Rt
399
c.1

A COMPARATIVE STUDY BETWEEN CACHE REPLACEMENT ALGORITHMS USED IN THE SCALABLE ASYNCHRONOUS CACHE CONSISTENCY SCHEME

by

LANA TURK

B.S., Computer Science, Haigazian University, 2001

Thesis submitted in partial fulfilment of the requirements for the Degree of Master of Science in Computer Science

Division of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

May 2006

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences - Beirut Campus

Thesis approval Form (Annex III)

Student Name: Lana Turk I.D. #: 200102455

Thesis Title :	A Comparative Study between Cache Replacement Algorithms used in the Scalable Asynchronous Cache Consistency Scheme
Program :	Master of Science
Division/Dept :	Computer Science and Mathematics
School :	**School of Arts and Sciences**
Approved by:	Ramzi A. Haraty
Thesis Advisor:	[illegible]
Member :	Faisal Abu Khzam [illegible]
Member :	Abdul Nasser Kassar [illegible]

Date 29/5/2006

Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Lana Turk

Signature: [illegible] Date: 30/05/2006

Acknowledgment

I would like to thank my advisor Dr. Ramzi Haraty for his guidance throughout my Thesis work. Thanks are also to Dr. Abed El-Nasser Kassar and Dr. Faysal Abu Khzam for being on my thesis committee.

I would like to thank my family and friends for their encouragement.

Finally, a special thanks to my husband for his support.

Abstract

The technology of PCs has been in progress in a fast rate for many years. Mobile computing is one of the technologies brought into the area of computers. Different problems have arisen from the narrow bandwidth and limited battery power of mobile clients. Therefore, algorithms have been proposed to provide cache consistency in mobile databases by using cache invalidation strategies. Scalable Asynchronous Cache Consistency Scheme (SACCS), a highly scalable, efficient and low complexity algorithm, is one of the cache consistency maintenance algorithms proposed. It counts on invalidation reports to maintain cache consistency between server databases and mobile user databases. Least-Recently-Used (LRU) is used as a cache replacement algorithm in SACCS. In this work, different cache replacement strategies are proposed to be applied in SACCS: MRU (Most-Recently-Used), MFU (Most-Frequently-Used), LFU (Least-Frequently-Used), FIFO (First-In-First-Out). A simulation of SACCS with these cache replacement algorithms is done and produces results that will be compared to show the variation of the performance of the system. Statistical results will point out the advantages and disadvantages of each algorithm concerning miss ratio, delay, total hit, and total miss.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past few years, mobile computing has been used profusely in computer technology. Many problems arise from this computer field because the wireless connection is expensive and the bandwidth is limited. To save time in exchanging data between base stations and mobile users and improve the performance of the system, data caching is imposed on mobile users in order to access the data recently used or most likely to be used in the future. To maintain cache consistency between mobile users and base stations, two approaches have been proposed: the stateful and the stateless approach.

A cache consistency maintenance algorithm called Scalable Asynchronous Cache Consistency Scheme (SACCS), gathering both stateless and stateful approaches, was proposed with the Least-Recently-Used (LRU) replacement algorithm (Wang, 2003). In this work, different replacement algorithms are used instead of the LRU in the SACCS, producing different results concerning the system performance. A comparison is made between the LRU and the replacement algorithms used to show the advantages and disadvantages of each of these algorithms in system performance.

1.1 Background

The progress of computer technology in the past ten years has been leading to a smaller size PC and an increase of the capacity of software and hardware functionality. The cellular, satellites and wireless LAN expanded technologies have added Mobile Computing (MC) technology to the field of computers. Mobile computing environments have faced physical constraints, such as low network bandwidth and high communication latency, which are subject to studies and a lot of work to solve such problems.

1.1.1 Mobile computing

Mobile computing is considered as an expanded version of distributed systems. A client/server network is an example of mobile computing. The clients act as mobile or fixed units and the servers as hosts or base stations for data (which serves as the communication link between MU and the entire network). The connection of mobile units to the servers can be started and terminated during different periods of time for many reasons, such as the limitation of battery power, the high cost of wireless connection, etc. Data caching on a mobile unit is one of the important solutions to this problem.

1.1.2 Mobile database system

A mobile database system is a distributed system with mobile connectivity (where a mobile client can communicate to a server whenever needed), full spatial mobility, and wireless and wired communication capability. A personal communication system is a mobile database system in which mobile components can communicate to the base station or server through a number of wireless channels.

