Efficient and Adaptive Schemes for Consistent Information Sharing in Wireless Mobile and Peer-to-Peer Networks

by

ZHIJUN WANG

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2005

Copyright © by Zhijun Wang 2005

All Rights Reserved

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisors Dr. Sajal K Das and Dr. Mohan Kumar, who gave me constant motivation and excellent guidance during my Ph.D. study. I would like to thank my committee members Dr. Hao Che, Dr. Bob Weems and Dr. Gergely Zaruba for their useful discussions and valuable comments on my dissertation. Let me say 'thank you' to my colleague, Huaping Shen, for his valuable discussions and help on my research work. I also thank the Texas Advanced Research Program (TARP 14-771032) grant, the Department of Computer Science and Engineering at the University of Texas at Arlington and Nokia Inc. for the financial support during my Ph.D program. Finally, I thank my family: my parents, Jingyun Wang and Yunying Shen, my wife, Qing Xia for their unconditional support and encouragement to pursue my doctoral degree.

August 8, 2005

ABSTRACT

Efficient and Adaptive Schemes for Consistent Information Sharing in Wireless Mobile and Peer-to-Peer Networks

Publication No. ______

Zhijun Wang, Ph.D.

The University of Texas at Arlington, 2005

Supervising Professors: Sajal K. Das and Mohan Kumar

With the tremendous growth of applications in wireless mobile and Peer-to-Peer (P2P) networks, significant research efforts have been made to improve the quality of sevice. Caching and replicating frequently used data objects or files in user's local buffers are popular mechanisms to effectively reduce the communication bandwidth requirement and thus improve the overall system performance. However, the frequent disconnections of users make data consistency a difficult task in wireless mobile and P2P networks.

In this dissertation, we design and analyze a *Scalable Asynchronous Cache Consistency Scheme* (SACCS) for single cell wireless cellular networks. SACCS is a highly scalable, efficient, and low complexity scheme and works well in error-prone wireless mobile environments. Analytical results indicate that SACCS provides very good cache consistency in error-prone wireless environments. Comprehensive simulation results show that SACCS offers more than 50% performance gain than that of existing Timestamp (TS) and Asynchronous Stateful (AS) schemes; We also propose *Dynamic SACCS* (DSACCS) for

multi-cell mobile environments. To the best of our knowledge, DSACCS is the first cache consistency scheme that optimizes cache performance in multi-cell mobile environments.

In P2P networks, some files are heavily replicated to enhance their availability and reduce the search cost. With the dramatic growth in P2P applications dealing with dynamic files, file updates and the file consistency maintenance become critical. To effectively propagate update information to the replica peers, we propose an efficient algorithm, called *Update Propagation Through Replica Chain* (UPTReC), in decentralized and unstructured P2P networks to provide weak file consistency. To provide strong file consistency, we develop another algorithm, called *file Consistency Maintenance through Virtual servers* (CMV). In CMV, each dynamic file has a virtual server and any file update must be accepted through the virtual server to maintain one copy serilizability of the file. To the best of our knowledge, CMV is the first strong file consistency algorithm for decentralized and unstructured P2P networks. Our simulation results show that UPTReC algorithm outperforms other existing algorithms, and CMV is an efficient file consistency algorithm with very low overhead messages.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

ABSTRACT iv

LIST OF FIGURES x

LIST OF TABLES xii

Chapter

1. INTRODUCTION 1

1.1 Cache Consistency in Wireless Mobile Networks 1

1.2 File Consistency in Peer-to-Peer Networks 4

1.3 Research Motivation and Contributions 6

1.4 Organization of the Dissertation 9

2. RESEARCH BACKGROUND 11

2.1 Cache Consistency Maintenance in Wireless Mobile Networks 11

2.1.1 System Architecture 11

2.1.2 Invalidation Methods 12

2.1.3 Stateless Schemes 13

2.1.4 Stateful Approaches 15

2.2 File Consistency in Distributed Databases and P2P Networks 16

2.2.1 Distributed Databases 16

2.2.2 P2P Networks 17

3. SCALABLE ASYNCHRONOUS CACHE CONSISTENCY SCHEME 21

3.1 Proposed Scheme SACCS 21

3.1.1 Data Structures and Message Formats 22

3.1.2 MUC Management . . . 22
3.1.3 Algorithm Description . . . 24
3.1.4 Illustration of MUC Management . . . 27
3.1.5 MUC Consistency Maintenance . . . 29
3.1.6 Efficiency and Cooperation . . . 30
3.1.7 Mobility . . . 31
3.1.8 Failure Handling . . . 32
3.2 Stale Cache Hit Probability . . . 33
3.2.1 Analytical Modeling . . . 33
3.2.2 Stale Cache Hit Probability for Rayleigh Fading Channels . . . 37
3.3 Performance Evaluation By Simulation . . . 39
3.3.1 Effect of the number of MUs . . . 42
3.3.2 Effect of Database Size . . . 44
3.3.3 Effect of Zipf Coefficient . . . 44
3.3.4 Effect of cache size . . . 47
3.4 Summary . . . 48
4. DYNAMIC SCALABLE ASYNCHRONOUS CACHE CONSISTENCY SCHEME 49
4.1 Intra-Roaming and Inter-Roaming . . . 49
4.2 Three Cache Consistency Strategies for Multi-Cell Cellular Networks . . 51
4.2.1 Homogeneous IR Strategy . . . 51
4.2.2 Inhomogeneous IR without Roaming Check Strategy . . . 52
4.2.3 Inhomogeneous IR with Roaming Check Strategy . . . 53
4.3 Performance Evaluation of Three Strategies . . . 54
4.3.1 Impact of Data Object Update Frequency . . . 58
4.3.2 Impact of the Number of Cells . . . 60
4.4 Dynamic Scalable Asynchronous Cache Consistency Scheme . . . 62

4.4.1 Two Consistency Maintenance Cost Functions 62
4.4.2 Description of DSACCS 65
4.4.3 Performance Evaluation 69
4.5 Summary 74
5. UPDATE PROPAGATION THROUGH REPLICA CHAIN (UPTReC) 76
5.1 Description of UPTReC 77
5.1.1 System Model and Assumptions 77
5.1.2 Push Update Through Replica Chain 79
5.1.3 Pull after Online 80
5.1.4 Chain Construction and Maintenance 81
5.2 Performance Analysis 83
5.2.1 Analytical Results 83
5.2.2 Numerical Results 87
5.3 Performance Comparisons 90
5.3.1 Overhead Messages for Each Push Process 91
5.3.2 Overhead Messages Per Query 94
5.4 Summary 95
6. FILE CONSISTENCY MAINTENANCE THROUGH VIRTUAL SERVERS . 97
6.1 Proposed CMV Algorithm 97
6.1.1 Logical Structure of RPs 97
6.1.2 Algorithm Description 99
6.1.3 File Consistency Maintenance 103
6.1.4 Failure Handling 104
6.1.5 Virtual Server Construction and Maintenance 105
6.2 Performance Modeling 106
6.2.1 Parameters and Notations 107

6.2.2 Performance Analysis 108
6.3 Numerical Results 114
6.3.1 Impact of File Update Rate 115
6.3.2 Impact of Online Probability of VRPs 117
6.3.3 Impact of Probability of VRP Changing its IP Address 119
6.3.4 Impact of the Number of RPs 120
6.4 Summary 122
7. CONCLUSIONS AND FUTURE WORK 123
7.1 Future Work 125
REFERENCES 127
BIOGRAPHICAL STATEMENT 138

LIST OF FIGURES

Figure		Page
1.1	Wireless Mobile Cache System Architecture	2
1.2	A Ubiquitous Information Access System	7
3.1	State Diagram of Cache Entry i	23
3.2	BSMain Procedure	25
3.3	MUMain Procedure	26
3.4	MUC Management Scheme	27
3.5	Data Object Update Process	34
3.6	Upper Bound on Stale Cache Hit Probability vs IR Miss Probability	35
3.7	Simulation of Stale Cache Hit Probability vs IR Miss Probability	36
3.8	Two State Markov Chain Describing the *Good* and *Bad* Channel Model	38
3.9	Average Access Delay vs Number of Data Objects	45
3.10	Uplink Per Query vs Number of Data Objects	45
3.11	Average Access Delay vs Zipf Coefficient	46
3.12	Uplink Per Query vs Zipf Coefficient	46
3.13	Average Access Delay vs MU Cache Size	47
3.14	Uplink Per Query vs MU Cache Size	48
4.1	Wireless Cellular Network Architecture	50
4.2	Three Configurations: (1) Cells 1 and 2, (2) Cells 1-7, (3) all cells	56
4.3	Average Access Delay (D) vs Average Data Update Interval	58
4.4	Average Uplink Per Query vs Average Data Update Interval	59
4.5	Average Access Delay (D) vs Number of Cells (B)	61

4.6 Average Uplink Per Query vs Number of Cells (B) 61
4.7 A multi-cell system 68
4.8 Average Access Delay vs Number of MUs per Cell 70
4.9 Average Uplink Per Query vs Number of MUs per Cell 71
4.10 Average Access Delay vs Number of Data Objects in the System 73
4.11 Average Uplink Per Query vs Number of Data Objects in the System . . 73
5.1 (a) Logical Replica Chain; (b) Update Propagation of RP_i 79
5.2 Replica Trees: (a) Root at RP_1; (b) New Tree with Root at RP_3 82
5.3 Replica Chain Constructing. Join at (a) Head or Tail; (b) Middle 83
5.4 Diagram of Calculating $P_h^s(m)$ 86
5.5 Probability of Successful Update Propagation vs Number of Probe Peer . 88
5.6 Probability of Successful Update Propagation vs Number of Hops 89
5.7 Number of Overhead Messages in Push vs Peer Online Probability 93
5.8 Stale Query Ratio vs Peer Online Probability 93
5.9 Number of Overhead Messages Per Query vs Update Period 95
5.10 Stale Query Ratio vs Average Update Period 96
6.1 Logical Structure of RPs in the CMV Algorithm 98
6.2 Procedure for VRP 100
6.3 Procedure for HRP 101
6.4 Procedure for LRP 102
6.5 Optimal Number of VRPs vs VRP Online Probability 118
6.6 Overheads and File Retrieval Per Query vs VRP Online Probability . . . 119
6.7 Optimal Number of VRPs vs IP Address Change rate 120
6.8 Overheads and File Retrieval Per Query vs IP Address Change Rate . . 120
6.9 Optimal Number of VRPs (N_v) vs Number of HRPs and LRPs 121
6.10 Overheads and File Retrieval Per Query vs Number of HRPs and LRPs . 121

LIST OF TABLES

Table Page

3.1 Communication Messages in SACCS 23

3.2 State Transition and *Bad* State Probabilities for Values of V and F . . . 37

3.3 Performance of SACCS at Different MU Speed 39

3.4 Parameter Definition 40

3.5 Ten Types of Data Objects in Database 42

3.6 average Access Delay D (sec) for varing number of MUs 43

3.7 The Uplink Per query (UPQ) for varying number of MUs 43

4.1 Features of Three Cache Consistency Maintenance Strategies 54

4.2 Ten Types of Data Objects in Database 57

5.1 Parameter Setup I 92

5.2 Parameter Setup II 94

6.1 Optimal N_v vs α 116

6.2 Messages per query (MPQ) vs α 116

CHAPTER 1

INTRODUCTION

Wireless mobile and peer-to-peer (P2P) networks are playing an increasingly important role in people's daily life to provide ubiquitous information access including integrated data, voice, and video services. With the significant growth of applications in such networks, the quality of service (QoS) issues become critical. Caching and replicating frequently used data objects or files at the users' local buffers are effective methods to improve the communication quality, and hence overall system performance. To guarantee consistent information access, efficient and scalable data consistency schemes must also be designed. However, limited network resources and frequent user disconnections make data consistency maintenance a challenging task. In this Chapter, we first introduce the research background of data consistency maintenance in wireless mobile and P2P networks, and then present the research motivation and contributions of this dissertation.

1.1 Cache Consistency in Wireless Mobile Networks

Wireless mobile communication has increasingly become an important means for accessing various kinds of dynamically changing data objects such as news, stock prices, and traffic information *anytime anywhere*. However, existing wireless mobile communication systems are primarily designed for supporting voice and are not efficient in handling the different characteristics and QoS requirements of data communications. Wireless mobile networks have two main scarce resources: communication bandwidth and battery power [23] [48] [51]. Moreover, they have to also deal with the user mobility and disconnections.

Thus, data communication in such networks is much more challenging than that in wired networks.

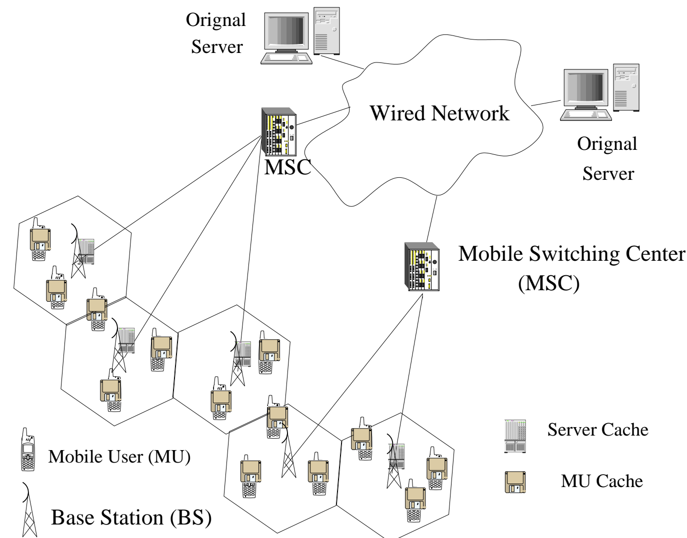


Figure 1.1 Wireless Mobile Cache System Architecture.

Figure 1.1 shows a wireless mobile communication system architecture. It includes a wired network, multiple original servers, mobile switching centers (MSCs), base stations (BSs) and mobile users (MUs). Each (hexagonal) cell has a BS that is connected to a MSC which is further connected to the original servers through wired networks. Each BS serves multiple MUs through wireless channels and has a server cache (SC) to store data objects. Each MU has also a local cache (MUC) storing some frequently used data objects.

Caching frequently accessed data objects at the local buffer of a mobile user (MU) is an efficient way to reduce data access delay, save bandwidth and improve the overall system performance. With caching, the average data access delay is reduced since some data requests can be satisfied from the local buffer, thereby obviating the need for data transmission over the scarce wireless links. But, limited communication bandwidth as

well as frequent disconnections and roaming of an MU make *cache consistency* a difficult task in wireless mobile computing environments. A successful scheme must be bandwidth efficient and must also effectively handle an MU's disconnection as well as mobility. Broadcast has the advantage of being able to serve an arbitrary number of MUs with minimum bandwidth consumption. Thus, the broadcast and cache management schemes must be carefully designed for an efficient mobile data transmission system to maximize *bandwidth utilization* and also to minimize *average data access delay*. Additionally, such schemes must be *scalable* to support large database systems as well as a large number of MUs.

To guarantee consistent information sharing in wireless mobile networks, the cache consistency must be strictly maintained. In the past decade, significant research efforts have been made in the development of cache consistency schemes for wireless mobile networks. In general, these existing schemes can be classified into two categorizes: stateless and stateful. In the *stateless* schemes [4] [8] [10] [28] [30] [39] [57] [64] [68], the server is unaware of the MUC's content. The server periodically broadcasts data objects' *invalidation reports* (IRs) to all the MUs. An IR of a data object includes the data object's identification (ID) and the latest update time. Even though stateless approaches employ simple database management schemes, their scalability and ability to support disconnection are poor. On the other hand, in *stateful* schemes [31] [11], the server maintains a cache state for each MU. Whenever a data object is updated, its IR is multicast to the MUs that have a copy of the data object in their caches. The stateful approaches can support disconnections and are scalable in terms of the number of data objects in the server. However, such schemes incur significant overhead in the server for maintaining cache state of each MU.

All the above existing caching schemes assume reliable communication between the BS and the MUs for IR broadcast. However, any reliable communication mechanism

requires acknowledgement of IRs from the MUs. After an IR is broadcast (or multicast for stateful schemes), the increased competition for uplink channel between the BS and MUs will have an impact on the uplink queries, and further on the average access delay and MU's battery consumption. On the other hand, if an MU is disconnected during the IR broadcast, the server cannot get its acknowledgement back, and must retransmit the IR because it does not know if the IR is lost or the MU is disconnected. Also the possible inconsistency and performance loss due to wireless channel errors are not studied in the existing schemes. Therefore, there is a need for scalable and efficient schemes for maintaining cache consistency in error-prone wireless channels. Moreover, existing cache schemes focus on the single cell mobile environments, and the impact of the MUs' mobility is ignored. A cache consistency scheme designed for single cell mobile environments may not be efficient for multi-cell environments. Hence it is important to develop cache consistency schemes that achieve the optimized cache performance in multi-cell mobile environments. The first part of the dissertation addresses the cache consistency issues in error-prone, single cell as well as multi-cell wireless mobile networks.

1.2 File Consistency in Peer-to-Peer Networks

Peer-to-Peer (P2P) networks are self-organizing distributed systems, in which all participating peers cooperatively provide and receive services from each other. P2P systems are rapidly growing due to such desirable characteristics as scalability, availability, anonymity and authentication. Today, P2P network traffic may account for up to 85% of the Internet bandwidth usage [27].

P2P systems can be classified as centralized and decentralized. Napster [26], is an example of a centralized P2P system. The decentralized P2P systems can be further categorized into structured and unstructured. In a *structured* P2P system, the topology is tightly controlled and the files are well deployed [13] [41] [50] [52] [56]. In P2P systems,

files are used instead of data object, because peers have much larger buffer space than those of mobile devices and a file may include multiple data objects. On the other hand, an *unstructured* P2P system has no control of its topology and file placement [2] [14] [17] [19] [24] [35] [40] [43] [46]. Chord [56] is an example of a decentralized and structured P2P system whereas Gnutella [43] is an example of a decentralized and unstructured P2P system. The structured P2P systems enjoy efficient file search mechanisms through distributed hash tables. However, they can not support key word search and incur high overheads to maintain very transient peers. In contrast, the unstructured P2P systems support key word searches. They also incur low overheads to maintain extremely transient peers, and are robust to transients and general search queries [14]. Based on these observations, we focus on decentralized and unstructured P2P systems in this dissertation.

In decentralized and unstructured P2P systems, each peer maintains information about its neighboring peers. Some files may be heavily replicated in the system to improve the file availability and system fault-tolerance. To access a file, a peer searches it through its neighboring peers. In order to make such P2P systems scalable and efficient, significant efforts have been made on the development of search and replication algorithms [14] [19][24][40]. In these algorithms, the locations of replicas for a file are well deployed based on partial knowledge of the system, thus minimizing the search cost and balancing the network load. Furthermore, these algorithms assume that files are rather static and updates occur very infrequently. Indeed, the impact of the file update has not received much attention.

With the tremendous growth in P2P applications, the issues related to file consistency maintenance become even more critical. For example, in such applications as shared calendar, P2P web cache [29], online game [34] and online auction [42] systems, the files are updated frequently and hence the consistency needs to be strictly main-

tained. Let us briefly sketch a scenario to justify the need for strict file consistency in P2P systems. In the buyer-seller market using P2P systems, it is very convenient to make a trading strategy if a trading file that lists all possible prices and quantities from the sellers and buyers, is maintained for each type of goods. A potential buyer/seller maintains a copy of the trading file and observes the trading progress. The offered price may be changed by modifying the trading file. After the trading file is updated by a buyer/seller, all others must view this update to track down the newest information to formulate their trading strategies. Thus, a strict file consistency maintenance scheme is essential for such P2P systems.

There exist very few file consistency maintenance algorithms [21] [36] for decentralized and unstructured P2P networks. Moreover, they can only provide weak file consistency, are not efficient and lack effective solutions for RPs with dynamic IP addresses. A peer is called a replica peer (RP) of a file if it has a replica of the file. In the second part of the dissertation, we design an efficient strong file consistency maintenance algorithm with effective solutions for RPs with dynamic IP addresses.

1.3 Research Motivation and Contributions

This dissertation aims at building ubiquitous, consistent information access systems for wireless mobile and P2P networks. Figure 1.2 shows a ubiquitous information access system in which users can access information through wireless and P2P networks. Caching and replicating frequently used data objects and files are effective methods to improve system performance and enhance information availability. To guarantee consistent information access, cache and replica consistency must be strictly maintained. We focus on how to provide consistent information access through mobile cache and replica peers.

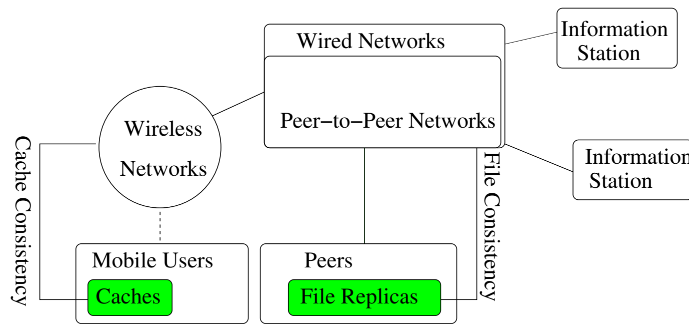


Figure 1.2 A Ubiquitous Information Access System.

Cache consistency maintenance in wireless mobile networks is a challenging task due to MUs' disconnection and mobility. Significant research efforts have been made on the development of cache consistency maintenance schemes in the past decade. As explained above, there is a need to develop cache consistency schemes that work efficiently in error-prone communication channels and achieve optimized cache performance in multi-cell mobile environments. This motivates our research work on the development of cache consistency maintenance schemes for wireless cellular networks.

Initially P2P networks were designed for facilitating download of static multimedia files that are not updated frequently. With the tremendous growth in P2P applications, file consistency maintenance becomes a critical issue. For example, in online auction systems, files are updated frequently and hence the file consistency needs to be maintained to guarantee consistent file sharing. In decentralized and unstructured P2P systems, it is necessary to develop efficient and scalable schemes for providing strong file consistency.

This dissertation make four major contributions.

- **S**calable **A**synchronous **C**ache **C**onsistency **S**cheme (SACCS) by designing for error-prone wireless mobile environments [58] [59].
- **D**ynamic SACCS (DSACCS) to achieve efficient cache consistency maintenance for multi-cell environments[60] [61].

- **U**pdate **P**ropagation **T**hrough **Re**plica **C**hain (UPTReC) algorithm to effectively propagate updates in decentralized and unstructured P2P networks [62].
- A strong file consistency maintenance algorithm, called file **C**onsistency **M**aintenance through **V**irtual servers (CMV), for decentralized and unstructured P2P networks [63].

Theses algorithms are briefly outlined as follows.

To improve mobile cache performance, we develop a novel algorithm, called *Scalable Asynchronous Cache Consistency Scheme* (SACCS) [58] [59], which maintains cache consistency between the BS cache, called the *server cache* (SC), and the MU caches (MUCs). SACCS is a highly scalable, efficient, and low complexity algorithm, and provides weak cache consistency with a small stale cache hit ratio under unreliable IR broadcast environments. The properties are accomplished through the use of flag bits at SC and MUC, an identifier (ID) in MUC for each entry after its invalidation and estimated time-to-live (TTL) for each cached entry, as well as rendering of all valid entries of MUC to uncertain state when an MU wakes up.

To address mobility of MUs in multi-cell environments, we design and evaluate a *Dynamic Scalable Asynchronous Cache Consistency Scheme* (DSACCS) in which the IR of a data object is broadcast globally or locally depending on which has the minimum consistency maintenance cost. A globally maintained data object implies that its IR is broadcast to every cell in the system. To the best of our knowledge, DSACCS is the first cache consistency algorithm focusing on the development of optimized cache performance in multi-cell mobile environments. An improvisation of DSACCS, called DSACCS-G, is also proposed which groups cells in order to facilitate effective cache consistency maintenance in multi-cell systems.

To effectively maintain file consistency in decentralized and unstructured P2P networks, we propose a novel algorithm, called *Update Propagation Through Replica Chain*

(UPTReC). UPTReC provides a probabilistically guaranteed file consistency. It is an efficient scheme for replica peers (RPs) with dynamic IP addresses and has low file consistency maintenance cost.

To provide strong file consistency in decentralized and unstructured P2P networks, we propose an algorithm called *file Consistency Maintenance through Virtual servers* (CMV) in which each dynamic file has a virtual server (VS) that maintains file consistency in the system. Any update to the file can only be accepted through the VS to ensure strong file consistency.

1.4 Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 presents a background of the data consistency maintenance in wireless mobile and P2P networks. The following two chapters cover the mobile cache consistency maintenance schemes. In Chapter 3, a Scalable Asynchronous Cache Consistency Scheme (SACCS) is proposed for error-prone wireless mobile networks. SACCS provides a weak data consistency with small stale cache hit probability. The upper bound of stale cache hit probability is analytically derived. Comprehensive simulations are also done to compare the performance of SACCS with existing schemes.

Chapter 4 first introduces three types of cache consistency IRs for multi-cell systems. Then we derive two consistency maintenance cost functions. Based on the two cost functions, we design and evaluate a Dynamic Scalable Asynchronous Cache Consistency Scheme (DSACCS) for multi-cell systems. Extensive simulations are done under various system conditions. An improvisation of DSACCS, called DSACCS-G, is proposed for grouping cells in order to facilitate effective cache consistency maintenance.

Chapters 5 and 6 include file consistency maintenance algorithms in decentralized and unstructured P2P networks. Chapter 5 presents the design,analysis and performance

of analgorithm called Update Propagation Through Replica Chain (UPTReC). In Chapter 6, a strong file Consistency Maintenance through Virtual servers (CMV) algorithm is proposed along with analytical derivation optimal parameter selections. The conclusion and future work are drawn in Chapter 7.

CHAPTER 2

RESEARCH BACKGROUND

Caching and replicating frequently used data objects in users' local buffers are effective means to improve the system performance in wireless mobile and peer-to-peer (P2P) networks. To guarantee consistent information access, the data consistency has to be strictly maintained. In the past decade, significant research efforts have been made on the development of efficient and scalable data consistency maintenance schemes. In this chapter, we present an overview of these research works.

2.1 Cache Consistency Maintenance in Wireless Mobile Networks

Although cache techniques are used to improve system performance in wireless mobile networks, the limited communication bandwidth as well as the mobility and disconnection of mobile users (MUs) make cache consistency maintenance a difficult task. This section presents the research background of mobile cache consistency maintenance.

2.1.1 System Architecture

As shown in Figure 1.1, a wireless mobile communication system includes a wired network, multiple original servers, mobile switching centers (MSCs), base stations (BSs) and mobile users (MUs). Each BS serves multiple MUs through wireless channels and has a server cache (SC) to store data objects. Each MU has also a local cache. When an MU requests a data object, it first searches the local cache. If the requested data object is valid in the cache, the request is answered immediately, thus reducing the data access time and saving the communication bandwidth and device energy. But the cache

consistency must be maintained to ensure consistent data access. The cache consistency maintenance includes two parts: one part between SC and original servers, and the other between SC and MUCs. The focus in this dissertation is on the second part. We assume the data consistency in the first part is maintained by existing cache consistency schemes in wired networks [12][38] [67].

2.1.2 Invalidation Methods

The content of data objects in MUCs can be changed by time and/or by MU's location. For example, news and stock prices change with time. A data object that describes the location of the nearest hotel to an MU, changes with the MU's location. Such data objects are called location-dependent. In this dissertation, we consider dynamic data objects that change with time but are location independent. The cache consistency maintenance schemes for location-dependent data objects including Bit Vector with Compression (BVC), Grouped Bit Vector with Compression (BVC) and Implicit Scope Information (ISI) [37] [66] [70].

The cache consistency in MUCs is maintained through invalidation report (IR) that includes the data object's ID and the latest update time. When an MU gets an IR of a data object, it invalidates the cached object. The different cache consistency schemes have different invalidation methods. Existing cache consistency maintenance schemes can be broadly categorized into stateless and stateful. *Stateless* schemes [4] [8] [28] [30] [39] [57] [64] [68] periodically broadcast IRs to MUs and employ simple database management in the BS, but their scalability and ability to support disconnection are poor. On the other hand, *stateful* schemes maintain a cache state for each MUC, and the IR is sent out asynchronously [31]. The stateful schemes are scalable in terms of the number of data objects, but incur significant overhead due to server database management. An overview of mobile cache consistency schemes is described in the following sections.

2.1.3 Stateless Schemes

A great deal of research efforts have been made on the development of stateless cache consistency schemes in the past years. In [4], three stateless schemes have been proposed for wireless mobile networks. These are Timestamps (TS), Amnesic Terminals (AT) and Signature (SIG) schemes, in which the server (i.e., BS) broadcasts IR messages to MUs every L seconds. An IR message includes the IDs and the update time of all data objects updated during the past kL seconds, where k is a positive integer. An MU can access the cached data objects only after it receives the next IR. If an MU has been disconnected longer than kL seconds, all cached data objects must be dropped. The advantage of these algorithms is that a BS does not maintain any state information about its MU's caches (MUCs), thus allowing simple management of the SC. However, they suffer from the following drawbacks:

- They do not scale well to large databases and/or fast updating data systems, due to increased number of IR messages;
- The average access latency is always longer than half of the broadcast period, simply because all requests can be answered only after the next IR;
- When the *sleep time* (during which an MU is disconnected from its BS) is longer than kL, all cache entries are deleted, thus leading to unnecessary bandwidth consumption particularly if the data objects are still valid.

In order to handle long sleep-wakeup patterns, several algorithms have been proposed. For example, in the bit-sequence (BS) algorithm [30], a hierarchical structure of binary bit sequences with an associated set of timestamps to represent users with different disconnection times can contain information about half of recently updated data objects (or all updated data objects if they are less than half of the total data objects in the system). Hence all cache entries are deleted only when half or more of the cache

entries have been invalidated. However, the model requires the broadcast of a larger number of IR messages than TS and AT schemes and cannot save the valid cache entry after very long disconnection. A validation check scheme is proposed in [64] to deal with long sleep-wakeup patterns. In this scheme, an MU sends back to the server the IDs of all cached data objects as well as their time stamps after a long disconnection. The server identifies the updated data objects and sends an individual IR to the MU. However, the scheme requires more uplink bandwidth and cannot handle arbitrary long sleep-wakeup patterns due to the fact that all the cached entries must be dropped after a disconnection longer that a time threshold. In order to reduce the IR messages, adaptive methods are developed in [28] to broadcast different IRs based on update frequency, MU access and sleep-wakeup patterns. In [68], an absolute validity interval (AVI) is employed for each data object, but it fails to reduce the access delay introduced by periodic broadcast cycles. In [8], an updated invalidation report (UIR) is introduced to reduce the access delay due to the wait for the next IR broadcast. An UIR comprises the IDs of the data objects that have been updated after the last IR has been broadcast. A fixed number of UIRs are broadcast during the interval between two IRs. Each MU can access the cached data object after each UIR receipt, thus reducing the waiting time for cache access. However, the inserted UIRs cost scarce communication bandwidth.

