decade. A mobile computing environment consists of stationary supporting servers and mobile computers connecting to the servers by low-bandwidth wireless networks. The main difference between a traditional client-server environment and a mobile computing environment is that the mobility of mobile computers is unrestricted. Therefore, a mobile computer can disconnect from a server and connect to a different server at any time. In addition, a server may not have any information about which mobile computers are currently connecting to it. Hence, the issue of maintaining a consistent cache becomes complicated. It is more complicated to maintain cache consistency in a partially replicated database system as a server only manages information of its local copies. Cache consistency may be destroyed when a mobile computer carries some partially replicated data items to a cell that does not support these data items.

In this thesis, a caching policy is proposed to maintain cache consistency for mobile computers such that locks are not required for read-only transactions from mobile computers, and yet serializability is guaranteed. Simulations are performed to study the performance of the proposed protocol under various scenarios with different parameter

settings. Results are then compared with other approaches such as the Amnesic Terminals (AT) method suggested in [6]. It shows that the proposed protocol is superior to the AT method under certain scenarios. The proposed protocol for fully replicated database system is further enhanced to tackle the problems in partially replicated database system.

Thesis Supervisor: **Prof. Man-Hon Wong**

Title: **Lecturer of Computer Science and Engineering Department**

Key Words: Mobile Computing, Transaction, Consistency, Replication

Acknowledgements

To my thesis supervisor, Prof. Man-hon Wong, for all his support and encouragement. Thanks a lots for his help, constructive criticisms and extraordinarily patience throughout this two-year research studies.

I am particularly grateful to Prof. John Lui and Prof. Ada Fu for kindly being my thesis committee. Special thanks to Prof. John Lui who kindly gave me helpful advice in performing simulations. I am indebted to him for his thoughtful suggestions and important improvements on my protocols and simulation system model.

I wish to thank Keith Hang-kwong Mak, my senior, who has been particularly supportive in this two-year. Many friends and colleagues have contributed greatly to the quality of this thesis. Additional valuable technical assistance in using Matlab was provided by Terry Wai-kwong Lau.

I specially thank Chin-ming Cheung, Chong-leung Chiu, Wai-wai Chan and Wai-kit Chan for giving me warmest friendship when I first joined this university so that I could be used to this new environment. Thanks for Edward Nai-biu Tam who provided all kinds of magazines so that I can keep myself up-to-date with the society; Jeff Cheung for telling me so many good stories. It has been a pleasure working with all my classmates like Hoi-yee Hwang, Alice Ching-ting Ng and Chiu-fai Chong who worked in the same research laboratory, they all have been very patient with me and made me feel at home during the past two years.

Finally, thanks to my family members for being part of my life during this time. The love, patience and encouragement made this research possible.

Contents

Abstract **i**

Acknowledgements **iii**

1 Introduction **1**

2 Background **7**

2.1 What is Mobile Computing? 7

2.1.1 Applications of Mobile Computing 8

2.1.2 New Challenges of Mobile Computing 9

2.2 Related Work 12

2.2.1 Lazy Replicated File Service 12

2.2.2 Dividing the Database into Clusters 14

2.2.3 Applying Causal Consistency 15

2.3 Summary 16

2.4 Serializability and Concurrency Control 17

3 System Model and Suggested Protocol **20**

3.1 System Model 20

3.2 Cache Management 21

3.2.1 Version Control Mechanism 22

3.2.2 Cache Consistency 22

3.2.3 Request Data from Servers 25

3.2.4 Invalidation Report 27
3.2.5 Data Broadcasting 30

4 Simulation Study **32**
4.1 Physical Queuing Model 32
4.2 Logical System Model 33
4.3 Parameter Setting 34
4.4 The Significance of the Length of Invalidation Range 37
4.4.1 Performance with Different Invalidation Range 38
4.4.2 Increasing the Update Frequency 40
4.4.3 Impact of Piggybacking Popular Data 41
4.4.4 Increasing the Disconnection Period 42
4.5 Comparison of the Proposed Protocol with the Amnesic Terminal Protocol 44
4.5.1 Setting a Short Timeout Period 45
4.5.2 Extending the Timeout Period 46
4.5.3 Increasing the Frequency of Temporary Disconnection 48
4.5.4 Increasing the Frequency of Crossing Boundaries 49
4.6 Evaluate the Performance Gain with Piggybacking Message 50
4.6.1 Adding Piggybacking Messages 51
4.6.2 Reducing the Number of Popular Data 52
4.6.3 Increasing the Frequency of Updates 53
4.7 Behaviour of the Proposed Protocol 54
4.7.1 Finding Maximum Number of Mobile Computers 54
4.7.2 Interchanging the Frequency of Read-Only and Update Transactions 55

5 Partially Replicated Database System **57**
5.1 Proposed Amendments 57
5.1.1 Not Cache Partially Replicated Data (Method 1) 58
5.1.2 Drop Partially Replicated Data (Method 2) 59
5.1.3 Attaching Server-List (Method 3) 59

5.2 Experiments and Interpretation 60
5.2.1 Partially Replicated Data with High Accessing Probability 61
5.2.2 Reducing the Cache Size 64
5.2.3 Partially Replicated Data with Low Accessing Probability 65

6 Conclusions and Future Work **70**
6.1 Future Work 72

Bibliography **73**

A Version Control Mechanism for Servers **76**

List of Tables

4.1 Fixed Simulation Parameters 35

5.1 Additional Simulation Parameters for Partially Replicated Database System 61

List of Figures

1.1 Problem with the Existing Strategy 3

2.1 Structure of a cell 8

2.2 Lazy Tree Organization 13

3.1 Algorithm for Processing Data Requests at Server 26

3.2 Algorithm for Processing Invalidation Report at a Mobile Computer 30

3.3 Algorithm for processing Data Broadcasted from a Server 31

4.1 Queuing Model of the Wireless Channel 33

4.2 Logical Model of the Simulation 34

4.3 Performance with Different Invalid Range 39

4.4 Increasing the Update Frequency 40

4.5 Impact of Piggybacking Popular Data 41

4.6 Impact of Piggybacking Popular Data (Higher Update Frequency) 42

4.7 Increasing the Disconnection Period to 300 seconds 43

4.8 Increasing the Disconnection Period to 500 seconds 43

4.9 Our Protocol VS AT Method (Timeout Period = 5 sec) 45

4.10 Our Protocol VS AT Method (Timeout Period = 60 sec) 47

4.11 Increasing the Frequent of Temporary Disconnection 49

4.12 Increasing the Frequency of Crossing Boundaries 50

4.13 Performance of Our Protocol With & Without Piggybacking 51

4.14 Reducing the Number of Popular Data 52

4.15 Increasing the Frequency of Updates 53

4.16 Tolerance of Our Proposed Protocol 55

4.17 Interchanging the Frequency of Read-Only and Update Transactions 56

5.1 A Scenario for a Partially Replicated Database 58

5.2 Partially Replicated Data with High Accessing Probability 62

5.3 Time Delay = 0.25 seconds 63

5.4 Time Delay = 0.05 seconds 63

5.5 Time Delay = 0.0 seconds 64

5.6 Time Delay = 0.25 seconds; Cache Size = 30 64

5.7 Popularity of Popular Data 30% 66

5.8 Popularity of Popular Data 60% 66

5.9 Popularity of Popular Data 80% 66

5.10 Ratio of Partially Replicated to Fully Replicated Data 67

5.11 Cache Size = 30 68

5.12 Cache Size = 10 69

Chapter 1

Introduction

Mobile computing is a new and challenging area in computer science. With this new technology, users will soon be able to access databases through wireless networks regardless of their physical locations [15, 21, 24]. Wireless networking greatly enhances the utilization of network resources and permits continuous access of data as users move.

A mobile computer user may frequently query databases by invoking a series of operations. The need to access databases implies the necessity to have a data management system which provides data efficiently regardless of location. Data management in mobile computing environments is completely different from that of traditional client-server environments. In traditional client-server environments, operations at remote terminals depend heavily on the rest of the networks [32]. In mobile computing environments, the nature of wireless communications makes traditional approaches not applicable.

Wireless connections are of lower quality than wired connections [21]. Moreover, the wireless networks usually deliver lower bandwidth than wired networks, hence mobile computing designs need to be more concerned about bandwidth consumption and constraints than that for stationary computing. Mobile computers may frequently disconnect from the fixed hosts and cross the boundaries to different cells for various reasons. The network addresses of mobile computers and the system configuration parameters may change dynamically. Hence, data that is considered static for stationary computing may become dynamic in mobile computing environments. Considering the constraints stated above, it is necessary to redesign the distributed services to support mobile computers. It is important to have dynamic replicated data management protocols that allow copies

of data to be created and migrate from one site to another site.

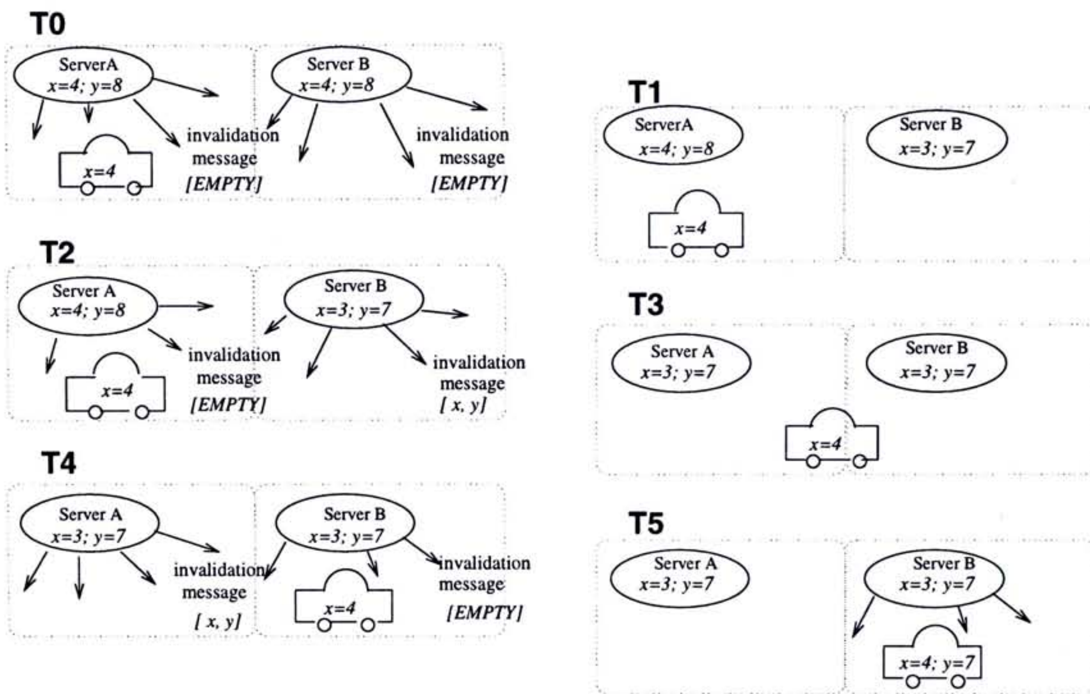
It is a new challenge to design strategies to distribute replicated data among supporting servers and mobile computers so as to maximize availability of data, maintain efficient executions of transactions and at the same time guarantee the correctness of the executions - serializability. Serializability is a widely accepted correctness criterion for the execution of transactions [19] and many commercial applications adopted this correctness criterion.

In order to reduce the contention on the narrow bandwidth of the wireless channels, usually part of a database is cached in mobile computers [5, 6, 14, 11]. Caching also implies the need for maintaining consistency. It is necessary to develop a new mechanism for maintaining cache consistency that takes into account the characteristics of mobile computing environments.

The issue of maintaining cache consistency for mobile computers is not as simple as that of a traditional client-server based database system. In a traditional client-server based database system, locks are required to be held in the server before data are cached in a client. It may be required for a server to invalidate outdated data in a client from time to time. By using the combination of locks and invalidation messages, it is easy to keep the data cached in a client consistent. However, in a mobile computing environment, mobile computers may disconnect from servers frequently without notifying the servers. This may be due to the failure in receiving the transceiver's signal of the server or running out of battery. A server may not even know the existence of every mobile computer [5]. Therefore, delivery of invalidation messages is not guaranteed. In addition, a mobile computer may connect to a different server when it crosses the boundary of a cell. Such a phenomenon further complicates the design of a caching strategy for mobile computing environments.

Consider a simple cache management policy such that mobile computers can request data from servers at any time and servers broadcast invalidation messages periodically to inform mobile computers to drop outdated data. Figure 1.1 illustrates a scenario that the data cached in a mobile computer become inconsistent, i.e. transactions reading these data may not be serializable. The main reason for such a phenomenon is that a mobile

computer may load data into its cache from different servers. It is well understood that no matter what replica control or commitment protocols are used, it is impossible to update all copies of a data on different servers simultaneously. Therefore, the data in the cache of a mobile computer may not correspond to a part of a database snapshot. Hence, transactions reading these data may not be serializable.



At Time T_0	The values of x and y at servers A and B are the same and a mobile computer is residing at cell A.
At Time T_1	The copies of x and y at server B are updated by a transaction.
At Time T_2	Before A receives the propagation message from B, server A and B broadcast invalidation messages to their local mobile computers respectively. The invalidation messages in these two cells are different.
At Time T_3	The mobile computer crosses the boundary to cell B and server A receives the propagation message from server B for updating X and Y.
At Time T_4	After the mobile computer has arrived at cell B, both servers send invalidation messages. However, the invalidation messages from server B is empty.
At Time T_5	If the mobile computer loads the value of y into its cache, the cache in the mobile computer is inconsistent in the sense that it does not correspond to a snapshot of the database.

Figure 1.1: Problem with the Existing Strategy

The most obvious solution to the above scenario is to drop the entire cache once a mobile computer crosses the boundary to another cell. However, such a strategy is very inefficient especially when a mobile computer is traveling to and fro through the boundaries of different cells [25]. In many cases, wireless communications can be supported

by cellular networks. However for some transmitters such as IR transmitters, the area covered by the transmitters' signal may be very small such as 10 meters [29]. It is realistic that a mobile computer crosses the boundaries of different cells frequently in real life.

Some caching strategies for mobile computers have been introduced in [6]. The performance of these strategies and the impact of client's disconnection durations on these strategies are evaluated. One of the suggested protocols in [6] is called *Amnesic Terminals*(AT), in which each mobile computer caches a portion of the database. The data are updated at servers only and the mobile computers carry copies of data for their own use. The servers broadcast invalidation reports periodically to inform the mobile computers to drop invalid cached data. Each invalidation report contains only the identifiers of the data that have been changed since the broadcasting of the last report. A mobile computer that has been disconnected for a while needs to start rebuilding its cache from scratch as some invalidation reports might have been broadcasted during its disconnection. In answering a transaction, a mobile computer has to listen to the invalidation report first in order to conclude whether its cache is valid or not.

In [24], various static and dynamic data allocation (caching) methods were proposed to optimize the communication cost between a mobile computer and a fixed host that stores the online database.

Performance is the main concern of these two papers. Their results are mainly based on the property that communication in a mobile computing environment is expensive and unreliable. However, the impact of the unrestricted mobility of mobile computers on data consistency is not addressed. Mobility can cause wireless connections to be lost or degraded. It is necessary to consider these problems in the design of protocols for the mobile computing environments. In addition, the synchronization issue for the execution of transactions under mobile computing environments is not studied in these two papers. In an attempt to complement their results, we are motivated to introduce a new caching strategy such that the cached data of a mobile computer must correspond to a snapshot of a database. Hence, read-only transactions can read data from the cache without issuing any lock to the servers and at the same time serializability is maintained. The basic idea

of this protocol is to allow a read-only transaction to observe the state of the database in the past. The data it read must correspond to a snapshot of the database. The main advantage of our strategy is that the wireless bandwidth can be saved by eliminating the communication required for locking. In addition, the round trip delay time needed for obtaining locks is saved. Furthermore, cache consistency is ensured without dropping the cached data after a mobile computer crosses the boundaries of cells.

For the sake of completeness, we have also investigated the performance characteristics of these protocols under different database environments by simulation. Through the simulations study we also identify the factors that lead to better performance. For example, the most appropriate amount of information that should be contained in each invalidation report is studied. It is found that piggybacking the new values of frequently accessed invalid data in the invalidation reports improves the performance of the proposed protocol significantly. The maximum number of mobile computers that can be supported by the proposed protocol is also investigated. After that, the performance of our protocol is compared with another approach, the Amnesic Terminals (AT) method suggested in [6]. The simulations show that the proposed protocol is superior to the AT method under certain scenarios. The factors that lead to superior performance are also investigated.

So far it is assumed that databases are fully replicated, however, in practice some location sensitive data may not be fully replicated. The issue of maintaining cache consistency of a partially replicated database system is more complicated as a server only manages information of its local copies. Invalidation reports from a server will not contain information of data which it does not support. The cache consistency of mobile computers cannot solely depend on the invalidation reports, since it may be destroyed when a mobile computer carrying some partially replicated data moves to a cell that does not support those data. Hence, enhancements of our protocol are proposed such that cache consistency can be ensured for such an environment.

The basic contributions of this research are summarized as follows:

- Cache consistency is guaranteed.

- Consumption of the wireless bandwidth is saved.

- Transaction response time is greatly reduced.

- Asynchronous database servers are considered.

This thesis is organized as follows. Chapter 2 presents the definition of mobile computing and gives a brief description of other approaches for transaction management in mobile computing environments. Chapter 3 presents the model of the mobile computing environment adopted in this paper and the proposed cache management policy. Simulation experiments and the corresponding results are described in Chapter 4. Chapter 5 discusses the caching issues for a partially replicated database system and evaluates the proposed solutions by simulations. Finally, we conclude our work and suggest future work in Chapter 6. The version control mechanism, which manages different versions of data, is summarized in Appendix A.

Chapter 2

Background

In this chapter, we will briefly describe the definition of mobile computing, its applications, technical challenges and some related work in this area. In the next chapter, we propose a new caching strategy that considers the impact of mobility and limited power of mobile computers on data management, which are often being neglected in previous work.

2.1 What is Mobile Computing?

The rapid development of cellular communications and satellite services make it possible for users to access information from anywhere and at anytime with their portable *mobile computers*. Mobile computers are able to access information through wireless networks regardless of their physical locations. This new mobile computing environment enables almost unrestricted mobility and hence provides flexible communication among people.

The databases of the system are stored at the supporting servers which are connected by a static network. This static network consists of a set of static *supporting servers* which serve as access points for mobile computers to connect to the static network. These supporting servers are static and connected by a fixed and reliable wired network.

Supporting servers communicate with mobile computers through *wireless channels*. The area covered by the signals of a server is defined as its *cell*. Cells may overlap physically but each mobile computer should logically connect to one server at a time only. The corresponding server is known as the *local supporting server* of that mobile computer. A mobile computer can connect to its local supporting server only if it is physically located

within the cell of the server. The structure of a cell is shown in Figure 2.1.

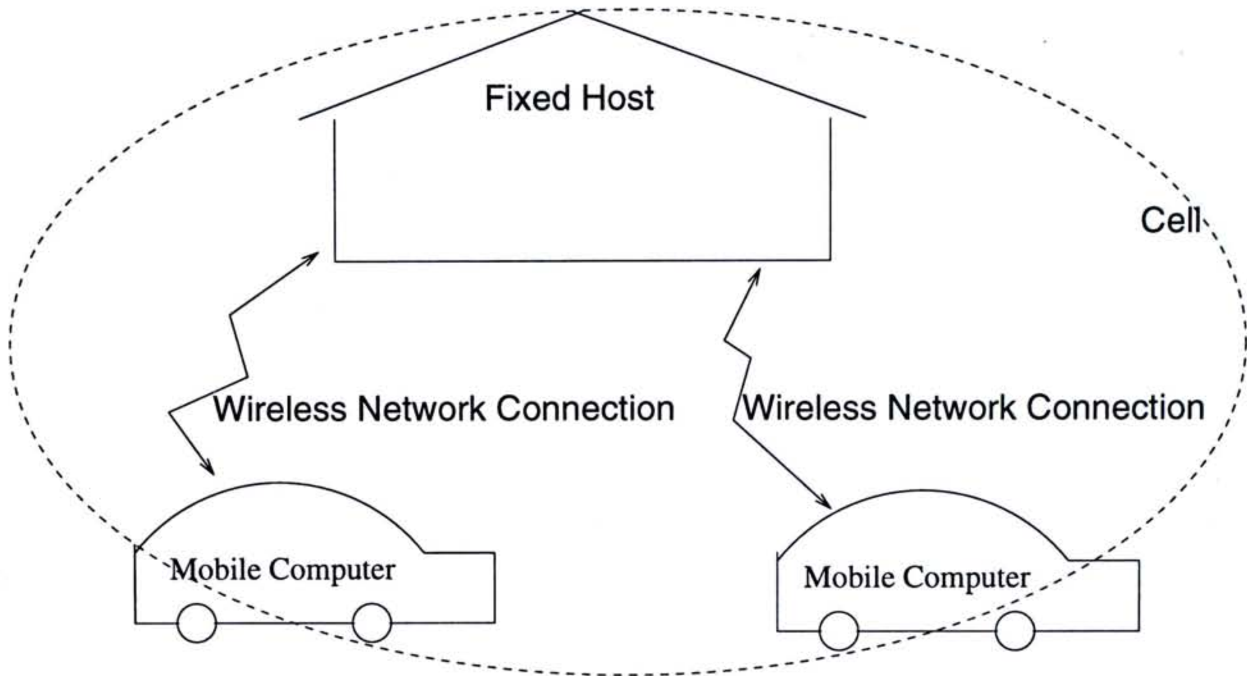


Figure 2.1: Structure of a cell

2.1.1 Applications of Mobile Computing

Wireless communication enables mobile computers to remain connected with the networks regardless of their physical locations. A user is not required to stay at fixed location in the networks, this enables unrestricted mobility of users. The small sizes of mobile computers also improve the portability. Wireless communication, mobility and portability will create an entire new class of applications and new markets combining personal computing and consumer electronics [26].