1.1.3 Data caching

Data caching is used in mobile databases to reduce the number of queries sent by clients to the server to access data needed. The recently used or likely to be used data is fetched in the caches of the mobile users in order to be reused in the future. Therefore, data caching improves the performance of mobile applications by reducing the expected data access delay. The caching performance is also affected by the cache replacement policy being used.

1.1.4 Cache replacement algorithm

It is an algorithm that determines which data in the cache will be replaced (i.e. choose a replacement victim) when the cache is full. Different strategies of selecting replacement victims are being used to increase the performance of mobile systems. Temporal locality, spatial locality, and semantic locality are three types of cache replacement policies: Temporal locality defines that the data being used will be reused in the future like MRU (most recently used). Spatial locality defines that the data in a near place with the recently used data will be used again in the future. The semantic locality is that a currently used area could be accessed in the future.

1.1.5 Invalidation report

To validate the data in their caches, mobile clients try to query the server congesting the network narrow bandwidth and getting stuck in the limited capability of the transmission. The invalidation method is a solution to this problem. It reduces the data transfer from the clients to the server and vice versa. Invalidation reports are proposed to invalidate out of use (or invalid) data in the mobile user caches. After disconnections, reconnected users are prevented, by using invalidation reports, from discarding wastefully all their caches since some data may still be valid for use.

Invalidation reports are composed of different methods like:

- The Broadcasting Timestamp Strategy:

  Updated data is represented in the report as a pair (ID,timestamp) where the ID is the updated data itself and the timestamp is the time at which the data was updated.

- The Bit-Sequence Approach:

  The invalidation report gets bigger as the amount of the updated data gets larger. The bit-sequence approach is proposed to solve this problem by presenting each data in the report by a bit where the bit sequence number in the report represents the data sequence number in the database. If the data has been updated, the value of the bit is 1 in the report or else 0.

1.2 Scope of the thesis

Mobile computing encounters a lot of problems regarding wireless connection and limited bandwidth. Solutions have been proposed to such problems like data caching and invalidation reports. These methods have been used in the cache consistency maintenance algorithm SACCS, which maintains the cache consistency between the mobile users and the servers by gathering the features of both the stateful and stateless approaches. SACCS uses the cache replacement algorithm LRU as a base replacement algorithm in the mobile user caches.

In this thesis, the LRU is replaced by different cache replacement algorithms: LFU (Least-Frequently-Used), MRU (Most-Recently-Used), MFU (Most-Frequently-Used), and FIFO (First-In-First-Out). The purpose of this thesis is to show and compare between the results of these cache replacement algorithms used in the SACCS (including the LRU already used). The comparison between the different outputs of the SACCS algorithm will be built on experimental and statistical results to show the advantages and disadvantages of each cache replacement algorithm in different characteristics of the system performance, for example the miss ratio and the delay in data access.

1.3 Organization of the thesis

The remainder of this thesis is composed of 5 chapters: Chapter 2 is a literature review of related techniques and approaches used in the field of mobile computing. This chapter provides for the reader some knowledge about what has been proposed in the past few years of methods related to caching and cache consistency between mobile users and servers in mobile computing. Chapter 3 describes the cache consistency maintenance SACCS in details: the circumstances under which it was proposed, the way how it works between mobile users and servers. It also describes the additional techniques applied to the SACCS: the different replacement algorithms used and the updated code of the SACCS for each replacement algorithm. Chapter 4 presents the experimental and statistical results of the SACCS, using different replacement algorithms. Chapter 5 presents the conclusion drawn from this thesis.

Chapter 2

Literature Review

Computer interconnection through wireless networks has been increasing at a high speed in mobile environments. Data is broadcasted repetitively from the server to the clients without the need for a client request. Thus, the server is not informed before a mobile transaction accesses a data object because the data is accessed by the clients while it is being broadcasted. A distributed database system, whose architecture is shown in figure 2.1, faces a lot of challenges concerning data consistency, especially with a large number of mobile clients.