In the preceding approaches, all MUs can benefit from broadcast only when they retrieve the same data objects from the BS in the same broadcast cycle. If the MUs retrieve the same data objects in separate broadcast cycles, they cannot share the broadcast data objects. This makes the broadcast inefficient and sensitive to the number of MUs in the cell. The TS strategy is modified in [8] by keeping the invalidated data objects in an MUC such that the MU can update a data object if it is received from the broadcast channel. This approach increases the broadcast channel utilization. However,

keeping invalid data objects in an MUC wastes precious cache memory. A comprehensive performance evaluation of the existing stateless algorithms is studied in [57].

2.1.4 Stateful Approaches

Very few stateful cache consistency maintenance algorithms have been proposed for wireless mobile environments. In [31], an Asynchronous Stateful (AS) algorithm is proposed to maintain cache consistency in which a BS records all retrieved data objects for each MU. When an MU first retrieves a data object after it wakes up, it needs to send a wakeup message to the BS. The BS sends an IR to that particular MU based on the MUC content record and sleep-wakeup time. Whenever a BS receives an update from the original server for each recorded data object, it immediately multicasts that object's IR to MUs that have a copy of the data object in their caches. The advantage of the AS scheme is that the BS avoids unnecessary IR broadcast to MUs. Moreover, MUs can deal with any sleep-wakeup pattern without losing valid data objects. However, in order to maintain a state of each MUC, the BS must record all cached data objects for each MU. Hence an MU can only download data objects which it requested through the uplink. This makes the broadcast channel utilization inefficient and sensitive to the number of MUs. More recently, a counter based scheme is used in [9] to identify the hot data and save unnecessary IR traffic. Whenever an MUC content is changed, the MU must piggyback the change to the server, thus consuming battery power and uplink bandwidth.

More recently, two hybrid cache invalidation schemes [3] have been proposed, namely Hybrid cache invalidation with Simple Broadcasting (HSB) and Hybrid cache invalidation with Attribute Bit Sequence Broadcast (HABSB). In these schemes, the server maintains a state for each MUC while periodically broadcasting IR. The maintained state is used for saving valid data objects after arbitrary long disconnection.

However, the database management in the server is complicated as in stateful schemes while the IR traffic is not minimized.

All above existing stateless and stateful caching consistency schemes are based on reliable communication between the BS and the MUs. They do not discuss how to handle possible inconsistency and performance loss due to wireless channel errors. Further more, the existing cache schemes only consider single cell mobile environments instead of multi-cell mobile environments. A cache consistency scheme designed for single cell mobile environments may not be optimized for multi-cell environments. Hence it is necessary to develop cache consistency schemes to achieve optimized cache performance in error-prone multi-cell mobile environments.

2.2 File Consistency in Distributed Databases and P2P Networks

Consistency maintenance for replicated files has been well studied in distributed databases, but only little attention has been paid to decentralized and unstructured P2P systems. In this section, we provide a brief review of the existing work in this domain.

2.2.1 Distributed Databases

Maintaining file consistency of replicas is the most crucial function in distributed databases. There are two types of replication algorithms: *eager* and *lazy* [25]. In eager algorithms, all replicas are exactly synchronized. They provide strong consistency, but are neither efficient nor practical. In lazy algorithms, on the other hand, the updates are asynchronously propagated to all replicas. Although lazy algorithms are efficient, they may cause inconsistency among replicas. The lazy replications can be categorized into lazy master [33][44] and lazy group communication [22] models. The lazy master model provides one-copy serializability for all replicas, whereas lazy group model allows any replica to update the file. One-copy serializability defined as the concurrent execution of

updates on the replicated files, is equivalent to a serial execution on a non-replicated file. When a file is updated, the update is sent to all other replicas through a multicast tree or epidemic scheme [22]. The above algorithms assume high availability of replica nodes.

Compared to the distributed databases, the P2P systems have to contend with the following issues:

1. Highly transient replica peers (RPs): the probability of an RP to be online may be as low as 0.1 [55].
2. Dynamic file acquisitions: some RPs may drop the replica and some other peers may replicate the file.
3. Highly dynamic IP addresses of RPs: the IP address of an RP may be changed at each reconnection.
4. Dynamic network topology: the network topology is changed from time to time.
5. Large number of replicas.

Due to the transient nature of the RPs, algorithms that assume master copy maintained by a single peer do not work for P2P systems. Algorithms based on group communications are also inefficient to provide strict file consistency in such highly unreliable systems. In other words, the existing consistency algorithms designed for distributed databases can not be directly applied to P2P systems.

2.2.2 P2P Networks

P2P applications are major consumers of the Internet infrastructure. The data management issues such as file consistency maintenance are becoming more and more important. An overview of the existing file consistency maintenance schemes of P2P networks is presented in this section.

As mentioned eariler, the decentralized P2P networks can be classified into structured and unstructured. The files in structured P2P networks are well deployed through distributed hash tables (DHTs). When a peer requests for a file, it first needs to get the hash key value and then search the file through the DHTs. It takes $O(log(n))(n$ is the number of peers in the system) hops to find the location of the file. They are very scalable and efficient for file search. However, they do not support keyword search and inefficient for transient peers. In unstructured P2P networks, peers search files through their neighbor peers. Recently, some file search schemes such as random walk and expanded ring are developed in [18] for effective file search. Although unstructured P2P networks do not scale well, they support key word search and play a more important role in some applications.

2.2.2.1 File Consistency in Structured P2P Networks

In the structured P2P networks, if a file is replicated, a single node stores the locations of all replicas and hence the file consistency maintenance becomes simple. However, it only works well for a small number of replicas.

To effectively maintain file consistency, an incentive-based algorithm called CPU is proposed in [53]. In CPU, the metadata of the lookup results is kept and updated. However, CPU only caches the metadata instead of the file itself, and the consistency among the replicas are not maintained. More recently, a file consistency algorithm, called Scalable COnsistency maintenance in structured PEer-to-peer systems (SCOPE) is proposed in [15]. In this scheme, a replica-partition tree (RPT) for each key is built to keep track of the locations of replicas and then propagate update notifications. However, the write-write protection is not considered.

2.2.2.2 File Consistency in Unstructured P2P Networks

Some research efforts have been devoted to the design of file consistency maintenance algorithms in unstructured P2P networks. Based on epidemic theory, a hybrid push/pull update propagation algorithm is proposed [21] for highly unreliable and unstructured P2P systems. The algorithm provides probabilistic guarantees rather than strict consistency. In this algorithm, each RP maintains a subset of all RPs as its responsible peers. When an RP initiates an update, the updated file is pushed to its responsible peers, which in turn propagate the updated file to their responsible peers with some probability. This process continues until all possible online RPs receive the update. When an RP gets reconnected, it queries multiple responsible peers to synchronize itself with the peer having the most updated file. The proposed algorithm is the first attempt to focus on the effective propagation of *updates* to RPs in decentralized and unstructured P2P systems. However, the overhead messages due to push updates are significant. Moreover, the maintenance of the subset of responsible peers is not easy, especially for RPs with dynamic IP addresses. If the subset of responsible peers is not well maintained, the file consistency is not guaranteed.

An invalidation report based on push and pull (PAP) algorithm is developed in [36]. In PAP, each file has a master peer and only the master peer can update the file. An estimated time-to-expire (TTE) and the master peer information are associated with each replica. When a file is updated, its IR is broadcast to the network. Any online peers that have replicas of the file invalidate the replicas. Once the TTE of a file expires, the file must be pulled from the master peer if it is accessed. Only the master peer updating the file is a strong constraint in P2P systems. Moreover, the master peer may change its IP address and go offline, thus resulting in a small probability of an RP successfully pulling a master peer.

The above algorithms can only provide weak file consistency. They are not efficient and cannot effectively handle RPs with dynamic IP addresses.

In this chapter, we briefly introduced the existing cache consistency schemes in wireless mobile networks and file consistency algorithms in P2P networks. We will present our solutions in the next four chapters.

CHAPTER 3

SCALABLE ASYNCHRONOUS CACHE CONSISTENCY SCHEME

As mentioned in the last chapter, existing stateless cache consistency schemes for wireless mobile networks [4] [8] [28] [30] [39] [57] [64] [68] are not scalable, whereas the stateful schemes [31] [9] entail complex data management tasks in the server database. Moreover, the existing schemes are based on error-free wireless channels. As wireless mobile networks are error-prone, existing cache schemes are often inadequate for such networks. In this chapter, a Scalable Asynchronous Cache Consistency Scheme (SACCS) is proposed for error-prone wireless mobile environments. SACCS is a hybrid of stateful and stateless schemes, and inherits the positive features of both stateful and stateless schemes. In the rest of the chapter, we first give the algorithm details of SACCS, then present the analytical results of stale cache hit probability and show the performance comparisons of SACCS with existing schemes.

3.1 Proposed Scheme SACCS

The proposed scheme, SACCS, provides a weak consistency of MUCs in error-prone wireless mobile environments. SACCS is a highly scalable, efficient, and low complexity scheme. It provides cache consistency for dynamic public data objects such as news, traffic information, live sport scores, etc. All such data objects are shared by MUs without security considerations. The IRs of these data objects are broadcast to all MUs without acknowledgement. If a reliable communication is used between the BS and an MU retrieving the data object, the MU needs to send acknowledgement to the BS. All

other awake MUs can download the data object without acknowledgement if they can correctly receive it.

3.1.1 Data Structures and Message Formats

Let us look at the data structure of cache entries in the MUC and SC. For each data object d_i with i as the unique ID, the data structures for SC and MUC are defined as follows.

In the SC:

- (d_i, t_i, l_i, f_i): where t_i is the last update time for the data object; l_i is the estimated TTL and f_i is the flag bit such that $f_i = 1$ indicates that the next IR will be broadcast.

In the MUC:

- (d_i, ts_i, ll_i, s_i): where ts_i is the time stamp denoting the last updated time for the cached data object d_i; ll_i is an associated TTL; and s_i is a two-bit flag identifying four data entry states: 0, 1, 2, and 3, indicating valid d_i, uncertain d_i, uncertain d_i with a waiting query, and ID-only, respectively.

The communication messages are as defined in Table 3.1. Each cache entry has three states: *valid*, *uncertain* and *ID-only*. Figure 3.1 shows how the entry i changes from one state to another.

3.1.2 MUC Management

Since we mainly focus on the cache consistency maintenance in this dissertation, we use the Least Recently Used (LRU) based replacement algorithm for the management of MUC. The impact of the cache replacement algorithms on SACCS is a subject of future study. In the adopted LRU based replacement scheme, a newly cached data object or

Table 3.1 Communication Messages in SACCS

Name	Sender	Receiver	Comments
$Update(i, d_i^{'}, t_i^{'})$	original servers	BS	d_i has been updated to $d_i^{'}$ at time $t_i^{'}$
$Vdata(i, d_i, l_i, t_i)$	BS	MUs	broadcast valid data object d_i with update time at t_i and TTL $= l_i$
$IR(i)$	BS	MUs	cached d_i is invalid
$Confirmation(i, l_i, t_i)$	BS	MUs	d_i is valid if $ts_i = t_i$ and TTL $= l_i$
$Query(i)$	MUs	BS	query for data object d_i
$Uncertain(i, ts_i)$	MUs	BS	querying if d_i in uncertain state with update time ts_i, is valid or not

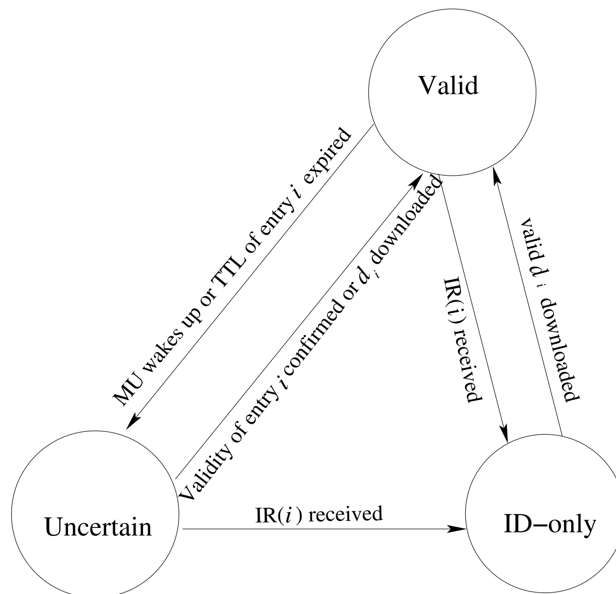


Figure 3.1 State Diagram of Cache Entry i.

one that receives a hit, is moved to the head of the cache list. When an object needs to be cached while the cache is full, data entries with $s_i \neq 2$ from the tail are deleted to make enough space for accommodating this new data object (the object with $s_i = 2$ must be kept because some requests are waiting for its confirmation). Any refreshed data objects from uncertain or ID-only state are placed in their original location and again, if necessary, enough data entries from the tail are removed.

We limit the number of ID-only entries that can be used at any given instant, to a certain value. This is to minimize frequent refreshment of old ID-only entries, which are likely to be replaced before they are requested. One may set this number close to the average number of data objects that can be cached. For example, if a cache can hold C objects, then the number of ID-only entries is limited to C. Because the MU has limited cache size, during a broadcast it caches only those objects whose ID-entries are already in the cache. The memory overhead of ID-only entries is insignificant since the size of a data object's ID is usually much smaller than the object itself.

3.1.3 Algorithm Description

We present two procedures for SACCS, namely BSMain() and MUMain(), as shown in Figures 3.2 and 3.3, respectively. The BS continuously executes the BSMain() procedure to handle MUs' query and data object update. Each MU continuously executes the MUMain() procedure to handle its query and broadcast messages. These procedures are described below.

BSMain(): When a BS receives a *Query* message, it broadcasts the queried data object to all MUs. When a BS receives an *Uncertain* message, it checks if the uncertain data object is valid or not by comparing the time stamps. If the uncertain data object is valid, a confirmation message is broadcast; otherwise, a valid data object is broadcast. Whenever a BS receives an *Update* message from the original server, it updates its cache entry in the database and also estimates a new TTL for the entry based on the last TTL and this update interval. If the corresponding flag bit is set for the entry, an IR is immediately broadcast and the flag bit is reset.

MUMain(): When an MU has a request for a data object, it first searches its local cache. If the requested data object is valid, the MU immediately answers the request using the cached object. If the cached data object is in the uncertain state, an *Uncertain*

```
BSMain() {
  For BS receiving a message
    IF (The message is Query(i))
         fetch data entry i from the database
         broadcast Vdata(i, d_i, l_i, t_i) to all MUs
         IF (f_i == 0)
             set f_i = 1
    IF (The message is Uncertain(i, ts_i))
         fetch data entry i from the database
         IF (t_i == ts_i)
             broadcast Confirmation(i, l_i, t_i) to all MUs
         ELSE
             broadcast Vdata(i, d_i, l_i, t_i) to all MUs
         IF (f_i == 0)
             set f_i = 1
    IF (The message is Update(i, d'_i, t'_i) from the original server)
         update the database entry i, as d_i = d'_i and t_i = t'_i
         update the TTL l_i
         IF (f_i == 1)
             broadcast IR(i) to all MUs and reset f_i = 0
    }
```

Figure 3.2 BSMain Procedure.

message is sent to the BS. If the cache entry is in the ID-only state or the data object is not in the local cache, a *Query* message is sent to BS to retrieve the data object. When the MU receives a *Vdata* message, if it has a query waiting for the data object, the MU answers the query and caches the data object. If the MU has an uncertain entry in the cache, the data object is downloaded. If the MU has an ID-only entry, the data object is also downloaded. Upon receiving an IR message, the MU sets the entry into ID-only state. When a *Confirmation* message is received, if the MU has a corresponding uncertain entry in the cache, it refreshes the entry if the time stamp on the *Confirmation* message is later than that on the MU. When the TTL of an entry expires, it is set to the uncertain state.

```
MUMain() {
  For MU receiving a message
    IF (The message is a Request for d_i)
        IF (d_i is valid in the cache)
          answer the request with cached data object d_i
          move the entry into the head of the cache list
        ELSE IF (d_i is in uncertain state )
          send Uncertain(i, ts_i) message to the BS
          add ID, i.e., i, to query waiting list
          set s_i = 2 and move the entry into the head of cache list
        ELSE IF (i is ID-only entry in the cache)
          send Query(i) message to the BS
          remove the entry i in the cache
          add i to query waiting list
        ELSE
          send Query(i) message to the BS
          add i to query waiting list
    IF (The message is a Vdata(i, d_i, l_i, t_i))
        IF (i is in query waiting list)
          answer the request with d_i
          remove the uncertain entry i if it exists in the cache
          add the valid entry i at the cache list head
        ELSE
          IF (i is ID-only entry in the cache)
            download d_i to the original entry location in the cache
          ELSE IF (i is an uncertain entry in the cache)
            IF (ts_i < t_i)
              download d_i to the original entry location in the cache
              set ts_i = t_i, ll_i = l_i and s_i = 0
            ELSE
              set s_i = 0
    IF (The message is an IR(i))
        IF (entry i is valid or uncertain in the cache)
          delete d_i and set s_i = 3
    IF (The message is a Confirmation(i, l_i, t_i))
        IF (i is an uncertain entry in the cache)
          IF (ts_i == t_i)
              set s_i = 0 and ll_i = l_i
              IF (i is in query waiting list)
                answer the request with d_i
          ELSE
              delete d_i and set s_i = 3
    IF (MU wakes up from the sleep state)
        set all valid (s_i = 0) entries into uncertain state (s_i = 1)
    IF (TTL expires for entry i)
        set the valid entry i into uncertain state (s_i = 1)
    }
```

Figure 3.3 MUMain Procedure.

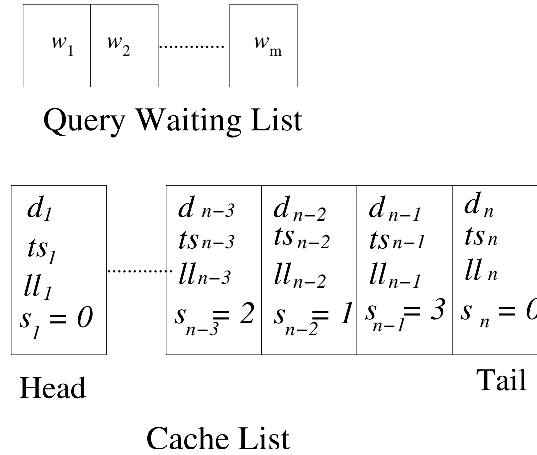


Figure 3.4 MUC Management Scheme.

3.1.4 Illustration of MUC Management

Let us illustrate the MUC management with the help of a simple example as shown in Figure 3.4. Assume there are n entries (1, 2, ..., n) in the cache list and m queries (w_1, w_2, ..., w_m) in the query waiting list. In particular, we illustrate various actions of cache management at an MU such as the ones upon receipt of a request, a valid data object, a confirmation message and an invalidation message.

(**1**) **Request** for d_i:

There are four cases for the scheme as shown in Figure 3.4:

- Case 1: $s_i = 0$; Suppose $i = n$. The MU responds with d_i and moves the entry i to the head of cache list; in other words, (d_i, ts_i, ll_i, s_i) is inserted to the head of the cache list.
- Case 2: $s_i = 1$; Suppose $i = n - 2$. The MU sends an $Uncertain(i, ts_i)$ message to the BS, sets $s_i = 2$, moves the entry i to the head of the cache list and adds i to the query waiting list.
- Case 3: $s_i = 3$; Suppose $i = n - 1$. The MU deletes the entry i from the cache list, sends $Query(i)$ message to the BS and adds i to the query waiting list.

- Case 4: entry i is not in the cache list. The MU sends $Query(i)$ message to the BS and adds i to the waiting list.

(**2**) **Vdata**(i, d_i, l_i, t_i):

There are three cases:

- Case 1: i is on the waiting list, the MU answers the request by downloading the data object d_i and adds the entry i at the head of the cache list. If the MUC has an uncertain entry i (e.g., $i = n - 3$ in Figure 3.4), the entry is deleted.
- Case 2: i is not in the query waiting list but in the cache list, e.g., $i = n - 1$ in Figure 3.4, the MU downloads d_i at the location of entry $n-1$ and sets $ts_{n-1} = t_i$, $ll_{n-1} = l_i$ and $s_{n-1} = 0$. Assuming $i = n - 2$, the MU sets $s_{n-2} = 0$ when $ts_{n-2} = t_i$, or downloads d_i instead of d_{n-2} and sets $s_{n-2} = 0$, $ts_{n-2} = t_i$ and $ll_{n-2} = l_i$ when $ts_{n-2} < t_i$.
- Case 3: i is not in the query waiting list and cache list, the MU does nothing on i.

Each time before the MU adds or downloads a data object, if the free buffer space in the cache is not enough, the entries n, $n - 1$, $n - 2$ from the tail are deleted such that $s_i \neq 2$. (The entry i with $s_i = 2$ must be kept because an uncertain message for the entry has been sent to the BS, and if it is still valid, it will be used for answering the query).

(**3**) **IR**(i):

There are two cases:

- Case 1: $s_i < 3$, the MU deletes d_i and sets $s_i = 3$. If the total number of ID-only entries is over a maximum limit, the last ID-only entry is deleted from the cache.

- Case 2: i is the ID-only entry or no entry i is in the cache list, the MU does nothing on it.

(4) **Confirmation**(i, l_i, t_i):

There are three cases:

- Case 1: $i = n - 3$; If $ts_{n-3} = t_i$, the MU answers the request with the cached data object d_{n-3}, then sets $s_i = 0$ and move the entry $n - 3$ to the head of the cache list. However, if $ts_{n-3} < t_i$, the request is still in the query waiting list; so delete d_{n-3} from the cache.
- Case 2: If $i = n - 2$, the MU checks the timestamp ts_{n-2}; if $ts_{n-2} = t_i$, the MU sets $s_{n-2} = 0$ and $ll_{n-2} = l_x$, else the MU deletes d_{n-2} from the entry and set $s_{n-2} = 3$.
- Case 3: Otherwise, the MU does nothing on it.

3.1.5 MUC Consistency Maintenance

In this section, we discuss the mechanism for consistency maintenance between the SC and the MUCs. We first assume an error free channel, where the MUs which are awake (i.e., the MU is not sleeping and connected to the BS) receive all IRs. The IR loss situation discussed below.

For each cached data object, SACCS uses a single flag bit, f_i, in SC in order to maintain the consistency between the SC and MUC. When d_i is retrieved by an MU, f_i is set indicating that a valid copy of d_i may be available in an MUC. If and when the BS receives an updated d_i, it broadcasts an $IR(i)$ and resets f_i. This action implies there are no valid copies of d_i in any MUC. Furthermore, while $f_i = 0$, subsequent updates do not entail broadcast of $IR(i)$. The flag bit f_i is set again when the BS services a request for retrieval (including request and confirmation) for d_i by an MU.

In mobile environments, an MUC belongs to one of two states: awake or sleep. If an MU is *awake* at the time of $IR(i)$ broadcast, the copy of d_i is invalidated and an ID-only entry is maintained by the MU. The data objects of an MU in the *sleep* state are unaffected until it wakes up. When an MU wakes up, it sets all cached valid data objects (including d_i) into the uncertain state. Consequently, MUs and their cached objects are unaffected if $IR(i)$ broadcast occurs during their sleep times.

It is probable for an MU to get a stale cache hit in the case of IR loss. A TTL is associated with each cache entry. When the TTL of a cache entry expires, an MU automatically sets it into the uncertain state, thus reducing the stale cache hit ratio. We give a detailed analysis on stale hit probability in Section 3.3.

3.1.6 Efficiency and Cooperation

As mentioned earlier, a good cache consistency maintenance scheme must be scalable and efficient in terms of the database size and the number of MUs. We claim that SACCS can handle large and fast updating data systems because the BS has some knowledge about the MUCs. Only data entries which have flag bits set, result in the broadcast of IRs when data objects are updated. Consequently, the IR broadcast frequency is the *minimum* of the uplink query/confirmation frequency and the data object update frequency. In this way, the broadcast channel bandwidth consumption for IRs is minimized. Besides IR traffic, all other traffic in SACCS is also minimized due to the strong cooperation among the MUCs. This is specifically due to the introduction of the uncertain state and the ID-only state for the MUCs. The retrieval of a data object, d_i, from the BS issued by any given MU brings the entries of i in the uncertain or ID-only state in all the awake MUCs to a valid state. Moreover, a single uplink confirmation for entry i makes all entries of i in the uncertain state in all the awake MUCs to be in the valid or ID-only state. The addition of the uncertain state also allows an MUC to keep

all the valid data objects when it wakes up after an arbitrary sleep time. In contrast, for asynchronous stateful (AS) and timestamp (TS) schemes, all the invalidated data objects are completely deleted from the MUC. This allows little cooperation among the MUs, resulting in a dramatic increase of traffic volume between the BS and the MUs as the number of MUs increases. Although the scalability of the TS scheme can be improved by retaining the invalid data objects [8], the cache efficiency is reduced by having to keep in the MUC the invalid data objects, rather than IDs as is the case in the SACCS scheme.

In contrast with the AS scheme which requires $O(MN)$ (M the number of MUs and N the number of data objects in the system) buffer space in the BS to keep all states of MUCs, SACCS requires only *one bit* per data object in the SC. The BS performs IR broadcast for an updated data object if the corresponding bit is set. Thus, the database management overhead is minimal requiring only a single bit check and set/reset.

The TTL expiration of a valid cache entry is checked only when its data object is accessed or available on the channel, otherwise we do not care about the entry status. When the data object of a valid entry is accessed or available on the channel, its TTL is first checked. If the TTL has not expired, the entry is treated as valid. Otherwise, it is handled as an uncertain entry. The cost of each TTL expiration check is the execution of an addition (namely, $ts_i + ll_i$) and a comparison (e.g., comparing $ts_i + ll_i$ with the current time). The cost of these two operations is not much as compared with the cost of a data object query and/or data object download. Thus, the cost of execution on the TTL expiration is not significant.

3.1.7 Mobility

One advantage of the AS scheme is the efficient handling of mobility. SACCS can also handle MUs' mobility effectively. When an MU roams, it is either in the awake or in sleep state. If an MU is in the sleep state, there is no extra action in SACCS as well

as in AS. In SACCS, an MU sets all its valid cache entries to the uncertain state after it wakes up. Hence, upon wake up, the mobility and location of an MU is irrelevant. In AS, when a roamed MU wakes up, its first query will be sent to the new BS which can retrieve its cache state from the previous BS which maintains the cache consistency.

If a roaming MU is awake, SACCS treats it as if it just woke up from the sleep state, i.e., all valid data entries are set to an uncertain state. The consistency is guaranteed with this approach and all valid data objects are retained. Also SACCS is a simple scheme in the sense that it is transparent to the BSs involved. But for AS, there exist two situations:

1. An MU is sending a message to the BS when the MU roams. In this situation, the BS can handle the MU's handoff and transfer the MU's cache state to the new BS. So there is no extra action for cache consistency maintenance.
2. An MU is not sending a message to the BS when it roams. A wakeup event is forced in this case. When the MU's next query comes, the MU sends to the BS a message, which includes the queried data object and cache state request. The cache state request contains the information of the previous BS and roaming time. After the BS gets the cache state request, it retrieves the MU's cache state from the previous BS and sends the data object IRs (which are updated after the MU roamed) to the MU. Thus the cache consistency is maintained.

3.1.8 Failure Handling

Handling of MU failures is the same as handling of MU disconnections. If an MU recovers from a failure, it sets all cached valid data entries into an uncertain state. SACCS treats this situation as a wakeup from the sleep state. Furthermore, SACCS handles server failures in a very simple way as described below. When a BS server is back after a failure, it simply broadcasts a server-down message to all MUs which in turn

set all valid data entries into the uncertain state. The MUs in the sleep state miss the server-down message, but after they wake up, all valid entries are automatically set to the uncertain state. Thus the cache consistency is maintained even if some cached data objects are updated during the BS server failure. This is because the validity of each cached data object is checked and refreshed before usage. After a server failure, all data object in an MU are retained, but set to uncertain state. Thus avoiding download of unchanged data objects.

3.2 Stale Cache Hit Probability

As mentioned above, SACCS provides a weak cache consistency. The stale cache hit ratio is an important metric to evaluate the performance of SACCS. A stale cache hit is counted when a cache access of a data object is the latest one. In this section, the stale cache hit probability as a function of IR loss probability is analyzed and simulated in Rayleigh fading wireless channels 3.8.

3.2.1 Analytical Modeling

The analytical model derives an upper bound of the *stale cache hit probability* in SACCS under various channel conditions. The following assumptions are made in the model:

1. The update process for data object d_i follows a Poisson distribution with average update rate, μ_i.
2. The TTL, l_i, of d_i equals the average update interval time T_{ui}. That is $l_i = T_{ui} = 1/\mu_i$.
3. The IR miss ratio of an awake MU is P_{mIR}.

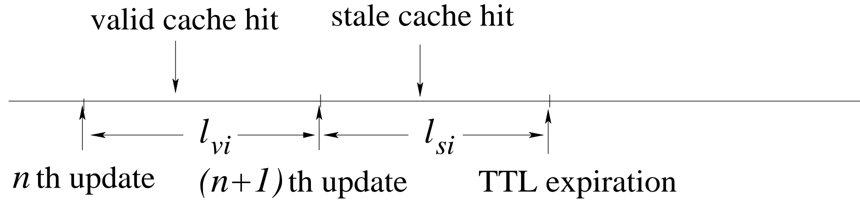


Figure 3.5 Data Object Update Process.

Let P_{si} be the stale cache hit probability for data object d_i. Figure 3.5 illustrates an object update process. After the nth update of d_i, the $(n+1)$th update may occur either before or after the TTL expiration. When an MU misses the $(n+1)$th IR, there is no stale hit if the update is after the expiration of the TTL, because the data entry is automatically set to an uncertain state. If the update is before the TTL expiration and the MU accesses the cached data object d_i between the $(n+1)$th update time and the TTL expiration time, it gets stale cache hits. Let P_{short_i} be the probability of the next update occurrence which is before the TTL expiration; l_{vi} be the average of all update intervals which are earlier than the TTL expiration; and l_{si} be the interval between the $(n+1)$th update and estimated TTL expiration. These terms can be calculated as:

$$P_{short_i} = \int_0^{l_i} \mu_i e^{-\mu_i t} dt = 1 - \frac{1}{e} \tag{3.1}$$

$$l_{vi} = \frac{\int_0^{l_i} \mu_i t e^{-\mu_i t} dt}{P_{short_i}} = (\frac{e-2}{e-1}) l_i \tag{3.2}$$

$$l_{si} = l_i - l_{vi} = (\frac{1}{e-1}) l_i \tag{3.3}$$

Suppose an MU is always awake and misses all IRs. Then, the stale cache hit for data object d_i in the MU is given by:

$$P_{si} = \frac{l_{si}}{l_i} = \frac{1}{e-1} \tag{3.4}$$

If an MU has IR broadcast miss probability of P_{mIR}, then the stale cache hit probability is:

$$P_{si} = \frac{l_{si} P_{mIR}}{l_i P_{mIR} + l_{vi}(1 - P_{mIR})} = \frac{P_{mIR}}{P_{mIR} + e - 2} \tag{3.5}$$

Equation (3.5) implies that the stale cache hit probability is only dependent on P_{mIR}. Thus we conclude that the stale cache hit probability for data object d_i is independent of i, i.e., $P_s = P_{si}$ for any i. Figure 3.6 shows the stale cache hit probability for various IR miss probabilities. The results show that the stale cache hit probability is about 10% for an MU with 10% IR miss probability, independent of the data object update frequency.