Mobile computers can download news or documents, access information and query remote databases from their local supporting servers through the wireless channels irrespective of time and location. Users can retrieve location dependent information like the closest hospital. For example, weather reports and traffic information will be sent to a user according to his/her location. Moreover mobile computing is applicable in business activities. Mobile computing is applicable to taxi services in which taxi drivers

can exchange traffic information and get the information of the locations about other drivers. Businessmen are capable of processing simple transactions such as inventory orders. Field technicians can access on-line manuals while they are on outdoors repair assignments. Salesmen are able to get the most up-to-date market information so as to make their investment decisions. A home buyer can enquire about properties from local estate agents.

As a conclusion, mobile computing systems will present a user easily accessible and online information without the need for the cumbersome packages now offered. It is expected that this inexpensive, portable, mobile telecommunications will become very popular in the near future. However, there are still some technical challenges need to be solved.

2.1.2 New Challenges of Mobile Computing

The challenges of mobile computing mainly come with its essential properties : wireless communication, mobility and portability [21]. The corresponding problems are presented in the following paragraphs.

Wireless Communication

It is much more difficult to achieve wireless communication than wired communication because a wireless channel is subject to signals blocking and noise, hence it faces more obstacles. In addition, wireless networks deliver lower bandwidth than wired networks [21]. Moreover, message transmission over the wireless channels is unreliable and is of lower quality than that over wired channels. It is also expected that the cost of wireless communication is expensive [18]. For these reasons, the users should access the wireless networks no more than necessary and the communication mechanism should also be fault tolerant. Furthermore, wireless communication also results in security risks. For example, mobile computers may attempt to access information from a database where data accesses are restricted. Even worse, the unreliable wireless channel cannot ensure the security of data

transmission.

Mobility

Unrestricted mobility of mobile computers enables users to take their mobile computers to places where users of traditional wired network do not go [18]. However, moving a mobile computer implies the need for reconfiguration of the system. Location dependent information like the local supporting servers and available printers changes as mobile computers move. Today's networking is not designed for dynamically changing addresses. It is necessary to have a mechanism to obtain these configuration information appropriate to the present location.

Traditional data management strategies which assume static topologies are not applicable in mobile computing environments. Query languages must be extended to take into account the mobility of users and the fact that data may be highly distributed. The effects on data distribution, query processing and transaction processing due to mobility are some of the challenges of this new area in computer science.

A mobile computer may travel beyond the coverage of the signals from its local supporting server. It will then be disconnected. Network failure is a greater problem in mobile computing environments than traditional client-server based systems. Frequent disconnection of mobile computers introduces new issues that are not presented in traditional client-server based systems. Mobile computers should be able to operate as stand-alone computers during disconnection.

The logical communication structure between mobile computers and supporting servers changes dynamically. A mobile computer may cross the boundary between two different cells while executing a transaction and the mobile computer may access information from two different databases. Mobility and disconnection should be completely transparent to users. Special strategies are required to handle the above problems which are characteristics of mobile computing environments.

Mobile computers may access different databases at different time and encounter more

heterogeneous network connections. Different quality of services will be provided at different places. This heterogeneity makes mobile networks more complex than traditional fixed networks. Privacy also becomes a problem in mobile computing environments. It may be possible for a person to know the locations of other mobile users.

Portability

In order to increase the portability of mobile computers, small size batteries are used. The battery weighs less but is also less powerful. This results in power limitation to mobile computers, it is a new resource limitation which is intrinsic to mobile computing environments. Minimizing power consumption can improve portability by reducing the battery size and providing efficient operations. Power management software can turn off a mobile computer when it is idle. Consequently, frequent disconnection becomes a characteristic of mobile computers so as to save power or due to shortage of power supplies.

The size constraints on mobile computers require small user interfaces. The size of screen is significantly reduced and it becomes difficult to operate with these portable devices.

Even though the mobile computing systems bring with them lots of hardware and software problems, it is still worth developing. It is the most powerful and exciting technological trend in computers and telecommunications [15]. It allows users to access the database continuously regardless of the geographical restrictions. Moreover, it enhances the utilization of portable devices which provide higher flexibility than fixed networks. Much researches have been done in this new area, some of the related work are presented in the following section.

2.2 Related Work

Many researchers believe that it is difficult to maintain perfectly synchronized cached data on a mobile host. Various degrees of consistency between cached data may be defined depending on the available bandwidth. Many researchers relax the correctness criterion such that a weaker notion of cache consistency is used. In this way, the cache may not correspond to a snapshot of a database system.

Before we start our investigation on cache management in mobile computing environments, we summarize some cache management protocols suggested by other researchers who adopt weaker notions of consistency.

2.2.1 Lazy Replicated File Service

In [16, 17, 35], the authors use a primary-secondary replication scheme to eliminate global communication by allowing a mobile computer only communicates with its primary servers. Updates by the mobile computer are read out (pickup) of the cache by the primary server of the mobile computer, i.e. only the primary server contains the updates. Primary server pickups only at moment of its choosing. However, it is possible for a client to request immediate pickup from its primary server due to a full cache during periods of heavy update activity. After a pickup, the primary server retains a volatile copy of the file and multicasts them to the secondary servers. Once the number of the secondary servers that have acknowledged saving the updates reaches a certain threshold, say N, the primary server will inform the client that the updates may be discarded from its cache by sending *purge notice* to the client. This purge notice guarantees that the updates have been replicated at the primary server and N secondary servers, and so the data is replicated widely enough to be N-fault tolerant. If there does not have enough secondary servers acknowledged the update, then the primary server does not send purge notice to the client. By this means, the service has some latitude to "wait out" failures of secondary servers without any apparent service disruption. This model is called "lazy tree" because of the lazy propagation of updates within the tree-like structure as shown in Figure 2.2

and the protocol is called as a *lazy "server-based" update operation.*

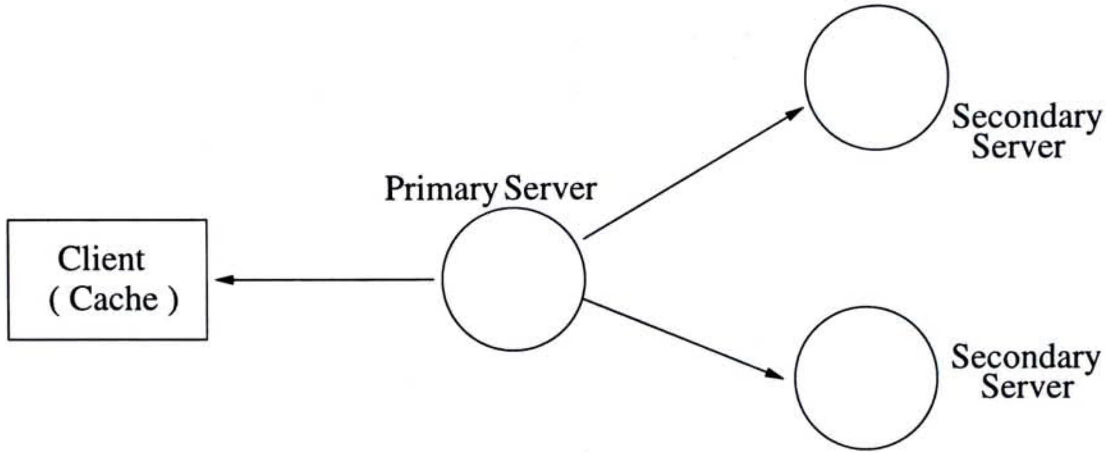


Figure 2.2: Lazy Tree Organization

This protocol allows clients to determine the level of consistency desired. The authors propose the idea of supporting two read calls, namely *strict_read* and *loose_read.* With *loose_read*, consistency is not guaranteed but it will make a "best effort" to provide the latest value. The entire cost of establishing one copy serializability is charged to the application which requires strong consistent data. A naive implementation of *strict_read* would contact all servers and all clients that had read the file and retrieve the most up-to-date copy it finds. *Currency tokens (CTs)* are used to avoid this naive approach. The idea of currency token enables the client holding the token has the powerful effect of allowing strict reads to be implemented as efficiently as loose reads and has the write permission to make a consistent update. CTs are defined in terms of *potential consistent writers*, or PCWs. A PCW is a client with a process that has strictly read a file, and that has write permission for that file; a PCW has both desire and ability to make a later update in a consistent fashion. A CT is given in response to a strict-read if there are no PCWs for that file or if the client is the only PCW.

A client that performs a strict read without a CT initiates a relatively complex series of actions. First, recall that an update must be replicated on at least N secondary servers before a client is allowed to purge the update from its cache. Therefore, assuming that there are a total of T secondary servers, at least $T - N + 1$ secondary servers must be

contacted as well as any PCWs that have the file in their caches - to ensure that the most recent value is located. This condition is simply that of quorum consensus, where the read and write quorums must overlap.

This protocol is unsuitable for applications that share a file simultaneously or that perform read/write transactions with a read operation closely following a write operation. Such applications will typically want to see each other's update as soon as possible.

2.2.2 Dividing the Database into Clusters

Maintaining consistency of data over all distributed sites imposes unbearable overheads in mobile computing environments. Hence, in [30, 31, 32], the authors introduce weaker notions of consistency. Strong and weak operations which exhibit different degrees of tolerance for inconsistency are defined. Strong operations are those operations that required strong consistency while weak operations are those that can be executed under weaker consistency requirements. Using this system model, the database is partitioned into a set of clusters. All data kept inside a cluster are mutually consistent and degrees of inconsistency are allowed among data at different clusters. The cluster configuration is dynamic. Clusters may be defined based on the semantics of data, such as location data. Location data represents the addresses of mobile computers and are fast changing data replicated over many sites. These data are often imprecise, since updating all their replicas impose unbearable overheads. The definition of clusters may be explicitly provided by users based on their requirements. Moreover, the system can use the information stored in users' profile to define clusters. A mobile computer can operate with its cached data during its disconnection if strict consistency is not required. Strict consistency is restored when clusters are merged as mobile users enter new cells or connect to and disconnect from the rest of the networks. Hence, no control messages are needed between transaction managers at different clusters for synchronizing weak transactions. To resolve conflicts in inter-clusters schedules, those weak write operations conflict with strict transaction need to be undone. The protocol requires that only weak transactions in the same clusters

read the values updated by weak transactions of that cluster. Therefore, even undoing a transaction normally causes *cascading abort* of weak transactions.

By offering to the applications the ability to specify explicitly when strict consistency is necessary for their executions, bandwidth utilization is reduced and data availability is improved.

2.2.3 Applying Causal Consistency

It is commonly accepted that strong consistency is difficult to obtain in mobile computing environments as suggested by the authors in [3]. Strong consistency such as serializability, which requires an interleaved execution to be equivalent to a serial execution, cannot be provided in systems where clients may temporarily disconnect. This is because communication between nodes is required before an access can be completed in cases where data are cached at mobile computers.

In [3], the authors defined *Causal Consistency* which is a weakly consistent shared data model based on causal orderings defined in [28] between accesses to shared data. Suppose two operations o_1 and o_2 are executed on the same set of data, S. According to [3], o_1 precedes o_2 if one of the following hold:

1. o_1 and o_2 are executed by the same process and o_1 is executed before o_2;
2. o_2 reads a data value that was written by o_1;
3. There is a sequence of operation $o_{i,1}, o_{i,2}, \ldots, o_{i,m}$ *s.t.* $o_{i,1} = o_1, o_{i,m} = o_2$, and for all $i \leq j \leq m$, $o_{i,j}$ *and* $o_{i,j+1}$ are related by either *(1) or (2)*.

If o_1 and o_2 do not causally precede each other then they are said to be *concurrent* and they cannot affect each other. Suppose the data updated by o_1 and o_2 be f_1 and f_2 respectively. Then cached copies of f_1 and f_2 are mutually consistent if one of the following holds.

1. o_1 and o_2 are concurrent and hence neither causally precedes the other.

2. The two operations are causally ordered. Assume that o_1 precedes o_2, then there do not exist and will not exist another operation o_3 such that o_1 precedes o_3 and o_3 precedes o_2. And o_3 updates a value of data f_1 that is not present in the cached copy of f_1.

Causal Consistency allows read and write operations to complete with cached copies and it captures causal relationships among operations that access data. *Causal consistency* are provided for each data by associating a pair of vector timestamps with each copy of replicated data. These timestamps are *creation time* and *validation time.* The *lifetime* of a version of a data is the duration from the time that version is created to the time it is invalided. They can be used to check if the cached data are consistent with the incoming data in case of cache miss. If one of the data has a creation time which is less than the creation time of the other and the lifetimes of these two data are not overlap, validation is required. The older copy has to be validated by more careful examinations.

The suggested protocol also has some drawbacks. For example, validation of a data requires that a server communicates with other mobile clients. It may not be possible to communicate with all such clients, since they may be disconnected. In this case, either the older data has to be removed from the cache or it can be marked as potentially inconsistent. Another important problem is the size of the vector timestamps which used to keep the timestamps of all the clients can be very large when the algorithm is applied in large systems.

2.3 Summary

All the above protocols support "disconnect operations" and coping with inconsistency problems brought by mobile computing environments by accepting different degrees of consistency. Weaker notion of consistency is defined such that read operations can be satisfied immediately with the cached data if strict consistency is not required. The cost of wireless communication is reduced.

However, these protocols cannot provide efficient solutions if serializability is necessary.

In this paper a cache management policy is proposed to support "disconnect operations" and guarantee serializability. Serializability is employed in our system such that we strictly require that the data available in each mobile computer are consistent.

2.4 Serializability and Concurrency Control

Users of mobile computers may frequently query information in databases through the servers. A *database* is a collection of data which may be distributed or replicated among the servers. It is assumed that the local supporting server is the only source of data for a mobile computer and that a data is the unit of a single access. In practice, a *data* may be a record, a relation, a page of memory or even an entire file. A *database state* is the values of the data in a database at a particular time and *consistent database state* means that the data values are consistent with each other [8]. A static image of the database taken at a given time is called *snapshot* [13]. It is a portion of a consistent database state.

Data are updated by transactions [8, 9]. Each transaction is composed of a series of read and write operations. A transaction which contains write operations is known as an *update transaction* and a transaction with read operations only is known as a *read-only transaction (query)* [19]. The execution of a transaction must be atomic, i.e, the transaction either commits or aborts. If a transaction commits, the effects of the transaction will be incorporated completely to the database and considered as permanent. Otherwise, if it aborts, none of the effects will be incorporated into the database. Atomic transactions transfer the database from one consistent state to another.

If all transactions in the system are executed serially, this sequence of operations performed by transactions are defined as correct by definition in [13]. However, if several transactions are executed at the same time, the processor can be shared among them and also higher the total transaction throughput. Hence, it is not desired to force transactions to execute serially but to execute them as concurrently as possible. When a set of transactions execute concurrently, their operations may be interleaved. The operations of one transaction may execute between two operations of another operation. This kind

of schedule may lead to an inconsistent database state. It is necessary to have a formal model to analyze the correctness of our system. The activity that coordinates the actions of processes that operate in parallel, access shared data, and therefore potentially interfere with each other is known as *concurrency control* [9].

The standard theory for analyzing database concurrency control algorithms is the *serializability theory* [8, 10, 12] which is a widely accepted correctness criterion for the execution of transactions. It guarantees that a concurrent execution of transactions must be equivalent to a serial execution of the transactions. A concurrency control mechanism restricts the order of operations of transactions by delaying some operations of transactions, restarting and aborting transactions if necessary.

One way to ensure serializability is to require that access of data be done in a mutually exclusive manner. When one transaction accesses a data, no other transaction can modify that data. The most common method used to implemented this is to allow a transaction to access a data only if it is currently holding a lock on that data [20]. Two phase locking is one of the protocols which adopts this strategy. Another method for determining the serializability order is to select an ordering among transactions in advance. The most common method for doing so is to use a timestamp-ordering scheme [20].

A data may be replicated among several servers, these copies of a data must appear as a single logical data to transactions. Similar to a one-copy database system, it is necessary to have a concurrency method to ensure correctness. It is generally required for a Database Management System to manage a replicated database behaves like a Database Management System managing a one-copy (i.e. non-replicated) database insofar as users can tell [9]. There are many different ways to perform the read and write operations of transactions. The system that we studied is a replicated database system in which multiple copies of some data are stored at multiple sites to increase data availability and improve performance.

When a replicated data is updated, it is required to update copies of replicated data which are being stored at more than one server. The replicated data should be transparent to users, such that the executions of operations of transactions on a replicated database

are equivalent to a serial execution of those transactions on a one-copy database. There are many approaches to do that. For example, according to the *Write-All Approach* [9], it is necessary to update all copies of an updated data. In this way, *Read* operations may run faster at the expense of slower *Write* operations. However, in real situation, sites may fail. If the DBS adheres to the write-all approach, it may need to delay the write operations until the site recovers. Therefore, different approaches like *Write-All-Available Approach*, which updates copies stored at all available servers when a replicated data is being modified, is introduced.

There is another replica control protocol such as the quorum protocol [23]. In the quorum protocol, every server is assigned with a non-negative weight and each server knows the weight of all the other sites in the system. A quorum is a set of sites with total weights more than half of the total weight of all the sites. A read operation reads the most up-to-date copy among all the data in the read quorum of sites. Whenever a write operation is performed at a server, it is necessary to update all the copies of that data in the quorum of sites. The purpose of quorums is to ensure *Read* and *Write* that access the same data must access at least one copy of that data in common [9].

We adopt one-copy equivalence as the correctness criteria in our system. A replica control protocol, together with a concurrency control protocol, can enforce one-copy serializability [23], a correctness criterion for replicated database.

After a write operation is executed on a data, a new version of the data will be created. If a new version always replaces the old version, then the system is a *monoversion* database system. Otherwise, if it keeps more than one version of the same data, it is a *multiversion* [9] database system. In this studies, it is assumed that the database is a multiversion database. Thus, each data x in the database is represented by a totally ordered sequence of versions, denoted by $x^1, x^2 \cdots x^n$, where the superscripts are the monotonically increasing *version numbers*. The version numbers measure how up-to-date each copy is. It is assumed in this studies that the version control protocol proposed in [34] is employed in the servers. The details of the version control mechanism is discussed in Appendix A.

Chapter 3

System Model and Suggested Protocol

3.1 System Model

There are two distinct sets of entities in the system model: servers and mobile computers. It is assumed that each server or mobile computer can be identified by a unique identity number. Servers are powerful stationary computers connected by a reliable network with high bandwidth. The wired network is not synchronized. On the other hand, mobile computers are portable computing devices that are usually low powered machines equipped with limited amount of memory [33].

The system allows unrestricted mobility and connectivity of the mobile computers among the cells. The exact location of a mobile computer is not known to the supporting servers. A mobile computer may be disconnected from its local supporting server for various reasons and crosses the boundaries to different cells without notifying the servers. Mobile computers can retain their network connections when they move to different cells. No special procedure will be executed between the previous and present local supporting servers when a mobile computer crosses the boundaries of these cells.

Data are fully replicated by storing copies of data at all the servers. The version control mechanism introduced in [34] is adopted to update data among the supporting servers. It is assumed that mobile computers cannot communicate directly with one another. The only source of information for a mobile computer is its local supporting server.

Data are updated by transactions at supporting servers only. Mobile computers can request data from their local supporting servers through the unreliable wireless channels. Message transmission between supporting servers and mobile computers are assumed to be delivered in a first-in-first-out manner over the wireless channels.

Mobile computers will cache portions of the database for their own use. Supporting servers broadcast invalidation messages to mobile computers periodically to invalidate cached data from mobile computers. All the messages from a supporting server are broadcasted to all mobile computers within its cell.