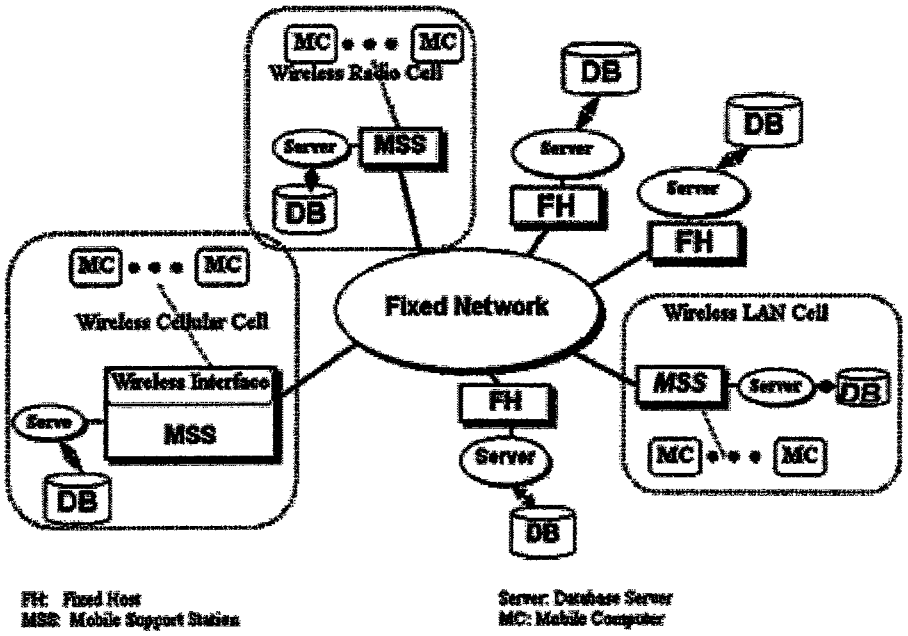


Figure 2.1 Architecture of a mobile database system (Seetha, 2001)

The server may broadcast data items during update transactions in its database. Thus, mobile clients observe data inconsistency through their transactions. During the past few years, different studies have been made on preserving data consistency in mobile distributed database systems. This chapter describes a number of cache invalidation strategies proposed to solve efficiently the problem of data inconsistency. These strategies lead to a better system performance by reducing bandwidth utilization and query latency, saving power and energy for mobile clients.

2.1 Cache invalidation model

The dynamic progress of mobile computing has led to a major concern regarding data consistency. Mobile clients want to connect to the server, send queries and receive data at different periods of time but they are facing different limitations, such as the limited battery power of the clients, and the narrow bandwidth of the network. In addition, the mobility and frequent disconnections of the mobile clients cause the data cached in the client side to become invalid. To solve this problem, different cache invalidation mechanisms have been proposed.

The IR-based cache invalidation approach was used to preserve cache consistency (Yin, 2006). In this approach, invalidation reports (IRs) are periodically broadcasted by the server to the clients. When a client receives a request for some data, it waits for the next IR to invalidate or validate the data in its cache accordingly. If the data is invalid, the client sends a request to the server for a valid copy of the data. Otherwise, the client returns the requested data directly.

2.1.1 UIR-based cache invalidation

The latency for any client to answer a query depends on the length of the IR interval; as the IR interval gets longer, the answer to the query gets delayed. A solution to that delay was proposed by Cao to replicate the IR m times within the IR interval (Yin, 2006). The replicated IRs called updated invalidation reports (UIRs) contain only the data items updated after the broadcast of the last IR. In this way, the client should wait the maximum of 1/m of the IR interval to answer a query for any data. This proposition reduced remarkably the query latency. Figure 2.2 presents a sample of replicating UIRs in IR intervals.

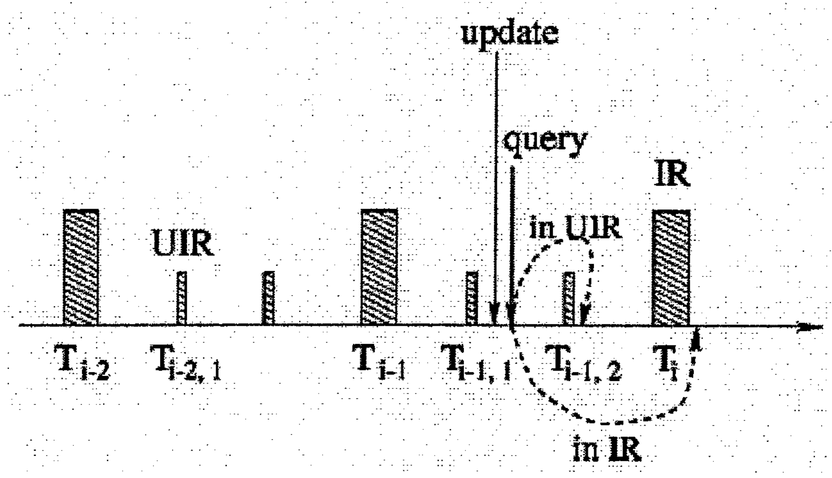


Figure 2.2 Reducing the query latency by replicating UIRs (Cao2, 2002)

2.2 Counter-based cache invalidation algorithm

The counter-based cache invalidation approach is designed to reduce the query latency and the congestion on the bandwidth by following three schemes.