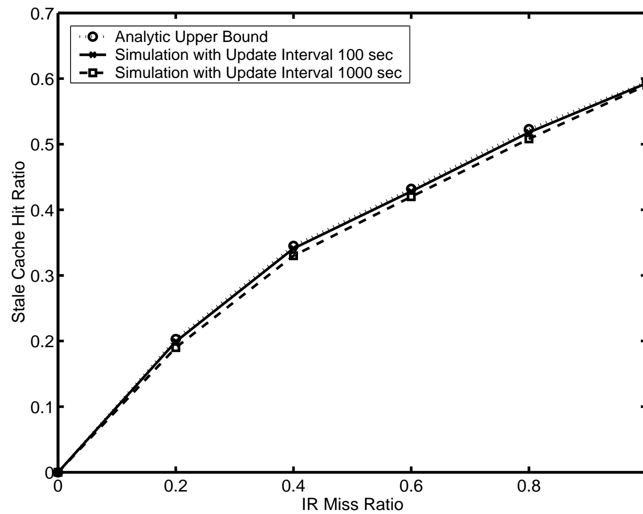


Figure 3.6 Upper Bound on Stale Cache Hit Probability vs IR Miss Probability.

The above analytical model assumes that an MU is always awake. If we consider the sleep-wakeup event for an MU, then the stale hit probability is reduced because when the MU wakes up from the sleep state, all valid entries will be checked prior to their usage. The stale cache hit probability for a frequently disconnected MU is much

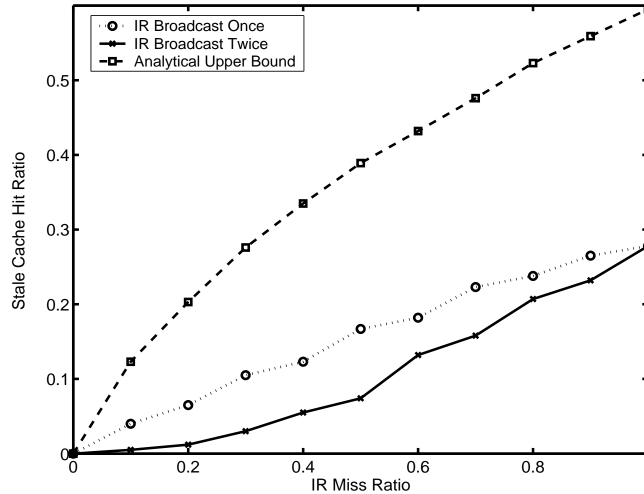


Figure 3.7 Simulation of Stale Cache Hit Probability vs IR Miss Probability.

smaller than the upper bound in Eqn. (3.5). A system with 100 MUs is simulated. In the simulation, the sleep-wakeup period (in *seconds*), the sleep ratio, and the request arrival rate are randomly picked from the set of values (600, 1200, 1800, 2400, 3000), (0.2, 0.35, 0.5, 0.65, 0.8), and (1/20, 1/40, 1/60, 1/80, 1/100), respectively. The detailed simulation setup is described in Section 3.3. Figure 3.7 shows the simulation results for stale cache hit probability of the system, which is reduced significantly as compared to that of MUs which are always awake. For example, when the IR miss probability is 10%, the stale hit probability is only about 4%; and for the IR miss probability of 20%, the stale hit probability is about 7%. In order to reduce the stale hit probability, we can broadcast the IR multiple times when a data object is updated earlier than its TTL expiration. Figure 3.7 shows the stale cache hit probability for broadcasting each IR twice if its update is earlier than the TTL expiration. Our results indicate that if the IR miss probability is less than 40%, the stale cache hit probability is less than 5%. These results demonstrate that SACCS can provide very small stale cache hit probability by broadcasting IR multiple times when the update is earlier than TTL expiration.

3.2.2 Stale Cache Hit Probability for Rayleigh Fading Channels

The stale cache hit probability of SACCS with Rayleigh-fading model for wireless channel is simulated in this section. We consider a Rayleigh-fading channel between the BS and an MU be modeled by a two-state Markov chain as shown in Figure 3.8. An MU can successfully receive packets from the BS if the channel between them is in *good* state, and lose packets if the channel is in *bad* state. The channel states between the BS and different MUs are independent. According to [49], a channel condition can be assumed to be effectively stable during a period of $T_c \approx 9s/(16\pi f_c V)$, where s is the speed of light, V is the MU speed (along the wave path) and f_c is the carrier frequency. The channel in the next T_c period has a probability of transiting to the other state. The state transition probabilities such as P_{gb} (from *good* to *bad* state) and P_{bg} (from *bad* to *good* state) are determined by V and the fading margin F, that is the maximum fading noise of received signal without system performance falling below a specified value [16]. The probability of a channel in *bad* state (P_B) is dependent on F [16]. Table 3.2 shows P_{gb}, P_{bg} and P_B values for a carrier frequency $f_c = 900\ MHz$ with various values of V and F. Note that $P_{gg} = 1 - P_{gb}$ is the transition probability from *good* to *good* state and $P_{bb} = 1 - P_{bg}$ is the transition probability from *bad* to *bad* state. The packet duration is set to 4 ms (millisecond), because $T_c \geq 4\ ms$ for $f_c = 900\ MHz$ and $V \leq 50\ Kmph$ (kilometer per hour).

Table 3.2 State Transition and *Bad* State Probabilities for Values of V and F

$V(kmph)$	5		10		20		40		For all V
	P_{gb}	P_{bg}	P_{gb}	P_{bg}	P_{gb}	P_{bg}	P_{gb}	P_{bg}	P_B
$F = 10\ dB$	0.013	0.125	0.026	0.248	0.050	0.474	0.079	0.747	0.095
$F = 15\ dB$	0.011	0.156	0.021	0.307	0.038	0.566	0.056	0.818	0.065

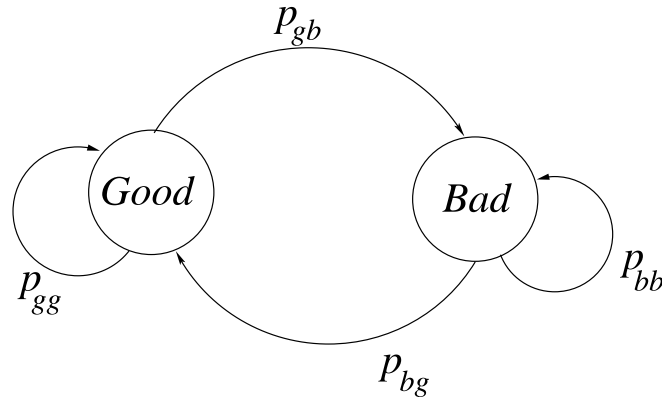


Figure 3.8 Two State Markov Chain Describing the *Good* and *Bad* Channel Model.

From Table 3.2, we note that the slower the MU speed, the smaller the state transition probability. This means that the channel condition is more stable for a slow moving MU than a fast moving one.

We study the performance of SACCS by simulation under Rayleigh fading channel model. In this simulation, an MU requesting a data object uses reliable communication, and all other awake MUs can passively download the data objects if they can successfully receive all the packets of a data object. Each packet duration is set to 4 ms. In this period of time, 100 $bytes$ (i.e., packet size) can be transmitted in a 200 $Kbps$ (kilobits per second) channel. A Go-Back-N (Automatically Repeat reQuest) (ARQ) scheme is used between the BS and the MU for retrieving data objects. The number of MUs is set to 50. Each MU's query access and sleep-wakeup pattern are set the same as in the previous section.

Table 3.3 shows the performance of SACCS with the MU speed. We conclude that the stale cache hit probability depends only on the fading margin. A smaller fading margin leads to a larger probability of a channel in *bad* state, resulting in a larger IR loss probability. For $F = 10\ dB$, $P_B = 0.095$ (note that $P_{mIR} = P_B$), the stale cache hit probability is about 4%; while for $F = 15\ dB$ (i.e., $P_B = 0.065$), the stale cache

Table 3.3 Performance of SACCS at Different MU Speed

Fading Margin	Performance Metrics	Speed $V(kmph)$			
		5	10	20	40
$F = 10\ dB$	Stale Cache Hit Probability	0.0358	0.0374	0.0367	0.0371
	Average Access Delay (sec)	1.387	1.595	1.650	1.762
	Data Download Ratio	0.844	0.857	0.867	0.873
$F = 15\ dB$	Stale Cache Hit Probability	0.0236	0.0230	0.0223	0.0228
	Average Access Delay (sec)	1.368	1.475	1.514	1.586
	Data Download Ratio	0.843	0.855	0.866	0.871

hit probability is about 2%. These results, similar to those of Figure 3.7, demonstrate that the proposed SACCS scheme can provide small stale cache hit probability in the error-prone wireless environments. The average access delay and the data download ratio (defined as the number of broadcast data objects divided by the number of queries) increase as the MU speed increases. This is because a slow moving MU has a more stable channel and stands a better chance to share the broadcast data objects. A larger F results in a smaller P_B, thus a smaller number of retransmissions and ultimately a smaller average access delay.

3.3 Performance Evaluation By Simulation

The performance of SACCS is evaluated and compared with timestamp (TS) and asynchronous stateful (AS) schemes. Recall that TS is a popular stateless scheme and has been widely compared with other schemes. For meaningful comparison, we extend TS with some advanced features of SACCS, such as: (1) introduction of uncertain state for an MU keeping its valid data entry after long disconnection; (2) use of ID-only state in MUC to trigger data object download; and (3) use of flag bits in SC to reduce the IR broadcast traffic. We call a TS with these additional features as *extended TS* (ETS) scheme. The AS scheme is also used for performance comparison because it is one of the

few stateful schemes that successfully handle the MU disconnection and mobility. For SACCS, an IR of a data object is broadcast twice if its update time is earlier than the TTL expiration to reduce the stale hit probability. The other error recovery costs, such as data retransmission, are ignored in all three schemes. The TTL, l_i, of a data object is dynamically calculated as: $l_i = l_i * 0.5 + l_{interval} * 0.5$, where $l_{interval}$ is the current update interval for the data object.

We consider a single cell system with one SC and multiple MUs with identical cache size. The parameters are defined as in Table 3.4.

Table 3.4 Parameter Definition

M	number of MUs in the system
N	number of data objects in the system
C	cache size for MU (*bytes*)
λ	average arrival rate of request for an MU
T_u	average update time interval for a data object (*sec*)
T_p	period for a sleep-wakeup cycle of an MU (*sec*)
s	ratio of the sleep time to the sleep-wakeup period for an MU
b_o	data object size (*bytes*)
b_u	uplink message size (*bytes*)
b_d	downlink invalidation or confirmation message size (*bytes*)
D	average query delay, i.e., the interval between the time request is issued and the time the result is received by the application (*sec*)
UPQ	uplink per query, defined as the total number of queries through uplink channel divided by the total number of queries
L	invalidation broadcast period for TS scheme (*sec*)
wsz	broadcast window size for TS scheme

Each MU's request process and the data object update process are assumed to follow Poisson distributions. The sleep-wakeup process is modelled as a two-state Markov chain (i.e., *sleep* and *awake*). The state transition probability from awake to sleep state is $\alpha = 1/(1-s)T_p$, and that from sleep to awake state is $\beta = 1/sT_p$.

In the simulations, we use two channels with bandwidth W_d and W_u for downlink and uplink data transmissions, respectively. In the uplink channel, all messages are buffered as FIFO (first in first out) queue. In the downlink, there are two FIFO queues, one having higher priority than the other. The IR messages are buffered in the higher priority queue. All other messages are buffered in the lower priority queue; this queue can be scheduled only if the higher priority queue is empty. All requests are ignored when an MU is in the sleep state. When a requested data object is available at an MUC, the average query delay (D) is counted as 0. We consider *Zipf-like* distribution for MU access pattern [7] [69] such that the access probability (p_i) for data object d_i is proportional to its popularity rank, $rank(x)$. More specifically, $p_i = const/rank(x)^z$, where $const$ is the normalization constant and z is the Zipf coefficient. We assume the most popular data has the smallest rank value.

In the following, we present the performance comparison of the proposed SACCS with AS, TS and ETS in terms of such metrics as D and UPQ for three different cases. The average waiting time (i.e., half of the IR broadcast period, $L/2$) is removed from D for ETS to make a better comparison with SACCS and AS in our simulation experiments. As shown in the previous section, for the same sleep-wakeup pattern, the stale cache hit probability is less than 5% if the IR miss rate is smaller than 40%, hence the stale hit probability is not presented as a metric in the result. In each case, $b_u = b_d = 20$ *bytes* for both SACCS and AS; and $b_u = b_d = 10$ *bytes* for TS and ETS. The bandwidth is set as $W_d = 200$ *Kbps* and $W_u = 1$ *Kbps*. The other parameters may be changed in some cases. Some default values are set as: N = 10,000, M = 100, C = 5 MBytes, z = 0.9, L = 20 *sec* and wsz = 5.

In all cases, we consider a system with 10 types of data objects. The data object update rate (T_u), size and percentage of each type of objects over the total objects are shown in Table 3.5. The chosen parameter values are based on the understanding that a

Table 3.5 Ten Types of Data Objects in Database

Data Type	1	2	3	4	5	6	7	8	9	10
$Size(Bytes)$	$1K$	$5K$	$10K$	$15K$	$20K$	$25K$	$30K$	$35K$	$40K$	$45K$
$T_u(sec)$	50	100	200	400	800	1600	3200	64000	12800	25600
$Percentage(\%)$	5	5	10	10	20	20	10	10	5	5

faster updated object usually has smaller size. The average data object size is about 25 Kbytes, which is based on the Internet measurements [5].

The MUs may be different from one another in terms of λ, s, T_p and T_r. These parameters for each MU can take values from the corresponding given sets. Each value has equal probability to be chosen in an MU. The sets of values used are as follows: arrival rate is $\lambda = (1/20, 1/40, 1/60, 1/80, 1/100)$; sleep ratio is $s = (0.2, 0.35, 0.5, 0.65, 0.8)$, and sleep-wakeup period time is $T_p = (600, 1200, 1800, 2400, 3000)$ *sec*.

The query patterns for each MU are assumed to follow Zipf-like distribution. The access popularity ranking for each MU is shifted by a random number between 0 and 99. For example, an MU picks up a shift number 50, which means the MU has the highest access popularity for data object number 51. Suppose there are 1000 data objects in the system. The popularity decreases from 51 to 1000, and 1000 to 1 and then from 1 to 50. The data object 50 has the lowest access popularity.

3.3.1 Effect of the number of MUs

In this case, we study the impacts of three features in SACCS on the system performance as compared with TS, ETS and AS. Let SACCS-nfg stand for SACCS without flag bit set in SC; SACCS-nid be SACCS without ID in MUC; and SACCS-nuc be SACCS without uncertain state in MUC. Recall that ETS is an extension of TS with three SACCS features. We use tables to present the results.

Table 3.6 average Access Delay D (sec) for varing number of MUs

M	20	40	60	80	100	120
$SACCS$	0.907	1.006	1.129	1.329	1.836	2.999
$SACCS$-nfg	0.912	1.021	1.153	1.372	1.932	3.393
$SACCS$-nid	0.968	1.129	1.346	1.736	3.128	13.300
$SACCS$-nuc	1.044	1.149	1.391	1.693	2.674	10.033
AS	0.969	1.139	1.376	1.824	3.619	18.429
TS	13.342	14.444	15.818	17.585	27.244	125.164
ETS	12.774	13.779	14.674	15.488	16.587	18.767

Table 3.7 The Uplink Per query (UPQ) for varying number of MUs

M	20	40	60	80	100	120
$SACCS$	0.902	0.894	0.874	0.866	0.854	0.838
$SACCS$-nfg	0.904	0.895	0.876	0.868	0.856	0.839
$SACCS$-nid	0.927	0.926	0.927	0.925	0.920	0.917
$SACCS$-nuc	0.909	0.896	0.884	0.871	0.862	0.850
AS	0.9000	0.904	0.906	0.910	0.909	0.901
TS	0.926	0.925	0.929	0.930	0.930	0.922
ETS	0.914	0.892	0.889	0.872	0.861	0.847

Tables 3.6 and 3.7 present the values of D and UPQ for varying number (M) of MUs. For all schemes, the average delay (D) increases as the number of MUs increases. The SACCS based schemes have much shorter D compared with AS, TS and ETS, especially when M >100. Moreover, the turning off flag bit of cache entries in SC has the least impact on D. This is due to the fact that the IR message is very small compared to the data object size. SACCS has about 10% less delay than SACCS-nfg when M = 120. Turning off ID or uncertain entries in MUC makes SACCS less scalable and leads to a larger D as M increases. This is because the ID-only and uncertain states allow MUs to share the broadcast data objects, thus saving the downlink bandwidth and consequently reducing the access delay. AS has smaller D than TS, but it does not scale as much as

ETS, which allows strong cooperation among MUs because ETS incorporates all three features of SACCS.

For SACCS based schemes and ETS, the UPQ metric decreases as M increases. But for AS and TS, it is almost constant. This is due to the cooperation among MUs in SACCS based schemes and ETS. Note that SACCS has the least UPQ, while turning off ID has the largest increase on the UPQ.

The simulation studies validate our initial claims, namely, ID-only entry and uncertain state in MUC are critical features of SACCS; and use of flag bit in SC reduces IR traffic. Thus ETS performs better than TS, and hence we will use ETS instead of TS in the following cases.

3.3.2 Effect of Database Size

Figures 3.9 and 3.10 present the simulation results showing the effects of database size. For ETS, the average query waiting time ($L/2 = 10$ *sec*) is not counted. In other words, only the queue delay and transmission time are counted for ETS in all the following cases. SACCS outperforms AS and ETS in both D and UPQ because SACCS avoids unnecessary IR traffic while retaining all the valid data objects in MUCs. As expected, with increased number of data objects, the performance metrics also increase for all three schemes, but SACCS has much smaller D than AS and ETS. Additionally, the average gain (in terms of D) of SACCS over AS and ETS is more than 50% throughout the range of database sizes. The UPQ of SACCS is a little bit lower than that of ETS, and about 6% less than that of AS.

3.3.3 Effect of Zipf Coefficient

In this case, we study the effect of Zipf coefficient, z, on the system performance. Here we choose a small database with $N = 1{,}000$ objects. This is because for small Zipf

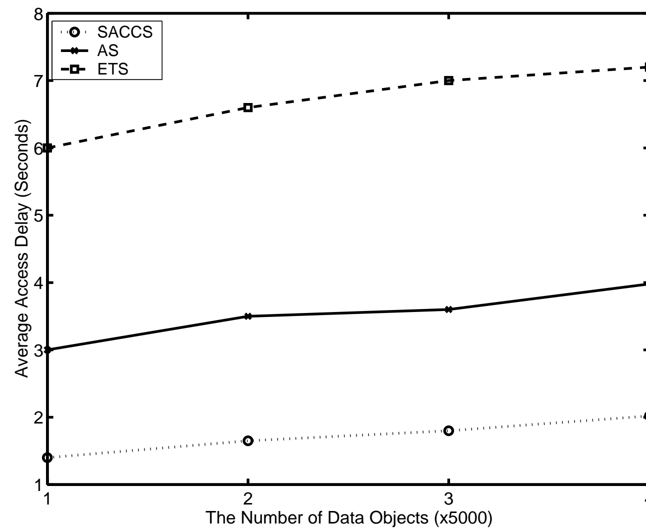


Figure 3.9 Average Access Delay vs Number of Data Objects.

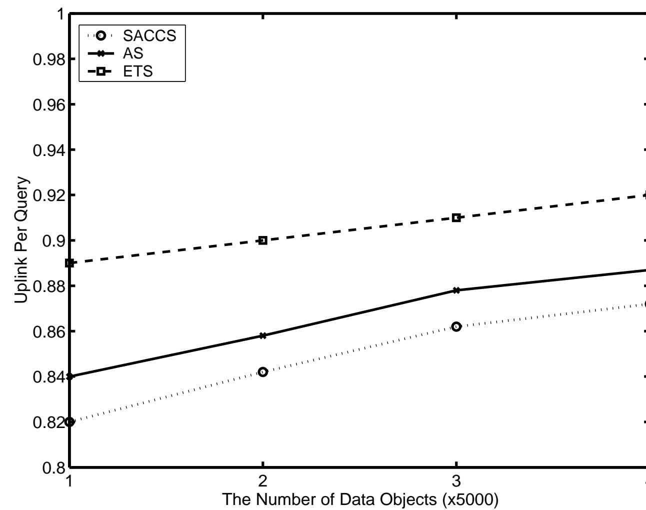


Figure 3.10 Uplink Per Query vs Number of Data Objects.

coefficient, the access frequencies for different data objects are very close to each other. Using a large database size results in very few cache hits, which makes the comparison meaningless.

From Figures 3.11 and 3.12, we conclude that SACCS has much smaller D than both AS and ETS. The average gain is more than 50% over other two schemes. AS has

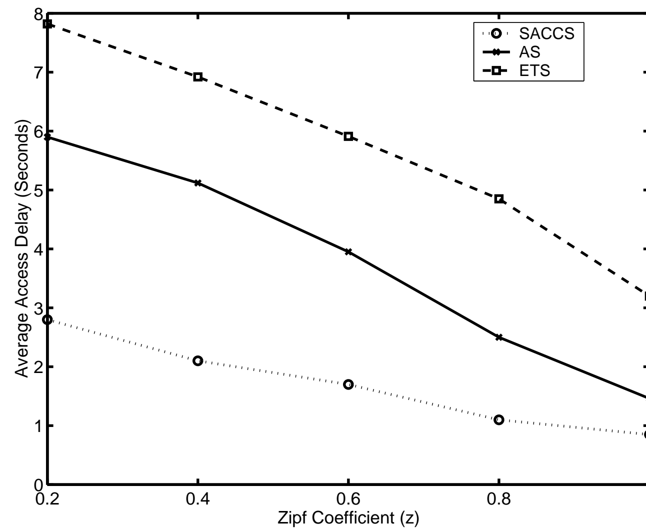


Figure 3.11 Average Access Delay vs Zipf Coefficient.

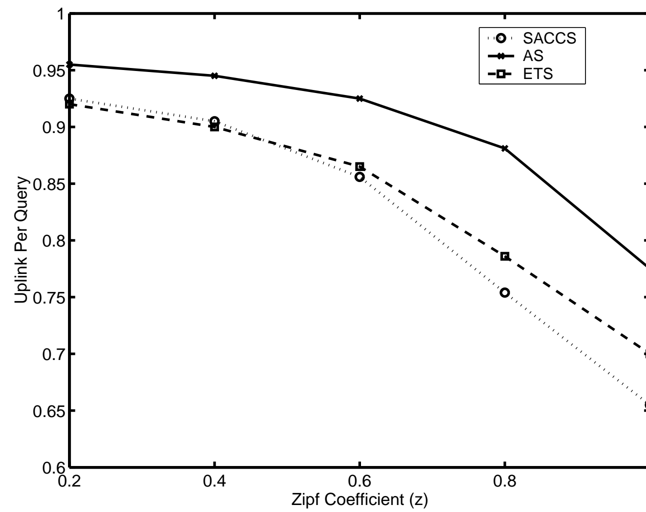


Figure 3.12 Uplink Per Query vs Zipf Coefficient.

the largest UPQ while SACCS has the lowest UPQ when $z > 0.6$, and a little bit more than ETS when $z < 0.6$. All three schemes perform better as z increases, because the data accesses are more concentrated for larger z, thus increasing the cache hit ratio and then reducing the access delay.

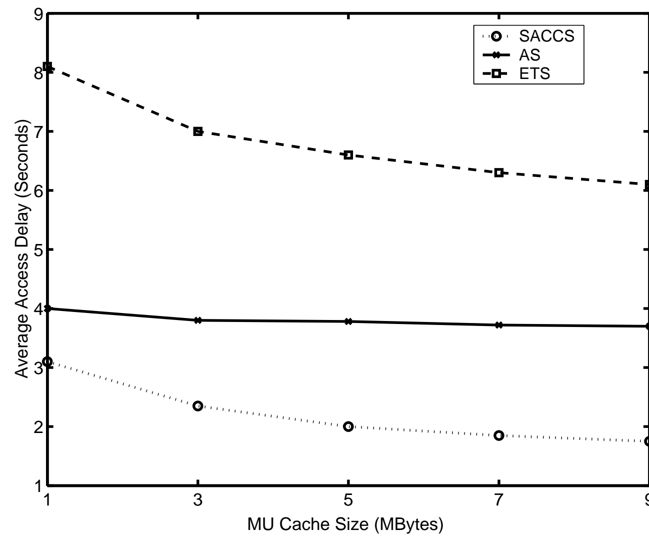


Figure 3.13 Average Access Delay vs MU Cache Size.

3.3.4 Effect of cache size

Figures 3.13 and 3.14, respectively, depict D and UPQ versus MU cache size for the three schemes. Once again, SACCS is much better than ETS and AS in terms of D, and almost the same UPQ for SACCS and ETS which is less than that of AS. Surprisingly, after the cache size reaches $C = 3$ Mbtyes, further increase in cache size does not help improve the performance for AS. This can be explained as follows. As the cache size increases, the time that a given data object stays in the cache before it is replaced, also increases. However, the longer the data object stays in the cache, the better the chance of its invalidation. Hence, as the cache size exceeds certain threshold, further increases in cache size does not help to retain the data object. This is true for AS. However, for SACCS and ETS, due to maintenance of only a data ID for the invalidated object, the corresponding data object has the chance to be downloaded again when other MUs retrieve the same data objects from the server cache (SC).

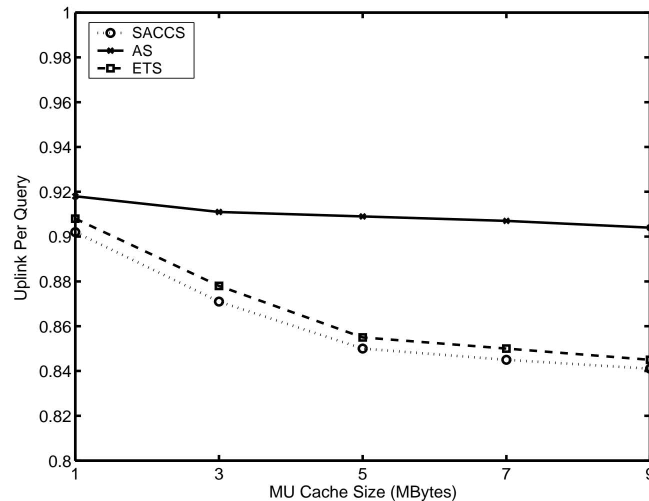


Figure 3.14 Uplink Per Query vs MU Cache Size.

3.4 Summary

In this chapter, we proposed the Scalable Asynchronous Cache Consistency Scheme (SACCS) for mobile environments and evaluated its performance analytically as well as experimentally. Unlike the previous methods, SACCS provides a weak cache consistency under realistic environments for an MU with IR broadcast miss.

Strictly speaking, SACCS is a hybrid of stateful and stateless schemes. However, unlike stateful schemes, SACCS maintains only one flag bit for each data object in BS to determine when to broadcast the IRs. On the other hand, unlike the existing synchronous stateless approaches, SACCS does not require periodic broadcast of IRs, thus significantly reducing IR messages that need to be sent through the downlink broadcast channel. SACCS inherits the positive features of both stateful and stateless schemes. Our comprehensive simulation results show that the proposed scheme offers significantly better performance than TS and AS schemes. So far we are focused on the single cell mobile environments, the impact of multi-cell environments is studied in the following chapter.

CHAPTER 4

DYNAMIC SCALABLE ASYNCHRONOUS CACHE CONSISTENCY SCHEME

In the previous chapter, we focused on the cache consistency maintenance in single cell wireless environments. However, due to the impact of MUs' mobility on the cache performance, a cache consistency scheme designed for single cell may not be efficient for multi-cell environments. In this chapter, we first introduce three types of cache consistency IRs for multi-cell systems. Then, we derive two consistency maintenance cost functions that define the costs associated with roaming of MUs for locally and globally maintained data objects. If a data object is locally maintained, then the IR is broadcast only to the cells in which the data object is retrieved, whereas if a data object is globally maintained, then its IR is broadcast to every cell in the system. The two cost functions are derived by considering the data objects' update frequency, the MUs' access pattern and the roaming frequency, the number of cells and the number of MUs in the system. Based on these two cost functions, we design and evaluate a Dynamic Scalable Asynchronous Cache Consistency Scheme (DSACCS) for wireless cellular networks. Finally, an improvisation of DSACCS, called DSACCS-G, is proposed for grouping cells in order to facilitate effective cache consistency maintenance in multi-cell systems.

4.1 Intra-Roaming and Inter-Roaming

We use *roaming* to indicate the movement of an MU from one cell into another. There are two types of MU roamings. In *intra-roaming*, an MU roams between cells that are all controlled by the same MSC. As shown in Figure 4.1, the movement of an

MU, say A, from Cell 2 to Cell 3 is an example of intra-roaming. In *inter-roaming*, on the other hand, an MU roams between cells belonging to different MSCs. The movement of the MU B from Cell 4 to Cell 3 is an example of inter-roaming. In this chapter, we assume the MSCs as well as BSs have caches, and the MSCs manage the IRs in the system.

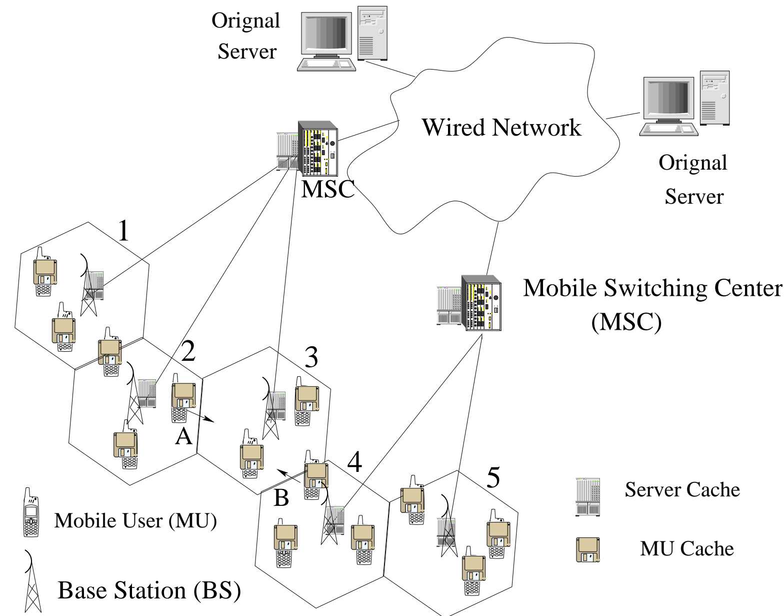


Figure 4.1 Wireless Cellular Network Architecture.

A roaming MU is in one of the two states: awake or sleep. In the sleep state, no extra action is needed for both SACCS and AS schemes. In SACCS, an MU sets all of its valid cache entries to the uncertain state after it wakes up. In AS, when a roaming MU wakes up, its first query is automatically sent to an MSC through the current BS; then the MSC retrieves or resets the cache state corresponding to the current BS, thus maintaining cache consistency. For a roaming MU in the awake state, the cache consistency maintenance is different from the one in single-cell systems because the IR

broadcast may be different in different cells. In the following subsections, we propose three types of strategies for roaming MUs that are awake.