3.2 Cache Management

In this section, a cache management policy is introduced to support read-only transactions in mobile computers. In a mobile computing environment, we have to face additional difficulties that do not exist in a stationary client-server environment. First, a mobile computer may cross the boundary and connect to a different cell without notifying the server. By using a quorum protocol for replica control, a write operation may not update all copies of a data. In addition, it is impossible to update different copies of the same data at different servers simultaneously. Therefore, the data in the new and the old servers may correspond to different snapshots of the database. Hence, the consistency in the cache may be destroyed if the invalidation messages and data from the current server are accepted blindly as in the example shown in Figure 1.1 of Chapter 1. Moreover, a mobile computer may be frequently disconnected from a server for various reasons, such as the failure in receiving the transceiver's signal. Therefore, a mobile computer may fail to receive some invalidation messages from the server. Hence cache consistency is destroyed. A trivial solution to these problems is that the entire cache is dropped when the mobile computer crosses the boundary, or when the missing of a message from the server is detected. However such an approach is very inefficient.

In this section, an efficient protocol which can maintain the consistency of cached data in mobile computers is proposed. Intuitively, the data cached in a mobile computer is

consistent, if the data corresponds to a snapshot of the database. In order to maintain this property, different versions of data, invalidation messages and data broadcast to mobile computers are required to be managed in a coordinated manner.

3.2.1 Version Control Mechanism

The version control mechanism in [34] is adopted in the servers for our protocol. The details of this mechanism is presented in Appendix A, readers who are familiar with this protocol may skip that section. A counter, *ctnc*, is introduced in the version control mechanism. *ctnc* stands for *completeness transaction number counter* which is maintained at each server. It's value is equal to a particular version number in our multi-version database system. There are two properties of this version control mechanism that are used in our cache management policy.

Property 1 No transaction in the system will commit with a timestamp less than *ctnc* of any server, i.e., no version of a data can be created in the system with a version number less than the *ctnc* of a server.

Property 2 A server S_i contains all versions of data with version number less than or equal to *ctnc*.

3.2.2 Cache Consistency

The most crucial property that must be maintained by the proposed caching policy is that the data cached in a mobile computer must be consistent in the sense that transactions reading these data are serializable. Therefore, the data in the cache must be part of a consistent database state. It is generally accepted that executing a transaction can bring a database from one consistent state to another consistent state. Hence, any database state resulting from a serializable execution must be consistent. The simplest method to obtain a consistent database state is to stop initiating new update transactions, and then obtain the database state after all update transactions have been completed. If

serializability is enforced, the state obtained in this way must be consistent. In practice, however, it is undesirable to obtain a consistent database state by blocking all update activities. Instead, multiple versions of data can be kept and a consistent database state can be obtained by selecting a version from each data properly. Obviously, for each data, the version that forms the most recent consistent database state must be created by the last update. Therefore, for any timestamp t, the latest versions of data with version numbers not greater than t form the most recent consistent database state before t, if no transaction will commit with a timestamp less than t. The formal definition for consistent database states is given as follows:

Definition 1 (Consistent Database State) *Suppose* $\{x_1, x_2 \cdots x_n\}$ *is the set of data in a database. The set of versions* $\{x_1^{j_1}, x_2^{j_2} \cdots x_n^{j_n}\}$ *is the most recent consistent database state at timestamp t, denoted* $DBS(t)$*, if for each data* x_i*, there does not exist and will not exist a version* $x_i^{j'_i}$ *such that* $j_i < j'_i \leq t$.

From the definition of consistent database states, it is not difficult to show the following lemma:

Lemma 1 For any timestamp t, $DBS(t)$ is unique.

Proof: Assume the lemma does not hold and let $\{x_1^{j_1}, x_2^{j_2} \cdots x_n^{j_n}\}$ and $\{x_1^{j'_1}, x_2^{j'_2} \cdots x_n^{j'_n}\}$ be two distinct consistent database states for $DBS(t)$. Then there must exist k such that $1 \leq k \leq n$ and $x_k^{j_k} \neq x_k^{j'_k}$. Since version numbers are totally ordered, without loss of generality, assume that $j_k < j'_k$. Hence, by definition, $\{x_1^{j_1}, x_2^{j_2} \cdots x_n^{j_n}\}$ is not a consistent database state. It leads to a contradiction and the lemma follows. ■

Due to the limitation of storage, usually only a subset of data is cached in a mobile computer. We assume that for each data in the cache, only one version of the data is kept in the cache. We call this set of versions a *version set*. The consistency of a version set is defined as follows:

Definition 2 (Consistent Version Set) *A version set is consistent with timestamp t, if the set of versions is a subset of $DBS(t)$.*

We say that a cache is consistent with timestamp t, if the set of versions kept in the cache is consistent with timestamp t. In the implementation, a counter will be used to store the timestamp of a cache. Logically, a cache can be represented by a tuple $\left\langle t, x_{i_1}^{j_1}, val_{i_1}, x_{i_2}^{j_2}, val_{i_2} \cdots x_{i_m}^{j_m}, val_{i_m} \right\rangle$, where t is the timestamp of the cache, x_i^j is the identity of the version and val_i is the value associated with the version. Intuitively, cache consistency with timestamp t can be maintained if we can enforce that the version of each data kept in the cache is the largest possible version with version number less than t. This property can be stated formally as follows:

Lemma 2 A version set is consistent with timestamp t, if the version for each data in the version set is the latest version with version number less than t and no transaction will commit with a timestamp less than t.

Proof: Since version numbers are totally ordered, if no transaction will commit with a timestamp less than t, then for each data, there exists a unique latest version of the data that is less than t. Therefore, the consistent version set with timestamp t for a set of data must be unique. From Lemma 1, $DBS(t)$ is also unique. By the definition of consistent database state, the version set must be a subset of the consistent database state. Hence the lemma follows. ∎

We can observe from the definition of cache consistency that the data in a consistent cache with timestamp t contains the effects of the transactions with timestamps less than t only. It is not difficult to show that any read-only transaction that reads data from a consistent cache with timestamp t can be serialized immediately after all the transactions with timestamps less than t. A similar proof for a multi-version database has been given in [8]. Therefore, if cache consistency can be maintained, any read-only transaction from a mobile computer can read the data in the cache without issuing any lock on the server and serializability is guaranteed.

As new versions of data are created continuously by update transactions, the cache in mobile computers should always be kept as up-to-date as possible. Updating the cache in a mobile computer may involve three different processes: dropping some old data, reading in new data and advancing the timestamp of the cache. However, these three processes cannot be performed independently. In order to maintain cache consistency, careful coordination is required. The process of advancing timestamps of cache and dropping outdated data from cache must be done in a single atomic step with respect to the transaction in a mobile computer. When a mobile user requests data from servers, the version that can be loaded into the mobile computer depends on the timestamp, t, of the cache. The details are discussed in the following sections.

3.2.3 Request Data from Servers

The data required by the user of a mobile computer may change from time to time. Therefore, unused data will be dropped from the cache and new data may be loaded into the cache gradually. From Lemma 2, it is not difficult to observe the following lemmas, since set difference and union do not destroy the assumption in Lemma 2:

Lemma 3 Let V be a consistent version set with timestamp t. Any $V' \subset V$ is also a consistent version set with timestamp t.

Lemma 4 Let V and V' be two consistent version sets with the same timestamp t. $V \cup V'$ is also a consistent version set.

Lemma 3 implies that a mobile computer can drop unused data at any time without destroying cache consistency. On the other hand, loading a data into a cache is more complicated. From Lemma 2, the largest possible version of a data with timestamp not greater than t is a consistent version set with timestamp t. Then by Lemma 4, we can conclude that the cache consistency of a cache with timestamp t will not be destroyed by loading the latest version of a data with version number less than t into the cache. Using this idea, the algorithm for requesting data from servers is derived.

When a query in a mobile computer requests data that is not cached, the mobile computer will send a *REQUEST* message to its local supporting server. A *REQUEST* message is a tuple, $\langle mobile_id, t, data_id \rangle$ where $mobile_id$ identifies the mobile computer, t is equal to the timestamp of the cache and $data_id$ is the identity number of the required data. After receiving a *REQUEST* message, the algorithm in Figure 3.1 is executed by the local supporting server.

```
Procedure ReceiveREQUEST
begin
    Suppose a request ⟨mobile_id, t, x⟩ is received.
    If ( old_vn > t )
        Message ← ⟨mobile_id, ABORT⟩ /* Abort the transaction */
        Broadcast Message
    else if (ctnc > t > old_vn )
            val ← x^j.val where (∄x^k s.t. j < k < t)
            Message ← ⟨t, x^j, val⟩
            Broadcast Message
        else if ( t > ctnv )
                Block until ctnv ≥ t
end
```

Figure 3.1: Algorithm for Processing Data Requests at Server

The situations can be classified into three cases:

- $t < old_{vn}$ (the oldest version that is kept in the server)

  Since the version requested by the mobile computer is no longer available in the server, the query needs to be aborted. The query should be re-executed after the timestamp of the cache is advanced.

- $old_{vn} < t < ctnc$

  From Property 1 stated in Section 3.2.1 of the version control mechanism, no transaction will commit with timestamp less than $ctnc$. In addition, from Property 2 stated in Section 3.2.1, the server has all the versions from old_{vn} up to $ctnc$. Therefore, x^j, which is the latest version of x with version number less than t is the

required version of the mobile computer. The server will then broadcast the message $\langle t,\ x^j,\ val\rangle$ through the wireless network. The semantics of the message is that x^j is the latest version of x with version number not greater than t and val is the value associate with this version. Therefore, all mobile computers with timestamp t can safely load the data into its cache. In fact, all mobile computers with timestamp t' such that $j < t' < t$ can also load the data. The details will be discussed in Section 3.2.5.

- $ctnc < t$

  A mobile computer may move to different cells at any time and since the servers are not synchronized, the timestamp of a mobile computer may be larger than that of the local supporting server. In this case, the version sequences in the server may only complete up to *ctnc* and it cannot be guaranteed that the latest version of the data, with version number not greater than t, is already contained in the server. Hence, the read request will be blocked until the value of *ctnc* is advanced to a value greater than or equal to t.

3.2.4 Invalidation Report

As new versions of data are continuously created by update transactions, the timestamp of a cache has to be advanced so that more updated versions of data can be loaded into the cache. This section discusses how to advance the timestamp of a cache and at the same time maintain cache consistency by invalidating data properly. Intuitively, when the timestamp of a consistent cache is advanced from t_1 to t_2, a data item has to be dropped if the version kept in the cache is not the largest possible version that is less than t_2; otherwise the cache consistency will be destroyed. This idea can be summarized in the following lemma:

Lemma 5 Suppose V is a consistent version set with timestamp t_1. Then $V - U$ is a consistent version set with timestamp t_2 ($t_2 > t_1$), if

1. no transaction will commit with a timestamp smaller than t_2, and

2. $U = \{x_i^j | x_i^j \in V \wedge \exists x_i^k \ s.t.\ t_1 < k \leq t_2\}$.

Proof: Suppose the assumptions in the lemma hold. Assume $V - U$ is not a consistent version set with timestamp t_2. By Definition 2, there exists a version $x_i^j \in V - U$ such that x_i^j is not the largest possible version that is less than t_2. That means there exists or will exist a version x_i^k such that $j < k \leq t_2$. Since no transaction will commit with a timestamp smaller than t_2, x_i^k is already in the system. By definition, $x_i^j \in U$. Therefore, $x_i^j \notin V - U$. It leads to contradiction and the lemma follows. ∎

From Lemma 5, we observe that when a server suggests to a mobile computer to advance its timestamp to t, the server has to make sure that no transaction will commit with a timestamp smaller than t. In addition, the server should know all the updates up to t, so that it can instruct the mobile computer to drop the data properly. Therefore, a server should not suggest to a mobile computer to advance its timestamp beyond *ctnc*.

Periodically, a server constructs an invalidation report $\langle t_0, U_list, ctnc \rangle$, where t_0 is a timestamp less than *ctnc* and *U_list* is the identities of the set of data which has been updated between t_0 and *ctnc*, i.e., $U_list = \{x | \exists x^k \ s.t.\ t_0 < k \leq ctnc\}$. The invalidation report is broadcast to every mobile computer in the cell. When a mobile computer receives the validation report, it can advance the timestamp of its cache to *ctnc* after dropping all the data in *U_list*, if the original timestamp of the cache is between t_0 and *ctnc*. The correctness of this approach is shown in the following lemma.

Lemma 6 Suppose V is a consistent version set with timestamp t, where $t_0 \leq t \leq ctnc$. Let V' be the version set obtained by removing all the data in *U_list* from V. Then V' is a consistent version set with timestamp *ctnc*.

Proof: Let $U_version = \{x_i^j | x_i^j \in V \wedge x_i \in U_list\}$ (the data in the cache that have been updated in the system between t_0 and *ctnc*) and $W = \{x_i^j | x_i^j \in V \wedge \exists x_i^k \ s.t.\ t < k \leq ctnc\}$ (the data in the cache that have been updated in the system between t and *ctnc*). Obviously, $W \subseteq U_version$, let $T = U_version - W$. By Lemma 5, $V - W$ is a consistent

version set with timestamp *ctnc*. By Lemma 3, $V' = V - W - T$, is also a consistent version set with timestamp *ctnc*. Hence the lemma follows. ∎

If the timestamp of a cache is smaller than t_0, Lemma 6 cannot apply. Obviously, from Definition 2, an empty version set is a consistent version set with any timestamp. Therefore, when the timestamp of a cache is smaller than t_0, the mobile computer can drop all the data in its cache and advance the timestamp to *ctnc*. Hence, the range of t_0 and *ctnc* must be carefully chosen so that not too many mobile computers are required to drop all the data in their cache. However, sometimes it is very difficult to choose a suitable timestamp t_0, especially when the variation of the timestamps of the mobile computers in a cell is great. When the range of t_0 and *ctnc* is too small, most of the mobile computers will drop all the data from their cache. When the range is too large, *U_list* will contain data from a large portion of the database. As a result, a significant amount of data in a cache has to be dropped. Simulations are presented in Section 4.4 to find an appropriate value for range of t_0 and *ctnc*. In order to alleviate this situation, the format of an invalidation report can be modified as follows:

$\langle t_0, U_0, t_1, U_1, \cdots t_j, U_j, t_{j+1} = ctnc \rangle$, where

1. $t_0, t_1 \cdots t_k$ are timestamps such that $t_0 < t_1 < \cdots t_k < t_{j+1} = ctnc$,
2. $U_j = \{x | \exists x^k \ s.t. \ t_j < k \leq ctnc\}$, and
3. for $0 < l \leq j-1, U_l = \{x | \exists x^k \ s.t. \ t_l < k \leq ctnc\} - \cup_{l<m\leq j} U_m$.

The intuitive idea of the design is that $U_l \cup U_{l+1} \cup \cdots \cup U_j$ contains all the data that have been updated between timestamps t_l and *ctnc*. Therefore, a mobile computer can advance its timestamp to *ctnc* by dropping all these data, if its original timestamp is between t_l and t_{l+1}. The algorithm to invalidate data is depicted in Figure 3.2, where t is the timestamp of the cache. Note that the algorithm is required to be executed *atomically* with respect to the read-only transactions in the mobile computer.

```
Procedure ReceiveInvalidate
t: the timestamp of the cache
begin
    Suppose an invalidation report ⟨t0, U0, t1, U1, ··· tj, Uj, tj+1 = ctnc⟩ is received
    If t0 > t
        drop the entire cache
        t ← ctnc
    else if ctnc ≤ t
        ignore the invalidation report
        else
            For l from 0 to j
                if tl ≤ t < tl+1
                    For each data x ∈ ∪_{l<i≤ctnc} Ui
                        Drop x from the cache
                    t ← ctnc
            Return
end
```

Figure 3.2: Algorithm for Processing Invalidation Report at a Mobile Computer

3.2.5 Data Broadcasting

After an invalidation report is broadcast, most of the mobile computers in the cell will drop some of their data. These mobile computers may then request new versions of these data at almost the same time. Such a sudden increase in the number of requests may jam the system. In order to alleviate this situation, the server can broadcast some frequently requested data immediately after an invalidation report has been broadcasted. The format of a $DATA$ message is as follows:

$$\left\langle ctnc, x_{i_1}^{j_1}, val_{i_1}, \cdots x_{i_n}^{j_n}, val_{i_m} \right\rangle,$$

where x_i^j are the latest version of the data x_i with version number smaller than $ctnc$ and val_i is the value associated with the version. Note that it is not difficult to merge a $DATA$ message with an invalidation report so that some redundancy can be removed. However, for the sake of explanation, we discuss invalidation reports and $DATA$ messages separately. Upon receiving a $DATA$ message from the local server, a mobile computer executes the algorithm in Fig 3.3 for loading new data into its cache.

```
Procedure ReceiveDATA
begin
    Suppose a DATA message ⟨ctnc, x_{i_1}^{j_1}, val_{i_1}, ··· x_{i_n}^{j_n}, val_{i_m}⟩ is received.
    If ctnc < t
        ignore the DATA message
    else
        For each data version x_i^j in the DATA message
            If t ≥ j and the mobile computer needs the data
                Cache x_i^j
end
```

Figure 3.3: Algorithm for processing Data Broadcasted from a Server

Note that although a $DATA$ message is broadcast immediately after an invalidation report, we cannot assume that the timestamp, t, of the mobile computer is $ctnc$. This is because the mobile computer may fail to receive the invalidation report or the mobile computer ignored the invalidation report because the timestamp of the mobile computer is greater than the ctnc of the server. In the algorithm, if $ctnc < t$, the $DATA$ message will be ignored. Otherwise, the mobile computer can load a data if t is greater than the version of the data. It is because for each data version x_i^j in the $DATA$ message, x_i^j must be the latest version with version number less than or equal to $ctnc$. Hence, if $ctnc \geq t \geq j$, x_i^j is also the latest version with version number less than or equal to t. Therefore, by Lemma 2, the mobile computer can load the data into its cache without destroying cache consistency.

Chapter 4

Simulation Study

This chapter presents a simulation study to find appropriate parameters for the protocol and to evaluate the performance gained by including piggybacking messages in the invalidation reports. We have built a simulating program using *CSIM17* which is a discrete event, process-oriented simulation package based on C++ language. A CSIM17 program models a system as a set of processes which interact with one another by using structures such as requesting service at facilities, waiting for events, etc. Special structures are provided in CSIM17 to collect simulation statistics during the execution of a model.

Besides, the maximum number of mobile computers that can be supported by the system is investigated. To make the simulation more concrete, the comparison of the performance between our protocol and the Amnesic Terminals (AT) method suggested in [6] is given.

4.1 Physical Queuing Model

The wireless communication channel within a cell is modeled as a M/G/1 feedback queue with a FCFS scheduling discipline as shown in Figure 4.1. The server is a fixed rate server with service rate equals the bandwidth of the channel. The service time of a message is equal to (length of the message)/(wireless bandwidth). In fact, a queue is an ideal model of a wireless Ethernet-like channel where the probability of collision is zero and messages are delivered in a FCFS manner.

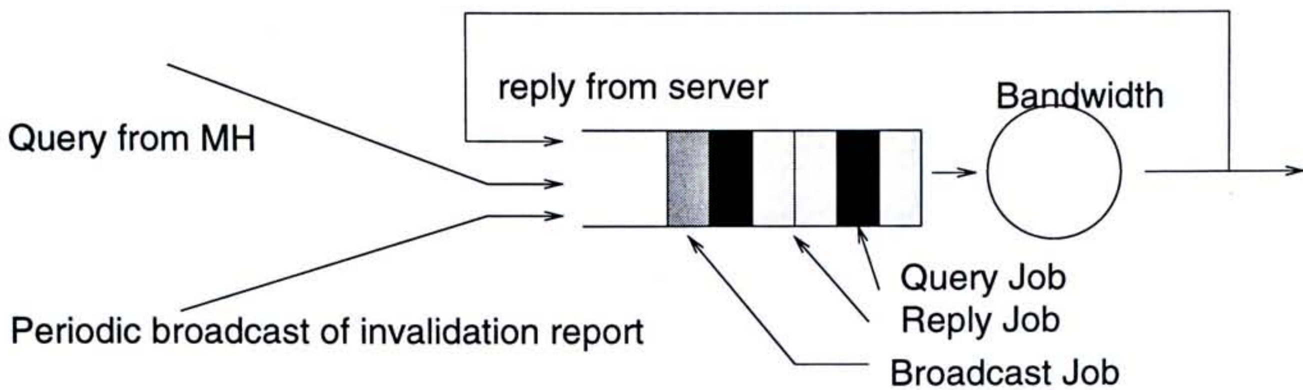


Figure 4.1: Queuing Model of the Wireless Channel

4.2 Logical System Model

The logical queuing model is illustrated in Figure 4.2. For simplicity, only one mobile computer and its local supporting server are shown. The model is based heavily on the one in [2] but is extended for the mobile computing environment.

Each cell consists of a single supporting server and a number of mobile computers. Each mobile computer has one active terminal so the number of mobile computers equals the number of terminals. The communication mechanism among servers is not shown here. It is assumed that the message transmission among supporting servers are reliable.