First, prefetching data that is likely to be used in the future by the clients makes a better utilization of the broadcast bandwidth. In this case, when the server broadcasts data items, the clients download the data that is invalid in their caches and do not have to send additional requests to the server for valid data.

Second, using a broadcast list saves power and bandwidth. The server broadcasts only the data that has been updated since the last IR (invalidation report). After the server broadcasts an IR, it broadcasts the id list of data being updated (Lbcast). Then it broadcasts the data itself. The clients, on their side, save the broadcast list and wait until the data arrives to wake up and download it.

Third, relying on counters also helps in saving bandwidth and in identifying the frequently accessed data to be broadcasted. The broadcast list includes only the ids of the data most frequently accessed by the clients. To identify the data to be included in the id list, a counter is associated with each data item. This counter is incremented by one on every request of the data from the server and decremented by one if the data has been discarded by the client. The data with a counter that is equal to 0 is not included in the IR, unlike the IR-based approach, which includes all updated data in the IR. Hence, this counter-based scheme saves the broadcast bandwidth. Figure 2.3 shows the design of the counter-based cache invalidation approach:

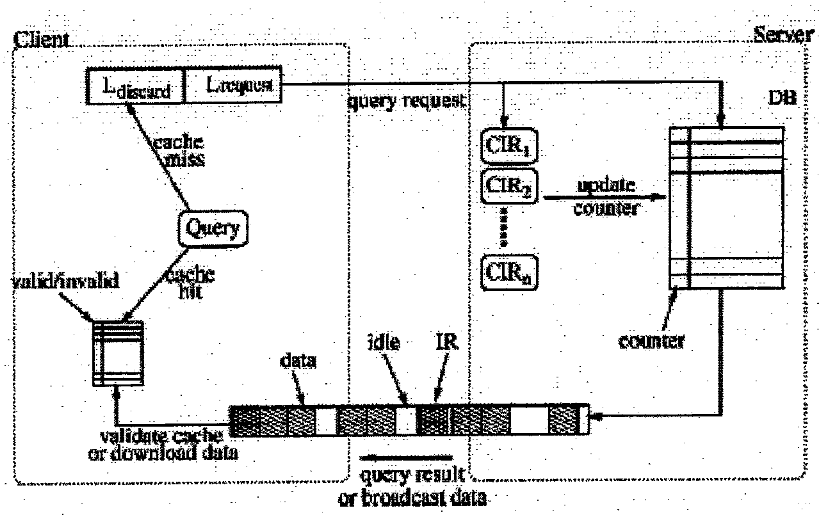


Figure 2.3 The counter-based cache invalidation (Cao2, 2002)

2.3 Cache invalidation strategy

In the previous years, different cache invalidation strategies existed, such as the bit-sequence (BT), the timestamp (TS), the dual-report (DRCI) and others, to maintain data consistency between data items in the server database and data items in the mobile client caches.

A new cache invalidation strategy is proposed to reduce the invalidation report sizes and invalidate the necessary data items in the mobile client caches (Chuang, 2004). In this strategy, when the user sends a query to the mobile client, the queried data will be checked for its validity; if the data exists in the client cache and is valid, it will be used by the user. On the other hand, if the data does not exist in the client cache, it will be requested from the server. If the client is disconnected for a period of time and reconnected again, the data in its cache will be in an uncertain state. In this case, if data is requested by a user and exists in the client cache, the client should check for the validity of all the data in its cache by sending the server the value of the time when the last invalidation report was received by the client (T_{lb}) and the requested data. Then, the server broadcasts to the client the invalidation report containing only the ids of the updated data since T_{lb} (including the last updated timestamp T) and the requested data. Thus, the client answers the user's request and replaces its T_{lb} by T if T_{lb} is less than T. In addition, the other mobile clients, receiving the IR, will update their T_{lb} if its value is less than T.

As a result, the IRs broadcasted by the server will be smaller in size consisting only of the ids of the invalidated data since the T_{lb} and the data in the mobile client caches would not be invalidated unnecessarily. Thus, this proposed cache invalidation strategy leads to better bandwidth utilization and less cache requests to the server.

2.4 Power-aware cache management

Improving performance and bandwidth utilization in a mobile environment have led to the suggestion of different cache solutions. The invalidation report (IR) approach was proposed in order to improve the performance in mobile systems. In this approach, IRs are broadcasted periodically by the server to the mobile clients. The clients listen to these IRs and validate their caches instead of querying the server for any invalid cached data. To improve the bandwidth utilization, the updated invalidation report or UIR-based approach, which is an additional technique to the IR-based solution, was proposed. In this approach, UIR is repeated during an IR interval so the client will not have to wait for the next IR to answer a query.