4.2 Three Cache Consistency Strategies for Multi-Cell Cellular Networks

In this section, we will introduce three cache consistency maintenance strategies [60] and apply them to SACCS and AS to study the impact of the MUs' mobility. These strategies are evaluated in various multi-cell environments.

4.2.1 Homogeneous IR Strategy

In this strategy, all BSs in the system broadcast the same IR to MUs. When an MU roams among these cells, there is no difference between a roaming and a static MU with regard to cache consistency maintenance. Consequently, the impact of mobility on the performance of an MUC is minimized. Therefore, such strategies are efficient for fast roaming MUs. However, the total IR traffic for each cell is increased as the number of cells increases. This is because any IR of a retrieved data object must be broadcast to every cell even if it is only queried in one cell. The homogenous IR strategy is not scalable for large number of cells as demonstrated in Section 3.2.2. The corresponding schemes of SACCS and AS will be called HM-SACCS and HM-AS, where "HM" stands for "homogeneous".

In HM-SACCS, only a *single* retrieving flag bit is needed for each data entry in an server cache (SC). When a data object is queried, its corresponding flag bit is *set*. Whenever a data object with a set flag bit is updated, its IR is broadcast to every cell, and the flag bit is *reset*. For a system with multiple MSCs, one MSC informs to all other MSCs to set the flag bit for the same entry. This guarantees the broadcasts of the same IR in every cell.

In HM-AS, the MSC maintains a cache state that records all queried data objects for each MU. When a recorded data object is updated, the MSC broadcasts its IR to every cell, then the cache consistency is maintained as in a single cell. For a system with multiple MSCs, an IR from one MSC is first broadcast to all other MSCs and from there to the cells in the system.

4.2.2 Inhomogeneous IR without Roaming Check Strategy

In this strategy, the broadcast IR varies in different cells. A wakeup event is forced by a roaming MU. The variations of SACCS and AS are called IM-SACCS and IM-AS, respectively, where "IM" stands for "inhomogeneous".

In IM-SACCS, a data object in an MSC's cache has multiple retrieving flag bits, each corresponding to a particular cell. If the data object is retrieved in one cell, the corresponding flag bit is set. When a data object is updated, the MSC informs the BSs. The BS in turn broadcasts the IR if the corresponding flag bit is set for that cell. All flag bits are reset after the data object is updated. A wakeup event is forced for each roaming MU in the sense that the MU sets all its valid data objects into uncertain state. The cache consistency is guaranteed because an uncertain entry must be refreshed or checked before its usage. In IM-SACCS, there is no difference between systems with one or multiple MSCs.

In IM-AS, an MSC maintains a cache state for each MU. The IRs of cached data objects in an MU are broadcast to the MU's current resident cell. A forced wakeup event is also used for a roaming MU. In other words, after an MU arrives in a new cell, its first query and roaming time must be forwarded through its new BS to an MSC. The *roaming time* is defined as the time when an MU switches the communication channel from its previous cell to the new cell. After the MSC receives the first query message, it resets the corresponding state to its new cell for an intra-roaming MU, or

retrieves the corresponding state from the previous MSC and sets it to the new cell for an inter-roaming MU. Then the MSC sends an invalidation-check message to the MU that contains all cached data objects updated in the interval between *roaming* and *current* time. This is because an MU may miss IR broadcast during roaming. After the MU receives an invalidation-check message, it drops all invalid data entries, thus maintaining cache consistency.

One advantage of this kind of strategy is that it minimizes IR traffic. But the extra uplinks are introduced by a forced wakeup event for a roaming MU. The strategy is superior to others in a system with slow roaming MUs and fast updating data objects. It also exhibits better scalability than homogeneous strategy.

4.2.3 Inhomogeneous IR with Roaming Check Strategy

The IR traffic varies in different cells, similar to that in the inhomogeneous IR without roaming check. The variations of SACCS and AS are called as CK-SACCS and CK-AS, where “CK” stands for “check”.

In CK-SACCS, after an MU arrives into a new cell, it immediately sends its roaming time and an ID list (that includes all valid cache data object IDs) to an MSC through the new BS. The MSC in turn sends an invalidation-check message containing all updated data objects in the MUC between the roaming and current time. The MU drops all invalidated cache entries. Then the MSC sets the corresponding retrieving flag bit for each valid entry in the new cell. After the flag bit is set, any future IRs for these valid entries will be broadcast in that cell to maintain cache consistency.

In CK-AS, a roaming MU immediately sends a roaming check message to an MSC through its new BS. The message includes its roaming time and previous MSC identifier. After the MSC receives the roaming check message, it resets the corresponding cache state to the new cell for an intra-roaming MU, or retrieves the cache state from the

previous MSC and sets it to the new BS for an inter-roaming MU. If any data object is updated between the roaming time and the current time, an invalidation-check message is sent to the MU to invalidate the updated cache entries. Consequently, the future IRs for these valid entries are broadcast to the new cell.

The cost for this strategy is an extra uplink for each roaming. But the valid entries of an MU's cache can be used immediately, resulting in reduced access delay for valid cache data objects. Finally, the IR traffic is also minimized.

The features of the above three strategies are summarized in Table 4.1.

Table 4.1 Features of Three Cache Consistency Maintenance Strategies

Strategy	Action on Roaming	Broadcast IR	Fast Roaming MU	Fast Updating Data Object	Scalability
Homogeneous	no action	all cells	efficient	not efficient	poor
Inhomogeneous without roaming check	force a wakeup event	one cell	not efficient	efficient	good
Inhomogeneous with roaming check	immediately send a roaming check message	one cell	not efficient	efficient	good

4.3 Performance Evaluation of Three Strategies

In this section, we study the performance of three proposed strategies under various system conditions through simulation experiments. The MU's roaming process is assumed to be Poisson with an average sojourn time, T_r. When an MU arrives in a new cell, the data object queries that were sent to the previous BS are resent to the current BS. After the current BS gets such requests, it sends a message to the previous BS indicating that the retrieved data objects by the roamed MU are not necessary to

be broadcast. The previous BS, in turn, removes those data objects from the broadcast queue if they are not retrieved by other MUs in the cell.

Assume each cell has an uplink channel with bandwidth W_u *bps* (bits per second) and a downlink channel with bandwidth W_d *bps*. There are two queues for downlink channel, one having a higher priority than the other. Each queue is scheduled on a FIFO (first in first out) basis. The IR messages are buffered in the higher priority queue, and all other messages are buffered in the lower priority queue, that is scheduled only if the higher priority queue is empty. All uplink messages in a cell are buffered in a queue scheduled on FIFO basis. The following parameters are defined in our simulations:

- B: number of cells
- T_r: average sojourn time for an MU in a cell (*sec*)

The other parameters are the same as those defined in Chapter 3. We still use average access delay (D) and uplink per query (UPQ) as two performance metrics. Recall that UPQ is defined as the number of total uplinks that include missing queries, uncertain queries, and roaming check messages divided by the number of queries in a system. Hence, for CK-SACCS and CK-AS schemes, UPQ may be larger than 1 due to an extra roaming check uplink for each roaming.

Figure 4.2 shows three multi-cell configurations. Configuration 1 (shaded cells 1 and 2) is a simple two-cell system. An MU from one cell can roam into another. Configuration 2 (cells 1-7) is a tier one system such that an MU from cell 1 can roam into any of the neighboring cells with a probability (1/6). An MU from one of cells 2 - 7 may roam into cell 1 with a probability 1/6 and another neighboring cell with a probability 5/12. Configuration 3 (cells 1-19) is a tier two system. Each of cells 1 through 7 has 6 neighbors, also called *inner* cells. An MU from an inner cell has equal probability (1/6) of roaming into any of its neighboring cells. A cell, which is not an inner cell, is called an *outer* cell. An even numbered outer cell has four neighboring cells. An MU

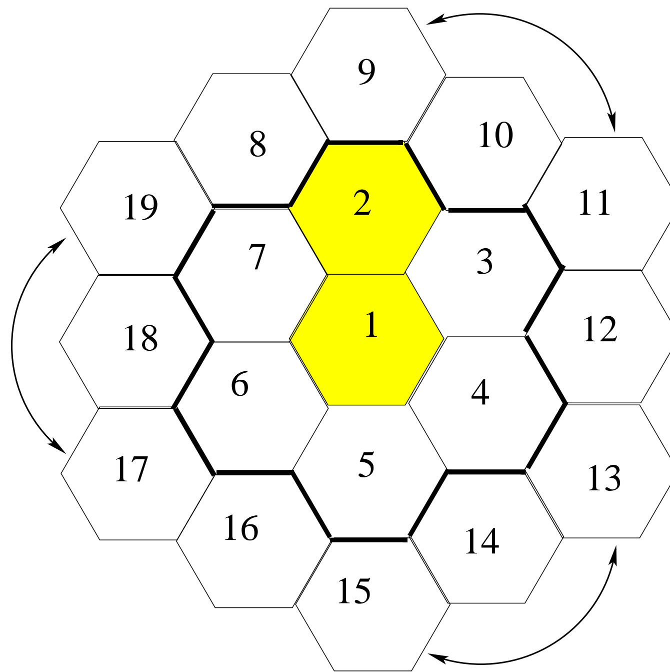


Figure 4.2 Three Configurations: (1) Cells 1 and 2, (2) Cells 1-7, (3) all cells.

from such a cell has 1/6 probability of roaming into one of its inner neighboring cells and 1/3 probability of roaming into one of its two outer neighboring cells. On the other hand, an odd numbered outer cell has three neighboring cells originally, for each such cell we set another odd numbered outer cell as its logical neighbor (indicated by arrow in Figure 4.2) so that each odd numbered cell also has four neighbors. For example, cells 9 and 11 are logical neighbors of each other. An MU from such a cell has 1/6 probability of roaming into its inner neighboring cell, 1/3 into each of its even numbered outer neighboring cells and 1/6 into the logical neighboring cell. This roaming probability balances the number of MUs in each cell. For studying the impact of data object update frequency (Section 4.3.1), only configuration 2 is used, and for studying impacts of the number of cells (Section 4.3.2), all three configurations are used.

In all two cases, we set $W_u = 4\ Kbps$, $W_d = 50\ Kbps$, $b_u = 20\ bytes$, $b_d = 20\ bytes$, $z = 0.9$, $N = 10,000$ and $C = 1\ Mbytes$. Note that the uplink roaming check message size

for an MU using CK-SACCS is set as the number of valid cache data objects multiplied by 4 *bytes*. Thus, for an MU with 50 valid cache entries, its roaming check message size is 200 *bytes*. The third generation (3G) wireless system has up to 144 *Kbps* bandwidth for high mobility traffic, the bandwidth includes all traffic in a cell. We consider a system of applications without security consideration, such as news, traffic information, etc. All other security based traffic can not be shared by MUs and must use a secure communication model, e.g., email, ftp, etc. Hence we choose 50 *Kbps* for public data communication traffic in our simulations.

Table 4.2 Ten Types of Data Objects in Database

Data Type	1	2	3	4	5	6	7	8	9	10
Size (*Bytes*)	500	1K	2K	3K	4K	5K	6K	7K	8K	9K
Update interval(T_u) (*Sec*)	60	120	240	480	960	1920	3840	7680	15360	30720
Percentage (%)	5	5	10	10	20	20	10	10	5	5

We consider a database with 10 types of data objects, and values of their update rate, size and percentage of the total database are shown in Table 4.2. In Section 4.2.1, the effects of data object update frequency are studied, and hence the update frequency of all data objects are set to be the same.

Each MU may be different from others. The parameters λ, s, and T_p for each MU can take values from the corresponding given sets. Each value has equal probability to be chosen for an MU. The values are set as λ = (1/20, 1/35, 1/50, 1/65, 1/80), s = (0.15, 0.3, 0.45, 0.60, 0.75), T_s = (500, 1000, 1500, 2000, 2500) *sec*, and T_r = (100, 400, 1600, 6400, 25600) *sec* . In Section 4.3.1, we assume all MUs are always in the awake state. In this case, sleep-wake up events do not affect the performance. The most popular data object in the Zipf distribution for each MU is shifted by a random number between 0

and 99. The maximum number of IDs that can be kept for each MUC is set to 200 for SACCS based schemes, since the average number of data objects that can be cached for an MU is about 200. Similar to the single-cell case, each IR is broadcast twice to reduce the stale cache hit probability.

4.3.1 Impact of Data Object Update Frequency

In this case, $M = 350$, i.e., 50 MUs per cell, and T_u of each data object is assumed to be the same. In order to isolate the performance impacts caused by the sleep-wakeup events, we assume that each MU is always in the awake state. Six schemes based on the three proposed strategies are evaluated. We also consider a naive scheme, where an MU has no cache in its local buffer and every data access needs to be retrieved from the BS, and no IR is broadcast in the system.

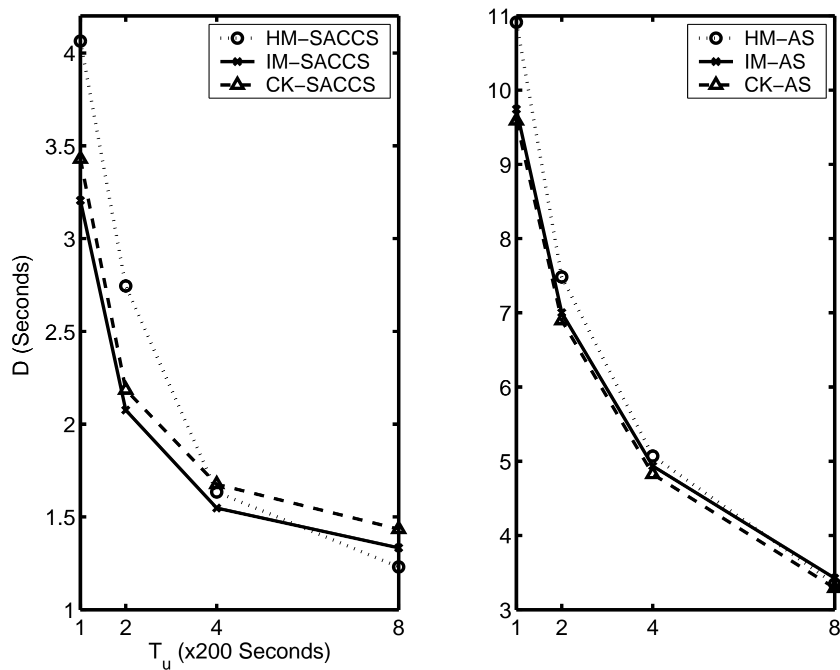


Figure 4.3 Average Access Delay (D) vs Average Data Update Interval.

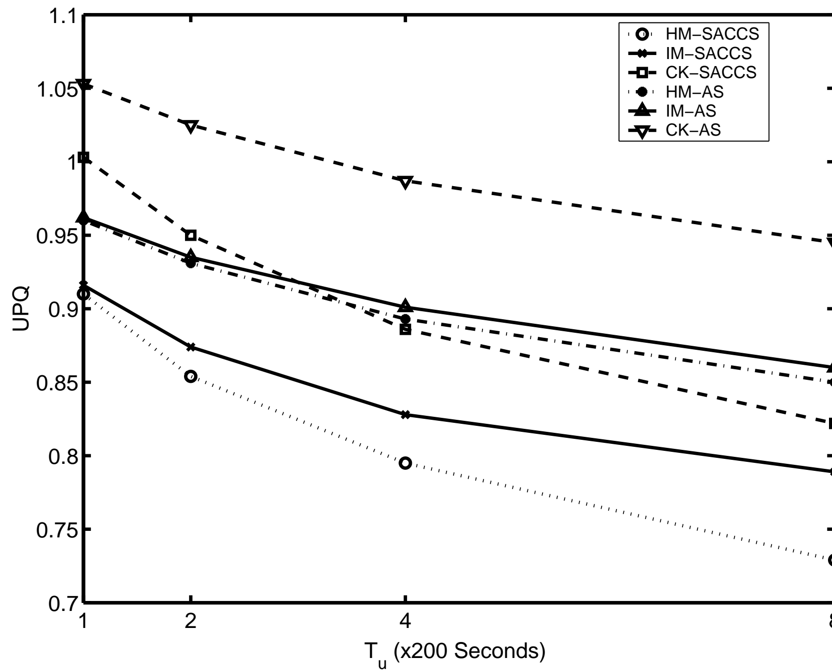


Figure 4.4 Average Uplink Per Query vs Average Data Update Interval.

Figures 4.3 and 4.4 show the relationship between T_u and the system performance in terms of D and UPQ for six schemes. In the naive scheme, each data access is retrieved from the BS and no IR is broadcast, hence D is independent on T_u. In the naive scheme, D is about 31 *sec*s (not shown in the Figure), observe that we know that D of the naive scheme is much longer than those of the other schemes. This indicates that caching data objects at the MU's local buffer is an effective method to improve system performance.

In all these schemes, both D and UPQ decrease as the update interval increases. The performance of SACCS based schemes is always better than that of AS based schemes when T_u is in the range 200 to 1600 *sec*s. SACCS based schemes have more than 50% gain in terms of D over AS based schemes. These results indicate that SACCS based schemes are superior to AS based schemes for multi-cell environments.

Among the variants of the SACCS based scheme, IM-SACCS has the best performance in terms of D at the cost of slightly more UPQ than HM-SACCS, for small T_u (less

than 800 *secs*). When T_u is larger than 800 *secs*, HM-SACCS has the best performance in terms of both D and UPQ. The data objects with small T_u are updated more frequently, leading to increased IR traffic in HM-SACCS. On the other hand, frequently updated data objects in the cache become invalid quickly. Thus, the cache hit ratio is reduced for fast updating data objects and results in fewer confirmation messages due to roaming in IM-SACCS. Hence, IM-SACCS has better performance for frequently updated data objects. For data objects with large T_u, the additional confirmation traffic is more than the extra IR traffic and makes HM-SACCS superior to IM-SACCS. Similar patterns are observed for the AS based schemes. For short T_u (less than 800 *secs*), HM-AS has the best performance. But IM-AS and CK-AS are superior to HM-AS in terms of D when T_r is over 800 *secs*.

In summary, the inhomogeneous strategies are more efficient for fast updating data objects, but inhomogeneous strategies are better for slow updating objects.

4.3.2 Impact of the Number of Cells

The relationship between the performance and the number of cells in the system is studied here. As shown in Figure 4.2, three multi-cell configurations are simulated. The number of MUs for each cell is set to 100 initially. Thus, configurations 1, 2 and 3 have 200, 700 and 1900 MUs respectively.

From Figures 4.5 and 4.6, we observe that the performance of both the SACCS and the AS based schemes are similar, though the SACCS based schemes perform better than the AS based schemes. In Configuration 1, homogeneous IR strategies have the best performance. In the other two configurations, inhomogeneous IR strategies have better performance. For the homogeneous IR strategies, D increases much faster than the inhomogeneous IR strategies as the number of cells (B) increases. UPQ is almost a constant for all strategies in all three multi-cell environments. In the homogeneous

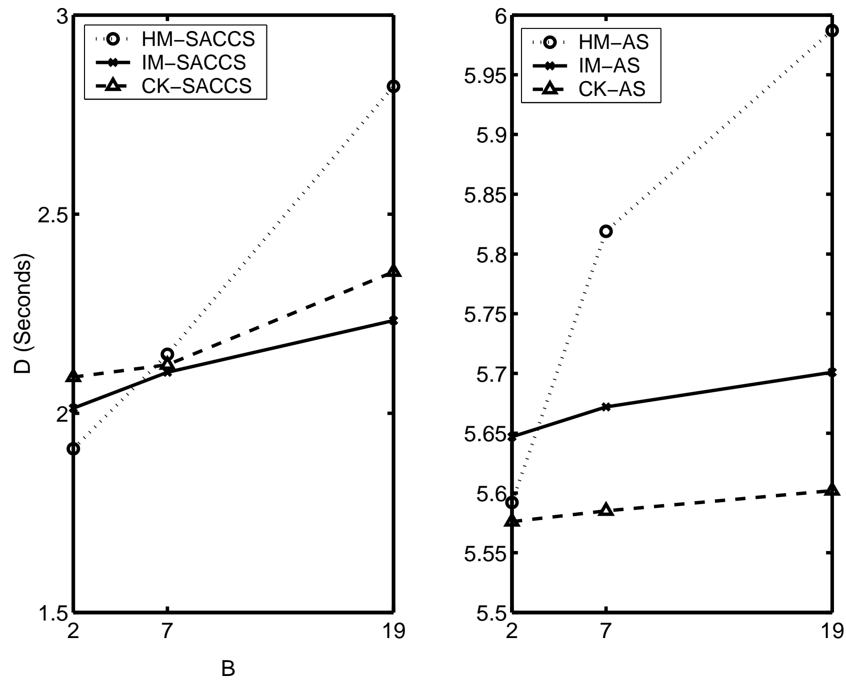


Figure 4.5 Average Access Delay (D) vs Number of Cells (B).

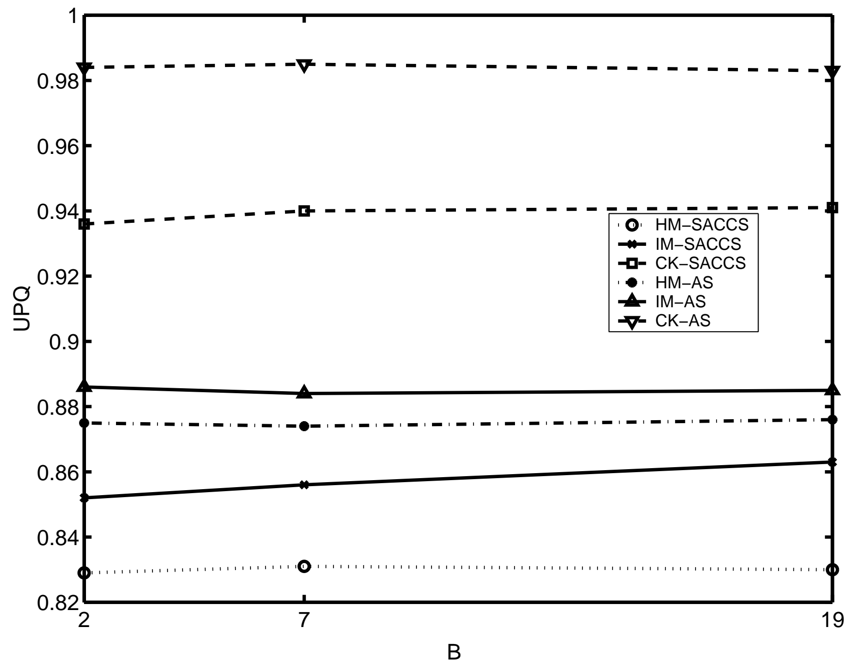


Figure 4.6 Average Uplink Per Query vs Number of Cells (B).

IR strategies, every cell broadcasts the same IR for any retrieved data object. As the number of cells increases, the additional IR traffic increases, thus leading to a longer D. However, there is no such effect on inhomogeneous IR strategies. The results indicate that the homogeneous IR strategies are not as scalable as inhomogeneous IR strategies due to the increased IR traffic.

The results of AS based schemes are similar to those of SACCS based schemes. But the difference among the three schemes is small. These results imply that the inhomogeneous IR schemes are more scalable than homogeneous IR schemes in terms of the number of MUs.

4.4 Dynamic Scalable Asynchronous Cache Consistency Scheme

From the above results, we know that no single cache consistency strategy for roaming MUs always performs better than others. Hence, it is necessary to develop a dynamic scheme that captures the positive features of these three strategies and achieves the optimized cache performance for roaming MUs. For this purpose, we first derive two consistency maintenance cost functions that define the costs associated with roaming MUs for globally and locally maintained data objects. Then based on these two cost functions, a Dynamic SACCS (DSACCS) is proposed to minimize the cache consistency maintenance cost in roaming MUs.

4.4.1 Two Consistency Maintenance Cost Functions

Through the performance evaluation of the three proposed strategies, we conclude that the homogeneous IR strategies are more efficient for slow updating data objects, fast roaming MUs and small systems. While the inhomogeneous IR strategies are more

efficient for fast updating data objects, slow roaming MUs and large systems. Hence, a good scheme should inherit positive features from both.

In a multi-cell wireless system, the consistency of a data object is said to be *globally* maintained if it is necessary to broadcast the IR to every cell in the system. Otherwise, it is said to be locally maintained. For a globally maintained data object, broadcasting IR to every cell results in additional IR traffic for those cells in which the data object is not retrieved. For a locally maintained data object, after an MU roams into another cell, the valid cache entry is set to the uncertain state even if the data object is still valid, thus leading to additional confirmation messages when the data object is accessed. Based on these observations, we derive two consistency maintenance cost functions that define the costs associated with the roaming MUs for globally and locally maintained data objects. The cost for a globally (locally) maintained data object is the number of additional IR (confirmation) messages. Let us first define some parameters:

- α_i: update arrival rate of ith data object d_i
- λ_{ij}: access arrival rate of d_i for jth MU m_j
- λ_{ik}^c: access arrival rate of d_i at cell k
- s_j: ratio of sleep time to the sleep-wakeup period of m_j
- h_{ij}^g: hit ratio of globally maintained d_i (say d_i^g) for m_j
- h_{ij}^l: hit ratio of locally maintained d_i (say d_i^l) for m_j
- C_i^g: consistency maintenance cost associated with roaming of MUs of d_i^g per update period
- C_i^l: consistency maintenance cost associated with roaming of MUs of d_i^l per update period

In our model, the data object update and MU's query, roaming and sleep wake-up processes are assumed to be Poisson. C_i^g is the number of extra IR messages for each update of d_i. During an update interval, if d_i is retrieved in the system, the IR

is broadcast to every cell. In this case, if d_i is not retrieved in a cell, C_i^g includes the number of IR messages of d_i in the cell, then

$$C_i^g = 2B \int_0^\infty \alpha_i e^{-\alpha_i t} \int_0^t \sum_{k=1}^{B} \lambda_{ik}^c e^{-\sum_{k=1}^{B} \lambda_{ik}^c \tau} d\tau dt - 2 \sum_{k=1}^{B} \int_0^\infty \alpha_i e^{-\alpha_i t} \int_0^t \lambda_{ik}^c e^{-\lambda_{ik}^c \tau} d\tau dt$$
$$= \sum_{k=1}^{B} \frac{2\alpha_i}{\alpha_i + \lambda_{ik}^c} - \frac{2B\alpha_i}{\alpha_i + \sum_{k=1}^{B} \lambda_{ik}^c} \quad (4.1)$$

In Equation (4.1), the factor 2 is due to broadcast of each IR twice. The first term is the number of IR messages for d_i^g in an update interval. The IR messages of d_i^g are broadcast to every cell when d_i is retrieved in the system. The total retrieving rate for d_i is $\sum_{k=1}^{B} \lambda_{ik}^c$. The second term is the number of IR messages for d_i^l in an update interval; the IR messages are only broadcast to cells in which d_i is retrieved.

The difference between h_{ij}^g and h_{ij}^l is caused by the roaming of m_j. For d_i^l in the cache of m_j, the valid entry for d_i is set to uncertain state after each roaming, and hence reduces the hit ratio of d_i. Here C_i^l is the summation of extra confirmation messages from all MUs in the system during an update interval of d_i. The average update interval of d_i is $1/\alpha_i$. During this period, on the average, m_j has $(1-s_i)\lambda_{ij}/\alpha_i$ accesses on d_i, thus the additional confirmation messages due to reduced hit ratio is $(h_{ij}^g - h_{ij}^l)(1 - s_i)\lambda_{ij}/\alpha_i$. Note that each confirmation incurs one uplink and one downlink message. Then C_i^l is given by,

$$C_i^l = \sum_{j=1}^{M} 2(h_{ij}^g - h_{ij}^l)(1 - s_i)\lambda_{ij} \frac{1}{\alpha_i} \quad (4.2)$$

Equations (4.1) and (4.2) define two cost functions associated with roaming of MUs for maintenance of data objects globally and locally. These cost functions are determined by the number of cells (B), the number of MUs (M), the data object update rate and the

hit ratio. The hit ratio is related to the data query pattern, data update rate and cache size. The reduced hit ratio for a locally maintained data object in an MUC depends on the MU's roaming and query pattern.

4.4.2 Description of DSACCS

Based on the above two cost functions, DSACCS is proposed to minimize the data consistency maintenance cost in a system. These costs for each data object are computed at the MSC, in order to determine whether the data object is globally or locally maintained.

Regardless of whether d_i is globally or locally maintained, C_i^g is measured as the average number of cells in which d_i is not retrieved during an update interval. Whenever d_i is updated, the number of cells in which d_i is not retrieved is counted, and then C_i^g is recalculated by the MSC. C_i^l is the average number of extra confirmations from all MUs during an update interval. For d_i^l, an extra confirmation may be introduced only when the uncertain entry is accessed. In this case, an uncertain uplink message must be sent to the MSC, and hence the MSC can record all extra confirmations in the system. For d_i^g, an MU can identify the saved extra confirmations and store at the local buffer. When d_i is accessed through the uplink query, the extra confirmations are passed to the MSC.

To measure the derived costs for each data object, some parameters are introduced for each cache entry in the MSC and MUs. In an MSC, five parameters (g_i, C_i^g, C_i^l, R_i^g, R_i^l) are associated with cache entry i. Here g_i is a global flag bit indicating a globally (value 1) or locally (value 0) maintained d_i. The parameters R_i^g and R_i^l are the current measured values of C_i^g and C_i^l. In an MUC, the entry i has three additional parameters (gf_i, rf_i, cm_i), where gf_i is a global flag bit, rf_i is a roaming flag bit (explained later) and cm_i records the number of saved extra confirmations for d_i^g associated with roaming. A

global flag bit is also associated with each downlink valid data and confirmation broadcast messages. rf_i and cm_i are attached to each uplink query message of d_i.

In an MUC, gf_i is used to identify a globally or locally maintained data object. After an MU roams into a new cell, all valid entries i ($i = 1, 2, ..., N$) whose gf_i= 0 is set to uncertain and those with gf_i= 1 are unchanged. Whenever an MU caches or refreshes entry i, the parameter gf_i is set to the same value as the global flag bit in the broadcast message.

We use rf_i to identify the extra confirmations for d_i. Note that rf_i and cm_i of entry i are initially set to 0 when it is cached. These two values are passed to the MSC when the MU retrieves d_i through the uplink query. When an MU roams into a new cell, rf_i is set to 1 for valid entry i. The roaming flag bit rf_i is reset to 0 if any of the following events occur: (1) the MU wakes up, (2) the MU receives an IR, a confirmation or a valid data object for entry i, and (3) the MU accesses d_i. For d_i^l ($i = 1, 2, ..., N$), the valid entry i is set to the uncertain state after each roaming. If d_i is not updated and refreshed, and the MU has no sleep wake up event after roaming, then access to d_i results in an extra confirmation due to roaming.

For d_i^g ($i = 1, 2, ..., N$), an extra confirmation is saved if it is accessed as a valid entry with $rf_i = 1$. This valid entry would be uncertain if d_i were locally maintained. $rf_i = 1$ indicates that the MU has no sleep wakeup event, did not receive an IR, refresh or access d_i after the last roaming. Hence, a saved confirmation is counted for d_i^g. Thus the MU increases cm_i by 1 and resets $rf_i = 0$. After the entry becomes an uncertain or ID-only, the next access to d_i is retrieved from the MSC. In this case, the value of cm_i is passed to the MSC and then reset to 0. The MSC in turn adds the cm_i into R_i^l.