A read-only transaction at a mobile computer is proceeded as follows. For each read operation of the transaction, *Transaction Manager* is called. Transaction Manager then accesses the cache of the mobile computer and the cached copy is returned if the data in question is cached. Otherwise, the mobile computer sends a request message to the local supporting server for the data. The supporting server then returns the data requested to the mobile computer and the cache of the mobile computer is updated to store the currently requested data. All the communications between the mobile computer and the local supporting server are through the wireless channel. The messages are transmitted in a First-Come-First-Serve manner as mentioned in Section 3.1 of Chapter 3. In order to model an interactive application, there is a time delay between two operations of a transaction and between two consecutive transactions.

If a mobile computer is disconnected while a transaction is active, the transaction will

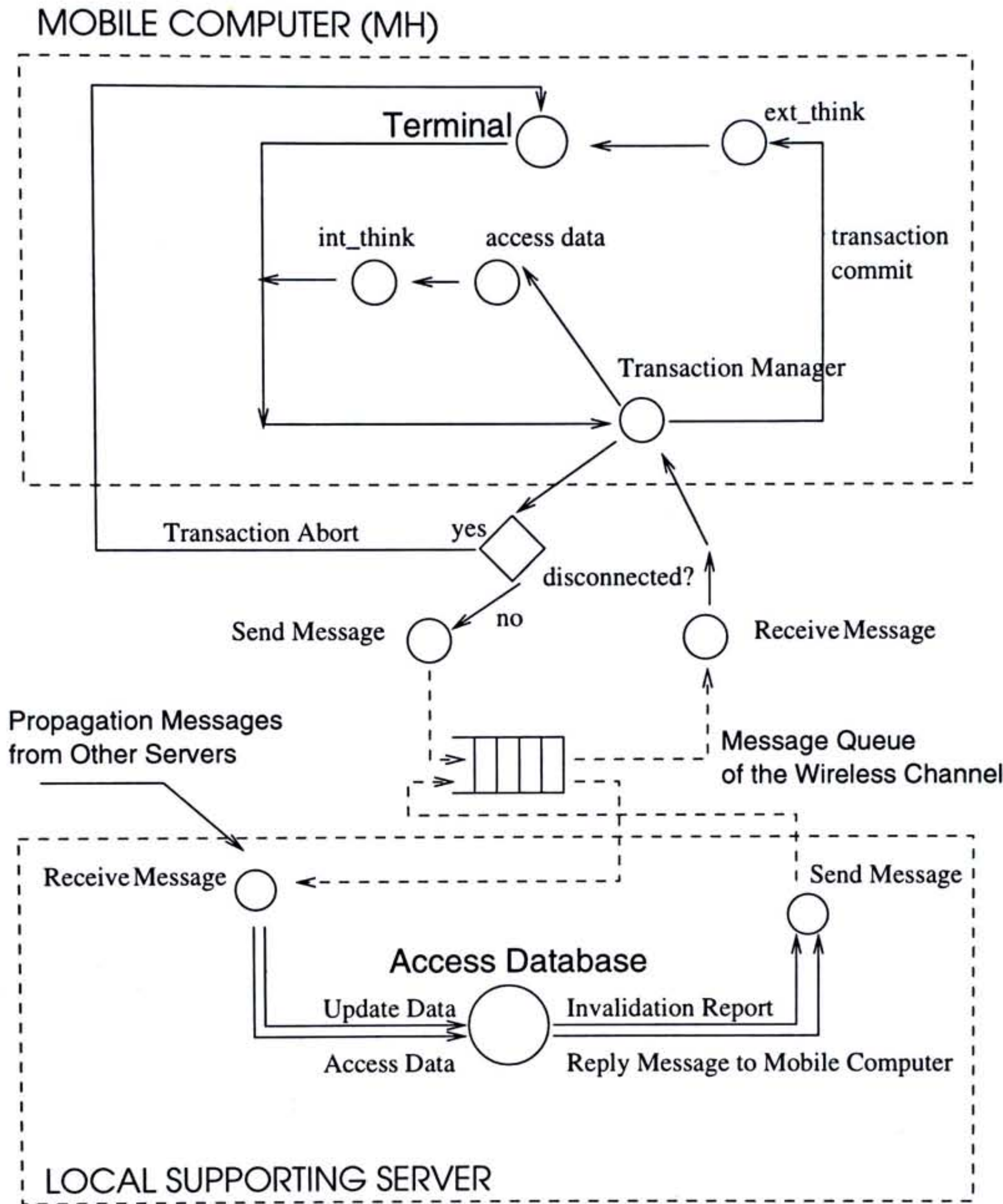


Figure 4.2: Logical Model of the Simulation

be aborted. As in [2], we adopt *fake restart* to implement transaction restart. Hence, a restarted transaction is replaced by a new and independent one.

4.3 Parameter Setting

The fixed parameters used in these experiments are listed in Table 4.1. Some of the values of the parameters are similar to those in [22, 2]. It is believed that the cost of concurrency control is negligible when compared with the cost of message transmission. Therefore,

the cost of concurrency control is ignored in our simulation.

Parameters	Meaning	Values
num_server	number of servers in the system	7
db_size	number of data in database	300
popular_obj	number of popular data	60
popularity	percentage of access fall into popular data	80 %
simtime	total simulation time in seconds	21600
prop_period	time interval between propagating messages in seconds	60
cache size	cache size of mobile computers	30
max_size	size of largest read-only transaction at MH	12
min_size	size of smallest read-only transaction at MH	4
max_up_date	maximum number of update per transaction at server	12
min_up_date	minimum number of update per transaction at server	4
invalid_range	range of data version include in invalidation report	300
int_think	intra-transaction think time	0.1
timeout	time limited to wait for reply from server	5.0
access_size	query size from MH to server in bytes	50
reply_size	message size from a server to MH in bytes	50
bandwidth	wireless communication bandwidth in bit per seconds	1,000,000
obj_size	size of data in bytes	1000
obj_id_size	size of data identity number in bits	100
obj_io	I/O time for accessing an data in seconds	0.035
obj_cpu	CPU time for accessing an data in seconds	0.015
cross_int	time interval a MH crosses boundaries in seconds	1800
disconnect_int	time interval a MH disconnects in seconds	1500
disconnect_period	time duration of each disconnection in seconds	10
int_read	time interval between read-only transactions at MH in seconds	10
int_update	time interval between update transactions at server in seconds	60

Table 4.1: Fixed Simulation Parameters

There are totally 7 local supporting servers in the system, each of them contains 300 fully replicated data. Our experiment simulates the system for 21600 seconds, i.e. 6 hours. The number of data accessed in each transaction in the system is uniformly distributed between 4 to 12 data. As mentioned earlier update transactions are originated at servers only. The data to be accessed are randomly chosen from all the data in the database. The average time interval between two consecutive update transactions within a server is 60 seconds. A server propagates update messages to other servers periodically with a

negative exponential mean of 120 seconds.

For a mobile computer, the *cache_size*, is 30. The time interval between two read-only transactions is 10 seconds. It is assumed that, the data accessed by the mobile computers follows the 80/20 access model, i.e. 80% of data accesses are directed to 20% of the data. We refer these data to as the *popular data* . Mobile computers cross boundaries to different cells periodically with a mean time interval of 1800 seconds. The mean time interval for a mobile computer to be disconnected is 1500 seconds and the mean disconnection duration lasts for 10 seconds.

Each data occupies 1000 bytes and each data identifier takes up 100 bits. Each query generated by a mobile computer has a size of 50 bytes and the size of a reply message is 50 bytes plus the size of the information it contains. After sending out a request message to the local supporting server, a mobile computer will assume that it is being disconnected if the reply message from the local supporting server cannot be received within the *timeout* period. The time required for I/O and CPU processing for a read operation at the server are 0.035 and 0.015 seconds respectively.

The value of *invalid_range* is the range of versions sequence included in each invalidation report. For example, if the value *invalid_range* is set to be 300, then each invalidation report includes all the data_id of the data in the database which are updated from version number ($ctnc - 300$) to *ctnc*. If the invalidation report includes piggyback messages, then the values of popular data which have been updated within this range are also included. In Section 4.4, it is found that 300 is the most appropriate value for *invalid_range* under the scenarios that we studied. Therefore, we employed this value in our following experiments.

The parameters that are varied are not listed in Table 4.1. They will be given in the description of the corresponding experiments. The performance of the proposed protocol is measured in terms of number of *cache drops*, *cache hit ratio* and *bandwidth utilization.* As explained in Section 3.2.4, a mobile computer may need to remove all data from its cache. In the following experiments, we refer to the step of removing all data from the caches of mobile computers to as *cache drop*. Whenever there is a data access at a mobile

computer, if that data is found in the cache, then it is referred to as a *cache hit*, otherwise, it is a *cache miss*. *Cache hit ratio* refers to the ratio between the number of cache hit and the total number of data accessed. *Bandwidth utilization* of the wireless channel is calculated by dividing the busy time by elapsed time.

Whenever there is a cache miss, the mobile computer sends a query to the local supporting server requesting the data that is not cached. We refer to this query as an *uplink query*. The mechanism involved is mentioned in Section 3.2.3. The experimental results are presented in the following sections.

4.4 The Significance of the Length of Invalidation Range

The aim of this section is to show the difference in performance of our proposed protocol with different values of *invalid_range*. The value of *invalid_range* represents the range of versions sequence included in each invalidation report. The value of invalid_range should be carefully selected especially when the variation of the timestamps of mobile computers is great as described in Section 3.2.4.

The timestamps of mobile computers would be varying when the value of *ctnc* among the local supporting servers have great differences or the mobile computers cannot update the values of their timestamps as a result of missing the invalidation reports. However, since it is assumed that the servers are connected by a wired network which is reliable, the differences among the timestamps of the servers should be small. Hence, the main cause for varying timestamps should be due to the fact that mobile computers always miss invalidation reports because of temporary disconnection or interference in the wireless networks. The following reasons make the selection of a suitable value for invalid_range difficult.

- If the value of invalid_range is *small*, most of the mobile computers will have the values of their timestamps less than the value of (*ctnc* - *invalid_range*). These

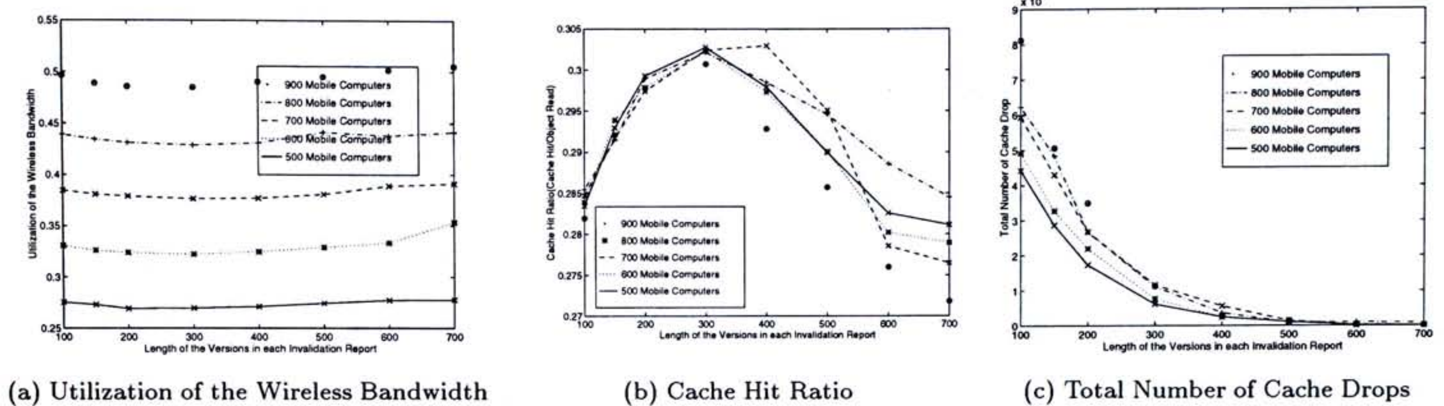
mobile computers will then have to remove all data from their caches. Cache drops increase the burden of the wireless network. As the caches become empty, the next data access cannot be answered by the mobile computer. The cache hit ratio hence decreases. It also cuts down the benefits brought with the caching strategy.

- It seems to be beneficial to have a *large* value of invalid_range. However, this will require more information to be included in each invalidation report. This raises the bandwidth consumption and at the same time, more data will be considered as invalid. Since each invalidation report contains more data, more invalid data need to be dropped after the invalidation report is received. Consequently, the cache hit ratio decreases and more uplink queries are required. As a result, the bandwidth consumption increases.

In the following, experiments conducted to find a suitable value for *invalid_range* which balances the number of cache drops, the cache hit ratio and the bandwidth utilization are shown.

4.4.1 Performance with Different Invalidation Range

As stated earlier, when the difference between the values of the timestamps of mobile computers is great, it is difficult to have a suitable value for the invalidation range. In this section, we try to evaluate the performance of the system with different values of invalidation range under a scenario that the variation in timestamps of the mobile computers is great. The values of parameters used in this experiment are the same as those listed in Table 4.1 except that the disconnection frequency and duration of disconnection are increased as we need to create a scenario where the mobile computers always miss invalidation reports. The timestamps among mobile computers would then have great differences. The mobile computers temporarily disconnect for every 500 seconds and each disconnection lasts for 100 seconds. The values given are exponentially distributed over a fixed mean value. The results are shown in Figure 4.3.



(a) Utilization of the Wireless Bandwidth (b) Cache Hit Ratio (c) Total Number of Cache Drops

Figure 4.3: Performance with Different Invalid Range

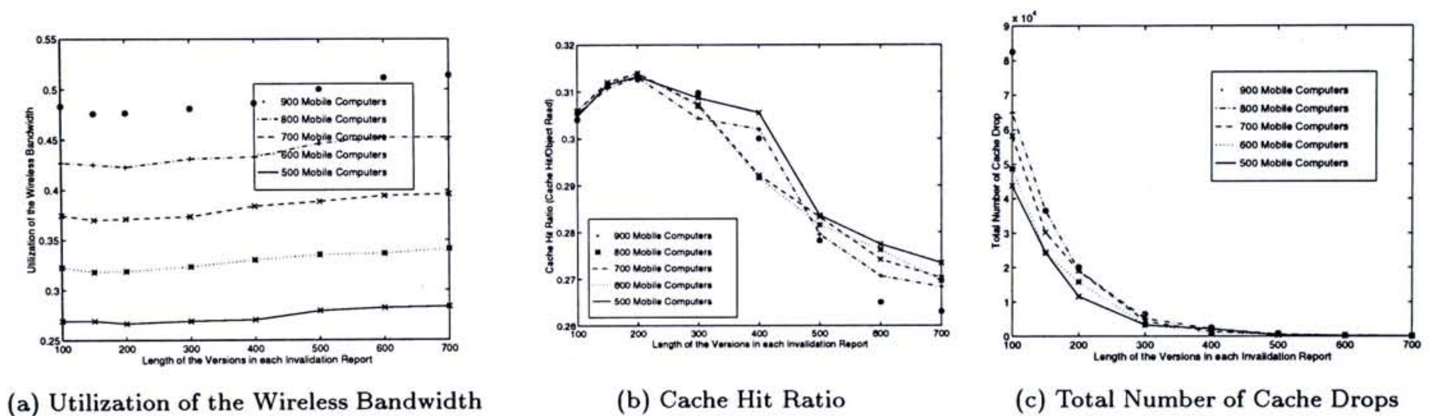
As stated above, the overall performance of our proposed protocol will be worsened if the value of invalidation_range is either too large or too small.

Figure 4.3(a) shows that the bandwidth utilization is the smallest when the value of invalid_range is about 300. It is found that the discrepancy in bandwidth utilization with different values of invalid_range is not significant while there are observable variance in cache hit ratio and total number of cache drops. This may be due to the fact that when the value of invalid_range is large, the increased burden in communication due to including more information in each invalidation report is compensated by the reduction in the number of cache drops. Fewer number of cache drops implies fewer number of cache misses and hence there are fewer uplink queries from mobile computers to local supporting servers. The bandwidth consumption becomes lower. When the value of invalid_range is small, most of the mobile computers may have the values of their timestamps out of the range. This results in more cache drops after invalidation reports are received. However, there are fewer data need to be included in each invalidation report.

The cache hit ratio reaches its maximum when the value of invalid_range equals 300 as shown in Figure 4.3(b). Figure 4.3(c) shows that the total number of cache drops is relatively low when the value of invalid_range is equal to 300. This shows that when the value of invalid_range equals 300 it gives an optimal performance both in terms of cache hit ratio and total number of cache drops.

4.4.2 Increasing the Update Frequency

In the following experiment, the mean time interval between each update at a server is decreased from 60 to 10 seconds. The values of other parameters are the same as that of the experiments in Section 4.4.1. With update frequency increased, more data will be updated during the same time interval than in the previous experiment. For the same value of invalid_range, more invalid data will be included in each invalidation report. To include the same number of invalid data within each invalidation period, the invalidation report should have a smaller value of invalid_range. It is predicted that the curves will shift to the left. The results of this experiment are shown in Figure 4.4.



(a) Utilization of the Wireless Bandwidth (b) Cache Hit Ratio (c) Total Number of Cache Drops

Figure 4.4: Increasing the Update Frequency

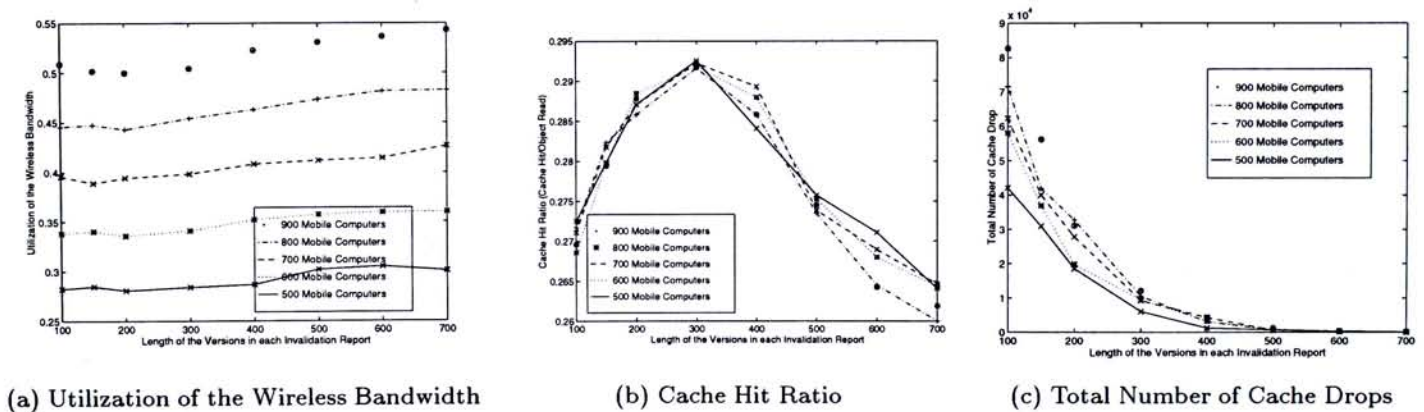
It is found that the cache hit ratio is the highest when the value of invalid_range is close to 200 and the bandwidth utilization is relatively small at this stage. The best value of invalid_range shifts to the left as expected.

As more data are dropped from the cache, the number of cached data should decrease and so as the cache hit ratio. However, as shown in Figures 4.3(b) and 4.4(b) the cache hit ratio increases with the update frequency. This may be due to the fact that after an invalidation report is received by a mobile computer, more data need to be removed from the cache. The space in the cache will be replaced by popular data immediately as the new values of popular data are included in the invalidation reports (the piggybacking messages). The number of cached popular data increases, which increases the probability

of cache hit. The cache hit ratio hence increases. In the following experiment, we justify the above argument by sending invalidation reports without including the new values of popular data (i.e. no piggybacking message).

4.4.3 Impact of Piggybacking Popular Data

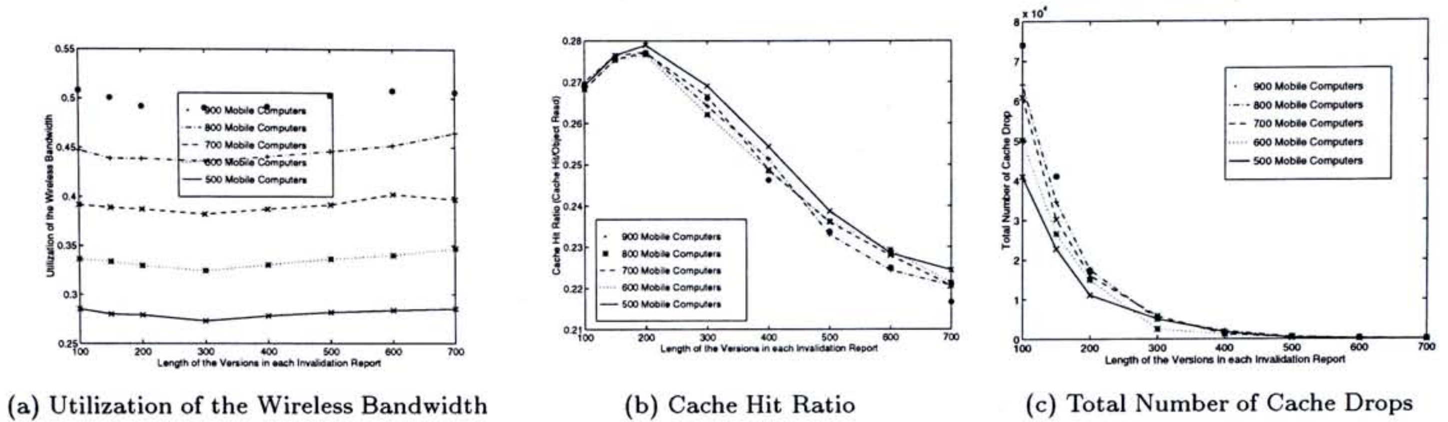
In this section, the experiments in Section 4.4.1 and Section 4.4.2 are repeated except that the new values of popular data are not included in the invalidation reports. The results are shown in Figures 4.5 and 4.6 respectively. It is expected that after the update frequency is raised, more data needs to be dropped out after an invalidation report is received. These dropped data will not be replaced by popular data as piggybacking messages are excluded in the invalidation reports in this experiment. The number of cached data decreases after an invalidation report is received. Besides, the cache hit ratio for the one with higher update frequency should be less than the one with lower update frequency.