Beside these approaches, prefetching data in the client caches helps in increasing the data hit ratio but does not consider power consumption in the client pcs. To solve the problem of power consumption, a power-aware cache management was proposed based on a prefetch-access ratio scheme (Cao1, 2002). After the server broadcasts an IR, it broadcasts the id list of data being updated. Then it broadcasts the data itself. The clients, on their side, save the broadcast list and wait until the data arrives to wake up and download it. This approach can save power and reduce bandwidth utilization.

The prefetch-access ratio (PAR) scheme uses the ratio of the number of prefetches over the number of accesses to evaluate the usefulness of prefetching the data by the clients. If the data is accessed frequently, then the PAR will be less than 1, which means that the data should be prefetched by the client. Otherwise, if the PAR is greater than 1, then the data is marked as non-prefetch because it has a low access rate. With this proposed prefetch-access ratio scheme, the power consumption will be reduced remarkably and the cache hit ratio will be increased, leading to an upgrade in performance.

2.5 Energy-efficient cache invalidation

In a mobile environment, two types of invalidation strategies exist.

The first strategy is when the server is called a stateful server, which knows about the state of data cached by its clients and the clients' relocation. When the data is updated, the server sends invalidation reports to the clients caching this data.

The second strategy consists of a stateless server, which is not informed about the state of the data cached in its clients' caches. Thus, whenever data is updated, it broadcasts invalidation reports containing this specific data to all its clients.

With the stateless approach, whenever a client disconnects from the server for a period of time and reconnects again, it discards all the data items in its cache. But this method is very costly, especially if there is a large number of data in the client cache still valid.

To solve this problem, an energy-efficient cache invalidation scheme is proposed where cache validity is checked after a client is reconnected to the server. Because checking cache validity consumes energy and utilizes bandwidth, this scheme called Grouping with COld update-set REtention (GCORE) is proposed where the validity of data is checked on a group level (Wu, 1996).

The server database is divided into groups of data objects for the only purpose of cache validity checking after the client's reconnection to the server. Thus, the GCORE scheme tries to retain the cold update set and exclude the hot update set from a group when the group validity is to be checked. A cold update set is the set of data items which are rarely updated by the server transactions and a hot update set is the set of data items which are frequently updated in the server database by transactions.

When checking the group's validity, the GCORE scheme retains the cold update set in the group while the simple-grouping caching scheme invalidates the whole group if a data item from the group gets updated after the client is disconnected. Therefore, the GCORE scheme is energy efficient because it reduces downlink costs by retaining more valid data objects in the client's cache than a simple-grouping caching scheme or any other cache invalidation schemes.

2.6 Cache consistency strategy in a disconnected distributed environment

As mentioned earlier, the mobile computing environment is prone to frequent disconnections of mobile clients or mobile hosts (MHs) from the server caused by low battery power. These disconnections make the cache consistency in the mobile hosts difficult to maintain. Several cache consistency schemes were proposed to solve this problem. Most of these schemes use the invalidation report approach (call-back mechanism).

An asynchronous invalidation reports scheme is proposed where invalidation reports are broadcasted only when data items are updated unlike periodic invalidation reports scheme where the server broadcasts invalidation reports periodically (Kahol, 2001). In the proposed caching scheme, an additional memory, called the Home Location Cache (HLC), is used for each mobile host in order to maintain each data item cached by the MH and the time-stamp of each data item last updated. The HLC of each mobile host is found at a Mobile Switching Station (MSS) as shown in figure 2.4 presenting the structure of the mobile computing system used in this scheme:

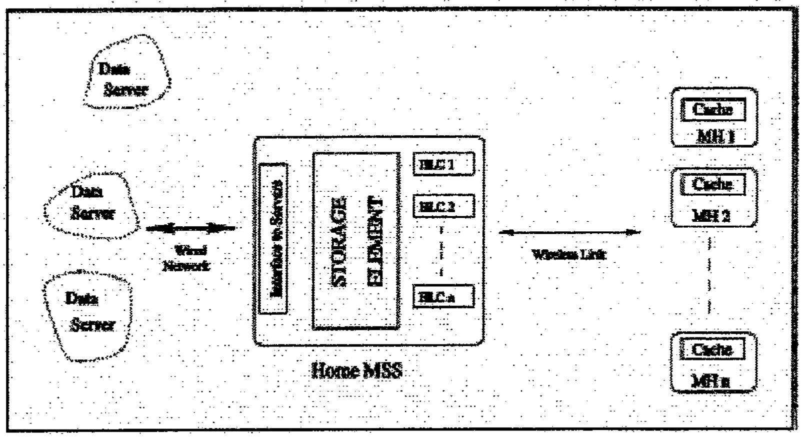


Figure 2.4 System architecture (Kahol, 2001)