For d_i^l ($i = 1, 2, ..., N$), an extra confirmation may be counted when an uncertain entry with $rf_i = 1$ is accessed. In this case, an uncertain query associated with rf_i is sent to the MSC, and if d_i is still valid, the MSC records an extra confirmation through

increasing R_i^l by 1. Hence, the value of R_i^l for d_i^l is exactly measured, but for d_i^g may be different from the real value during each update period of d_i. However, the average value (C_i^g) is close to its real average value.

In an MSC, the retrieving flag bits for each cache entry are managed in the same way as IM-SACCS scheme. When the MSC receives an update for d_i, if the retrieving flag bit is not set for any cell in the system, the MSC just updates the cached d_i. Otherwise, the MSC first checks g_i. If g_i is set, the IR of d_i is broadcast to every cell, else the IR is only broadcast to the cells in which the retrieving flag bits are set. Now R_i^l is counted as the number of cells in which the retrieving flag bits are not set. Then C_i^g and C_i^l are updated by using $C_i^g = \beta_1\ C_i^g + (1 - \beta_1)R_i^g$ and $C_i^l = \beta_2\ C_i^l + (1 - \beta_2)\ R_i^l$, respectively. Here β_1 and β_2 are two smoothing coefficients in the range 0 to 1. If $C_i^g < C_i^l$, g_i is set to 1 (i.e., globally maintained), otherwise it is set to 0. Then the retrieving flag bits, R_i^g and R_i^l are reset to 0. When the MSC receives a query for d_i, it sets the retrieving flag bit for the corresponding cell, adds cm_i which is attached to the uplink query message into R_i^l. If the query is uncertain with $rf_i = 1$ and the uncertain d_i is still valid, an extra confirmation is added into R_i^l. When a valid data object or confirmation message is broadcast, the global flag bit of the entry in the MUC is set to the same value as that in the MSC cache. If there are multiple MSCs in the system, for each update of d_i, the values of R_i^g and R_i^l in each MSC must be forwarded to all other MSCs.

DSACCS can be dynamically applied to different systems with some advanced features. In the multi-cell system in Figure 4.7, all 16 cells are controlled under two MSCs. Shaded Cells (i.e., cells 1-3, 5-8, and 12-13) are controlled by one MSC, and the remaining cells are controlled by another MSC. Suppose, a university campus is located in a region corresponding to Cells 7-11. Students in the university frequently roam within the campus and rarely roam out. Students frequently access some common interesting data objects that may not be of interest to MUs outside the campus. In this scenario,

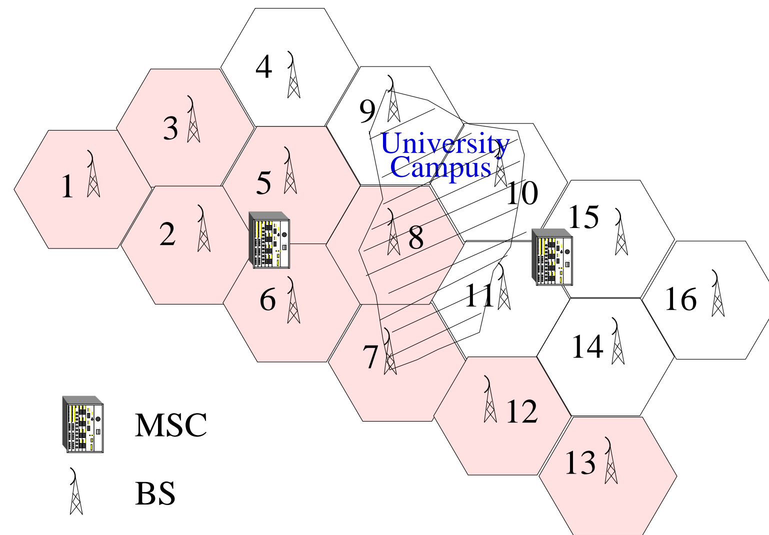


Figure 4.7 A multi-cell system.

some data objects may be highly accessed in the campus cells but rarely in other cells, thus the additional confirmations for these data objects are caused by roaming within the campus. We can define a cell group that includes all campus cells and allows some data objects to be maintained as group data objects. For a group data object, its IR is broadcast to every cell in the group if it is retrieved in any cell of the group. The valid entry for a group data object is unchanged when an MU roams among the grouped cells. This is an efficient mechanism because: (1) it minimizes the mobility impacts on the MUs' roaming within the grouped cells, and (2) all additional IRs from the grouped cells are only broadcast in themselves, without affecting other cells.

Now we define DSACCS-G with cell grouping scheme. In DSACCS, the access popularity of a data object in a cell can be measured through the retrieving flag bit for that cell. When some data objects are observed to be very popular in some cells but not popular in other cells, a cell group can be defined by grouping these cells into a logical cell. After the group is defined, some data objects can be maintained as group data objects. The IR of a group data object is broadcast to all grouped cells if it is retrieved

in any cell of the group. When an MU roams among the grouped cells, the entry states for group data objects in the cache are unchanged.

To manage group data objects, a group flag bit (f_i) is introduced for cache entry i in the caches of MSCs and MUs. $f_i = 1$ implies that d_i is a group data object, and $f_i = 0$ implies it is not a group object. Four parameters (E_i, L_i, e_i, l_i) are set for entry i in the MSC. Here E_i and e_i are the average and current measured extra IRs of d_i per update interval in the grouped cells (i.e., the number of cells in the group in which the data object is not retrieved during an update interval). L_i and l_i are the average and current number of extra confirmations during an update interval due to MUs roaming among the grouped cells. Two extra parameters (r_i, cr_i) are associated with each entry in the cache of an MU. r_i is used to identify the extra confirmations for the MU roaming among the grouped cells. It is set to 1 for valid entry i when the MU roams from one grouped cell to another. cr_i stores the number of extra confirmations of d_i. The management of these extra parameters is the same as the management of parameters introduced in DSACCS.

In DSACCS-G, after d_i is updated, the MSC first checks if d_i is globally or locally maintained by comparing C_i^g with C_i^l. If d_i is globally maintained, there is no difference between DSACCS and DSACCS-G. However, if d_i is locally maintained, the MSC sets it as a group data object if $E_i < L_i$, otherwise d_i is locally maintained.

4.4.3 Performance Evaluation

The performance of DSACCS is evaluated by varying the number of MUs per cell. Section 4.4.3.2 evaluates DSACCS and DSACCS-G under various database sizes. Due to the wired communication between the MSCs, the communication delay is insignificant compared with low bandwidth wireless communications. Hence we do not consider the cost for communications among the MSCs. In the simulations, the smooth coefficients β_1 and β_2 are set as 0.5.

As mentioned above, cm_i of d_i^g in an MUC is passed to the MSC in its next uplink query for d_i. When entry i is invalid, cm_i is associated with the ID-only entry. If the number of ID-only entries is too small, some saved confirmations may be lost due to the dropped ID-only entries. However, the globally maintained data objects should be popular and hence have more chance of being retained in the cache. We varied the number of ID-only entries in the simulation (not presented in here), and found that 200 ID-only entries are enough for the MSCs to record most of the saved extra confirmations for globally maintained data objects. Thus we choose 200 ID-only entries in the simulation.

4.4.3.1 Impact of the Number of MUs Per Cell

In this case, we use a 19-cell system similar to Configuration 3 shown in Figure 4.2, and set N = 10,000 and z = 0.95. All other parameters for the MUs and the database are set to the same as those in Section 4.3.2.

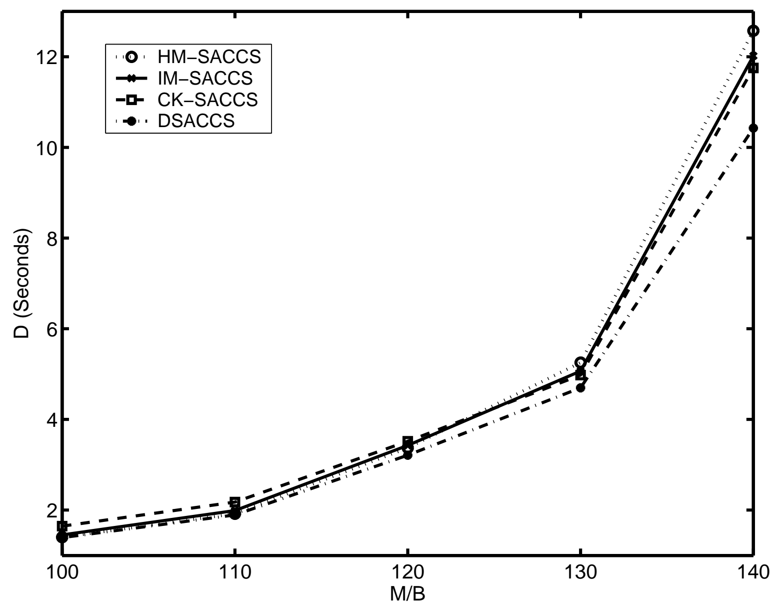


Figure 4.8 Average Access Delay vs Number of MUs per Cell.

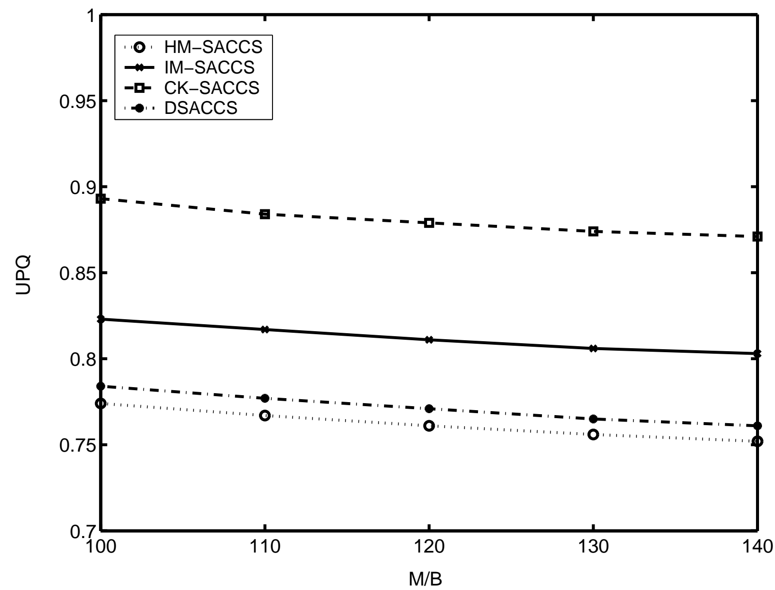


Figure 4.9 Average Uplink Per Query vs Number of MUs per Cell.

Figures 4.8 and 4.9 show the performance of three extended SACCS schemes and DSACCS. We can see that DSACCS has the best performance in terms of delay (D) for varying number of MUs per cell. When the number of MUs per cell reaches 140, DSACCS performs about 10% better than the other three schemes. Among the other schemes, HM-SACCS has better performance for small number of MUs per cell; both CK-SACCS and IM-SACCS are superior to HM-SACCS when the number of MUs per cell is over 130. HM-SACCS has the minimum UPQ and CK-SACCS has the maximum UPQ. DSACCS has a little bit larger UPQ than HM-SACCS varying number of MUs per cell. The consistency maintenance cost of each data object is minimized in DSACCS, thus resulting in better performance. These results indicate that DSACCS inherits both positive features from homogenous and inhomogeneous strategies, and makes it more scalable and efficient.

4.4.3.2 Impact of Database size

In this case, DSACCS and DSACCS-G are evaluated for a system as shown in Figure 4.7. Two types of MUs are set in the simulation. There are 1100 of the first type MUs (initially 100 per cell) located in cells 1-6 and 12-16, and 1000 (200 per cell) MUs of the second type (students) in cells 7-11.

The parameters for the first type of MUs are set the same as that of previous section. For a student in the campus, the only different parameter from the first type of MUs is the set of values of T_r (in *sec*); he/she randomly picks up a T_r value from the set (100,400,1600). A student in a campus cell (7-11) has 90% probability roaming into another campus cell, and 10% probability into a non-campus cell. A student in a non-campus cell roams back to a campus cell. The first type of MUs have 10% probability of roaming from a non-campus cell to a campus cell, and they roam back to a non-campus cell if they are in a campus cell. The downlink bandwidth for cells 1-6 and 12-16 is set to 40 *Kbps* and for cells 7-11, it is set to 80 *Kbps*. The uplink channel is set to 4 *Kbps* for cells 1-6 and 12-16, and 6 *Kbps* for cells 7-11. The most popular data object for students is randomly picked from the ID 0-199, and for the first type of MUs is randomly picked from the ID 500-599. The cache size is set as 1 *Mbytes* for each MU and the ID-only entry is set to 200. Ten types of data objects are set in the database as shown in Table 6.1 except the set for T_u (in *sec*), for which the set is (100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200).

Figures 4.10 and 4.11 show the performance of three extended SACCS schemes, DSACCS and DSACCS-G. From the figures, we know that DSACCS-G has smaller delay (D). DSACCS has better performance than other three schemes but poorer than DSACCS-G. When N reaches 30,000, the D for DSACCS-G is about 5% smaller than DSACCS, and 10% - 30 % smaller than IM-SACCS, CK-SACCS, and HM-SACCS. The

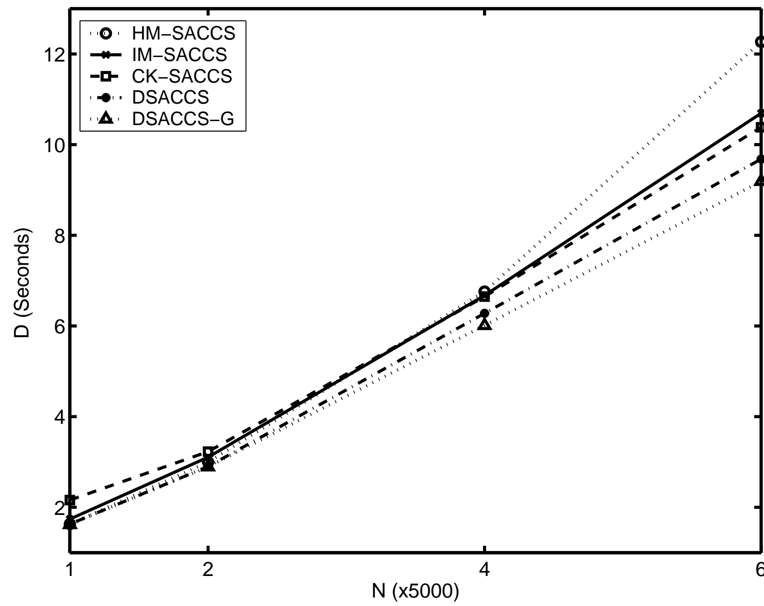


Figure 4.10 Average Access Delay vs Number of Data Objects in the System.

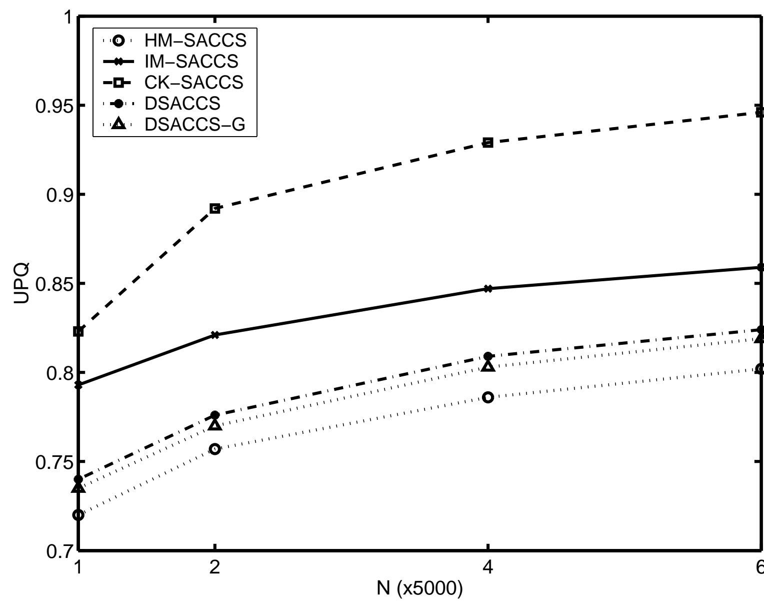


Figure 4.11 Average Uplink Per Query vs Number of Data Objects in the System.

better performance of DSACCS-G is due to the saved extra confirmations of group data objects while the IRs of group data objects retrieved in the grouped cells are only broadcast to themselves, resulting in very few additional IR traffic. For the three extended

schemes, for small N, HM-SACCS has the smallest D and CK-SACCS has the longest one. But for large N, CK-SACCS has the smallest D. HM-SACCS has the smallest UPQ among the five schemes. But UPQ for DSACCS and DSACCS-G is very close to HM-SACCS for the entire range of N. Similar to the result in the previous section, CK-SACCS always has the most UPQ.

The above results show that DSACCS and DSACCS-G inherit the positive features from both homogenous and inhomogeneous strategies, thus minimizing the additional consistency maintenance cost for roaming MUs.

4.5 Summary

In this chapter, we first introduced three types of cache consistency maintenance strategies: homogeneous IR, inhomogeneous IR without roaming check, and inhomogeneous IR with roaming check for multi-cell wireless cellular networks. These strategies are evaluated under various multi-cell environments. Simulation results revealed that no single strategy performs better than the other strategies for all parameter ranges. More specifically, the homogeneous IR strategies perform better for slow updating data objects, fast roaming MUs and small systems; on the other hand, the inhomogeneous IR strategies are more efficient for fast updating data objects, slow roaming MUs and large systems.

To achieve optimized cache performance in multi-cell environments, we derived two functions to determine the cost of maintaining consistency of data objects locally and globally. Based on these two costs, a Dynamic Scalable Cache Consistency Scheme (DSACCS) is proposed for wireless cellular networks. In DSACCS, the IR of a data object is broadcast globally or locally depending on which has the minimum consistency maintenance cost. DSACCS inherits positive features of all the three strategies and hence

outperforms these strategies in the entire parameter ranges. In addition, a variation of DSACCS, called DSACCS-G, is developed for grouping cells in order to facilitate effective cache consistency maintenance in multi-cell systems.

CHAPTER 5

UPDATE PROPAGATION THROUGH REPLICA CHAIN (UPTReC)

Wireless mobile and peer-to-peer (P2P) networks are employed to provide ubiquitous information access, for which data consistency has to be maintained strictly. In the previous two chapters, we developed cache consistency maintenance schemes for information access in wireless mobile networks. In this chapter, we focus on the development of algorithms for file consistent in P2P networks. Recently, P2P networks have been widely used as self-organized distributed systems for information access and sharing. It is also known that the Internet traffic based on P2P applications takes up to 85% of the Internet bandwidth usage [27].

Initially, P2P networks were designed for downloading multimedia files which are assumed to be rather static and where updates occur infrequently. Due to the tremendous growth in P2P applications, the issues related to file consistency maintenance become even more critical. For example, in many applications such as the shared calendar, P2P web cache, online gaming and online auction systems, the files update frequently and hence the consistency needs to be strictly maintained to guarantee consistent file sharing. However, there is no efficient and scalable file consistency maintenance algorithms that can be used in real P2P networks, hence there is a need to design efficient file consistency algorithms for existing P2P networks. Two file consistency algorithms have been developed in this and the next chapters. These algorithms can be effectively used in the existing P2P networks to provide consistent information access and sharing. In this chapter, we develop an *update propagation through replica chain* (UPTReC) algorithm to

provide weak file consistency in P2P networks. We first give the details of the algorithm and then present the performance analysis and comparisons with existing algorithms.

5.1 Description of UPTReC

There are two types of decentralized P2P networks: structured and unstructured. The structured P2P networks enjoy efficient file search through distributed hash tables (DHTs). However, they cannot support keyword search and incur high overheads to maintain very transient peers. In contrast, unstructured P2P systems support keyword searches and also incur low overheads to maintain extremely transient peers, and are robust to general search queries [14]. Based on these observations, we focus on decentralized and unstructured P2P networks.

The main motivation behind Update Propagation Through Replica Chain (UPTReC) algorithm is to minimize the overhead messages for propagating updates of a file to Replica Peers (RPs) which have replicas of the file in decentralized and unstructured P2P systems. The detailed description of the proposed algorithm is given in the following sections.

5.1.1 System Model and Assumptions

We consider a decentralized and unstructured P2P system, such as Gnutella [43] where all peers are equal and no peer has a global view of the system. Each peer frequently joins (online) and leaves (offline) the system and change its IP address with some probability for each reconnection. We assume that any two online peers can communicate with each other if one knows the IP address of the other. The system model and assumptions are summarized as follows:

1. A probabilistically guaranteed file consistency rather than a strong file consistency is required.
2. The write-write conflict is ignored.
3. All peers frequently join and leave the system.
4. An online peer that gets an update has the ability to finish its push process.
5. An online peer can communicate with any other online peer if it knows the IP address of that peer.
6. The physical connectivity and system topology are ignored.
7. Each RP has an ID and an IP address, the ID is fixed but the IP address may be changed for each reconnection.
8. Each file is associated with a version and generation time used for synchronization.

Assumption 2 above is justified due to the lower write-write conflict rate [45] in P2P systems. We make assumption (3) to simplify our algorithm analysis. The probability of online peer to successfully finish its push process is usually over 0.95 [21]. If the probability is low for a system, the assumption (3) above can be remedied by using a reliable push process. In a reliable push process, the push process of RP_1 does not stop after it propagates the update to online RP_2, which in turn forwards the update to other RPs. RP_1 must wait for the confirmation from RP_2 indicating the update has been successfully propagated. If the confirmation is not received within a certain period, RP_1 probes RP_2 again; and if RP_2 is offline, RP_1 contacts with another RP to continue the push process. The reliable push process incurs some additional overhead messages for confirmation.

5.1.2 Push Update Through Replica Chain

Figure 5.1 (a) shows a logical replica chain for a file with N replicas. Each RP is a node (we use RP and node interchangeably) on the chain and has a unique ID associated with it. Each RP maintains ID and IP address information about k, typically O(10), nearest RPs in each (left and right) direction of the chain[1]. These $2k$ RPs are called *probe* RPs. Two RPs are said to have h-hop distance if there are $h-1$ RPs between them. For example, RP_i and RP_{i+k} are k-hop distance apart.

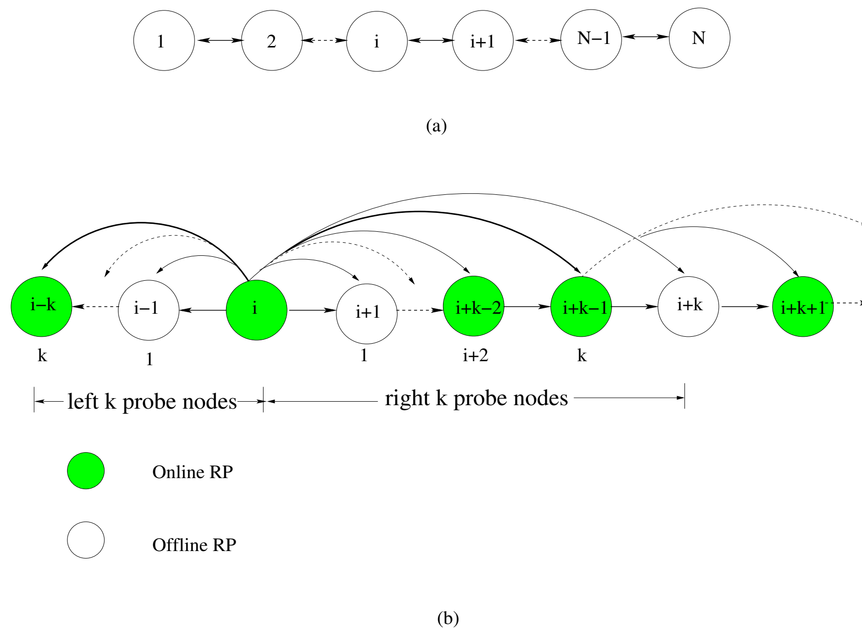


Figure 5.1 (a) Logical Replica Chain; (b) Update Propagation of RP_i.

Figure 5.1 (b) shows the update propagation process of RP_i. When RP_i initiates an update, the update is pushed symmetrically along both left and right directions of the chain. Now let us look at the process of RP_i pushing the update to RP_N (right end of the chain). RP_i has information of k probe RPs to the right (from RP_{i+1}, called the 1st probe RP, to RP_{i+k}, called the kth probe RP). To push an update, RP_i sends a probe

[1]Note that the RPs at or near the head or tail have less than k probe RPs in one direction

message to each of the probe RPs in this direction (i.e., RP_{i+k}, ..., RP_{i+1}). The farthest online probe RP (here RP_{i+k-1}) is chosen to be the update *relay* RP, which will further propagate the update through the chain along the direction. All other online probe RPs of i, such as RP_{i+k-2}, will receive but do not propagate the update. After RP_i determines its update relay RP_{i+k-1}, it first sends the update to that RP with the relay flag bit set as 1 and then sends the update to all other online probe RPs with the relay flag bit set as 0. When an online probe RP receives the update, it first checks the update relay flag bit. If the bit is 0, it only needs to receive the update. Otherwise, it needs to propagate the update through the chain along the direction. The process of the update propagation is similar to RP_i except not to send the probe messages to its probe RPs which are also the probe RPs of i. Because all these RPs are probed by i and they should be offline. As shown in Fig. 5.1(b), when RP_{i+k-1} gets the update, it finds that the update relay flag bit is 1, and hence it immediately sends the probe messages to its probe RPs on the right hand side which are not the probe RPs of i, i.e., RP_{i+2k-1}, ..., RP_{i+k+1}. The update propagation process is repeatedly executed through the replica chain. If all k probe RPs of an update relay RP are offline, the propagation process is stopped and the update cannot be propagated in this direction. The similar process is executed for RP_i to propagate the update to RP_1.

5.1.3 Pull after Online

During the offline period of an RP, it may miss some updates of the file and/or some information on the chain changes. Hence, when an offline RP gets reconnected, it needs to pull some online RPs to synchronize the status of the file and its probe RPs. An RP can probe RPs from nearest to farthest in each direction. Whenever an online RP is probed in one direction, the file and the information of its probe RPs are synchronized. If its IP address is not changed, the pull process in this direction is finished. The similar

process is executed in the other direction. If the IP address of the reconnected RP is changed, it needs to send its ID and new IP address to all its probe RPs. Then the pull process is finished. If no online probe RP can be pulled (due to probe RPs going offline or changing IP addresses), the reconnected RP needs to connect its probe RPs through flooding search to synchronize the status of the file and the information of its probe RPs if its IP address is changed.

5.1.4 Chain Construction and Maintenance

We discuss how to construct and maintain the replica chain in this subsection. After a peer initiates a file in the system, the file can be searched, fetched and replicated by other peers. Each replica is copied from one of the other replicas. If each RP maintains the information of all RPs that fetched a file from it, then a replica tree is naturally constructed. Figure 5.2 (a) shows a replica tree composed of 5 RPs as the root RP at RP_1. If all RPs are always online, any update from any RP can be successfully propagated to any other RPs. For example, when RP_3 initiates an update, it sends the update to RP_1, RP_4 and RP_5. Each RP in turn updates its replica and then relays the update to all its children and parent except the one which sent the update. The update is successfully propagated through all RPs. A new replica tree with RP_3 as the root is shown in Figure 5.2 (b). However, frequently disconnected peers make such a replica tree ineffective in terms of update delivery. In order to increase the probability of successfully propagating the update, each RP must maintain the information of multiple RPs along each path. Due to the properties of the general tree, some RPs may maintain the information of a large number of RPs, while some other RPs maintain information of very few RPs. To balance the overhead associated with the file maintained by each RP, a replica chain can be constructed from the replica tree as explained below.

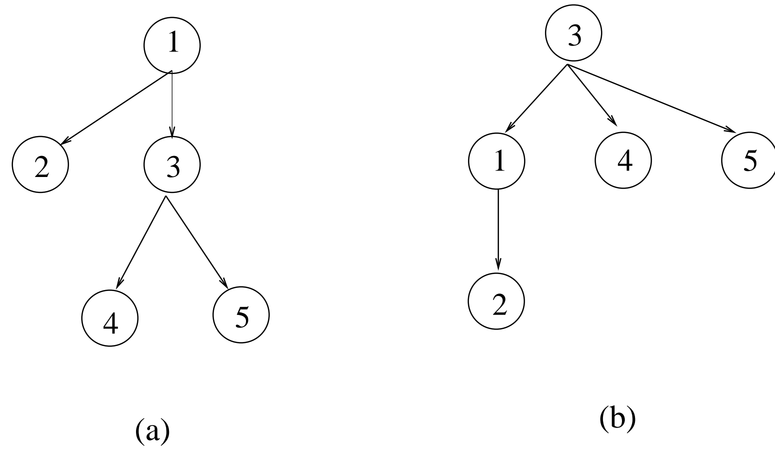


Figure 5.2 Replica Trees: (a) Root at RP_1; (b) New Tree with Root at RP_3.

Figure 5.3 shows the process of constructing a replica chain during the replica process. Figure 5.3 (a) presents the first four RPs which naturally form a chain. In this case, a new RP locates at the head or tail of the chain. When a new peer replicates the file fetched from another RP, the corresponding chain information is also fetched. The information of a new RP is forwarded to all possible RPs which should have the information of the new RP. Figure 5.3 (a) illustrates the process for an RP, such as RP_4 joining the chain. When RP_4 fetches the file from RP_3, the replica chain including information about RP_1 and RP_2 is also fetched. Then RP_3 adds RP_4 into the chain, and pushes information about RP_4 to RP_1 and RP_2. However, if RP_1 for example is offline at that time, it needs to probe either RP_2 or RP_3 to get the latest chain information.

If a new peer joins in the middle of a chain, it needs to push its information to at most k RPs in the chain along the direction opposite to the RP which provides the file. For example, when RP_5 joins the chain by obtaining the chain information from RP_3, RP_5 pushes the information to RP_4.

When RP_i removes a replicated file, it sends a message to each of its probe peers to get removed from the replica chain. All online probe peers get the message and in

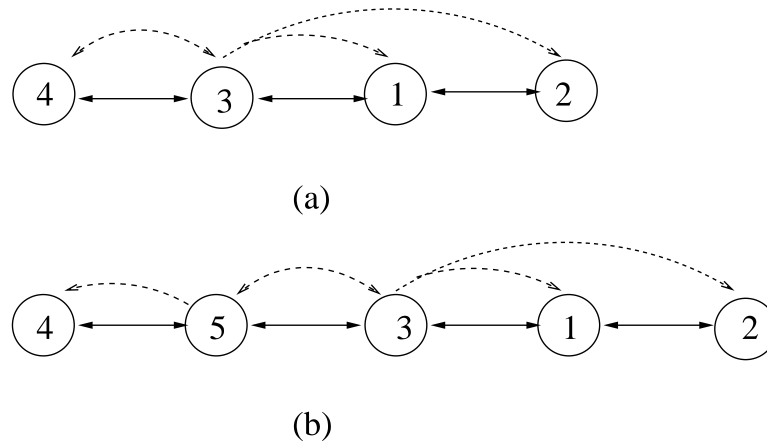


Figure 5.3 Replica Chain Constructing. Join at (a) Head or Tail; (b) Middle .

turn remove RP_i from the chain. All offline probe peers get this message when they reconnect. If all probe peers are offline, RP_i is not removed from the chain and informs the reconnecting peers when they probe. The process of adding or removing a replica requires up to $2k$ messages.