(a) Utilization of the Wireless Bandwidth (b) Cache Hit Ratio (c) Total Number of Cache Drops

Figure 4.5: Impact of Piggybacking Popular Data

As shown in Figures 4.5(b) and 4.6(b), the cache hit ratio is smaller as compared with that in Figures 4.4(b) and 4.3(b) in Section 4.4.1 and Section 4.4.2 respectively. This is because without piggybacking messages, invalid data are not replaced after invalidation. The cache hit ratio in Figure 4.5 is higher than that of Figure 4.6 as expected. This shows that the reasoning in the previous paragraph is correct.



(a) Utilization of the Wireless Bandwidth (b) Cache Hit Ratio (c) Total Number of Cache Drops

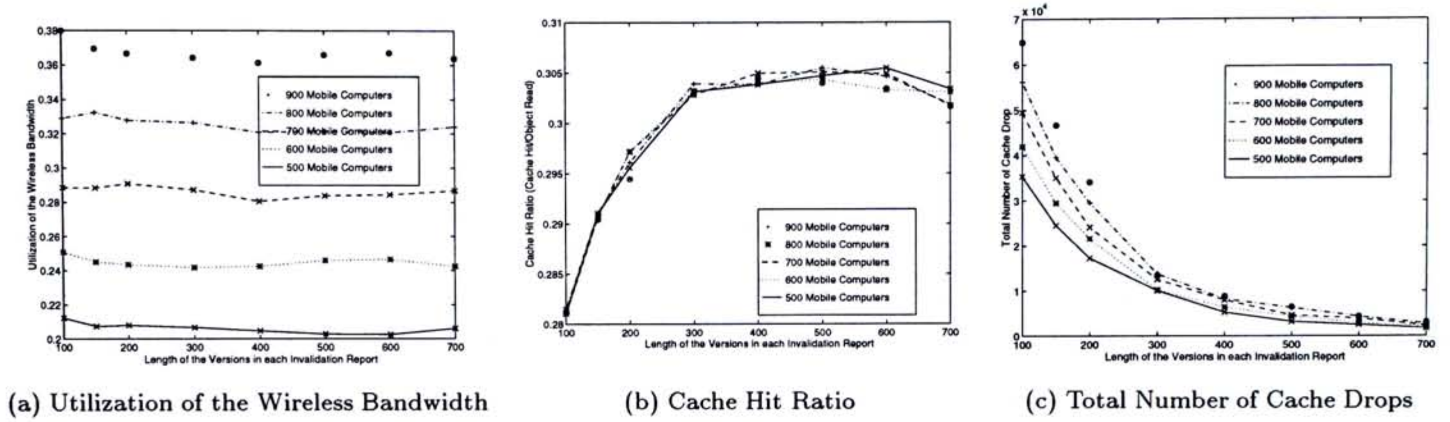
Figure 4.6: Impact of Piggybacking Popular Data (Higher Update Frequency)

The maximum cache hit ratio is found when the value of invalid_range equals 300 and 200 respectively that is similar to the results in Section 4.4.1 and Section 4.4.2. The optimal value of invalid_range becomes smaller as the update frequency increases.

4.4.4 Increasing the Disconnection Period

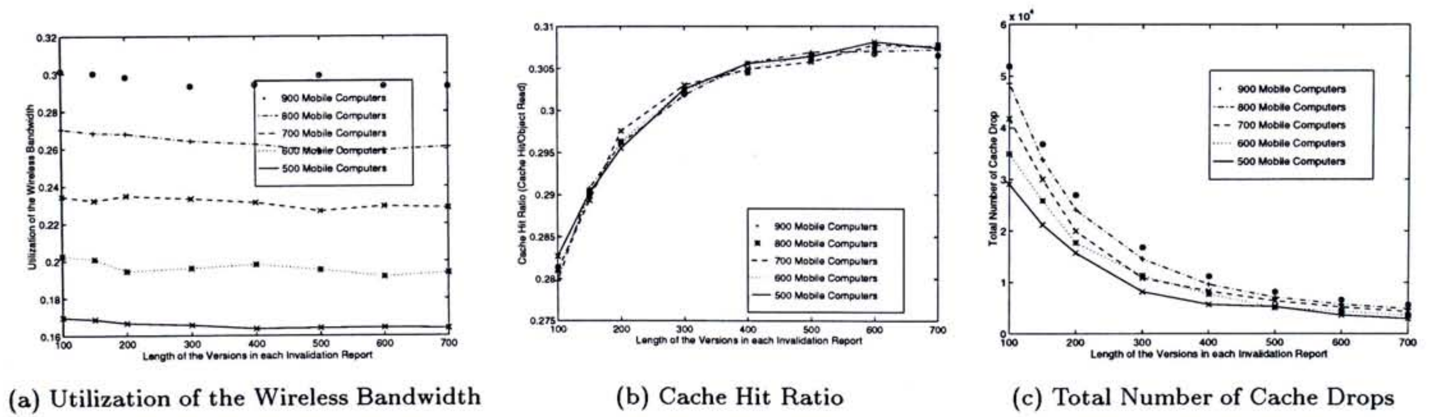
Besides increasing the disconnection frequency, extending the disconnection period would also result in large variation of timestamp values among mobile computers. It is because, the values of the timestamps of mobile computers are advanced after the invalidation reports are received. If a mobile computer is always disconnected, it will miss most of the broadcasted invalidation reports. Therefore, mobile computers cannot advance the values of their timestamps and the values diverse. In the previous experiments, the disconnection duration is 100 seconds. In this section, the disconnection duration is increased to 300 and 500 seconds respectively. The disconnection interval is 500 seconds as is in the previous experiments in Section 4.4.1. The remaining parameters are the same as in Table 4.1. The experimental results are shown in Figure 4.7 and 4.8.

It is found that the bandwidth utilization decreases as the duration of disconnection increases as shown in Figures 4.3(a), 4.7(a) and 4.8(a) (Their disconnection periods are 100, 300 and 500 seconds respectively). Figures 4.3(c), 4.7(c) and 4.8(c) show that the total number of cache drops decreases as the total number of connected mobile computers



(a) Utilization of the Wireless Bandwidth (b) Cache Hit Ratio (c) Total Number of Cache Drops

Figure 4.7: Increasing the Disconnection Period to 300 seconds



(a) Utilization of the Wireless Bandwidth (b) Cache Hit Ratio (c) Total Number of Cache Drops

Figure 4.8: Increasing the Disconnection Period to 500 seconds

decreases. It is because the number of mobile computers that are connected with the system decreases as the disconnection duration increases. Therefore, fewer number of uplink queries and reply messages are transmitted through the wireless networks. The bandwidth utilization cuts down. Besides, and so as the number of cache drops.

It is shown in Figures 4.3(b), 4.7(b) and 4.8(b) that the variation in cache hit ratio decreases and the curves become flattened. A mobile computer would miss the invalidation reports which are broadcast when it is disconnected. As the disconnection duration increases a mobile computer would miss most of the broadcasted invalidation reports. Missing invalidation reports implies that the mobile computers in question will not perform cache invalidation. Increasing the value of the invalid_range will not significantly affect the contents of caches of mobile computers. Therefore, the curves for cache hit ratio flatten.

Figures 4.3(c), 4.7(c) and 4.8(c) show that the total number of cache drops decreases as the disconnection period increases. The reasons behind is obvious. As the disconnection period increases, the number of mobile computers that are connected with the system decreases. Fewer mobile computers perform cache drops, hence the total number of cache drops decreases. However, it should be noticed that as the disconnection period of mobile computers increases, the data that the mobile computers read become more outdated.

It is found that the optimal value of invalid_range varies in different scenarios. Since it is better to have an unique value of invalid_range in all the experiments for the sake of comparison, we will have the value of invalid_range set to be 300 in the following experiments. It is the best value found in Section 4.4.1.

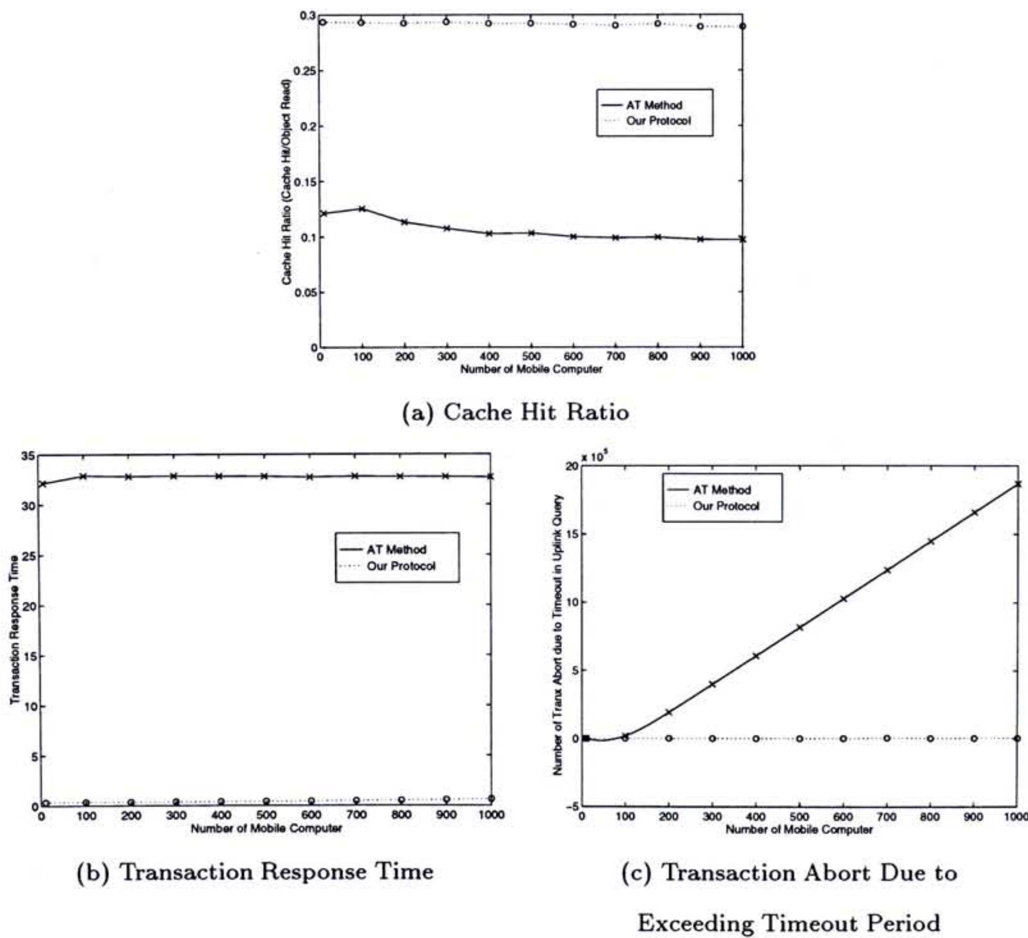
4.5 Comparison of the Proposed Protocol with the Amnesic Terminal Protocol

The objective of this section is to study the performance of the proposed caching protocol under different scenarios. The Amnesic Terminals (AT) method suggested in [6] is used as

a control experiment. Since the invalidation reports of the AT method do not include any piggybacking messages, piggybacking messages are excluded in the invalidation reports of the proposed protocol in this section. In the experiments followed, the parameter values as shown in Table 4.1 are used.

4.5.1 Setting a Short Timeout Period

In the first set of the experiments, the *timeout* period is set to be 5 seconds. It is found that the performance of the AT method is poor in terms of cache hit ratio and transaction response time as shown in Figure 4.9.



(a) Cache Hit Ratio

(b) Transaction Response Time

(c) Transaction Abort Due to Exceeding Timeout Period

Figure 4.9: Our Protocol VS AT Method (Timeout Period = 5 sec)

In Figure 4.9(a), it is found that the cache hit ratio drops and the number of transaction

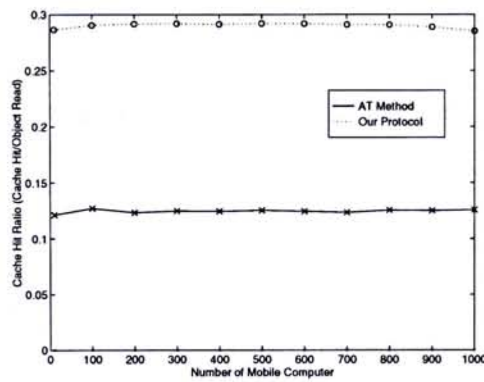
aborts due to exceeding the timeout period increases linearly when there are more than 100 mobile computers in the system. According to the AT method, uplink queries are sent after the invalidation reports are received. All the uplink queries arriving during each invalidation period will be sent simultaneously. Sending a large number of uplink queries at the same time results in a keen competition for the wireless bandwidth. It is believed that the round-trip delay for the uplink queries and reply messages takes up the whole *timeout* period when there are 100 mobile computers in the system. Hence in the AT method, as the number of mobile computers increases, additional queries cannot be processed but aborted. Maximum transaction response time is reached when there are about 100 mobile computers and the average transaction response time remains to be the maximum value afterwards. The system is believed to be saturated when there are 100 mobile computers in the system if the AT method is adopted.

4.5.2 Extending the Timeout Period

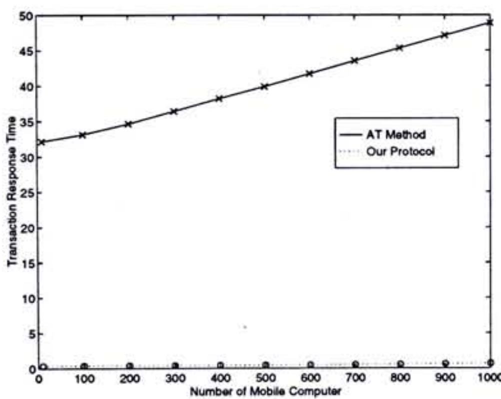
It is believed that the poor performance of the AT method in Section 4.5.1 is due to the short timeout period. In the AT method, uplink queries are sent after invalidation reports are received which are broadcasted every 60 seconds. The invalidation period is much longer than the timeout period of 5 seconds. In this experiment, the timeout period is extended to 60 seconds which is the same as the invalidation period. Other parameters are the same as those in Table 4.1. The results are illustrated in Figure 4.10.

Figure 4.10(a) shows that our protocol still outperforms the AT method in terms of cache hit ratio. It is necessary for a mobile computer to perform cache drop in order to ensure cache consistency in the AT method when a mobile computer crosses boundaries to different cells or is disconnected. However, it is not necessary in the proposed protocol. It is obvious that the cache hit ratio of the proposed protocol will be higher than that of the AT method.

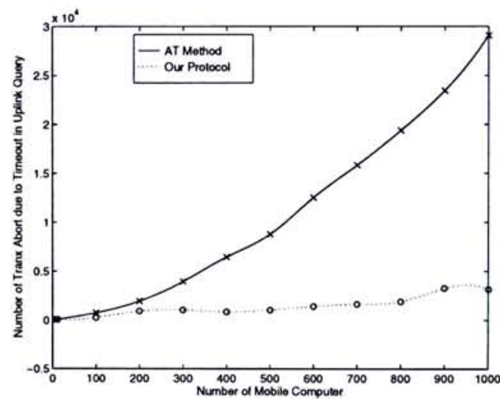
Figure 4.10(b) presents the transaction response time. In the AT method, uplink queries are sent after receiving an invalidation report. Since the average time interval for



(a) Cache Hit Ratio



(b) Transaction Response Time



(c) Transaction Abort Due to Exceeding Timeout Period

Figure 4.10: Our Protocol VS AT Method (Timeout Period = 60 sec)

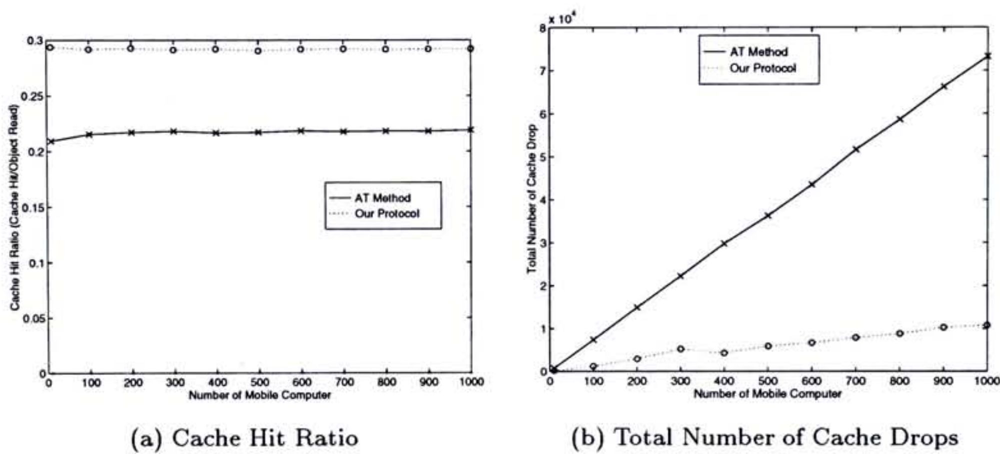
a server to broadcast invalidation reports is 60 seconds, the average waiting time for an uplink query to be sent should be 30 seconds. Hence the transaction response time of the AT method should be at least 30 seconds as shown in Figure 4.10(b). The transaction response time of the AT method increases as the number of mobile computers increases while in the experiment of Section 4.5.1, the transaction response time is nearly constant. This is because, as the timeout period is extended, those transactions that need to be aborted in the previous experiment can be processed. As a result, the average transaction response time increases as the number of mobile computer increases.

Finally, Figure 4.10(c) shows that the total number of transactions abort due to exceeding the *timeout* period is greatly reduced as compared with the previous experiment. In the previous experiment, the order of magnitude is up to 10^5; in this experiment, the number is decreased to the order of 10^4. The number of transactions aborted in the AT method is still higher than that of the proposed protocol. It should be noticed that besides the high contention on the wireless channel, there are other reasons that result in message loss. For example, when a mobile computer crosses boundaries to different cells or temporarily disconnects it may miss the reply messages. For the AT method, the size of the uplink queries is larger than that of our proposed protocol due to its poor cache hit ratio. This leads to a longer transmission time. After sending out an uplink query, a mobile computer using the AT method has a higher chance to cross the boundary to different cells than our protocol. Consequently, the probability of missing the reply message from the local supporting server is higher. Therefore, more transactions need to be aborted with the AT method even after extending the timeout period.

4.5.3 Increasing the Frequency of Temporary Disconnection

In this section, the performance of the proposed protocol and the AT method under a scenario where mobile computers frequently disconnect and each disconnection lasts for a longer time is studied. In the previous experiments, each mobile computer disconnects periodically according to a negative exponential distribution with a mean value of 1500

seconds and its disconnection duration is exponentially distributed over a mean value of 10 seconds. In this section, we increase the disconnection frequency such that each mobile computer disconnects every 500 seconds for 100 seconds. Other parameters are listed in Table 4.1. As stated in Section 3.2.4, the AT method cannot ensure cache consistency if a mobile computer loses the last invalidation report. As a result, a mobile computer has to remove all the cached data in order to ensure cache consistency after each disconnection. This will increase the number of cache drops and lower the cache hit ratio. The total number of cache drops and cache hit ratio of the two methods are shown in Figure 4.11.