When a MH receives a query, it checks for the data in its cache; if the data is valid, then the MH answers the query immediately by using its local cache. Otherwise, the MH sends a request for the data to the MSS, which in its turn sends a request to the server for the data. When the MSS receives the data, it saves an entry of each data item in the HLC and forwards the data to the MH.

Each MH preserves a time-stamp for its cache, called the cache time-stamp, which is the time of the last invalidation report received by the MH. The cache time-stamp is used to decide which invalidation reports to discard or to resend to the MH just reconnected. The invalidation reports with a time-stamp less than the cache time-stamp of the MH are discarded and the ones with a time-stamp greater than the cache time-stamp in the HLC are resent to the MH. In fact, when a MH reconnects to the server after a disconnection for a period of time, it sends a probe message containing the cache time-stamp. The HLC responses to the MH message by sending an invalidation report. Thus, the MH can recognize which data was updated during its disconnection by using the cache time-stamp. This scheme handles frequent disconnections of the MHs by maintaining the consistency of their caches with the cost of extra memory used for HLCs.

2.7 Ordered Update First with Order

Data broadcast has been one of the important methods in coping with the limitations of the mobile computing environment, such as the narrow bandwidth of the network and the frequent disconnections of the mobile clients. When the server periodically broadcasts invalidation reports, it informs all its clients about the out-dated data in their caches; hence, the clients do not have to congest the network with their queries to the server concerning data validation and the data consistency will be ensured in their caches after frequent disconnections. But this broadcast method does not control the flow of the update transactions on the data in the server while the data is being broadcasted. The clients might read inconsistent data values during their mobile transactions (MTs).

A broadcast method called Ordered Update First with Order (OUFO) has been designed to solve the problem of receiving inconsistent data by the client mobile transactions and to reduce the delay in accessing the data by these transactions (Lam, 2000). OUFO is an extension of another broadcast method called Update First with Ordering (UFO). The UFO maintains the concurrency control between the update transactions in the server and the mobile transactions of the clients but it is designed for unordered operations in mobile transactions. OUFO is defined as the UFO extended to deal with ordered data requests in mobile transactions.

2.7.1 OUFO protocol description

OUFO is designed to ensure consistency in data items broadcasted by the server and read by ordered operations in mobile transactions. Hence, the goal of the OUFO protocol is to ensure that a broadcast transaction (BT) should always occur after an update transaction (U) of the data in the server database (U $\rightarrow$ BT); that means the order of these transactions is the most important (Lam, 2000). Data inconsistency occurs when an update transaction is processed in the server database during a broadcast transaction of the data. In this case, the broadcast transaction would come before the update transaction (BT $\rightarrow$ U).

2.7.2 Data inconsistency resolution

On the server side, to resolve the data inconsistency occurrence between the update transaction and the broadcast transaction, data in the server database is re-broadcasted. By re-broadcasting a data item, the updated value of the data item will be included in the last BT. Thus, the order of the transactions is reversed: (BT $\rightarrow$ U) becomes (U $\rightarrow$ BT).

On the mobile client side, when a mobile transaction composed of a set of ordered read operations is processed, the read operations will be started in sequence. The executing operation will access a data item if the latter is received from a broadcast transaction. If a data item has already been read by a read operation and were re-broadcasted by the server, the MT will be restarted from the read operation requesting that data item.

Consequently, The OUFO protocol reduces the data inconsistency between the update transaction and the broadcast transaction of the data by re-broadcasting the data and preserving the order U $\rightarrow$ BT, which provides the last updated values of the data items to the mobile client transactions.

2.8 PRO-MOTION

In mobile databases, the frequent disconnections of the clients, due to limited battery power and the expense of the wireless network connection, constitute a big challenge for the continuity of a transaction processing. An infrastructure for transaction processing called PRO-MOTION was proposed in a mobile environment (Mazumdar, 1999). PRO-MOTION permits a transaction to continue execution while a client or a mobile host (MH) is disconnected from the server database.