5.2 Performance Analysis

An analytical model is developed in this section. One critical issue concerning the UPTReC algorithm is to determine the value of k. If k is too small, an update may fail to propagate through the chain. If k is too large, the overhead cost of chain maintenance is high.

5.2.1 Analytical Results

Our analytical modeling is based on the assumptions made in Section 5.1.1. Some parameters and performance metrics are defined below:

- N: number of RPs in the chain, i.e., the total number of replicas for a file
- k: number of probe RPs in one direction

- P_{on}: probability of an RP to be online
- P_{off}: probability of an RP to be offline ($P_{off} = 1 - P_{on}$)
- P_{cIP}: probability of an RP to change the IP address after reconnecting
- h: number of hops to an online RP from the update initiating peer
- T: average period of a peer online and offline cycle
- λ: access rate of a file for the whole system
- T_{up}: average file update period
- P_h^s: probability of successfully propagating an update to an online RP with h-hop distance
- $P_h^s(m)$: probability of successfully propagating an update to an online RP with h-hop distance while the online RP only counts the contributions of its m farthest ($1 \leq m \leq k$) probe peers, i.e., the kth, ..., $(k-m+1)$th probe peers
- $P_{pull}^s(k)$: probability of a reconnected RP to successfully pull an online RP
- C_{flood}: average number of messages to find an online probe peer through flooding search
- $C_{push}(N)$: maximum number of messages to push an update through a replica chain with N RPs
- $C_{pull}(k)$: average number of messages in each pull procedure of a reconnected peer
- OHQ: number of overhead messages per query of file consistency maintenance (including overhead of push and pull)
- $P_{stale}(N)$: stale query probability for a file with N replicas.

In UPTReC, the maximum number of messages to push an update is N, because each RP at most receives one probe message. Thus, we have

$$C_{push}(N) \leq N \tag{5.1}$$

When an offline RP rejoins the system, it pulls an online RP from its probe peers in each direction to synchronize the file status and probe peers' information, the pull process in one direction stops whenever an online RP is pulled. If a probe peer is offline or online but with different IP address from its previous IP address that recorded by the reconnected RP, it cannot be pulled. We use $P_{fail} = P_{off} + P_{on}P_{cIP}$ to represent the probability that a probe peer cannot be pulled by a reconnected peer. Then the probability of a reconnected RP to successfully pull an online RP is

$$P^s_{pull}(k) = 1 - (P_{fail})^{2k} \tag{5.2}$$

If the IP address of the reconnected RP is changed, it needs to contact with all probe peers once, hence $2k$ probe messages are needed. If no online probe peer is connected, it needs to search a probe peer through flooding. So the average number of probe messages for each pull process is:

$$\begin{aligned} C_{pull}(k) &= 2(1 - P_{cIP})[(1 - P_{fail})\sum_{i=1}^{k-1} i(P_{fail})^{i-1} + k(P_{fail})^{k-1})] \\ &\quad + P_{cIP}[2k + (1 - P^s_{pull}(k))C_{flood}] \\ &= P_{cIP}[2k + (1 - P^s_{pull}(k))C_{flood}] + 2(1 - P_{cIP})(\frac{1 - P^k_{fail}}{1 - P_{fail}}) \end{aligned} \tag{5.3}$$

In Equation (5.3), the first term is the pull cost when its IP address is not changed, the second term is the pull cost for a reconnected RP with changed IP address. In the second term, if an online probe peer is pulled in a direction, the pull process is stopped in that direction; and if no online probe peer is pulled, all k probe peers are needed to be pulled once. The pull process is symmetrical in both directions.

Based on the definitions, we have $P^s_h = P^s_h(k)$, and $P^s_h(m)$ can be recursively calculated. Figure 5.4 shows the calculation diagram of $P^s_h(m)$. Here RP_h has h-hop distance from the update initiating peer. $P^s_h(m)$ represents the probability for RP_h to

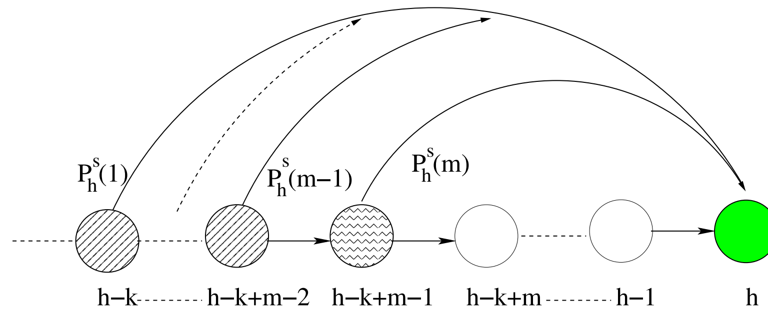


Figure 5.4 Diagram of Calculating $P_h^s(m)$.

get the update if only its farthest m probe peers (i.e., k-th, (k-1)-th, ..., (k-m)-th probe peers) are considered, these probe peers are $h-k$, $h-k+1$, ..., $h-k+m-1$ hops distance from the update initiating peer, we call these RPs as RP_{h-k}, RP_{h-k+1}, ..., and $RP_{h-k+m-1}$ as shown in Figure 5.4. For example, $P_h^s(2)$ is the probability for RP_h to get an update if only probe peers RP_{h-k} and RP_{h-k+1} are considered to push the update to RP_h, and all probe peers RP_{h-k+2}, ..., RP_{h-1} are not considered. All these probabilities can be recursively calculated by the following three equations:

If $h \leq k$ and $1 \leq m \leq k$,

$$P_h^s(m) = 1 \tag{5.4}$$

If $h > k$ and $m = 1$,

$$P_h^s(m) = P_{on} P_{h-k}^s(k) \tag{5.5}$$

If $h > k$ and $1 < m \leq k$,

$$P_h^s(m) = P_h^s(m-1) + P_{on} P_{off}^{m-1} P_{h-k+m-1}^s(k-m+1) \tag{5.6}$$

Equation (5.4) means that if an online RP is a probe peer of the update initiating peer, it will definitely receive the update. Equation (5.5) indicates that RP_i considers its farthest probe peer RP_{h-k}, that is online and successfully receives the update, then RP_h can successfully get the update. Equation (5.6) can be explained by considering the

mth farthest probe peer $RP_{h-k+m-1}$, the probability of successfully receiving the update by RP_h is the probability of successfully receiving the update through its farthest $m-1$ probe peers plus the contribution of the mth farthest probe peer. The mth probe peer has contributions only if all farthest $m-1$ probe peers are offline, because if any of these peer is online, the contribution would be accounted through that peer. In this case, the probability of successfully receiving the update at the mth farthest probe peer is only through its $k-m+1$ probe peers (its first $m-1$ probe peers are offline), i.e., $P^s_{h-k+m-1}(k-m+1)$.

The number of overhead messages per query of file consistency maintenance is:

$$OHQ = \frac{1}{\lambda}\{\frac{C_{push}}{T_{up}} + N\frac{C_{pull}(k)}{T}\} \tag{5.7}$$

For a replica chain with N RPs, the maximum number of hops from an update initiating peer to an online RP is $N-1$. Hence, any online RP has a probability larger than P^s_N to get the update. An offline RP has $P^s_{pull}(k)$ probability to synchronize with an online RP, thus each online RP has at least $P^s_N P^s_{pull}$ probability with a valid file. Then the stale query probability is upper bounded by:

$$P_{stale}(N) \leq 1 - P^s_N(k)P^s_{pull}(k) \tag{5.8}$$

The performance of UPTReC is formulated by equations (6.1) - (6.7). All these measurements are determined by P_{on}, P_{cIP}, k and N.

5.2.2 Numerical Results

Some numerical results are shown in this subsection to characterize typical value of k under some probabilistically guaranteed file consistency. The difference between the

numerical and simulation results (not presented in the here) is within 2% for all these cases.

5.2.2.1 Probability of successfully propagating an update through the chain

We study the impact of the number of probe peers (k) on the probability (P_h^s) of successfully propagating an update to an online RP when $h = 10,000$ hops. The relationship between P_h^s and k is shown in Figure 5.5. When $P_{on} \geq 20\%$, P_h^s is very close to 1 for $k \geq 60$. To achieve P_h^s close to 1, $k = 40$ is enough for $P_{on} = 30\%$ and k is reduced to 20 for $P_{on} = 50\%$. For very small $P_{on} = 10\%$, we get $k = 110$. The results indicate that $k = 60$ ensures a near to 1 probability to propagate an update through a 10,000-node chain for $P_{on} \geq 20\%$. As stated in the previous section, a larger k leads to more overhead messages for the replica chain maintenance. But the overheads per update propagation is independent of k.

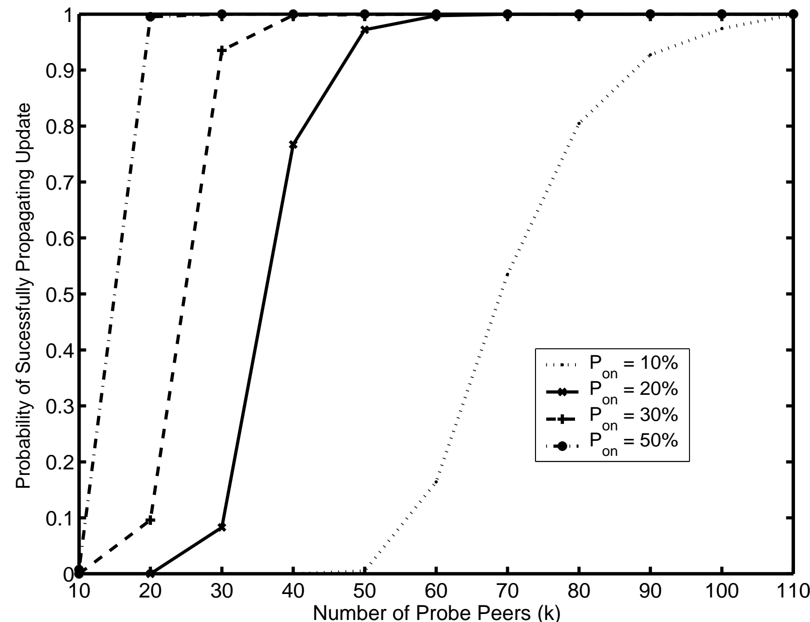


Figure 5.5 Probability of Successful Update Propagation vs Number of Probe Peer .

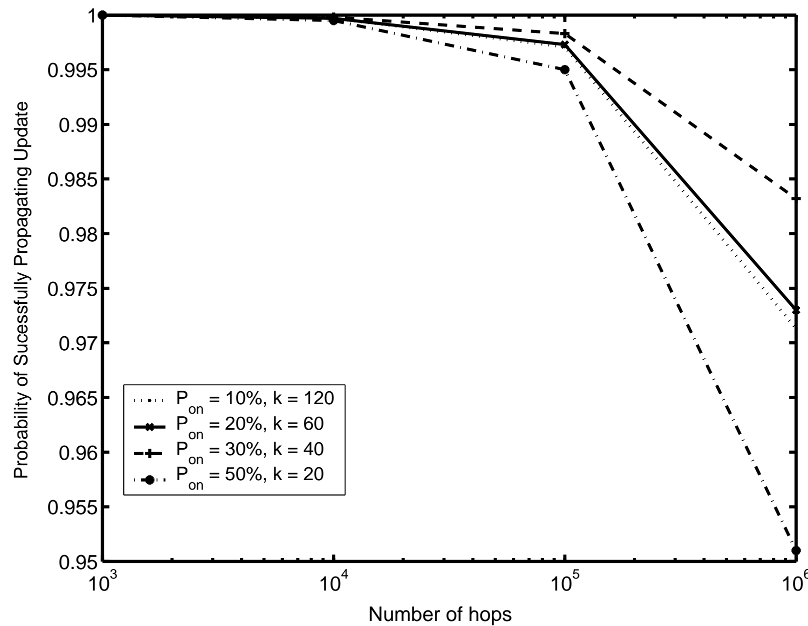


Figure 5.6 Probability of Successful Update Propagation vs Number of Hops .

5.2.2.2 Scalability on the number of replicas

The maximum number of hops of a replica chain increases as the number of RPs increases. In P2P systems, the typical number of replicas for a file varies from tens to thousands. We investigate the scalability of the algorithm on the number of RPs. Figure 5.6 shows the results of P_h^s as h increases from 1,000 to 1,000,000. For a system composed of peers with high online probability ($P_{on} \geq 50\%$), a small number of probe peers $k = 20$ can ensure a larger than 0.95 probability of successful propagation an update to an online RP with 1,000,000-hop distance. For a system with peers having very low online probability, $k = 120$ makes $P_h^s > 0.98$ for $h = 1,000,000$. The probability of successful propagation drops slowly as the number of hops increases. The results indicate that the UPTReC algorithm has good scalability in terms of the number of RPs.

5.3 Performance Comparisons

The performance comparisons between UPTReC and the update propagation scheme based on the rumor spreading algorithm (in short, *Rumor*) proposed in [21] are presented in this section. The overhead messages of file consistency maintenance come from push and pull processes, the major messages of a fast (slow) updating file is from the push (pull) process. We use simulations to study the impact on the performance of update frequency because the analytical performance of update frequency is not presented in [21].

Both algorithms, i.e., UPTReC and Rumor, focus on efficient update propagation to all online RPs. Note that the update propagation is only through the RPs. Moreover, both algorithms are independent of file search and replication. Therefore, we simulate only RPs instead of a whole P2P system to focus on the file consistency maintenance cost. The system topology and physical connectivity are ignored.

In the simulations, we assume each RP alternatively leaves and joins the system as a Poisson process. The file update is also assumed to be a Poisson process. When an update occurs, the initiating RP is randomly chosen from an online RP. In a real P2P system, a file can be searched and replicated by other peers, and an RP may drop a replica. As stated in the previous section, adding a new RP or removing an RP costs $2k$ messages to maintain the chain, but the subset maintenance is not discussed in the Rumor algorithm [21]. Hence, we ignore the comparison on the costs of the chain and subset maintenance in the simulation by assuming a static chain and subsets. Moreover, all RPs are considered to have static IP addresses, because no method is discussed to deal with dynamic IP address in the Rumor algorithm. The chain is randomly built, i.e., each RP has equal probability to appear at any location on the chain. Each RP keeps information of k probe peers in each direction. In the Rumor algorithm, each peer randomly picks up L RPs as its responsible peers. In the 0th push round, the update as

well as a replica list are forwarded to its all responsible peers. The replica list records all RPs in which the update has been sent. In the $t(\geq 1)$ push round, a peer has a probability $P_F(t) = f^t$ to push the update to any responsible peers that are not on the replica list, where f is a constant between 0 and 1. An RP that receives an update is assumed to have ability to finish its push process. The pull process in both algorithms is similar. In UPTReC, when an online probe peer is probed in a direction, the pull process in this direction is finished. In the Rumor algorithm, two online probe peers are probed in each pull process.

Let the file have an access rate λ for the whole system, each access randomly fetches the file from an online RP. When an online RP answers a query, if the file is generated in its newest version, a valid query is counted; otherwise a stale query is counted. Due to focus on the efficiency of file consistency maintenance, the parameters λ, T, and T_{up} are set to unit time.

In our simulation model, for instance, when RP_1 pushes an update to RP_2, it first probes RP_2. If RP_2 is online, the update is forwarded. Thus the total number of updates sent out is equal to the number of online RPs which have received the update. This number is almost equal in both algorithms if the stale query ratio is close to each other. We compare the overhead messages for probing all RPs rather than the number of updates themselves. Of course, the update can be sent out instead of the probe messages. However, if the update is large, this may cause a large amount of extra traffic for sending the update to offline RPs.

5.3.1 Overhead Messages for Each Push Process

The number of overhead messages in the push process and the stale query ratio are studied under various probabilities of online peers. The probability of successfully propagating an update is determined by the probability of a peer being online and the

Table 5.1 Parameter Setup I

N	λ	T	T_{up}
10000	1	10000	10000

number of probe (responsible) peers. Based on the analytical results in the previous section, we set $2kP_{on} = 20$ (or $LP_{on} = 20$) to ensure a low stale hit probability. Thus, $P_{on} = 10\%$ corresponds to $k = 100$ ($L = 200$), and $P_{on} = 50\%$ corresponds to $k = 20$ ($L = 40$). The other parameters are set as in Table 5.1. Based on these setups, there are one query per RP and one update in each RP online and offline cycle (T period) on the average. Two different f values (0.8 and 0.9) are used in the Rumor algorithm to show the relationship between the stale query ratio and the number of overhead messages. The number of overhead messages in the Rumor algorithm is determined by the stale query ratio, a larger f or L makes a lower stale query ratio. We set the L value as $2k$ which is the total number of probe peers kept by an RP in UPTReC. For such L value, high f values are needed to ensure a similar stale query ratio between UPTReC and *Rumor* algorithms, therefore f is set to 0.8 and 0.9.

Figures 5.7 and 5.8 show the number of overhead messages in the push process and the stale query ratio of both algorithms. As shown in these figures, a smaller f reduces the number of overhead messages in the Rumor algorithm, but the stale query ratio is increased. When f drops from 0.9 to 0.8, the number of overhead messages drops about 20%, but the stale query ratio is almost doubled.

The results show that the number of push overhead messages divided by the number of RPs (N) in UPTReC is almost 1, and this value is more than 2.4 in Rumor. The stale query ratio for the UPTReC is less than 1.2% for all ranges of P_{on} from 10% to 50%. But the stale query ratio for the Rumor algorithm increases from 1.2% to 4.8% when P_{on} increases from 10% to 50% with $f = 0.8$. The stale query ratio can be reduced

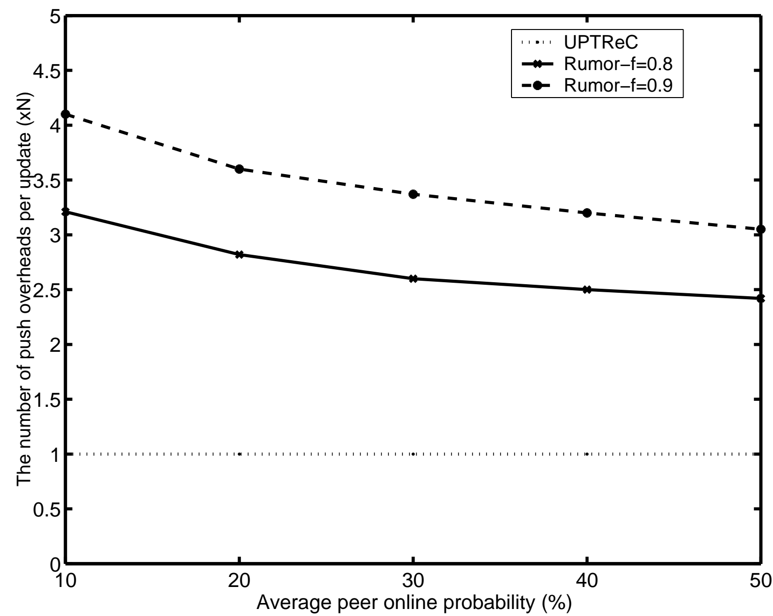


Figure 5.7 Number of Overhead Messages in Push vs Peer Online Probability.

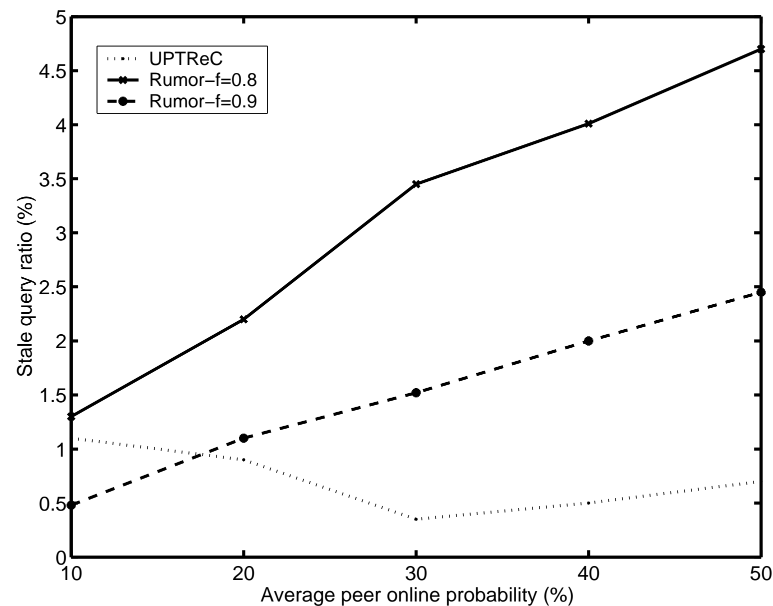


Figure 5.8 Stale Query Ratio vs Peer Online Probability.

to less than 2.5% but incurs more than 20% overhead messages if f is set to 0.9. The results indicate that compared with Rumor, UPTReC reduces more than 60% overhead messages to put an update while achieving a smaller stale query ratio.

5.3.2 Overhead Messages Per Query

The number of overhead messages per query for various update frequencies is investigated in this experiment. We measure two performance metrics: (1) the number of overhead messages per query, and (2) stale query ratio. The number of overhead messages per query is defined as the total number of consistency maintenance messages which include overhead messages of the push and pull processes divided by the total number of queries in the system. We set two L values (80 and 100) for the Rumor algorithm in the simulation to study the effects of L. The other system parameters are specified as in Table 5.2.

Table 5.2 Parameter Setup II

N	λ	P_{on}	T	k	f
10000	1	30%	10000	40	0.9

Figures 5.9 and 5.10 show the results of the number of overhead messages per query and stale query ratio versus different update frequencies. When the average update period ($T_{up} = 10^5$) is much larger than the peer online and offline cycle, the overhead messages of the pull process are the major source. Due to the similar pull process, the number of overhead messages per query for two algorithms is close in this case. As the update period decreases, the number of overhead messages from the push process increases and dominates the number of overhead messages from the pull process. This leads to a better performance of UPTReC than that of the Rumor algorithm. When the update period is much shorter than the peer online and offline cycle, the number of overhead messages per query in UPTReC is more than 70% lower than that of the Rumor. The stale query ratio in UPTReC is less than 0.1% in all range of update periods. In the Rumor algorithm, when the update frequency is high, the stale query ratio is about 2% for $L = 80$, and

it is reduced to less than 1% when $L = 100$. The effect of L is similar to f. A larger L or f gives a lower stale query ratio but costs more overhead messages. The results show that the UPTReC can save up to 70% overhead messages while providing better probabilistically consistency guarantee for highly update files compared to the Rumor algorithm.

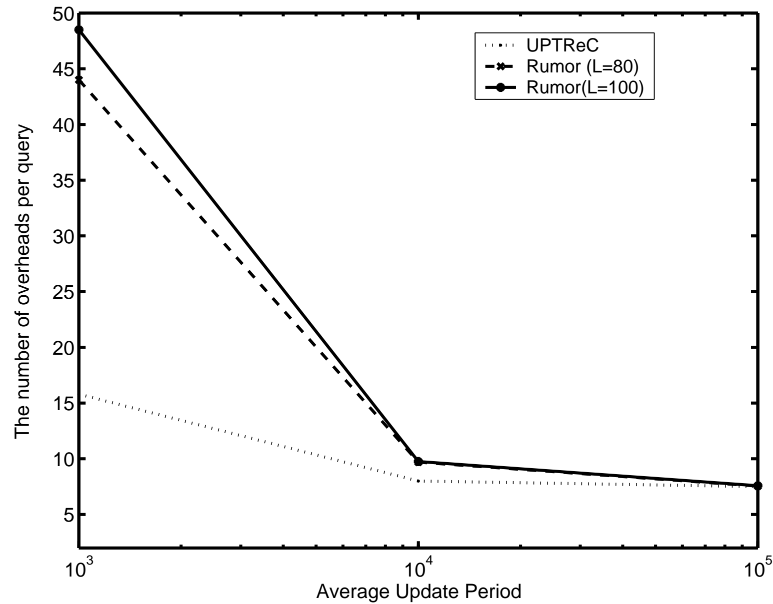


Figure 5.9 Number of Overhead Messages Per Query vs Update Period.

Through these comparisons, we know that the UPTReC algorithm can significantly reduce overhead messages to propagate an update with a smaller stale hit ratio compared with the Rumor algorithm.

5.4 Summary

In this chapter, we propose a novel algorithm, UPTReC, to propagate update through replica chain for decentralized and unstructured P2P systems. In UPTReC, each file has a logical replica chain composed of all RPs. Each RP has a partial knowledge

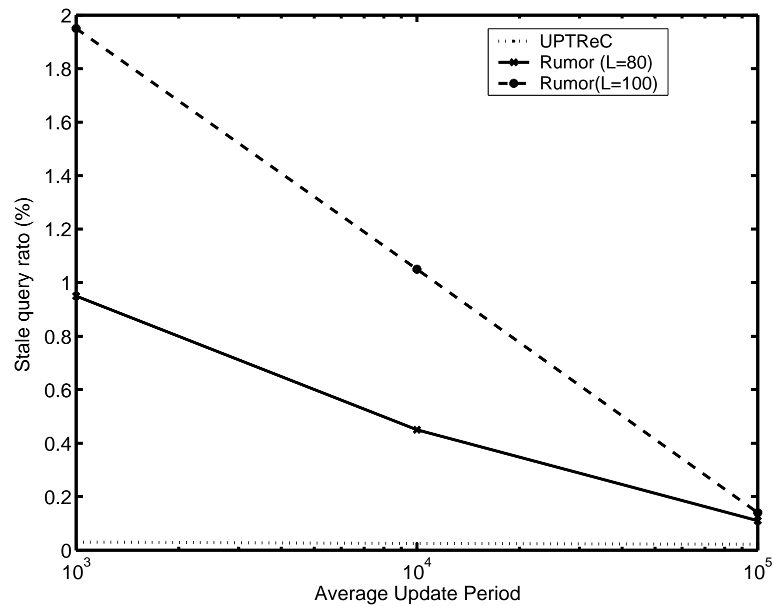


Figure 5.10 Stale Query Ratio vs Average Update Period.

of the chain. When an RP updates the file, it pushes the update to all possible online RPs through the replica chain. When an offline RP gets reconnected, the file status is synchronized by pulling an online RP.

An analytical model of the proposed algorithm is derived. The performance comparison of UPTReC with the Rumor algorithm show that UPTReC reduces up to 70% overhead messages to propagate updates with a smaller query ratio for highly updated files.

UPTReC provides probabilistically guaranteed file consistency, and can be used in some systems such as P2P web cache and bulletin board systems. In these systems, a file access with small stale probability is acceptable. However, in some applications, such as online auction and online game systems, the file consistency needs to be strictly maintained, thus requiring a strong file consistency algorithm.

CHAPTER 6

FILE CONSISTENCY MAINTENANCE THROUGH VIRTUAL SERVERS

The UPTReC algorithm in the last chapter provides weak file consistency. With the growth of applications, the file consistency needs to be strictly maintained, such as in online games and auction systems. However, to the best of our knowledge, there is no strong file consistency algorithm designed for decentralized and unstructured P2P networks. This motivates us to develop a strong file consistency maintenance scheme for such networks to ensure strictly information access/sharing. To meet this goal, we develop a file **C**onsistency **M**aintenance through **V**irtual servers (CMV) algorithm for decentralized and unstructured P2P networks. In this chapter, we first present the algorithm details, then analyze the algorithm performance. Finally we give the numerical performance results under various system parameters.

6.1 Proposed CMV Algorithm

The proposed CMV algorithm maintains one-copy serializability of the file in decentralized and unstructured P2P systems, such as Gnutella, in which no peer has a global view of the system. The details of CMV are described in the following sections.

6.1.1 Logical Structure of RPs

In the CMV algorithm, there are three types of RPs: (1) virtual RPs (or VRP) executing the role of the virtual server (or VS); (2) HRPs using push-based file consistency maintenance scheme; and (3) LRPs using pull-based file consistency maintenance scheme.

Figure 6.1 shows a logical structure of these RPs, where 1-6 are VRPs, 7-9 are HRPs, and 10-15 are LRPs.

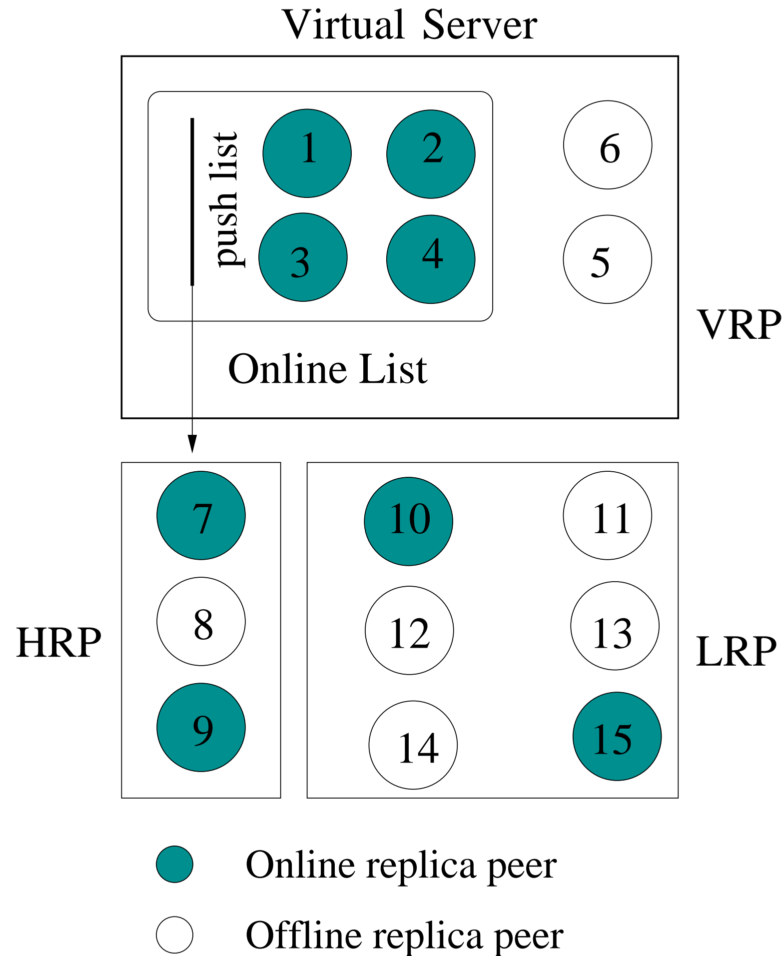


Figure 6.1 Logical Structure of RPs in the CMV Algorithm.

Each RP has the ID and IP address information of all VRPs used for pulling the VS. The VRPs are selected from the highly trusted [20] [32] [47] [65] and reliable RPs and can be dynamically changed. Each online VRP maintains a list of other online VRPs with a view to use this information for reducing communication costs associated with file consistency maintenance. The information of all online HRPs is recorded in a

dynamically maintained list by the VRPs. An LRP (or HRP) can join (or leave) the push list when the cost of file consistency maintenance after the join (or leave) event is less than the current cost.

6.1.2 Algorithm Description

The execution of the CMV algorithm in VRPs, HRPs and LRPs are described by the procedures in Figures 6.2, 6.3 and 6.4, respectively.