(a) Cache Hit Ratio (b) Total Number of Cache Drops

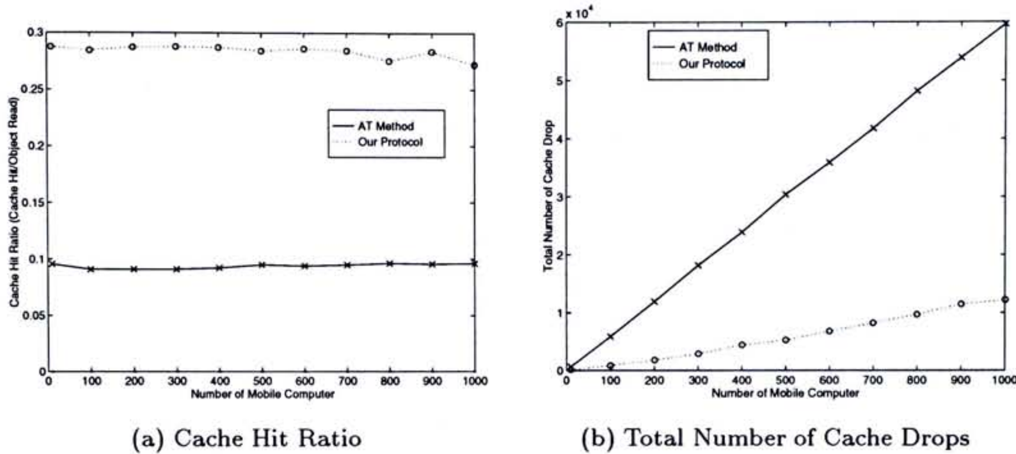
Figure 4.11: Increasing the Frequent of Temporary Disconnection

The graphs show that the number of cache drops of the AT method is much higher than that of our protocol and the high number of cache drops lowers the cache hit ratio.

4.5.4 Increasing the Frequency of Crossing Boundaries

It is well understood that servers are never perfectly synchronized in practise. Therefore, in the AT method a mobile computer has to drop its cache after crossing boundaries to different cells in order to ensure cache consistency. In the previous experiment, the time interval between a mobile computer crosses boundary to different cells is exponentially distributed over a mean value of 1800 seconds. In this experiment the mean of the random variable is decreased to 500 seconds. Other parameters are the same as those listed in

Table 4.1. The results are shown in Figure 4.12. As stated before, high frequency of cache drop will worsen the overall performance of the system. The cache hit ratio is therefore reduced.



(a) Cache Hit Ratio (b) Total Number of Cache Drops

Figure 4.12: Increasing the Frequency of Crossing Boundaries

The proposed protocol outperforms the AT method in terms of cache hit ratio and transaction response time in the above scenarios. In the following experiments, we will concentrate on the difference in performance of the proposed protocol by including and excluding piggybacking messages.

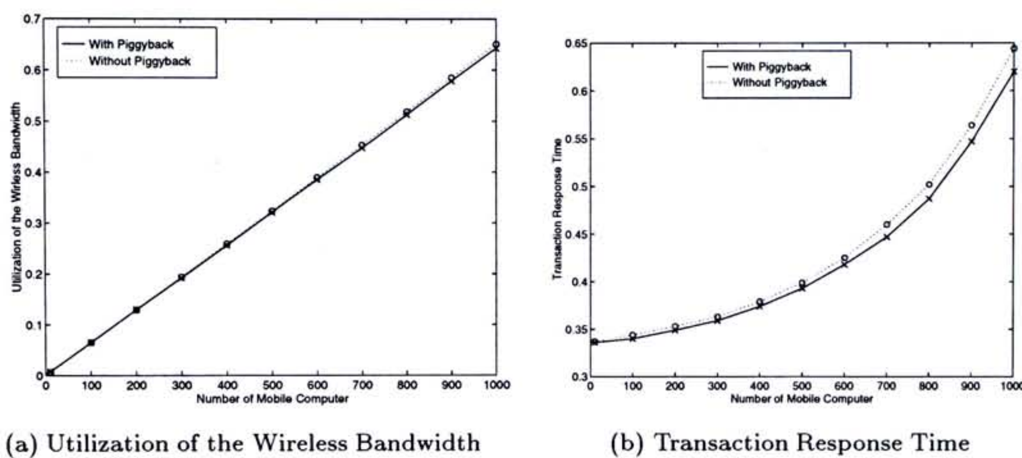
4.6 Evaluate the Performance Gain with Piggybacking Message

This section evaluates the difference in performance of the proposed protocol by adding piggybacking messages to the invalidation reports. Adding piggybacking messages reduces the total number of uplink queries by supplying the new values of popular data to the caches of mobile computers. Consequently, bandwidth consumption is reduced. On the other hand, it enlarges the size of each invalidation report, as more information are included in the invalidation reports. Bandwidth consumption is raised. Therefore, piggybacking extra information is not a sufficient condition to improve the overall performance

of the proposed protocol. In the following experiments, we will study different scenarios to verify the performance gain of piggybacking messages.

4.6.1 Adding Piggybacking Messages

The scenario studied in this section is the same as those in the above experiments. The parameters values are listed in Table 4.1. The results are presented in Figure 4.13.



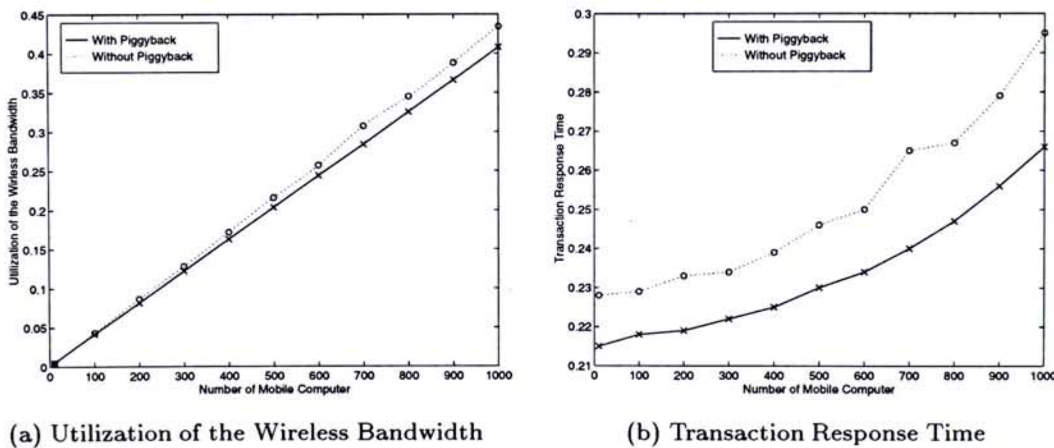
(a) Utilization of the Wireless Bandwidth (b) Transaction Response Time

Figure 4.13: Performance of Our Protocol With & Without Piggybacking

As shown in Figure 4.13(a), piggybacking extra information with the invalidation reports does not intensify the bandwidth consumption of our protocol significantly but shortens the transaction response time. This is because, as updated values are attached to the invalidation messages, the invalid data dropped from the cache of mobile computers will be replaced by popular data. These popular data are likely to be accessed by the following queries, therefore more queries can be satisfied with the cached data. Fewer uplink queries implies lower wireless bandwidth consumption and shorter transaction response time as shown in Figure 4.13(b). Furthermore, since the popular data are likely to be queried soon, piggybacking these data can supply the data to the mobile computers before they are required. The increase in bandwidth consumption due to a larger invalidation report is compensated by the reduction in the number of uplink queries. Therefore, there is not a significant difference in bandwidth consumption between the two cases.

4.6.2 Reducing the Number of Popular Data

After receiving an invalidation report, invalid data are removed from the cache of a mobile computer. Piggybacking the values of popular data replaces the cache with update values of these popular data. Since *Least Recent Used* strategy is adopted as the cache replacement strategy, those cached popular data may not be accessed before they are swapped out for other data. If the number of popular data is reduced, and the probability of accessing popular data is unchanged, the probability of accessing each popular data will increase. Cached popular data have a higher probability to be accessed before being swapped out. More queries can be satisfied by cached data, this reduces the bandwidth consumption and at the same time shortens the transaction response time. In the following experiment, we will verify the above hypothesis by reducing the number of popular data from 60 to 30. The values of other parameters are given in Table 4.1. The results are shown in Figure 4.14.



(a) Utilization of the Wireless Bandwidth (b) Transaction Response Time

Figure 4.14: Reducing the Number of Popular Data

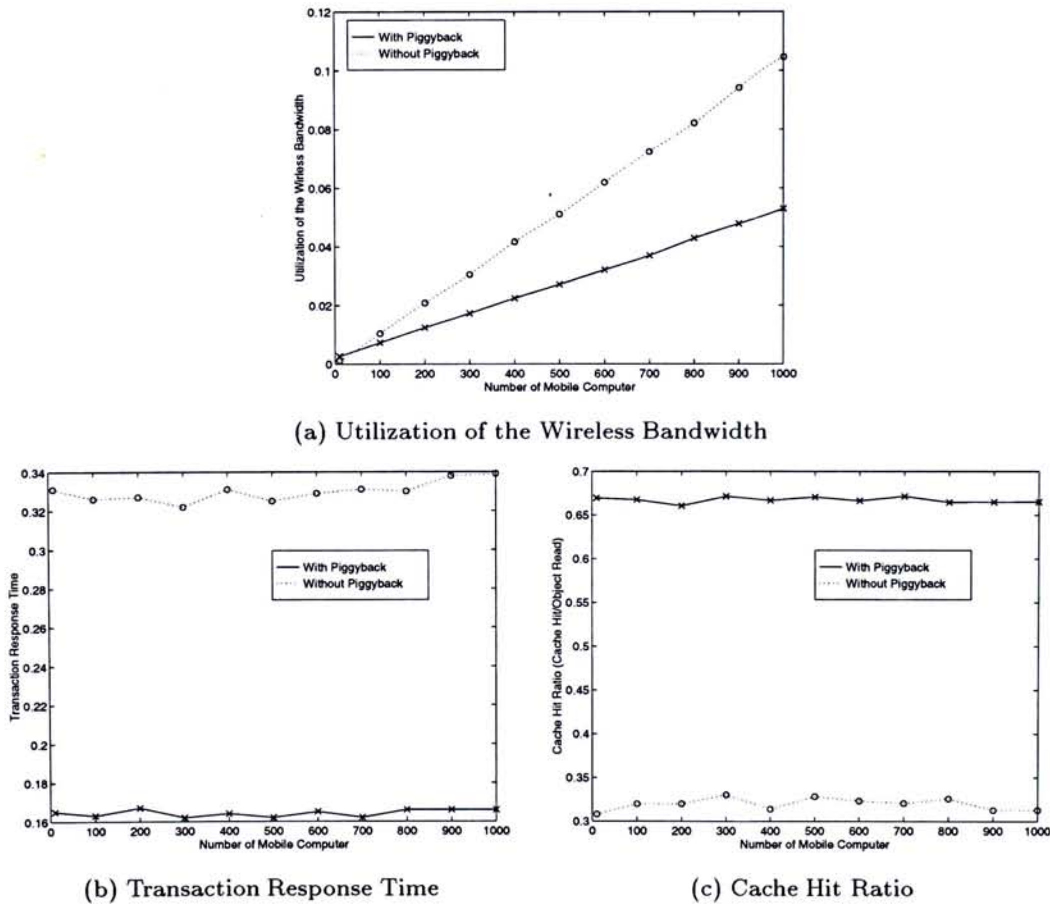
As shown in Figure 4.14(a), the bandwidth consumption is significantly reduced when compared with the previous experiment. The transaction response time is also shortened as shown in Figure 4.14(b).

If the update frequency is higher than that of query, it is believed that more invalid data will be dropped from cache after receiving invalidation reports. Piggybacking the values

of popular data with the invalidation reports replaces the cache with updated popular data, the performance of the protocol should be improved significantly. We evaluate the above prediction in the following section.

4.6.3 Increasing the Frequency of Updates

This section studies the performance of the proposed protocol under a scenario where updates are more numerous than queries. Previously, the mean time interval for updates and queries are 60 and 10 seconds respectively. In this section, the mean time interval for updates and queries are interchanged, that is 10 and 60 seconds respectively. The values of other parameters are listed in Table 4.1. The results are shown in Figure 4.15.



(a) Utilization of the Wireless Bandwidth

(b) Transaction Response Time

(c) Cache Hit Ratio

Figure 4.15: Increasing the Frequency of Updates

Figure 4.15(a) shows that bandwidth utilization is greatly reduced by piggybacking

extra information. As the update frequency increases, more data needs to be dropped from the caches of mobile computers after each invalidation report is received. Piggybacking the values of popular data replaces the cache immediately. As a result, the cache hit ratio increases as shown in Figure 4.15(c), and the number of uplink queries decreases. Consequently bandwidth consumption reduces. Average transaction response time is also shortened as shown in Figure 4.15(b), because more queries can be satisfied by the cached data. Moreover, since the number of queries is reduced, more transactions can be processed without spending a long time in queuing in the wireless channel.

Adding piggybacking messages to our protocol significantly improves the performance when the frequency of updates is high and the number of popular data is small such that the probability of accessing each popular data is high.

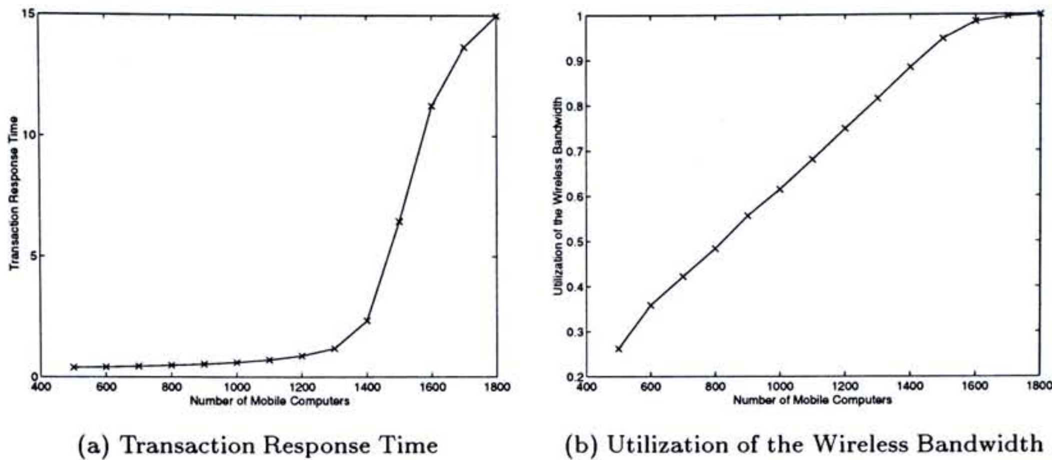
4.7 Behaviour of the Proposed Protocol

This section investigates the maximum number of mobile computers that can be supported by the proposed protocol.

4.7.1 Finding Maximum Number of Mobile Computers

The parameter values in this section are the same as those listed in Table 4.1. The results are shown in Figure 4.16.

It is found that when the number of mobile computers is close to 1400, the wireless channel is saturated, i.e. the bandwidth utilization approaches 100%. The transaction response time also increases significantly when there are about 1400 mobile computers in the system.



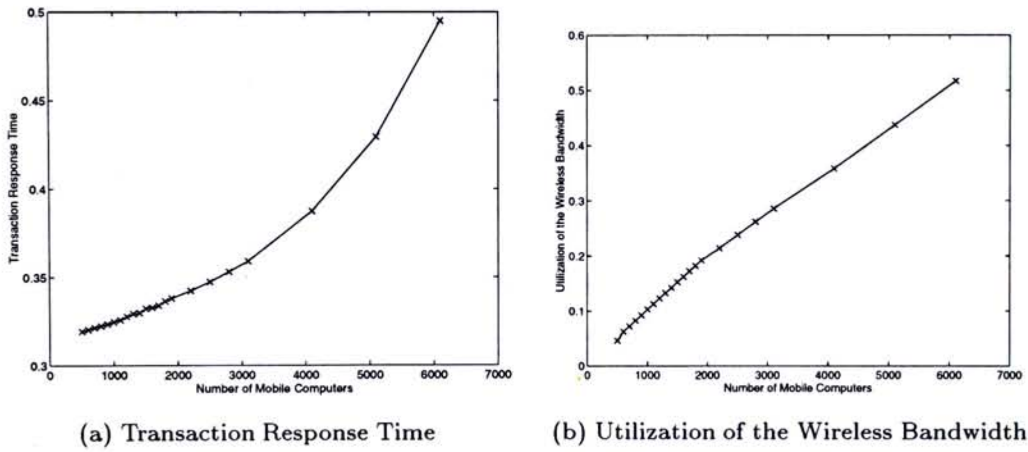
(a) Transaction Response Time (b) Utilization of the Wireless Bandwidth

Figure 4.16: Tolerance of Our Proposed Protocol

4.7.2 Interchanging the Frequency of Read-Only and Update Transactions

Since the bandwidth consumption is related to the number of uplink queries and the size of each invalidation report, changing these parameters will result in the change in performance. In the following experiment, the frequencies of update transactions and read-only queries are interchanged. The mean time interval between read-only transactions is 60 seconds and update transaction is 10 seconds. In this section their values are 10 and 60 seconds respectively. The remaining parameters are given in Table 4.1. The graphs are shown in Figure 4.17.

Increasing the update frequency increases the size of each invalidation report by including more *data_id* of invalid data and the values of those updated popular data. However, the increase is not significant when compared with the reduction in bandwidth consumption due to the reduction in the number of read operations. It is found that the bandwidth utilization is much lower and the transaction response time is shortened. The system at this stage is not saturated.



(a) Transaction Response Time

(b) Utilization of the Wireless Bandwidth

Figure 4.17: Interchanging the Frequency of Read-Only and Update Transactions

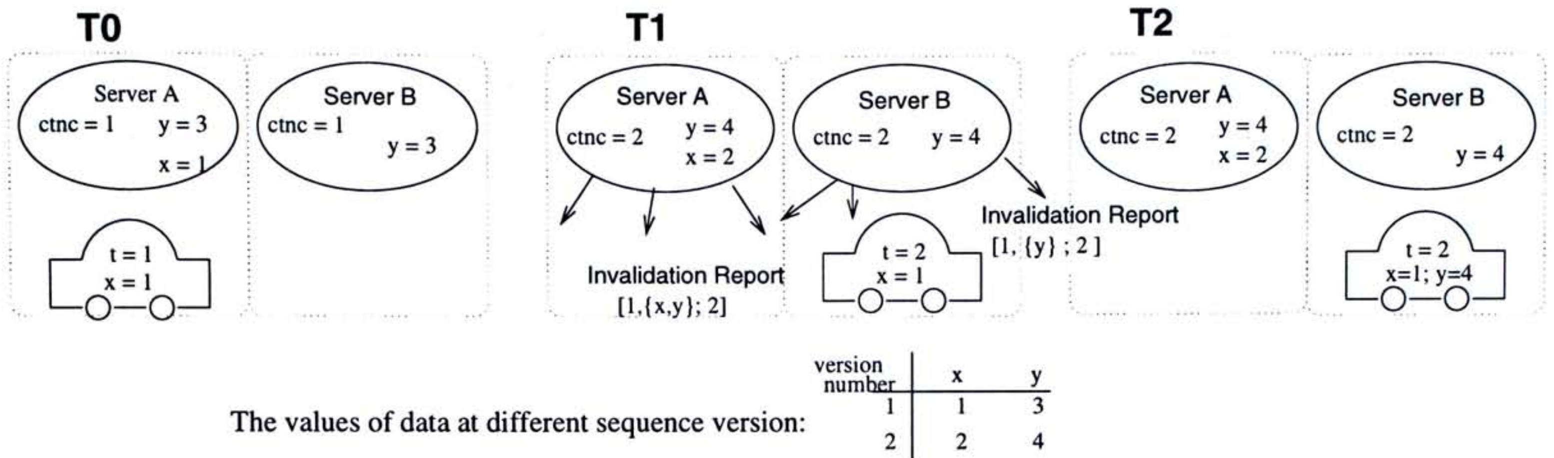
Chapter 5

Partially Replicated Database System

So far we assume that the database is fully replicated, i.e., every server stores a copy of each data. In practice some location sensitive data may not be fully replicated and the database is referred to as a partially replicated database. The issue of maintaining cache consistency in mobile computing environment with a partially replicated database is more complicated, since a server only manages information of its local copies. Therefore, the invalidation reports of a server will never contain data that are not supported by itself. Problems arise when a mobile computer carries some partially replicated data to a cell that does not support these data. Since the data will not be invalidated in this cell, cache consistency may be destroyed when the mobile computer advances its timestamp. Such a scenario is illustrated in Figure 5.1.

5.1 Proposed Amendments

For the reasons stated above, the proposed protocol in Chapter 3 for a fully replicated database system cannot be applied to a partially replicated database system directly. In order to provide a complete solution in maintaining cache consistency, necessary amendments to the proposed protocol are investigated. Three amendments using different approaches are presented in the following section. Simulation results are shown in Section 5.2 to evaluate the performance of these approaches under different scenarios.



At Time T_0	The mobile computer loads x from server A and sets its timestamp, t, to 1.
At Time T_1	The mobile computer crosses the boundary to server B, at the same time both x and y are updated by a transaction. These events are indicated in the invalidation reports which are sent afterwards. However, the mobile computer at server B is not informed about the update of data item x, but the timestamp of the cache advances to 2.
At Time T_2	Cache consistency is destroyed when the mobile computer loads y from server B.