A compact is an object encapsulating replicated or cached data. PRO-MOTION uses compacts to achieve the goal of a transient process of a transaction. When a mobile host requests data from the server, the data is cached in the form of compacts composed of obligations, methods, consistency rules, data, and state information as shown in figure 2.5:

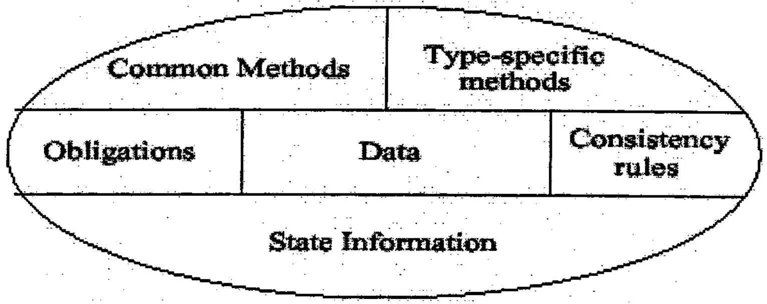


Figure 2.5 Compacts as objects (Mazumdar, 1999)

On the server and the client sides, there are agents dealing with the communication between the server database and the MH. On the server side, the mobility manager helps in continuing the process of a transaction in case a client disconnects directly after sending its transaction message to the server. On the client or mobile host side, the compact agent handles the compacts stored in the MH and controls the execution of transactions.

A compact is created by the compact manager (front-end of the server) including the data itself and all related control and status information. The compact is stored in a compact store before it is transmitted to the MH. Once it is received by a MH, a compact is saved in a compact registry for tracking its status. A compact is deleted from the compact registry and stored when it is needed neither by the MH nor by the server. With this infrastructure, a transaction can continue execution even if the client is disconnected from the server.

2.9 Policy-based management for distributed systems

Large-scale distributed systems need security and management for the large number of data objects that they contain. The management system has to be automated in order to diminish the cost of turning off the whole system in case there is a change in requirements. The management behaviour is represented by a set of policies which control the access on the data objects and provide levels of security for the distributed systems. A policy can be applied to a group of people with the same access rights for a group of data objects. Thus, a policy is not defined for a single data object but for a group of data objects to organize a large-scale distributed system (Lupu, 1999).

2.9.1 Domains

Objects in a large system can be divided into groups, called domains, according to different characteristics such as their types and functionalities. Domains help the management system to define a policy and provide authority to a group of objects. A domain can be a part of another domain and is called a subdomain of the latter. A component in a subdomain B of a domain A is not considered a direct member of A but an indirect member of A. An overlap is when a component exists in more than one domain in the same time. For example, a component E exists in both domains B and C so there is an overlap between B and C as shown in figure 2.6:

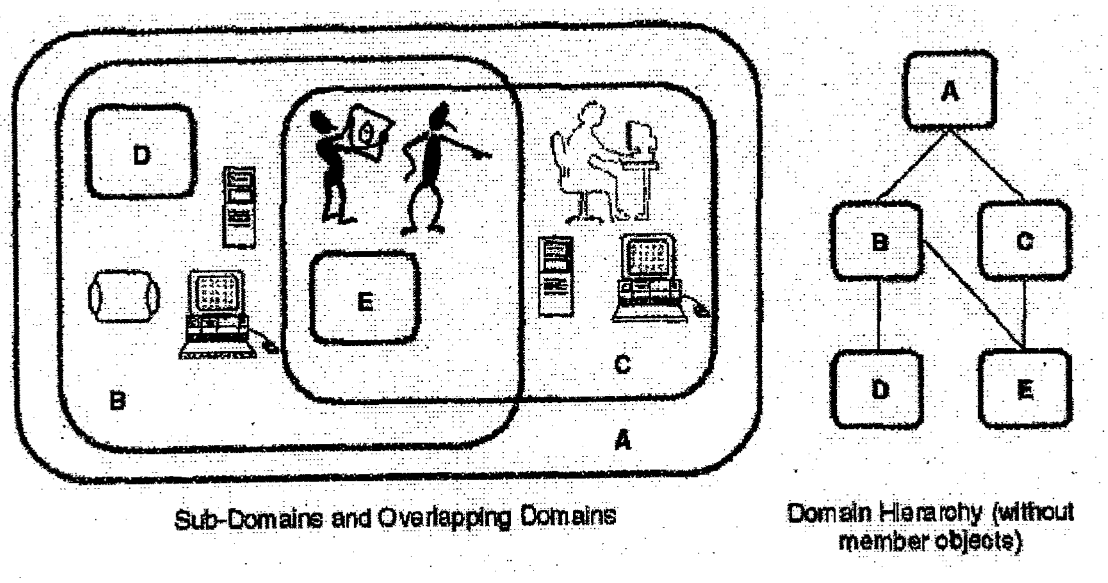

D
E
C
B
A
Sub-Domains and Overlapping Domains
A
B
C
D
E
Domain Hierarchy (without member objects)

Figure 2.6 Domains (Lupu, 1999)

A policy, applying to a domain, is also applied to its subdomain. So, in figure 2.6, if a policy is applied to domain B is also applied to its subdomains D and E. If a policy is not applied to a specific object anymore, the object is removed from its domain without any change in the policy itself. Thus, policies, being applied to the level of domains of objects, make the management system simple to handle in large distributed system.