6.1.2.1 The VRP Procedure

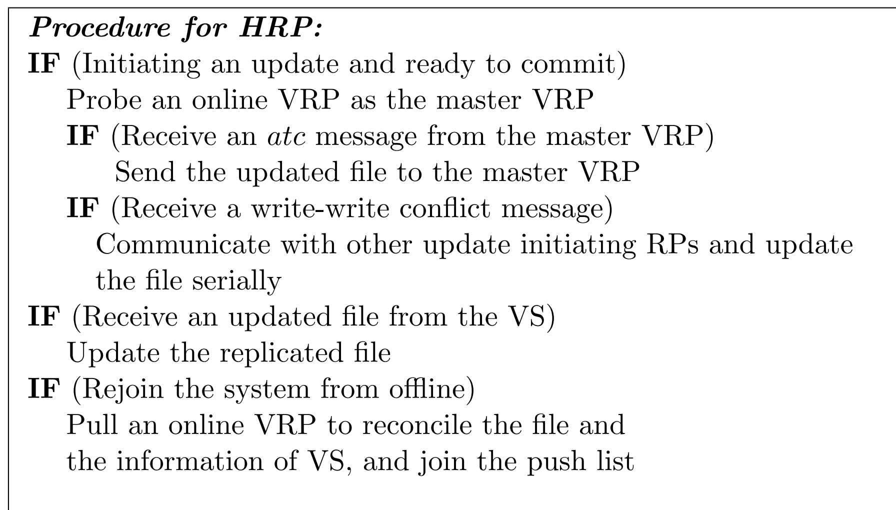
When a VRP initiates an update or is selected as the master VRP (i.e., in charge of committing the update to the VS) by an update initiating RP, the VRP first checks whether the file is being updated by another RP or not. If so, the VRP cannot commit the update until the last update is finished. Then the VRP sends a ready-to-commit message to each online VRP. If the VRP gets an agree-to-commit message back from every other online VRP, the update is ready to be committed. if the file is not updated by itself, the VRP first gets the updated file from the update initiating RP. Then the updated file is pushed to all online VRPs. Finally, the VRP probes all HRPs. The VRP pushes the updated file to each online HRP. Otherwise, the HRP is removed from the push list. After the update is successfully committed, an update completed message is sent back to the update initiating RP, and if the push list is changed, a message is sent to each online VRP to update the push list. If a write-write conflict message is received from an online VRP, the conflict information is sent back to the update initiating RP. All the conflicting updates are serially done based on the update generation time through all update initiating RPs, after which the last updated file is committed to the VS.

```
Procedure for VRP:
fg = 0: flag bit for detecting write-write conflict
/*0: no conflict and 1: existing conflict */
rtc: ready-to-commit            atc: agree-to-commit
IF (Initiating an update and ready to commit
      OR selected as the master VRP by an RP)
    IF (fg == 1)   /* the file is being updated by another RP */
        Wait till the update finished
    Set fg = 1 and send an rtc message to each online VRP
    IF (Receive an atc message from every online VRP)
       Push the updated file to all online VRPs
       Set fg = 0
       FOR (Each HRP)
          IF (It is online )
              Send the updated file
          ELSE
              Remove the HRP from the push list
       IF (The push list is changed)
          Send a message to each online VRP to update the push list
    ELSE  /* resolve the write-write conflict */
        IF (The update is from the other RP)
            Send the conflict information to that RP
       ELSE
          The file is updated serially through communication with other
          update initiating RPs
IF (Get an rtc message)
    IF (fg == 0)
        Record the information of the master VRP
        Send an atc message back and set fg = 1
    ELSE   /* find a write-write conflict */
       Send a write-write conflict message back
IF (Get an updated file from a master VRP)
    Update the replicated file and set fg = 0
IF (Rejoin the system from offline)
    Pull an online VRP to join the VS
```

Figure 6.2 Procedure for VRP.

When a VRP gets a ready-to-commit message, it first checks the write-write conflict bit. If no conflict is detected, an agree-to-commit message is sent back. Otherwise, a write-write conflict message including the conflict information is sent back. When the

VRP receives an updated file from the master VRP, it updates the replicated file and resets the write-write conflict bit. If the VRP is rejoining the system, it needs to pull an online VRP to reconcile the file, the push list and the online VRP list. If any VRP in the online list is offline, the VRP sends the corresponding information to all other online VRPs to update the online list.



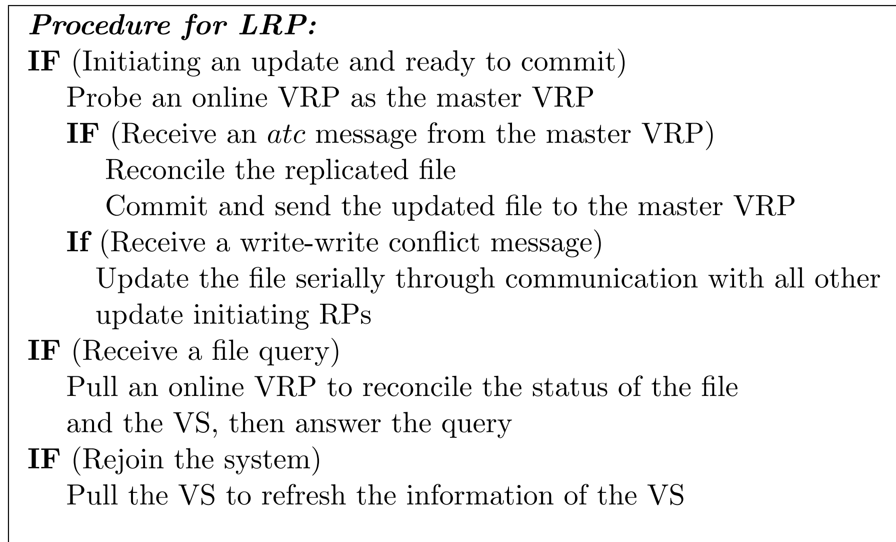
Procedure for HRP:
IF (Initiating an update and ready to commit)
  Probe an online VRP as the master VRP
  IF (Receive an *atc* message from the master VRP)
    Send the updated file to the master VRP
  IF (Receive a write-write conflict message)
    Communicate with other update initiating RPs and update the file serially
IF (Receive an updated file from the VS)
  Update the replicated file
IF (Rejoin the system from offline)
  Pull an online VRP to reconcile the file and the information of VS, and join the push list

Figure 6.3 Procedure for HRP.

6.1.2.2 The HRP Procedure

When an HRP is ready to commit an update, it first probes an online VRP as the master VRP for the update. If an agree-to-commit message is received from the master VRP, the HRP commits the update and sends the updated file to the master VRP. If a write-write conflict is detected, the HRP contacts all other update initiating RPs. The updates are done serially based on the update generation time. When the HRP receives an updated file from the VS, the replicated file is updated. When the HRP rejoins the

system, it sends a message to the VS to join the push list and reconciles the status of the file and the information of the VS.



Procedure for LRP:

IF (Initiating an update and ready to commit)
  Probe an online VRP as the master VRP
  IF (Receive an *atc* message from the master VRP)
    Reconcile the replicated file
    Commit and send the updated file to the master VRP
  If (Receive a write-write conflict message)
    Update the file serially through communication with all other update initiating RPs
IF (Receive a file query)
  Pull an online VRP to reconcile the status of the file and the VS, then answer the query
IF (Rejoin the system)
  Pull the VS to refresh the information of the VS

Figure 6.4 Procedure for LRP.

6.1.2.3 The LRP Procedure

When an LRP is ready to commit an update, it first probes an online VRP as the master VRP for the update. If an agree-to-commit message is received from the master VRP, the LRP first reconciles the file. The rest update committing process is the same as that of an HRP. When an LRP gets a file query, it answers the query after checking with the VS. If the file is updated, the updated file is retrieved. For each reconnection, the LRP pulls an online VRP to refresh the information of the VS.

6.1.2.4 RP Pull Procedure

When an RP comes back online, it needs to pull the VS to reconcile the information of the VS and the status of the file. The backup RP first probes all possible VRPs one by one through the recorded information. If an online VRP is found, the corresponding information is synchronized and the pull process is finished. If no online VRP can be probed, the RP must start a flooding search because there may exist some online VRPs whose information is not maintained by the RP or whose IP addresses have changed.

6.1.3 File Consistency Maintenance

The CMV algorithm uses a lazy-master copy replication model to maintain one-copy serializability of the file. All updates are first accepted by the VS, and then pushed to all possible online HRPs. Hence, the replicas in all online VRPs and HRPs are synchronized. Any file query from these RPs is valid. An LRP may have a stale replica, but any file query from it must be reconciled from the VS and thus guarantees the file consistency. If a VRP rejoins the system, it needs to reconcile the replicated file from another online VRP. This ensures the replicated file in a backup VRP to be consistent. A backup HRP also needs to reconcile the replicated file and then joins into the push list.

The above description assumes the VS is always on the system. If no VRP is online, the VS fails. No update can be committed until the VS is recovered. The VS recovery process is described next.

6.1.4 Failure Handling

First we discuss the master VRP recovery. If the master VRP goes offline during the update process, the update may not be successfully committed. Hence, a time threshold is set. If the update initiating RP has not received an update completed message from the master VRP within the threshold time, it probes the master VRP again. If the master VRP is still online, it sets another threshold time waiting for the update completed message. Otherwise, it immediately probes another online VRP to commit the update. This procedure is repeatedly executed until the update is successfully committed.

Next let us explain how to handle virtual server recovery. If all VRPs are offline, the virtual server goes into the $fail$ state. In this case, no update can be committed and no file access from LRPs can be accepted. However, the file accesses from online HRPs[1] are accepted because an HRP has the newest version of the file, and no new update can be accepted when the VS is in the fail state.

The VS comes back when any VRP rejoins the system. The rejoining VRP can detect the VS in the fail state because it cannot find another online VRP. If two VRPs come back online at the same time, they can probe each other. But neither of them has valid information of the push list and online VRP list, thus they can detect the VS in the fail state. A VS recovery message including the IP address and the file version of the backup VRP is broadcast to the system. Each HRP must send a message to the VRP to join the push list after it gets the VS recovery message. If an HRP has a more recent version of the file, the VRP reconciles the file from that HRP. An LRP, which has a more recent version of the file than that of the backup VRP, needs to contact the backup VRP. The replicated file of the VRP is synchronized with the most recent version among all

[1] A backup HRP that fails to probe the VS is considered as an LRP, and cannot accept file access.

online HRPs and LRPs. If all RPs with the most recent version of the file are offline, the most recent update cannot be viewed in the system and considered to be aborted. When an RP with the most recent updated file comes online, it needs to redo the update and commit the update into the system again.

The recovery cost can be very high if the VS enters the fail state even for a short period of time. Thus, a high availability of the VS is critical to the performance of the CMV algorithm. However, a VS with high availability needs more VRPs and incurs additional maintenance costs. The performance analysis is carried out in the next section with an objective to determine the parameters for optimal performance of the CMV algorithm.

6.1.5 Virtual Server Construction and Maintenance

The number of VRPs must be large enough to provide a highly available VS. When a file is initiated by a peer, a certain number of peers (analyzed in the next section) must be selected as the VRPs to form a VS. These VRPs can be chosen by applying replication rules [18]. An RP with high trust value has more chance to be selected. Any selected peer can accept or reject to be a VRP depending on its traffic load, local buffer space, etc.

After the VS of a file is constructed, the file can be searched and replicated by other peers. If a more reliable peer replicates the file, that peer should be selected to be a VRP to enhance the VS availability. On the other hand, the replicated file in a VRP may be replaced by other more useful files, and hence the RP should leave from the VS. Thus the VS should be dynamically managed.

When a VRP decides to leave the VRP list, it selects an online HRP with high trust value as the new VRP. Then the information of this exchange is sent to all online VRPs which in turn update the corresponding information. All offline VRPs miss the information but will be informed when they synchronize the VS information from an online VRP. The next updated file associated with the newest information of the VS will be pushed to all HRPs. An LRP gets this information when it connects to the VS. The VRP with changed IP address has the same effect, the changed IP address must be informed to all RPs so that they can update the corresponding information.

A VRP may permanently leave the system without notice. Such VRPs should be removed from the VS to enhance the VS availability. To handle this case, the last appearing time for each VRP is recorded by all VRPs (managed in the same manner as its IP address). If a VRP does not appear in the system over a period time (e.g., one day), it is removed from the VS and replaced by a new VRP.

If the VRPs and/or their IP addresses are changed too fast, an RP may fail to probe an online VRP through the maintained information of the VS. If so, a flooding search is needed to probe an online VRP. This significantly increases the number of overhead messages for file consistency maintenance. The members of the VS should be slowly changed so that an RP has very high probability to successfully probe an online VRP using the maintained information.

6.2 Performance Modeling

One critical concern of the CMV algorithm is the number of VRPs. If the number is too small, the VS may go down, thus resulting in a significant VS recovery cost. If the number is too large, the costs of committing updates and maintaining the VS are high.

An analytical model is derived in this section to determine the optimal number of VRPs under various system conditions.

6.2.1 Parameters and Notations

In our analytical model, all RPs are assumed to be independent. The file update and access, as well as the RP process for going online and offline are assumed to follow Poisson distribution. The file access and update are only from the online RPs. The parameters and notations are defined below:

- N_v: number of VRPs
- N_h: number of HRPs
- N_l: number of LRPs
- N: total number of RPs, i.e., $N = N_v + N_h + N_l$
- P_i^{on}: online probability of RP_i
- P_i^{off}: offline probability of RP_i (i.e., $P_i^{off} = 1 - P_i^{on}$)
- P_i^{cIP}: probability of RP_i changing its IP address at each reconnection
- P^{vf}: probability of the VS in fail state
- P_i^p: probability of RP_i successfully pulling an online VRP using the maintained information
- T_i^c: on-off cycle time for RP_i
- T^{vf}: average time of a VS in fail state
- α_i: file update arrival rate from RP_i during its online period
- α: file update arrival rate of the system ($\alpha = \sum_{i=1}^{N} \alpha_i P_i^{on}$)
- λ_i: file access arrival rate of RP_i during its online period
- λ: total file access rate in the system ($\lambda = \sum_{i=1}^{N} \lambda_i P_i^{on}$)

- β_i: arrival rate of RP_i coming online from offline ($\beta_i = 1/P_i^{off}T_i^c$)
- C^{fld}: cost of search of an online VRP using flooding method
- C^{brd}: cost of broadcasting a message in the system
- C_i^p: cost of each pulling process by RP_i
- C^r: cost of the VS recovery
- H_i^v: number of overhead messages per unit time of VRP_i maintaining the file
- H_i^h: number of overhead messages per unit time of HRP_i maintaining the file
- H_i^l: number of overhead messages per unit time of LRP_i maintaining the file
- H: total number of overhead messages per unit time in the system for maintaining the file ($H = \sum_{i=1}^{N_v} H_i^v + \sum_{i=N_v+1}^{N_v+N_h} H_i^h + \sum_{i=N_v+N_h+1}^{N} H_i^l$)
- F_i^{vh}: file retrieval rate of RP_i as a VRP or an HRP
- F_i^l: file retrieval rate of LRP_i
- F: total file retrieval rate in the system ($F = \sum_{i=1}^{N_v+N_h} F_i^{vh} + \sum_{i=N_v+N_h+1}^{N} F_i^l$)

6.2.2 Performance Analysis

In our model, there are a total of N RPs in the system. Among them, N_v are VRPs, N_h are HRPs and are N_l LRPs. Without loss of generality, if $i \leq N_v$, we assume the RP_i is a VRP; if $N_v < i \leq N_v+N_h$, it is an HRP; otherwise, it is an LRP. The on-off cycle time, T_i^c, of RP_i is defined as the interval between its two successive rejoining time. Because the effects of a VRP leaving the VRP list are the same as those due to IP address change, we only consider the probability of its IP address change.

If all VRPs are offline, the VS is down, and therefore the probability of the VS in fail state is given by:

$$P^{vf} = \prod_{i=1}^{N_v} P_i^{off} \tag{6.1}$$

If the VS is in fail state, it recovers when any VRP comes online. The average time of the VS in fail state is:

$$T^{vf} = \int_0^{\infty} t e^{-\sum_{i=1}^{N_v} \beta_i t} dt = \frac{1}{\sum_{i=1}^{N_v} \beta_i} \tag{6.2}$$

The probability of successfully pulling an online VRP is critical to the performance of the CMV algorithm. Each RP maintains the information of all VRPs, the RP can randomly pull these VRPs one by one, and the process continues until an online VRP is pulled. For a VRP, it cannot be successfully pulled by the RP if one of the following two cases occurs: (1) the VRP is offline; or (2) the VRP is online but with an IP address different from that recorded by the RP. Moreover, RP_i pulls the VS at least once in each of its on-off cycle to get the newest VS information.

Let n_{ij} denote the number of on-off cycles of VRP_j between two successive pull times of RP_i. Then

$$n_{ij} = T_i^c / T_j^c \tag{6.3}$$

In each on-off cycle, VRP_j changes its IP address with probability P_j^{cIP}. Then the probability of RP_i successfully pulling an online VRP through recorded information is given by,

$$P_i^p = 1 - \prod_{j=1, j \neq i}^{N_v} P_{ij}^u \tag{6.4}$$

where P_{ij}^{u} is the probability of RP_i for unsuccessfully pulling VRP_j as an online VRP. It is given by:

$$P_{ij}^{u} = P_j^{off} + P_j^{on}[1 - (1 - P_j^{cIP})^{n_{ij}}] \tag{6.5}$$

If no online VS can be pulled through the recorded information, an RP will try to pull one through flooding search which costs C^{fld} messages. Thus the cost of each pull process including these two steps is computed as:

$$C_i^p = \sum_{j=1}^{N_v-1} j(1 - P_{ij}^{u}) \prod_{k=1}^{j-1} P_{ik}^{u} + N_v \prod_{k=1}^{N_v-1} P_{ik}^{u} + (1 - P_i^p)C^{fld} \tag{6.6}$$

If a VRP comes online during the period the VS is in fail state, the VS is recovered. Also a VS-recovery message including the VRP's information as well as the file version are broadcast to the system. The online HRPs as well as LRPs that have more recent version of the file need to contact the VRP. Then the VS recovery cost is upper bounded by:

$$C^r = C^{brd} + \sum_{i=N_v+1}^{N} P_i^{on} \tag{6.7}$$

Now we calculate the number of overhead messages for file consistency maintenance. A back up VRP needs to pull another online VRP to rejoin the VS. If an online VRP is found, the rejoining VRP retrieves the online VRP list and the push list, reconciles the file status and the information of the VS. Then a message is sent to each online VRP for joining the online list. If some VRPs on the online list have gone offline, an extra message is sent to inform all other online VRPs to remove these offline VRPs from the list. In this case, two messages are needed for each online RP.

If the VS is in fail state, the VRP broadcasts a VS-recovery message in the system to recover the VS at a cost of C^r. For each file update, each online VRP receives a read-to-commit message and sends an agree-to-commit message back. Moreover, if the push list is changed, another message is incurred to update the push list. In summary, each online VRP incurs up to 3 messages for each update commitment. When the updates are in conflict, extra communication messages are needed to resolve the conflict through all conflicting RPs. However, only the final updated file is pushed to the VRPs, thus reducing the number of messages to transmit the updated file. Based on this observation, we ignore the write-write conflict cost in our analysis. Then, H_i^v is obtained as:

$$H_i^v = \frac{1}{T_i^c}[C_i^p + 2(1 - P^{vf}) \sum_{j=1}^{N_v} P_j^{on} + P^{vf} C^r] + 3\alpha P_i^{on} \tag{6.8}$$

An HRP pulls the VS when it rejoins the system or updates the file. For each rejoining, a message is sent to each online VRP to update the push list. Hence, H_i^h is given by:

$$H_i^h = \alpha_i P_i^{on} C_i^p + \frac{1}{T_i^c}(C_i^p + \sum_{j=1}^{N_v} P_j^{on}) \tag{6.9}$$

For each file access or update from an LRP, the LRP must pull an online VRP to reconcile the file status or commit the update. For each reconnection, the LRP also needs to pull the VS to get the newest VS information. Then H_i^l is given by:

$$H_i^l = [(\alpha_i + \lambda_i) P_i^{on} + \frac{1}{T_i^c}] C_i^p \tag{6.10}$$

The total number of message for file consistency maintenance per unit time in the system is:

$$H = \sum_{i=1}^{N_v} H_i^v + \sum_{i=N_v+1}^{N_v+N_h} H_i^h + \sum_{i=N_v+N_h+1}^{N} H_i^l \tag{6.11}$$

If RP_i is a VRP or HRP, and also online, it receives the updated file for each update. When RP_i rejoins in the system, the replicated file is retrieved if the file was updated during its offline time. The file retrieval rate for RP_i is:

$$F_i^{vh} = \alpha P_i^{on} + \frac{1}{T_i^c}(1 - e^{-\alpha P_i^{off} T_i^c}) \tag{6.12}$$

The file retrieval rate of LRP_i is the minimum of the file access rate and file update rate. Note that if the file is updated when LRP_i is offline, the next access needs to retrieve the file, thus resulting in additional update rate of $(1 - e^{-\alpha P_i^{off} T_i^c})/T_i^c$. Then the file retrieval rate of LRP_i is given by:

$$F_i^l = \min\{\lambda_i P_i^{on}, \alpha P_i^{on} + \frac{1}{T_i^c}(1 - e^{-\alpha P_i^{off} T_i^c})\} \tag{6.13}$$

The total file retrieving rate in the system is

$$F = \sum_{i=1}^{N_v+N_h} F_i^{vh} + \sum_{i=N_v+N_h+1}^{N} F_i^l \tag{6.14}$$

Equations (6.1) - (6.14) represent the file maintenance cost of RPs under various system conditions. The file maintenance cost of an RP is determined by the file access and update rate, on-off cycle and the online probability of the RP, the file update rate in the whole system and the probability of the VS in fail state. If the file size equals s overhead messages, the total communication costs per unit time including both overhead messages and the updated files for an HRP or LRP are given by M_i^h and M_i^l respectively, as shown below:

$$M_i^h = H_i^h + sF_i^{vh} \tag{6.15}$$

$$M_i^l = H_i^l + sF_i^l \tag{6.16}$$

The file maintenance cost for RP_i as an LRP or an HRP is dependent on M_i^h and M_i^l. Considering only the file maintenance cost[2], RP_i chooses to be an LRP when $M_i^h > M_i^l$, and an HRP when $M_i^h \leq M_i^l$. The file update rate in the whole system and the probability of the VS in fail state can be measured by the VS. When a peer first fetches a file, it retrieves the corresponding parameters from the VS and decides to be an HRP or LRP depending on the computed file maintenance cost. The file maintenance cost can be periodically calculated based on the most recent parameters of the system.

From Equations (6.12) - (6.14), we know that the number of updated files retrieved by an RP is the same whether it is a VRP or an HRP. In other words, the number of updated files retrieved does not depend on the number of VRPs. Thus the value of N_v is determined by minimizing the number of overhead messages in the system, i.e., $\partial H / \partial N_v = 0$. To derive an explicit expression for N_v, we assume that there are three types of RPs, each type having the same parameters. For example, the first type is VRP, each with the same P_v^{on}, P_v^{cIP}, T_v^c, β_v, α_v and λ_v. The second type is HRP, the corresponding parameters being specified with the subscript h, such as P_h^{on} and so on. The third type is LRP, the parameters of which are indicated with the subscript l. Based on these, N_v is determined by the following equations,

[2]The other factors, such as file access delay, may be considered more important for an RP to be an HRP or LRP in real systems. We do not discuss these issues because our focus is on the file maintenance cost.

$$H_v^v + N_v(\frac{\partial H_v^v}{\partial N_v}) + N_h(\frac{\partial H_h^h}{\partial N_v}) + N_l(\frac{\partial H_l^l}{\partial N_v}) = 0 \tag{6.17}$$

where

$$\frac{\partial H_v^v}{\partial N_v} = \frac{1}{T_v^c}[2P_v^{on}(1 - P^{vf}(1 + N_v \ln(P_v^{off}))) +$$
$$C^r P^{vf} \ln(P_v^{off}) + (P_v^u)^{N_v} \ln(P_v^u)(\frac{-1}{1 - P_v^u} + \frac{C^{fld}}{P_v^u})] \tag{6.18}$$

$$\frac{\partial H_h^h}{\partial N_v} = (\alpha_h P_h^{on} + \frac{1}{T_h^c})(P_h^u)^{N_v} \ln(P_h^u) \times (\frac{-1}{1 - P_h^u} + C^{fld}) + \frac{P_v^{on}}{T_i^c} \tag{6.19}$$

$$\frac{\partial H_l^l}{\partial N_v} = ((\alpha_l + \lambda_l)P_l^{on} + \frac{1}{T_l^c}) \times (P_l^u)^{N_v} \ln(P_l^u)(\frac{-1}{1 - P_l^u} + C^{fld}) \tag{6.20}$$

where ”ln” stands for natural logarithm.

6.3 Numerical Results

The numerical results are presented in this section to quantify the performance of the CMV algorithm. The characteristics of RPs in the Gnutella system are measured in [6] and [55]. The results in [55] show that about 60% peers have 0.2 or less online probability, and 10% peers have more than 0.8 online probability; Moreover, about 50% peers have 60 minutes or less and 10% peers have 300 minutes or more in each online session. The IP address of a peer may be changed for each rejoining; the results in [6] indicate that about 40% peers changed their IP address in one day and about 50% in

seven days. The parameters of RPs in the numerical results are selected based on these measurements.

The optimal number of VRPs (N_v) varies with different system conditions, we assume all updates are generated by HRPs and LRPs to maintain a constant file update rate (i.e., $\alpha = N_h \alpha_h P_h^{on} + N_l \alpha_l P_l^{on} =$ constant) in the whole system. We set $C^{brd} = 1,000,000$ and $C^{fld} = 10,000$ messages. A flooding search costs fewer messages because multiple VRPs may be online with changed IP address. Moreover, an online HRP may be searched and hence a more recent information of the VS can be obtained, thus resulting in reduced search cost. The performance impacts of file update arrival rate (α), VRP online probability (P_v^{on}), the probability of VRP changing its IP address for each rejoining (P_v^{cIP}) and number of RPs (i.e., $N_h + N_l$) are studied in the following.

6.3.1 Impact of File Update Rate

In this case, we assume that all RPs except VRPs are the same. The total number of messages per query (MPQ) including both overhead and file retrieval messages are calculated by assuming that all these RPs are HRPs or LRPs, respectively. The MPQ for a system is defined as $M = (\sum_{i=1}^{N_v} M_i^v + \sum_{i=N_v+1}^{N_v+N_h} M_i^h + \sum_{i=N_v+N_h+1}^{N} M_i^l)/\lambda$, where $M_i^v = H_i^v + sF_i^{vh}$. If RP_i is not a VRP, we set $\lambda_i = 0.2$ per minute (min^{-1}), $T_i^c = 300$ minutes (min), $P_i^{on} = 0.25$ and the number of RPs as 4000. The parameters of VRPs are set as $P_v^{on} = 0.7$, $P_v^{cIP} = 0.4$, $T_v^c = 330$ (min), and $\lambda_v = 0.2$ (min^{-1}). Each RP has the same update rate and the total update rate in the system is α. The file size can be different, we set $s = 5$ and 50 representing a small and middle sized file. For very large files, the update description can be propagated, thus the cost of propagating update for large file is the same as that of a small file.

Table 6.1 Optimal N_v vs α

α (min^{-1})		0.01	0.1	1	10
$s = 5$	LRP	27	26	23	19
	HRP	16	16	15	15
$s = 50$	LRP	27	26	23	19
	HRP	16	16	15	14

Table 6.1 shows the optimal N_v for various file update rates (α) in a system that all RPs except VRPs are HRPs or LRPs . The optimal N_v is less than 30 in all update rates. As α increases, the updates in the system also increase, and hence the optimal N_v decreases to reduce the overhead messages for each update. The optimal N_v for a system with LRPs is larger than that of a system with HRPs. Because the pull list needs to be updated for each HRP rejoining, the optimal N_v is decreased to save the overhead messages for each HRP rejoining. The file size has no effect on the optimal N_v which is determined by minimizing overhead messages.

Table 6.2 Messages per query (MPQ) vs α

α (min^{-1})		0.01	0.1	1	10
$s = 5$	LRP	3.00	5.35	8.11	13.05
	HRP	1.52	3.85	26.74	255.53
$s = 50$	LRP	7.99	31.28	56.75	87.99
	HRP	6.49	29.61	257.12	2530.59

Table 6.2 presents the total number of messages per query (MPQ) with different file update rates (α). The results show that the push based scheme is more efficient for slow updating file while the pull based scheme is more efficient for fast updating files. For slow updating files, an HRP has multiple file accesses on each updated file. These

file accesses are directly answered, thus saving the file reconciling messages as compared to file accesses from an LRP. When the update rate is larger than the file access rate, some updated file pushed by the VS may not be accessed, thus making the push based scheme inefficient compared with the pull based scheme. The MPQ increases for larger files (larger s) due to the increased cost of propagating the updated files. In summary, an RP selects to be an LRP or HRP depending on the file update and access rates.

6.3.2 Impact of Online Probability of VRPs

In the rest of this chapter, we assume a system with both HRPs and LRPs, and the parameters are set as: $P_l^{on} = 0.2$, $P_h^{on} = 0.4$, $T_l^c = 270$ (min), $T_h^c = 300$ (min), $\lambda_l = 0.05$ (min^{-1}), $\lambda_h = 0.2$ (min^{-1}), and $s = 10$. The parameters of the VPRs are set as $T_v^c = 330$ (min^{-1}), $\lambda_v = 0.5$ (min^{-1}). Two different file update arrival rates ($\alpha = 0.002$ and 0.2 min^{-1}) are used, both LRPs and HRPs have the same file update arrival rate during their online time. The other parameters may change in different cases. Here, we set $P_v^{cIP} = 0.3$, $N_h = 500$ and $N_l = 4000$. We use the number of overhead messages per query (OHPQ) and the number of retrieved files per query (FPQ) as two metrics. They are defined as $OHPQ = H/\lambda$, and $FPQ = F/\lambda$. The parameter setup ensures that the file maintenance cost of an HRP and LRP is minimized based on the Equations (6.15) and (6.16).

Figure 6.5 shows optimal N_v as a function of P_v^{on}. We observe that the optimal value of N_v decreases from about 33 to approximately 10 as P_v^{on} increases from 0.4 to 0.9. This is due to the fact that a VS composed of a smaller number of VRPs with larger P_v^{on} can provide the same availability as a VS composed of larger number of VRPs with smaller P_v^{on}. The file update rate has very little impact on the optimal N_v in this case.

This is because the total update rate is much smaller than the total file access rate. This is true in many database and file sharing systems.