Figure 5.1: A Scenario for a Partially Replicated Database

5.1.1 Not Cache Partially Replicated Data (Method 1)

If partially replicated data are not cached in mobile computers, the problem stated in Figure 5.1 can be avoided. Moreover, it should be noticed that the cache of a mobile computer should alway be full except during its initial state. Whenever a new data has to be added into the cache memory, the *Least Recently Used (LRU)* method is employed to remove some cached data in order to free the cache memory. In some cases, fully replicated data are more frequently accessed than partially replicated data. If all kinds of data can be cached in mobile computers, a frequently accessed popular data may be swapped out to free cache space for a partially replicated data. This scenario can be avoided if a mobile computer do not cache partially replicated data. At any time a mobile computer needs a partially replicated data, it accesses the local supporting server. The main advantage of this method is that infrequently accessed partially replicated data will not occupy the limited space of the cache memory. However, a mobile computer needs to access the local server for each partially replicated data whenever it is requested.

5.1.2 Drop Partially Replicated Data (Method 2)

Some of the partially replicated data may be frequently accessed by all the mobile computers in the system. In this case, caching the values of these partially replicated popular data should improve the performance of the proposed protocol in terms of cache hit ratio and transaction response time. Hence, it is suggested that mobile computers should cache some of the popular partially replicated data. However, as stated in the example in Figure 5.1, it is necessary to have a mechanism to invalid partially replicated data from mobile computers. It is suggested that mobile computers drop partially replicated data once an invalidation report is received. In this way, it is able to invalidate partially replicated data without including additional information in the invalidation reports. However, mobile computers need to download necessary partially replicated data through the wireless channels after each invalidation report is received.

5.1.3 Attaching Server-List (Method 3)

Another way to solve the problem is to attach a *server-list* to each partially replicated data. The *server-list* contains the identity numbers of all the servers which support the data. Whenever a partially replicated data is loaded in the cache of a mobile computer, the server-list of the data is also stored in the cache. Each server is required to attach its identity number to every invalidation report. When a mobile computer receives an invalidation report, it compares the identity number of the server with those in the server-list of each cached partially replicated data. The mobile computer drops a partially replicated data if the data is not supported by the local supporting server. After considering partially replicated data, the normal algorithm in Figure 3.2 is executed to invalidate fully replicated data and advance the timestamp.

When a mobile computer requests a data that is not supported by the local supporting server, the local supporting server will forward the request to a server which contains the data and possesses the latest version with version number less than the timestamp of the mobile computer (based on $V[].ctnc$). After receiving the response, the local supporting

server forwards the value of the data to the mobile computer. By using the above invalidation mechanism, a data will be dropped when a new invalidation report is received from the local supporting server that does not support the data.

When the frequency for accessing a partially replicated data that is not supported in a server is high, the server can decide to support the data. Similarly, when the frequency for accessing a partially replicated data that is supported in a server is low, the server can decide not to support the data. The servers only need to modify the server-list of the data. The server-list is updated as if a new version of the data is created after executing a write operation. Similar to an update transaction, this "write" operation updates the server-list of the copies in the write quorum. Therefore, simultaneous updates of the server-list is avoided by the replica control protocol. This new version of data together with the new server-list will then be propagated to other servers not in the write quorum by the version control mechanism presented in Appendix A. By using this approach, the successive invalidation reports will contain the modified server-list of this data. Therefore, a mobile computer which carries the data with an outdated server-list will drop the data once it receives an invalidation report from the server. It can load the data (with the updated server-list) again by requesting it from the server or when it receives a $DATA$ message which contains the data. The details of the $DATA$ message is presented in Section 3.2.5 of Chapter 3.

5.2 Experiments and Interpretation

In this section, the probability of accessing partially replicated data and the cache size are varied. If the probability of accessing partially replicated data is high and the cache size is large, it is expected that the strategies which cache partially replicated data are more favorable, i.e. methods 2 & 3. Otherwise, method 1 which does not cache partially replicated data should be more appropriate.

5.2.1 Partially Replicated Data with High Accessing Probability

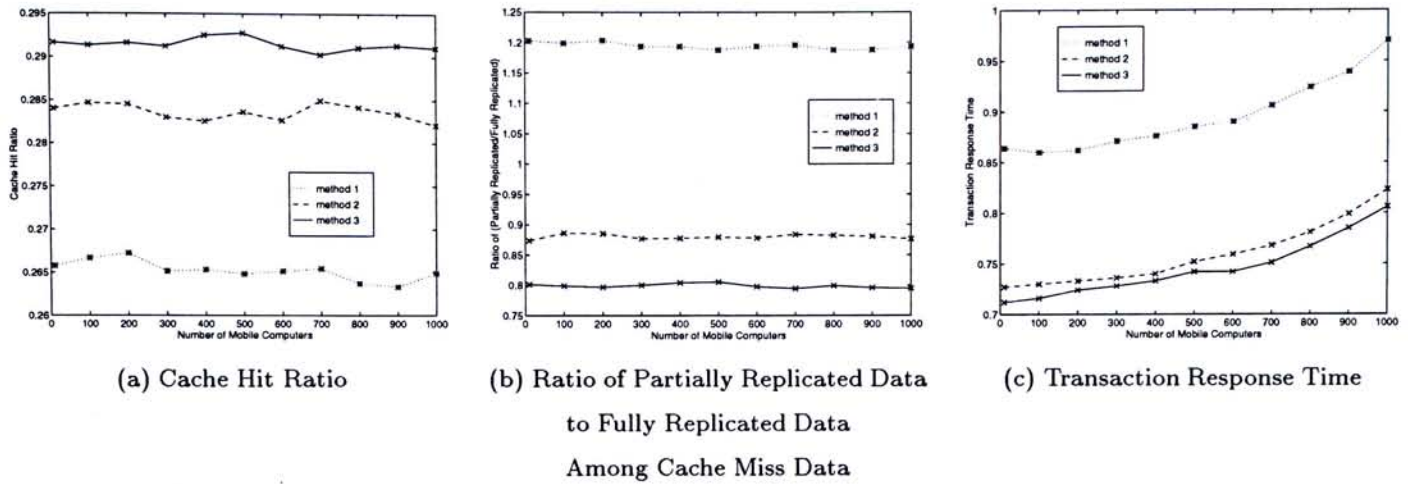
In the following experiment, there are three types of data. There are 60 partially replicated data with the accessing probability equals 40%. There are two types of fully replicated data, 60 of them being popular data, with accessing probability equals 40%; the remaining 180 data are fully replicated with accessing probability equals 20%. Each server has a probability of 40% to hold each partially replicated data. Periodically (according to a negative exponential distribution with a mean value of 300 seconds) a server will decide whether or not to support each partially replicated data. The parameters setting is shown in Table 5.1. The results are shown in Figure 5.2.

Parameters	**Values**
Probability that a server support a partially replicated data	0.4
Number of fully replicated data	60
Time delay for accessing partially replicated data from other servers	0.3 sec
Number of partially replicated data	240
The size of server-list	4 bytes
Mean time a server drop / support partially replicated data	300 seconds

Table 5.1: Additional Simulation Parameters for Partially Replicated Database System

As shown in Figure 5.2(b), most of the cache misses of method 1 are caused by partially replicated data. It is because 40% of the accesses are directed to the partially replicated data, the number of cache misses for these data will absolutely increase if they are not cached. Besides, as shown in Figure 5.2(a), the cache hit ratio for method 1 is the lowest. This shows that the total number of cache misses for method 1 is the largest, which in turn shows that excluding the partially replicated data in cache can result in the worst performance if these data are frequently accessed. Besides, the transaction response time of method 1 is the longest as shown in Figure 5.2(c) since it is necessary to access data through the wireless channels whenever there are cache misses.

It is assumed that there is a time-delay for a local supporting server to request data



(a) Cache Hit Ratio

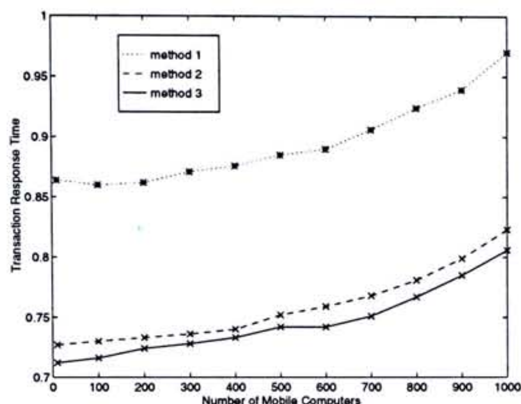
(b) Ratio of Partially Replicated Data to Fully Replicated Data Among Cache Miss Data

(c) Transaction Response Time

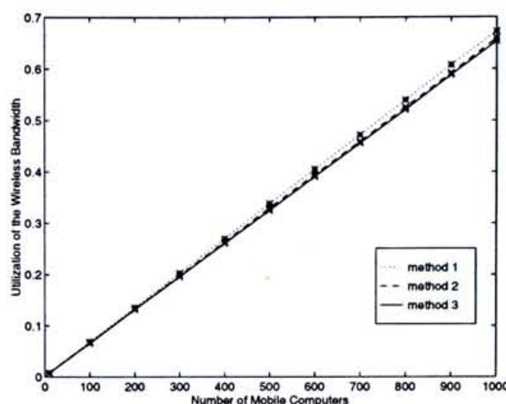
Figure 5.2: Partially Replicated Data with High Accessing Probability

from other servers if that data is not locally supported. Those mobile computers in a system that adopt method 1 suffer cache misses more frequently that those systems that adopt methods 2 and 3. As a result, these mobile computers require a longer time delay in message transmission which increases the transaction response time. Suppose that the time required for message transmission over the wired network can be neglected. The time-delay for requesting data from other servers should be reduced. If the time delay is reduced, the difference in transaction response time should also be reduced. In the following experiments, we decrease the time delay from 0.25 seconds to 0.05 seconds and finally 0.0 second. The results are shown in Figures 5.3, 5.4 and 5.5 respectively.

It is found that the transaction response time of these methods are getting closer. However, even when the time-delay for accessing partially replicated data through the wireless channel is neglected, the transaction response time of methods 2 and 3 still outperforms method 1. There is not a significant difference in the bandwidth utilization of all these methods. It is because, the amount of messages traveling over the wireless channels is not affected by the time-delay.

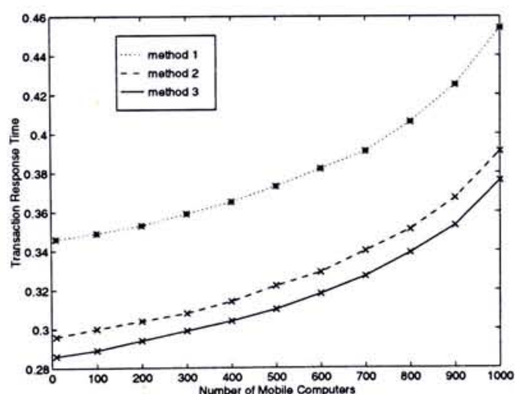


(a) Transaction Response Time

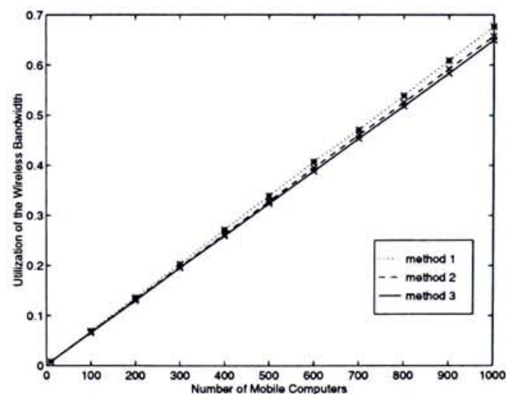


(b) Utilization of the Wireless Bandwidth

Figure 5.3: Time Delay = 0.25 seconds

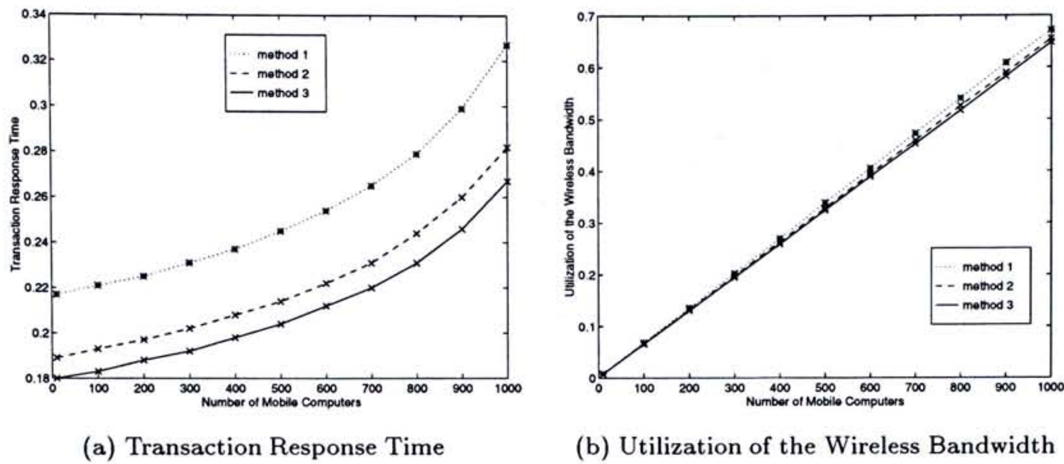


(a) Transaction Response Time



(b) Utilization of the Wireless Bandwidth

Figure 5.4: Time Delay = 0.05 seconds

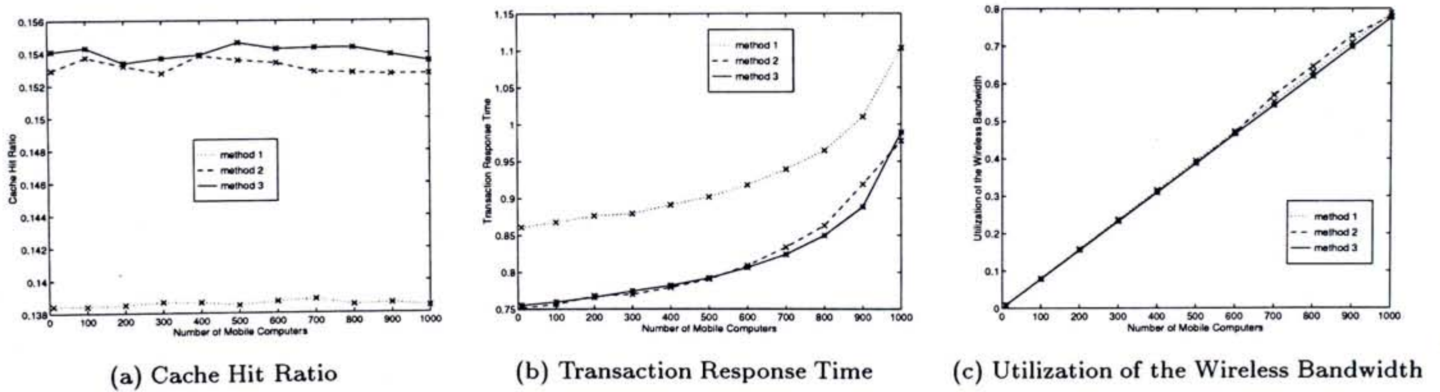


(a) Transaction Response Time (b) Utilization of the Wireless Bandwidth

Figure 5.5: Time Delay = 0.0 seconds

5.2.2 Reducing the Cache Size

If the size of cache is reduced, the cache hit ratio should decrease. Since the partially replicated data are popular data which are frequently accessed, reducing the cache size should lead to a more significant difference in cache hit ratio among the three methods. In the following experiments the cache size is now reduced from 60 to 30, the values of other parameters are the same as those in Figure 5.3. The results are shown in Figure 5.6.



(a) Cache Hit Ratio (b) Transaction Response Time (c) Utilization of the Wireless Bandwidth

Figure 5.6: Time Delay = 0.25 seconds; Cache Size = 30

It is found that the difference in cache hit ratio among the three methods is more significant as expected. The lower cache hit ratio results in a longer transaction response

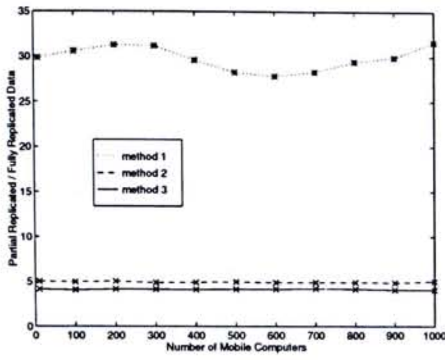
time as shown in Figure 5.6(b). Regardless of the cache size, the cache hit ratio of methods 2 and 3 is higher than that of method 1 in general. It is concluded that when the accessing probability of partially replicated data is high, it is advised to cache these data into the cache. It should be noticed that among the methods which cache partially replicated data, method 3 (which attaches server-list to each partially replicated data) outperforms method 2 (which drops partially replicated data whenever invalidation reports are received) in terms of cache hit ratio and transaction response time in all the above experiments.

5.2.3 Partially Replicated Data with Low Accessing Probability

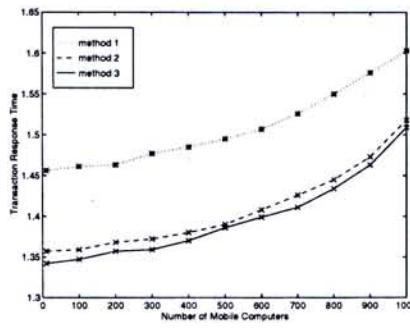
Caching data enables mobile computers to answer queries immediately with their local copies. However, caching should be used to store those data that are expected to be frequently accessed. If the accessing probability of partially replicated data is lower than that of fully replicated data, it is better not to cache partially replicated data. In the following experiments, there are only two types of data: *popular data*, which are fully replicated among servers and are frequently accessed, and *non-popular data* which are only partially replicated and are infrequently accessed.

It is assumed that there are 60 popular data out of 300 data in the database. These popular data are frequently accessed and are fully replicated among the servers. The remaining 240 data are infrequently accessed and are not fully replicated. The cache size of a mobile computer is 60 and we vary the accessing probability of popular data to be 30%, 60% and 80%. The results are shown in Figures 5.7, 5.8 and 5.9 respectively.

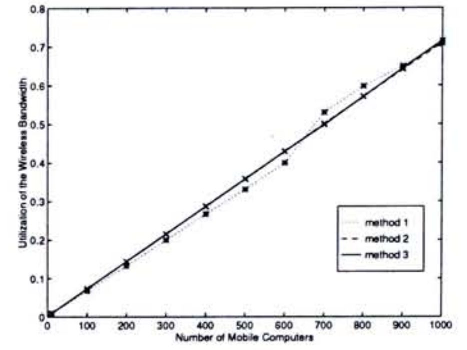
It is found that if mobile computers do not cache partially replicated data as in method 1, it would result in a higher cache hit ratio. The cache hit ratio increases as the accessing probability of popular data increases. It is because, the size of the cache memory at mobile computers is limited. Caching partially replicated data may cause the frequently accessed popular data (fully replicated data) to be swapped out in order to free the cache for the



(a) Cache Hit Ratio

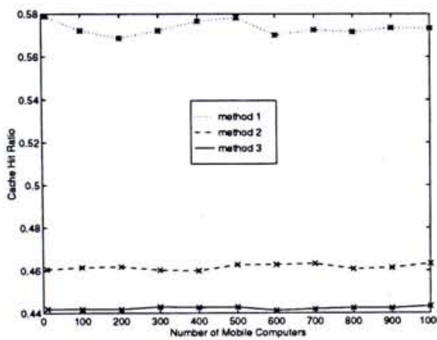


(b) Transaction Response Time

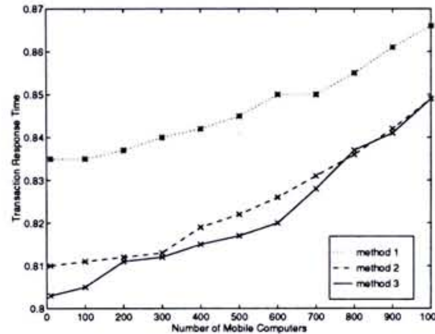


(c) Utilization of the Wireless Bandwidth

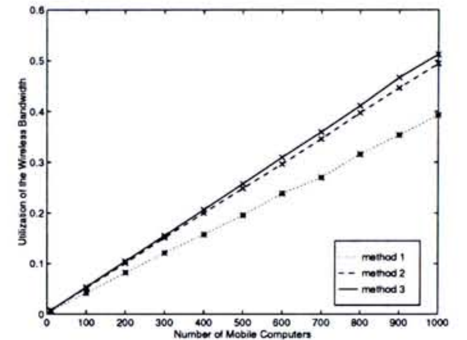
Figure 5.7: Popularity of Popular Data 30%



(a) Cache Hit Ratio

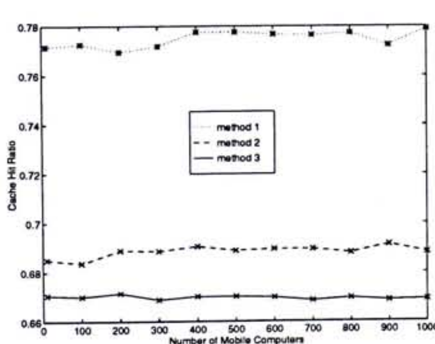


(b) Transaction Response Time

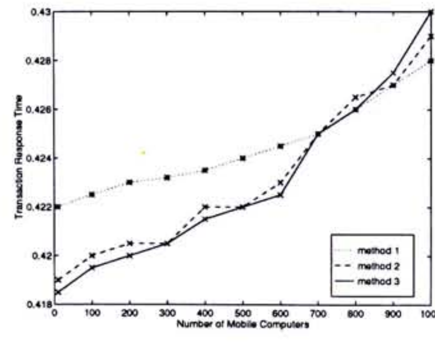


(c) Utilization of the Wireless Bandwidth

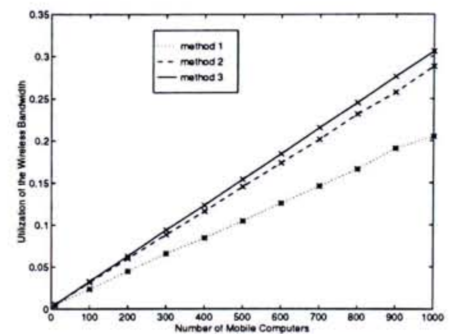
Figure 5.8: Popularity of Popular Data 60%



(a) Cache Hit Ratio



(b) Transaction Response Time



(c) Utilization of the Wireless Bandwidth

Figure 5.9: Popularity of Popular Data 80%

data in case of the cache is full. Later when the popular data are requested, the mobile computer needs to download it through the wireless channel again, thus lowers the cache hit ratio and increases the bandwidth utilization. This effect is intensified as the accessing probability of fully replicated popular data increases as shown in Figures 5.7, 5.8 and 5.9.

Figure 5.10 shows the ratio of partially replicated data to fully replicated data among those data that cause cache miss when the probability of accessing popular data is 80%. It shows that most of the data that cause cache miss in method 1 are due to partially replicated data. This can be used to explain the longer transaction response time of method 1 as shown in Figures 5.7, 5.8 and 5.9. There is a longer delay time whenever a mobile computer requests partially replicated data from other servers. Hence, the transaction response time of method 1 is the longest even though its cache hit ratio is also the highest.

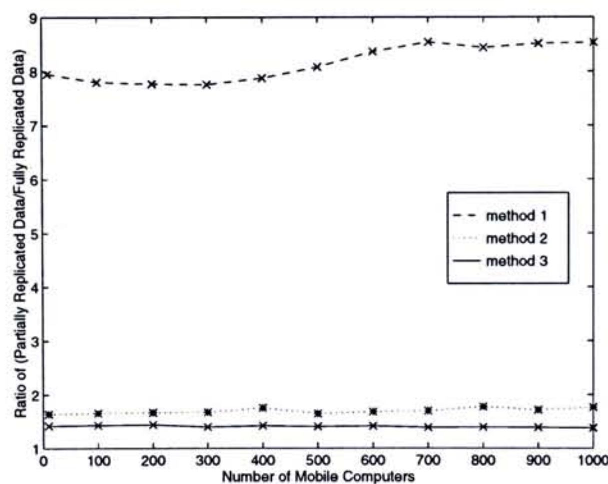


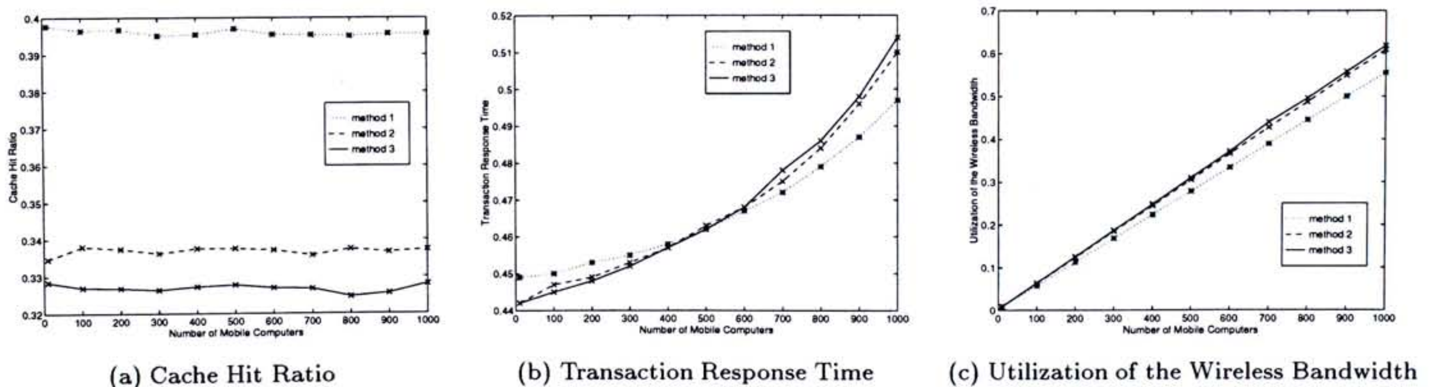
Figure 5.10: Ratio of Partially Replicated to Fully Replicated Data

Comparing the two methods which cache partially replicated data, the cache hit ratio is lower if each partially replicated data is attached with a *server-list*. Consequently, the transaction response time is also longer. It may be due to the fact that partially replicated data may remain in the cache after an invalidation report is received. These data will consume the space in cache memory for a longer time, hence, fewer popular data (fully replicated data) can be cached in the mobile computers. This effect is more significant when the probability of accessing fully replicated data increases.

A caching strategy is more efficient if each cached data has a higher chance of being accessed. If the cache size decreases, the set of data to be cached should be selected more carefully. In the following two sets of experiments, the parameters setting is the same as those in Figure 5.9 except that the cache size decreases from 60 to 30 and finally 10. We can then evaluate the efficiency of the caching strategies in different methods. The results are shown in Figures 5.11 and 5.12.

The cache hit ratio decreases as the cache size decreases. However, it is found that the difference in performance is becoming more significant as shown in Figures 5.11(a) and 5.12(a). It shows that method 1 is more efficient in caching data when compared with method 2 and 3.

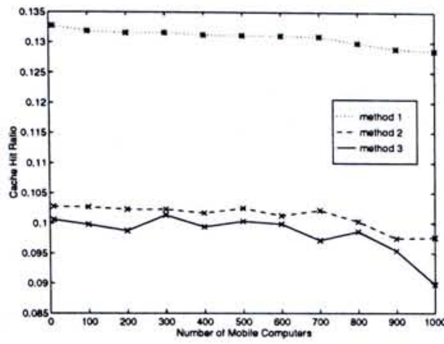
Comparing the results of Figure 5.9 with that in Figures 5.11 and 5.12 (the cache sizes are 60, 30 and 10 respectively) it is found that the difference in transaction response time and bandwidth utilization decreases. It may be due to the fact that as the cache size decreases, more queries cannot be answered by the locally cached data. Therefore, mobile computers need to access data through the wireless channels regardless of their caching strategies. More uplink queries are generated which in turn increases the transaction response time and the bandwidth utilization.

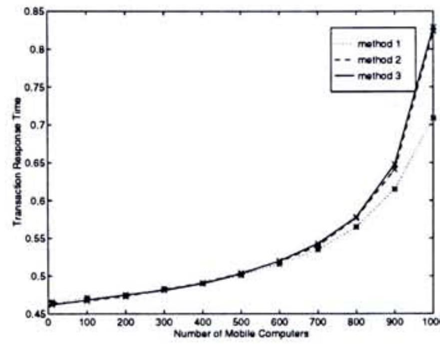


(a) Cache Hit Ratio (b) Transaction Response Time (c) Utilization of the Wireless Bandwidth

Figure 5.11: Cache Size = 30

In short, it is advised that if partially replicated data are frequently accessed by mobile computers, mobile computers in the system should store these data in their cache in order


(a) Cache Hit Ratio


(b) Transaction Response Time

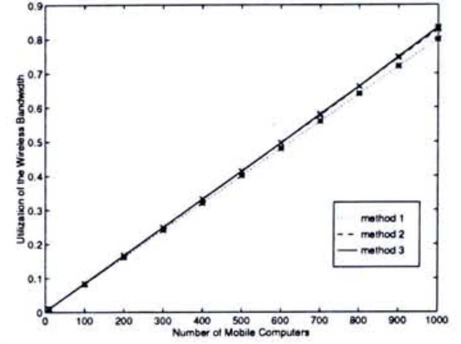

(c) Utilization of the Wireless Bandwidth

Figure 5.12: Cache Size = 10

to lower bandwidth consumption and reduce transaction response time. The simulation results show that method 3 in partially replicated database system provides a solution which ensures cache consistency and provides efficient access to data under a scenario where popular data are not fully replicated among all the servers.

Chapter 6

Conclusions and Future Work

Different protocols are proposed to maintain cache consistency in the mobile computing system [1, 4, 5, 7, 17, 25, 27, 30, 31, 32, 35]. However, the impact of the unrestricted mobility of mobile computers on cache consistency and transaction management is seldom addressed. In order to complement the previous results, the notion of cache consistency for mobile computers is defined and the issues for maintaining cache consistency are investigated. In addition, a new caching policy for mobile computers in mobile computing environments is proposed.

The new caching strategy guarantees that cached data of a mobile computer must correspond to a snapshot of a database. Hence, read-only transactions can read data from the cache without issuing any lock to the servers and at the same time serializability is maintained. Wireless bandwidth consumption is reduced by eliminating message transmission over the networks to acquire lock. Simulations are conducted for the cases with a fully replicated database system to investigate the performance of the proposed protocol under different scenarios.

Suppose the accessing probabilities of data in the database are different and are known in advance. It is found that adding piggybacking messages which include the new values of some frequently accessed data to the invalidation reports can improve the performance of the proposed protocol. The improvements include shorten transaction response time and lower wireless bandwidth utilization. The improvement is more significant when the accessing frequency of these frequently accessed data increases.

The simulations show that the proposed protocol outperforms the AT method suggested in [6] in a mobile computing environment. Our protocol considers the problems that may arise in an asynchronized system and provide a method to maintain cache consistency where cached data can still be kept in cache when a mobile computer crosses the boundaries of cells. In the contrast, it is necessary for a mobile computer to drop all cached data to ensure cache consistency whenever it crosses boundaries to different cells if the AT method is employed. Using our proposed protocol, message communications for building the cache from scratch is saved. As more queries can be answered by the contents of cache, the round trip delay time for accessing data is reduced. Furthermore, with the assumption that message transmission over the wireless network is expensive and unreliable, the proposed protocol provides an efficient mechanism to answer queries from mobile computers. Consequently, the transaction response time is reduced.

In summary, the experiments show that this new caching policy not only reduces the contention on the wireless network and shortens the transaction response time, but also maintains cache consistency.

However, if data are not fully replicated at all the servers, invalidation reports from a server will not include information of those partially replicated data that the server does not contain. It may result in inconsistency of the caches at mobile computers when they advance their timestamps solely depend on the invalidation reports received. Several amendments of the original protocol are presented to solve the problems and simulations are performed to evaluate their differences in performance. One of the proposed amendments which attaches a *server-list* to each partially replicated data is proved to be better than others when the partially replicated data is frequently accessed. The server-list contains the identity numbers of those servers that support the corresponding partially replicated data. Suppose an invalidation report is received at a mobile computer, the mobile computer compares the identity number of its local supporting server with those in the server-list of each partially replicated data. If the mobile computer finds that its local supporting server is not included in the server_list of any partially replicated data, the

mobile computer removes that data from its cache. This amendment is particularly appropriate in a scenario where the accessing probability of partially replicated data is high. However, if the accessing probability of partially replicated data is lower than other fully replicated data, it is suggested not to cache partially replicated data at mobile computers.

6.1 Future Work

The policy introduced in this paper is by no means perfect. The present work requires some continuations. Some applications may not require to adopt serializability as the correctness criterion. If the correctness criterion can be relaxed, a weaker notion of cache consistency can be used. As a result, the synchronization constraints for cache management may be relaxed. Another issue that may be worth investigating is to reduce the length of the invalidation reports. In the current approach, an access unit is also an invalidation unit. If the database is large and updates are frequent, an invalidation report may consume a large amount of the wireless bandwidth. The length of an invalidation report may be reduced if we can group several data as one invalidation unit. However, the way to group the data and the impact on the performance need further investigation. We hope that the results will benefit the design of the cache management policy for the future mobile computer environments.

Bibliography

[1] A. Acharya and B. R. Badrinath. Delivering Multicast Messages in Networks with Mobile Hosts. In *Proceedings the 13th International Conference on Distributed Computing Systems*, pages 292–299, 1993.

[2] Rakesh Agrawal. Models for Studying Concurrency Control Performance Alternatives and Implications. In *Proceedings of the 1985 SIGMOD Conference, Austin, TX*, pages 108–121, May 1985.

[3] 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.

[4] 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*, pages 388–392, 1993.

[5] B.R. Badrinath and T. Imielinski. Replication and Mobility. In *Second Workshop on the Management of Replicated Data*, pages 9–12, 1992.

[6] D. Barbara and T. Imielinski. Sleepers and Workaholics: Caching Strategies in Mobile Environment. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 1994.

[7] Daniel Barbara-Milla and Hector Garcia-Molina. Replicated Data Management in Mobile Environment: Anything New Under the Sun? In *Proceedings of the IFIP Conference on Application in Parallel and Distributed Computing*, April 1994.

[8] P. A. Bernstein and N. Goodman. Multiversion Concurrency Control – Theory and Algorithms. In *ACM Transactions on Database Systems, Vol 8, No. 4*, pages 465–483, 1983.

[9] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems.* Addison-Wesley, 1987.

[10] P.A. Bernstein, D.W. Shipman, and W.S. Wong. Formal aspects of serializability in database concurrency control. In *IEEE Trans. Softw. Eng. SE-5,3*, pages 203–215, May 1979.

[11] Michael J. Carey, Michael J. Franklin, Miron Livny, and Eugene J. Shekita. Data Caching Tradeoffs in Client-Server DBMS Architectures. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 357–366, 1991.

[12] M.A. Casanova. The Concurrency Problem of Database Systems. In *Lecture Notes in Computer Science, vol.116,Springer-Verlag, New York*, 1981.

[13] Stefano Ceri and Giuseppe Pelagatti. *Distributed Databases Principles & Systems.* McGraw-Hill, 1984.

[14] D. Duchamp. Issues in Wireless Mobile Computing. In *Proceedings. Third Workshop on Workstation Operating Systems*, pages 2–10, 1992.

[15] D. Duchamp, S.K. Feiner, and G.Q. Maguire. Software Technology for Wireless Mobile Computing. In *IEEE Network Vol. 5. No.6*, pages 12–18, 1991.

[16] Dan Duchamp and Carl D. Tait. An Efficient Variable-Consistency Replicated File Service. In *File Systems Workshop, USENIX, Ann Arbor MI*, pages 111–126, May 1992.

[17] Dan Duchamp and Carl D. Tait. An Interface to Support Lazy Replicated File Service. In *Second Workshop on the Management of Replicated Data*, pages 6–8, 1992.

[18] Maria R. Ebling, Lily B. Mummert, and David C. Steere. Overcoming the Network Bottlenect in Mobile Computing. In *IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, US*, December 1994.

[19] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I. L. Traiger. The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, 19(11):624–633, November 1976.

[20] Henry F.Korth and Abraham Silberschatz. *Database System Concepts.* McGraw-Hill, 1986.

[21] G.H. Forman and J. Zahorjan. The Challenges of Mobile Computing . In *Computer Vol.27 No.4*, pages 38–47, 1994.

[22] Ada Fu, John C.S. Lui, and M.H. Wong. Dynamic Policies in Selecting a Caching Set for a Distributed Mobile Computing Environment. In *Technical Report of The Chinese University of Hong Kong*, May 1995.

[23] D. K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, pages 150–159, December 1979.

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

[25] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *Proceedings of the 18th VLDB Conference Vancouver, British Columbia, Canada*, pages 41–52, 1992.

[26] T. Imielinski and B.R. Badrinath. Mobile Wireless Computing: Solutions and Challenges in Data Management. In *Technical Report DCS-TR-296/WINLAB-TR-49,Department of Computer Science, Rutgers University*, 1993.

[27] Tomasz Imielinski and B. R. Badrinath. Data Management for Mobile Computing. In *Sigmod Record Vol. 22. No.1*, pages 34–39, March 1993.

[28] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[29] Abhay K. Parekh. Selecting Routers in Ad-hoc Wireless Networkds.

[30] Evaggelia Pitoura and Bharat Bhargava. Revising Transaction Concepts for Mobile Computing.

[31] Evaggelia Pitoura and Bharat Bhargava. Building Information Systems for Mobile Environments. In *The 3rd International Conference on Information and Knowledge Management*, December 1994.

[32] Evaggelia Pitoura and Bharat Bhargava. Maintaining Consistency of Data in Mobile Distributed Environment. In *Technical Report TR-94-025, Purdue University, Department of Comp. Sciences*, 1994.

[33] M. Satyanarayanan. Mobile Computing. In *Computer Vol. 26. No.9*, pages 81–82, 1993.

[34] O.T. Satyanarayanan and D. Agrawal. Efficient Execution of Read-Only Transactions in Replicated Multiversion Databases. In *IEEE Transactions on Knowledge and Data Engineering, Vol 5, No 5*, pages 849–871, 1993.

[35] C.D. Tait and D. Dunchamp. Service Interface and Replica Management Algorithm for Mobile File System Clients. In *Proceedings of the First International Conference on Parallel and Distributed Information System*, 1991.

Appendix A

Version Control Mechanism for Servers

The version control mechanism in [34] is adopted in the servers. Readers who are familiar with this protocol may skip this appendix. By using this version control mechanism, each update transaction, T, is assigned a timestamp $tn(T)$, which is unique and corresponds to the serialization order of the transaction. Therefore, if a locking concurrency control protocol is used, a timestamp is assigned at the end of a transaction when the serialization order is known. In addition, all data updated by transaction T are stamped with the version number $tn(T)$, i.e., the timestamp of the transaction. A counter called the *visible transaction counter*, *vtnc*, is kept at each server. The visible transaction counter for a server is in fact the lower bound of the timestamp for the next transaction committed in the server. The counter is implemented in this way. When a transaction T executed in a server commits, *vtnc* of the server will be incremented to $tn(T)$, if no transaction T', at the local server, will commit with a timestamp $tn(T')$ such that $tn(T') < tn(T)$. All transactions that start on the server after T has committed will be serialized after T and obtain a timestamp greater than $tn(T)$, i.e., greater than *vtnc*. Therefore, no new version can be created with version number less than or equal to *vtnc* at the server. Note that the *vtnc* on different servers may be different; hence, a transaction executed on a remote server may commit with a timestamp less than the *vtnc* of a local server.

By using a quorum protocol, a write operation may not update all the copies of a data. Therefore, a server may only contain a partial version sequence of each data. In order to

obtain a complete version sequence of each data, a background propagation mechanism is employed to propagate the updates to every server. A server's local sequences are said to be complete, up to the value of a given transaction counter, if it contains all versions of the data with version number less than the counter globally. The extent of completeness of a server's version sequences is represented by a *completeness transaction number counter, ctnc* maintained at each server.

Each server, S_i also maintains two vectors, $V_i[j].vtnc$ and $V_i[j].ctnc$. $V_i[j].vtnc$ indicates S_i's knowledge of the value of *vtnc* at server S_j; i.e., S_i knows that versions with version numbers less than or equal to $V_i[j].vtnc$ cannot be created at S_j. Similarly, $V_i[j].ctnc$ represents S_i's knowledge of the value of *ctnc* at server S_j. In addition, each server S_i, maintains a log, L_i, which contains the update history of the data maintained in the system. The log is composed of a sequence of records $\langle id, vn, val \rangle$, where *id* identifies the data, *vn* identifies the version number of the data and *val* is the value associated with the version of the data. Periodically, each server sends propagation messages to all other servers. A propagation message includes a copy of the server's local vectors $V_i[].vtnc$, $V_i[].ctnc$ and a portion of its *log* which contains updates that may not be known by the receiver (records with *vn* greater than $V_i[j].ctnc$). Upon receiving the propagation message from server S_i, server S_j merges the incoming log with its local log. Then vectors $V_j[i].vtnc$ and $V_j[i].ctnc$ will be updated by taking the pairwise maximum of the local and the received vectors. It has been shown, in [34], that no version of a data can be created with a version number less than $MIN_i(V_j[i].vtnc)$ globally. In addition, the local version sequences at server S_j are completed up to $MIN_i(V_j[i].vtnc)$. Hence, server S_j can roll forward its *ctnc* to $MIN_i(V_j[i].vtnc)$. Readers are encouraged to refer to [34] for the detailed description of the version control mechanism.

CUHK Libraries

003510952