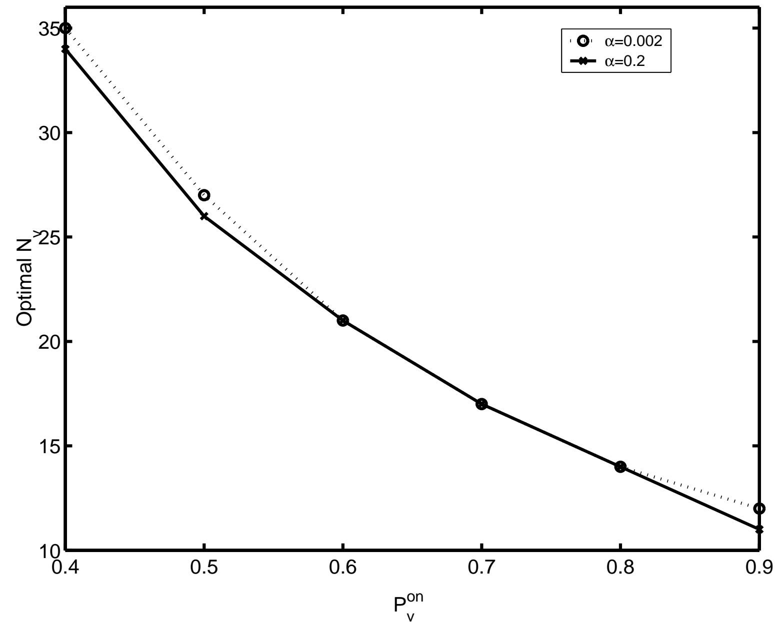


Figure 6.5 Optimal Number of VRPs vs VRP Online Probability.

From Figure 6.6, we can see that the FPQ is only dependent on the file update rate. A fast updating file corresponds to a larger FPQ, because the replicas of a fast updating file become stale quickly. Subsequently, the file accesses from these replicas need to be retrieved from the VS, thus resulting in a larger FPQ. The OHPQ is decreased from approximately 2.5 to just above 1 as P_v^{on} increases from 0.4 to 0.9. A larger P_v^{on} leads to a smaller value of optimal N_v and hence smaller OHPQ. The results indicate that the overhead messages for file maintenance are very low in the CMV algorithm.

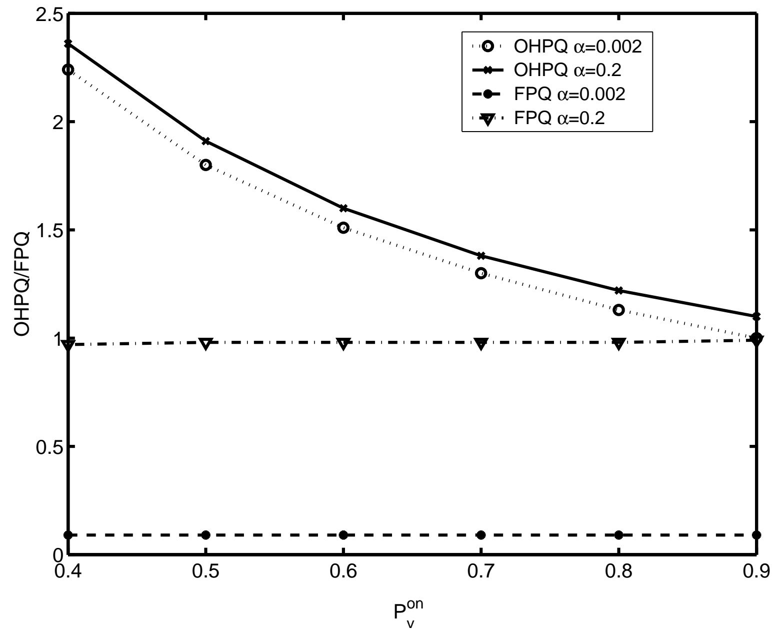


Figure 6.6 Overheads and File Retrieval Per Query vs VRP Online Probability.

6.3.3 Impact of Probability of VRP Changing its IP Address

We set $P_v^{on} = 0.7$, and all other parameters are the same as in the previous case. In our model, the effects of VRPs leaving the VRP list are considered as equivalent to the IP address changing. Thus, we vary the value of P_v^{cIP} from 0.1 to 0.5 in this study.

We observe that P_v^{cIP} has negative impact on the performance of the CMV algorithm as opposed to P_v^{on}. The VS composed of VRPs with faster changing IP address needs larger N_v and results in higher value of OHQP. The optimal N_v is about 13 for $P_v^{cIP} = 0.1$ and increases to about 24 for $P_v^{cIP} = 0.5$. The average number of online VRPs increases as N_v increases, thus resulting in more overhead messages for each file update and hence a larger OHPQ. However, FPQ has no such effect, and only depends on the update rate.

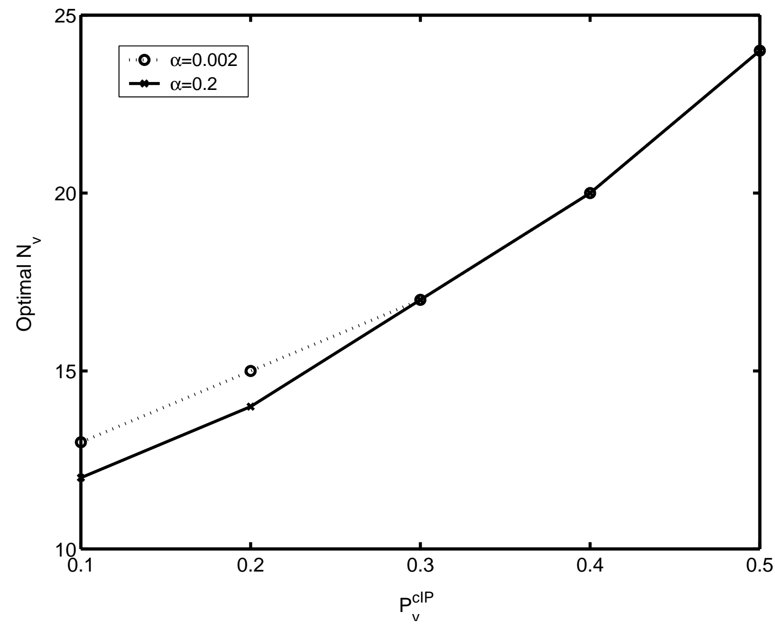


Figure 6.7 Optimal Number of VRPs vs IP Address Change rate.

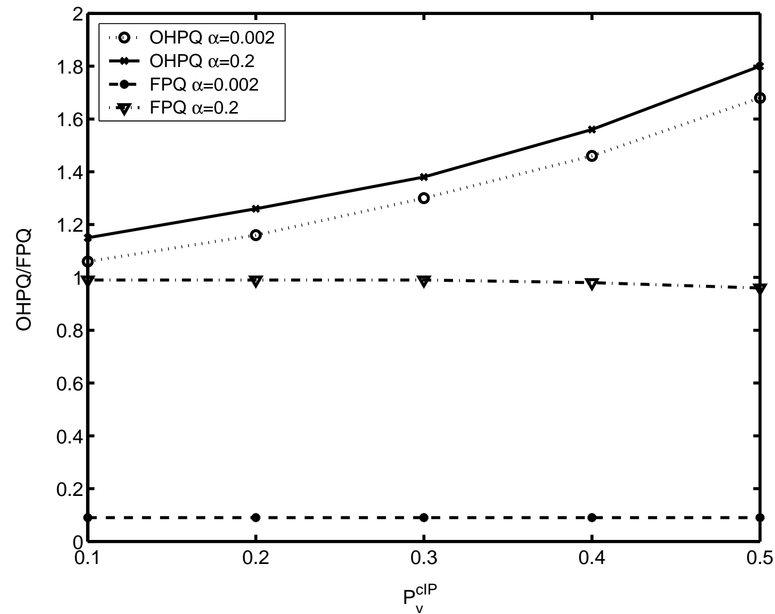


Figure 6.8 Overheads and File Retrieval Per Query vs IP Address Change Rate .

6.3.4 Impact of the Number of RPs

The impacts of the number of RPs are studied here. P_v^{on} is set as 0.7, and all other parameters are the same as in Section 6.3.2.

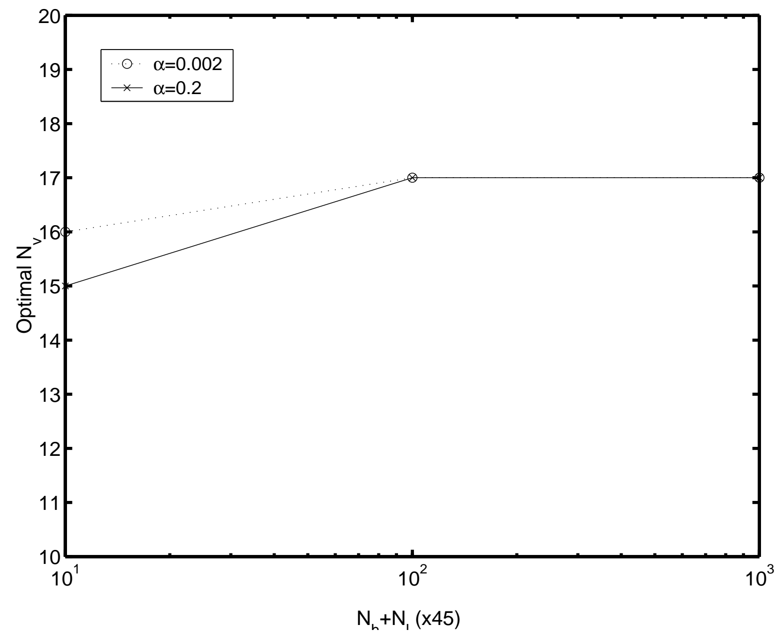


Figure 6.9 Optimal Number of VRPs (N_v) vs Number of HRPs and LRPs.

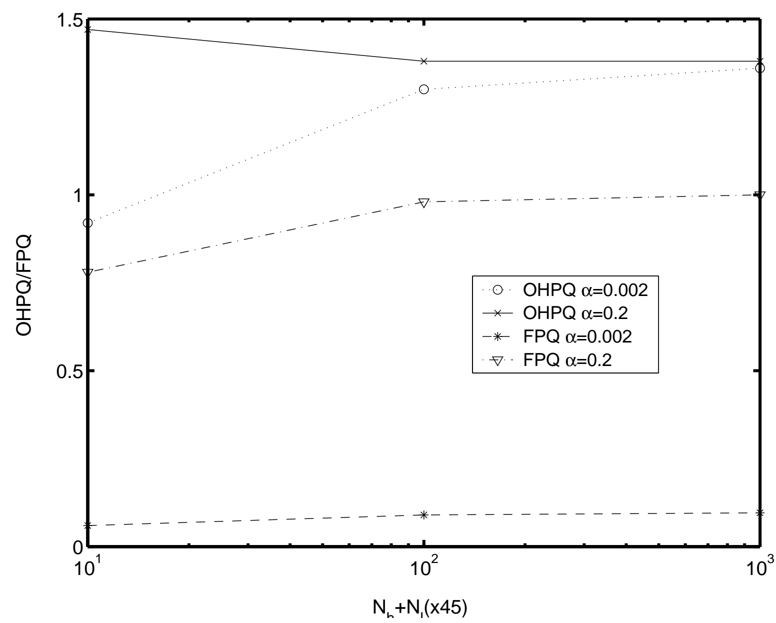


Figure 6.10 Overheads and File Retrieval Per Query vs Number of HRPs and LRPs.

Figures 6.9 and 6.10 show that the CMV algorithm has very good scalability in terms of the number of RPs. The performance of the CMV algorithm is the same when the number of RPs increases from 4,500 (4,000 LRPs and 500 HRPs) to 45,000 (40,000 LRPs and 5000 HRPs). When the number of RPs reduces to 450 (400 LRPs and 50 HRPs), the optimal value of N_v, OHPQ and FPQ have very small difference. For a large number of RPs in the system, the effect of VRPs is insignificant. However, the effect is significant when the number of VRPs is close to the number of HRPs and LRPs.

In the above cases, P^{vf} is in the order of 10^{-9} or less and T^{vf} is only a few (<10) minutes for all parameter ranges. These results show that the CMV algorithm is efficient to provide one-copy serializability file consistency for decentralized and unstructured P2P networks.

6.4 Summary

File consistency is a critical problem in P2P systems that are subjected to continual file updates. In this chapter, we proposed a novel algorithm for file Consistency Maintenance through Virtual servers (CMV) in decentralized and unstructured P2P systems. In the CMV algorithm, each dynamic file has a VS composed of multiple RPs. The virtual server RPs (or VRPs) cooperatively maintain the master copy of a file. In order to maintain one-copy serializability, any update can only be accepted through the VS. Mathematical analysis is carried out to determine the optimal number of VRPs and the overhead messages for file maintenance under various system parameters. The results indicate that the CMV algorithm is an efficient file consistency maintenance algorithm and can be used in P2P based electronic business systems such as online auctions and online games.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This dissertation addressed the issue of consistent information sharing in wireless mobile and Peer-to-Peer (P2P) networks.

To improve mobile cache performance, we proposed a *Scalable Asynchronous Cache Consistency Scheme* (SACCS). The performance of SACCS is evaluated analytically and experimentally. The results show that SACCS is a highly scalable, efficient, and low complexity algorithm, and provides *weak* cache consistency with a small probability of stale cache hit under unreliable invalidation report (IR) broadcast environments.

Strictly speaking, SACCS is a hybrid of stateful and stateless algorithms. However, unlike stateful algorithms, SACCS maintains only one flag bit for each data object in mobile support station (MSS) to determine when to broadcast the IRs. On the other hand, unlike the existing synchronous stateless approaches, SACCS does not require periodic broadcast of IRs, thus significantly reducing IR messages that need to be sent through the downlink broadcast channel. The SACCS inherits the positive features of both stateful and stateless algorithms.

To design effective cache consistency schemes and achieve the optimized cache performance in multi-cell wireless networks, we introduced three strategies including: 1) homogeneous IR; 2) inhomogeneous IR without roaming check; and 3) inhomogeneous IR with roaming check. These strategies are evaluated under various multi-cell environments. Our simulation results revealed that the homogeneous IR strategies perform

better for slow updating data objects, fast roaming MUs and small systems whereas the inhomogeneous IR strategies are more efficient for fast updating data objects, slow roaming MUs and large systems. We also derived two consistency maintenance cost functions to determine the cost of maintaining data objects locally and globally. Based on these two cost functions, a Dynamic Scalable Cache Consistency Scheme (DSACCS) is proposed for wireless cellular networks. In the DSACCS scheme, the IR of a data object is broadcast globally or locally depending on which action results in a minimum consistency maintenance cost. The simulation results show that DSACCS outperforms three extended cache strategies for various multi-cell environments.

To effectively maintain probabilistic file consistency in decentralized and unstructured P2P networks, we proposed a novel algorithm, called *Update Propagation Through Replica Chain* (UPTReC). In UPTReC, each file has a logical replica chain composed of all RPs and each RP has a partial knowledge of the chain. When an RP updates the file, it pushes the update to all possible online RPs through the replica chain. When an offline RP gets reconnected, the file status is synchronized by pulling an online RP. An analytical model of the proposed algorithm is derived. The performance comparison of UPTReC with the Rumor algorithm shows that UPTReC reduces up to 70% overhead messages to propagate updates with a smaller query ratio for highly updating files.

To provide strong file consistency in decentralized and unstructured P2P networks, we proposed an algorithm for *file Consistency Maintenance through Virtual servers* (CMV). In the proposed approach, each dynamic file has a virtual server (VS) composed of multiple RPs. The VRPs cooperatively maintain the master copy of a file. Any update can only be accepted through the VS to maintain the one-copy serializability. We analytically determine the optimal number of VRPs and the overhead messages for file

maintenance under various system parameters. The results indicate that CMV is an efficient file consistency maintenance algorithm for decentralized and unstructured P2P systems.

7.1 Future Work

The future research works related to the dissertation include the data management issues in mobile P2P networks and trust management in P2P networks.

Data management in mobile peer-to-peer networks

Wireless mobile P2P networks inherit all the challenges in wireless cellular and P2P networks, such as limited bandwidth and battery power, frequent disconnections and dynamic IP addresses. Moreover, the network topology changes from time to time. Thus, providing data consistency for mobile P2P networks is more challenging than that for traditional wired P2P networks. Our future plan on this topic is to develop design principles for data consistency management in wireless mobile P2P networks.

Trust Management in Peer-to-Peer networks

Anonymity is one of the most attractive features of P2P networks. However, this feature makes the network more vulnerable to malicious peers. To protect the system, some P2P systems introduce the community-based reputations to estimate the trust value of a peer which is accessed by other peers in the system. A high trust value indicates a good reputation of a peer. Using the trust value, a malicious peer can be identified. The trust value can effectively protect the system by identifying the malicious peers. The trust

value is a dynamic file and must be efficiently and accurately maintained. Our research will explore efficient trust management schemes in P2P networks.

REFERENCES

[1] K. Aberer, "P-Grid: A Self-organizing Access Structure for P2P information Systems". In *Proceedings of the Sixth International Conference on Cooperative Information Systems*, pp 179-194, 2001.

[2] K. Aberer, Z. Despotovic, "Managing Trust in a P2P Information System". In *Proceedings of the 10th International Conference on Information and Knowledge management*, pp 310-317, ACM press 2001.

[3] Y. Bao, R. Alhajj and K. Barker, "Hybrid Cache Invalidation Schemes in Mobile Environments". In *Proceedings of IEEE/ACS International Conference on Pervasive Services*, pp 209-218, 2004.

[4] D. Barbara and T. Imielinksi," Sleeper and Workaholics: Caching Strategy in Mobile Environments". *In Proceedings of the ACM SIGMOD Conference on Management of Data*, pp 1-12, 1994.

[5] P. Barford, M. Crovella,"Generating Representative Web Workloads for Network and Server Performance Evaluation". *Proceedings of the ACM SIGMETRICS Conference*, pp 151-160, 1998.

[6] R. Bhagwan, S. Savage and G. M. Voelker, "Understanding Availability". In *Proceedings of the 2nd International Workshop on Peer-to-peer systems*, pp 256-267, 2003.

[7] L. Breslau, P. Cao, J. Fan, G. Phillips and S. Shenker, "Web caching and Zipf-Like Distributions: Evidence and Implications". *In Proceedings of IEEE INFOCOM*, pp 126-134, 1999.

[8] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments". *ACM Intl. Conf. on Computing and Networking (Mobicom)*, pp 200-209, August, 2000.

[9] G. Cao, "On Improving the Performance of Cache Invalidation in Mobile Environments". *ACM/Kluwer Mobile Network and Applications*, 7(4), pp 291-303, 2002.

[10] G. Cao, "Proactive Power-Aware Cache Management for Mobile Computing Systems". *IEEE Transactions on Computers*, 51(6), pp. 608-621, 2002.

[11] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments". *IEEE Transactions on Knowledge and Data Engineering*, 15(5), pp. 1251-1265, 2003

[12] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World-Wide Web". *In Proceedings of the International Conference on Distributed Computing Systems*, pp 12-21, 1997.

[13] M. Castro, P. Druschel, A. Ganesh, A. Rowstron and D. Wallach, "Secure Routing for Structured Peer-to-peer Overlay Networks". *In Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation*, 2002.

[14] Y. Chawathe, S. Ratnasamy, L. Breslau, N. lanham and S. Shenker, "Making Gnutella-like Peer-to-Peer Systems Scalable". In *Proceedings of ACM SIGCOMM*, pp 407-418, 2003.

[15] X. Chen, S. Ren, H. Wang and X. Zhang, "SCOPE: Scalable Consistency Maintenance in Structured P2P Systems", In *Proceedings of IEEE INFOCOM*, 2005.

[16] A. Chockalingam, M. Zorzi, L. B. Milstein and P. Venkataram, "Performance of a Wireless Access Protocol on Correlated Rayleigh-Fading Channels with Capture". *IEEE Transaction on Communication*, pp 644-655, 1998.

[17] I. Clarke, O. Sandberg, B. Wiley and T. Hong, "Freenet: a Distributed Anonymous Information Storage and Retrieval System". In *Sedigning Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[18] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks". In *Proceedings of the ACM SIGCOMM* , pp 177-190, 2002.

[19] E. Cohen, A. Fiat and H. Kaplan, "Associative Search in Peer-to-Peer Networks: Harnessing and Latent Semantics". In *Proceedings of IEEE INFOCOM*, pp 1261-1271, 2003.

[20] F. Cornelli, E. Damiani, S. Vimercati, S. Paraboschi, P. Samarati, "Choosing Reputable Servents in P2P network". In of the International World Wide Web Conference, pp 376-386, 2002.

[21] A. Datta, M. Hauswirth and K. Aberer, "Updates in Highly Unreliable, Replicated Peer-to-Peer Systems". In *Proceedings of IEEE ICDCS*, pp 76-88, 2003.

[22] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, and J. Larson, "Epidemic Algorithms for Replicated Database Maintenance". In *Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC)*, pp 1-12, 1987.

[23] L. Feeney and M. Nilsson,"Investigating the Energy Consumption of a Wireless Network Interface in an Ad hoc Networking Environment". In *Proceeding of IEEE INFOCOM*, pp 1548-1557, 2001.

[24] B. Gedik and L. Liu, "PeerCQ: A Decentralized and Self-Configuration P2P Information Monitoring System". In *Proceedings of IEEE ICDCS*, pp 490-499, 2003.

[25] J. Gray, P. Helland, P. O'Neil and D. Shasha, "The Dangers of Replication and a Solution". In of ACM SIGMOD, pp 173-182, 1996.

[26] http://www.napster.com.

[27] http://linuxreviews.org/news/2004/11/05-p2p.

[28] Q. Hu and D. K. Lee, "Cache Algorithms Based on Adaptive Invalidation Reports for Mobile Environments". *Cluster Computing*, 1(1), pp 39-50, 1998.

[29] S. Iyer, A. Rowstron and P. Druschel," Squirrel: A Decentralized Peer-to-peer Web Cache". In *Proceedings of the 21th ACM Symposium on Principles of Distributed Computing (PODC)*, pp 213-222, 2002.

[30] J. Jing, A. Elmagarmid, A. Heal, and R. Alonso. "Bit-Sequences: an Adaptive Cache Invalidation Method in Mobile Client/Server environments". *ACM Mobile Networks and Applications*, 2(2), pp 115-127, 1997.

[31] A. Kahol, S. Khurana, S.K.S. Gupta and P.K. Srimani," A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment". *IEEE Trans. on Parallel and Distributed Systems*, 12(7), pp 686-700, 2001.

[32] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks". In *Proceedings of International Word Wide Web Conference*, pp 536-543, 2003.

[33] B. Kemme and G. Alonso, "Don't be Lazy, be Consistent: Postgres-R, a New Way to Implement Database Replication". In *Proceedings of VLDB*, pp 134-143, 2000.

[34] B. Knutsson, H. Lu, W. Xu and B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games". In *Proceeding of IEEE INFOCOM*, pp 96-107, 2004.

[35] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, and D. Geels, "Oceanstore: an Architecture for Global-Scale Persistent Storage". In *Proceedings of ACM ASPLOS-IX*, pp 190-201, 2000.

[36] J. Lan, X. Liu, P. Shenoy and K. Ramaritham, "Consistency Maintenance in Peer-to-Peer File Sharing Networks". In *Proceedings of Third IEEE Workshop on Internet Applications*, 2002.

[37] D. Lee, W. Lee, J. Xu and B. Zhang, "Data Management in Location-dependent Information Services: Challenges and Issues". *IEEE Pervasive Computing*, 1(3), pp 65-72, 2002.

[38] D. Li and R. Cheriton, "Scalable Web Caching of Frequently Updatted Objects Using Reliable Multicast". *In Proceedings of The USENIX Symposium on Internet Technologies and Systems*, October, pp 1-12, 1999.

[39] G.Y. Liu and G.Q. McGuire Jr, "A Mobility-Aware Dynamic Database Caching Scheme for Wireless Mobile Computing and Communications". *Distributed and Parallel Databases*, 4(5), pp 271-288, 1996.

[40] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer networks". In *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, pp 84-95, 2002.

[41] R. Govindan and H. Tangmunarunkit, "Heuristics for internet map discovery". *Proceedings of IEEE INFOCOM*, pp 1371-1380, 2000.

[42] E. Ogston and S. Vassiliadis, "A Peer-to-peer Agent Auction". In *Proceeding of ACM International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp 151-159, 2002.

[43] "Open Source Community, Gnutella". In *http://gnutella.wego.com*, 2001.

[44] E. Pacitti, P. Minet and E. Simon, "Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases". In *Proceeding of VLDB*, pp 126-137, 1999.

[45] T. W. Page, R. G. Guly, J. S. Heidemann, D. Reiher, A. Goel, G. H. Kuenning, and G. J. Popek, "Perspectives on Optimistically Replicated Peer-to-Peer Filing". *Software-Practice and Experience*, pp 155-180, 28(2), 1998.

[46] Y. Saito, C. Karamanolis and M. Mahalingam, "Taming Aggressive Replication in the Pangaea Wide-Area File System". In *Proceedings of USENIX OSDI*, 2002.

[47] S. Park, L. Liu, C. Pu, M. Srivatsa, J. Zhang, "Resilient Trust Management for Web Service Integration". In *Proceedings of the 3rd IEEE International Conference on Web Services*, 2005.

[48] R. Powers, "Batteries for Low Power Electronics". *Proceedings of IEEE*, pp 687-693, 83(4) 1995.

[49] T. S. Rappaport, *Wireless Communication: Principles and Practice*, Prince Hall, 1996.

[50] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network". In *Proceedings of ACM SIGCOMM*, pp 161-172, 2001.

[51] V. Rodoplu and T. Meng, "Minimum Energy Mobile Wireless Networks". *IEEE Journal on Selected Areas in Communications*, pp. 1333-1344, 1999.

[52] A. Rowstron and P. Druschel," Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". *International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[53] M. Roussopoulos and M. Baker, "CUP: Controlled Update Propagation in Peer-to-Peer Networks". In *Proceedings of the 2003 Annual USENIX Technical Conference*, June 2003.

[54] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems". In *Middle Ware*, pp 329- 350, 2001.

[55] S. Saroiu, P. K. Gummadi and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems". In *Proceeding of SPIE Conference on Multimedia Computing and Networking (MMCN)*, 2002.

[56] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". In *Proceedings of ACM SIGCOMM*, pp 149-160, 2001.

[57] K. Tan, J. Cai and B. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments". *IEEE Trans. on Parallel and Distributed Systems*, 12(8), pp 789-807, 2001.

[58] Z. Wang, S. K. Das, H. Che and M. Kumar, "SACCS: Scalable Asynchronous Cache Consistency Scheme for Mobile Environments". *In Proceedings of International Workshop on Mobile and Wireless Networks*, pp 797-802, 2003.

[59] Z. Wang, S. K. Das, H. Che and M. Kumar, "Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments". *IEEE Transactions on Parallel and Distributed Systems*, 15(11), pp 983-995, 2004.

[60] Z. Wang, M. Kumar, S. K. Das and H. Shen, "Investigation of Cache maintenance Strategies for Multi-Cell Environments". *In Proceedings of 4th International Conference of Mobile Data Management (MDM)*, pp29-44, 2003.

[61] Z. Wang, M. Kumar, S. K. Das and H. Shen, "Dynamic Cache Consistency Schemes for Wireless Cellular Networks". *To appear to IEEE Transactions on Wireless Communications*, 2006.

[62] Z. Wang, M. Kumar, S. K Das and H. Shen, "Update Propagation Through Replica Chain in Decentralized and Unstructured P2P Systems". *In Proceedings of IEEE International Conference on Peer-to-Peer Computing*, pp 64-71, 2004.

[63] Z. Wang, S. K Das, M. Kumar and H. Shen, "File Consistency Maintenance through Virtual Servers in P2P Systems". *Technique Report, University of Texas at Arlington*, 2005.

[64] K. L. Wu, P. S. Yu and M.S. Chen, "Energy-Efficient Caching for Wireless Mobile Computing". *In Proceedings of 20th International Conference on Data Engineering*, pp 336-345, 1996.

[65] Li Xiong and Ling Liu, "PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities". *IEEE Transactions on Knowledge and Data Engineering*, 16(7), pp 843-857, 2004.

[66] J. Xu, X. Tang and D. Lee, "Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments", *IEEE Transactions on Knowledge and Data Engineering*, 15(2), pp474-488, 2003.

[67] H. Yu, L. Breslau and S. Shenker, "A Scalable Web Cache Consistency Architecture". *In Proceedings of the ACM SIGCOMM*, pp 163-174, 1999.

[68] J. C. Yuen, E. Chan, K. Lam and H.W.Leung, "Cache Invalidation Scheme for Mobile Computing Systems with Real-time Data". *SIGMOD Record*, pp 34-39, 2000.

[69] J. Zhang, R. Izmailov, D. Reininger and M. Ott, "Web Cache Framework: Analytical Models and Beyond". *In Proceedings of IEEE Workshop on Internet Applications*, pp 132-141, 1999.

[70] B. Zheng, J. Xu and D. Lee, "Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments". *IEEE Transactions on Computers*, 51(10), pp 1141-1153, 2002.

Publications

1. Z. Wang, M. Kumar, S. K. Das and H. Shen, "Investigation of Cache maintenance Strategies for Multi-Cell Environments", *In Proceedings of 4th International Conference of Mobile Data Management (MDM)*, pp 29-44, 2003.
2. Z. Wang, S. K. Das, H. Che and M. Kumar, "SACCS: Scalable Asynchronous Cache Consistency Scheme for Mobile Environments", *In Proceedings of International Workshop on Mobile and Wireless Networks*, pp 797-802, 2003.
3. H. Shen, M. Kumar, S. K Das and Z. Wang, "Energy-Efficient Caching and Prefetching with Data Consistency in Mobile Distributed Systems", *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp 67-76, 2004.
4. H. Shen, S. K Das, M. Kumar and Z. Wang, "ECOR: Energy Efficient Cooperative Caching with Optimal Radius in Hybrid Wireless Networks", *IFIP Networking Conference*, pp 841-853, 2004.
5. Z. Wang, M. Kumar, S. K Das and H. Shen, "Update Propagation Through Replica Chain in Decentralized and Unstructured P2P Systems", *In Proceedings of IEEE International Conference on Peer-to-Peer Computing*, pp 64-71, 2004.
6. Z. Wang, S. K. Das, H. Che and M. Kumar, "Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments", *IEEE Transactions on Parallel and Distributed Systems*, 15(11), pp 983-995, 2004.
7. Z. Wang, H. Che, M. Kumar and S. K Das, "CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Table Locking", *IEEE Transactions on Computers* 53(12), pp 1602-1614, 2004.

8. H. Shen, M. Kumar, S. K Das and Z. Wang, "Energy-Efficient Data Caching and Prefetching for Mobile Devices Based on Utility", *ACM Mobile Networks and Applications (MONET), Special Issue on Mobile Services*, 10(4), pp 475-486, 2005.

9. Z. Wang, M. Kumar, S. K. Das and H. Shen, "Dynamic Cache Consistency Schemes for Wireless Cellular Networks", *To appear in IEEE Transactions on Wireless Communications*, 2006.

10. Z. Wang, S. K Das, M. Kumar and H. Shen, "File Consistency Maintenance through Virtual Servers in P2P Systems", Technique report, University of Texas at Arlington, 2005.

BIOGRAPHICAL STATEMENT

Zhijun Wang was born in Hunan, China. He received his Ph.D. degree in Computer Science and Engineering from The University of Texas at Arlington in 2005 and his Master degree in Electrical Engineering from the Pennsylvania State University in 2001. Before he came to USA, he graduated from the Huazhong University of Science and Technology in Physics major. His current research interests including data management in wireless mobile and peer-to-peer networks, network processor, network security and distributed systems.