2.9.2 Policies

Because distributed systems need management, policies are made to provide security and control the access on data objects contained in these systems. There are two types of policies to identify management behaviour: Authorization policies and obligation policies. Authorization policies provide access control to a group of objects. The access to objects can be either authorized by positive authorization policies or unauthorized by negative authorization policies. Obligation policies identify the actions that should or should not be done to a set of data objects. Thus, the obligation policies can be positive or negative. All policies have a high abstraction level in large distributed systems because they provide authorizations and obligations to objects through the level of domains.

The subject of a policy is defined as an agent or a manager to whom the previously mentioned policies are applied. The target of a policy is the data object on which a policy performs an action.

The action is what an authorization or obligation policy performs to control the access or provide obligations on data objects. A policy may contain a set of sequential actions to perform.

2.9.3 Roles

Managers or agents are organized to fit into manager positions or domains which comprise a set of policies that defines the authorizations and obligations of each position. A role is defined as a manager position with the group of all policies identifying the privileges for that manager position. Figure 2.7 represents the structure of a management role.

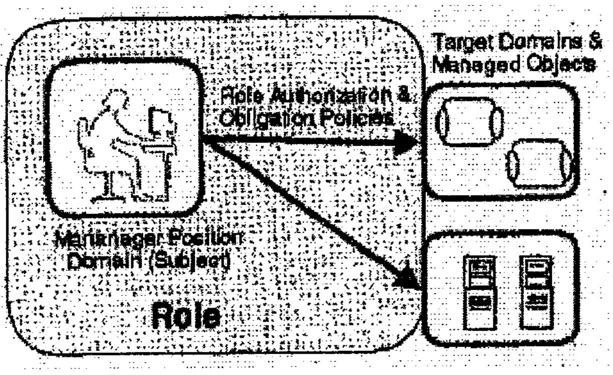

Target Domains & Managed Objects
Role Authorization & Obligation Policies
Manager Position Domain (Subject)
Role

Figure 2.7 Management roles (Lupu, 1999)

2.9.4 Conflicts

Conflicts occur when policies with opposite signs have subjects, targets and actions in common, i.e. an overlap occur between the subjects, targets and actions as shown in figure 2.8. In this case, the conflicts are called modality conflicts.

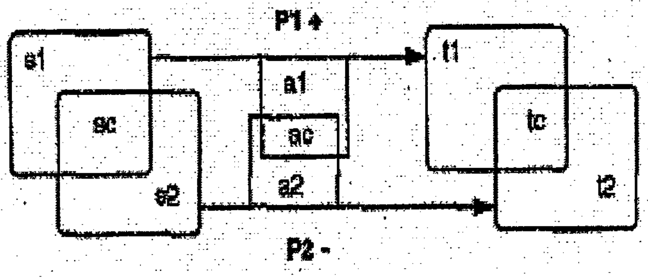


Figure 2.8 Overlap between subjects, targets, and actions (Lupu, 1999)

A solution to resolve a conflict is to give higher priority to negative policies over positive policies. In fact, when a conflict is detected, subjects are forbidden to perform any operation on objects because precedence is given to negative policies over positive ones. In this way, all conflicts are resolved while security is provided in distributed systems but a lack of flexibility exists.

2.10 Peer database management system

In the database technology, studies have been made on database mobility in order to develop coordinating mobile databases. Coordination between databases is a very important issue especially on the network where database disconnections are frequent. In addition, locations of databases can change leading to a problem in accessing these databases when there is a user request for data. Therefore, a Peer Database Management System (PDBMS) is proposed to resolve these problems (Fausto, 2004).

2.10.1 PDBMS characteristics

PDBMS is a self-reliant database system and has a small size which makes it easy to carry. Also, PDBMS is independent of any platform and can connect with other databases in order to answer queries and data requests. It runs on a peer-to-peer networking model which faces problems concerning the mobility or lack of availability of other databases.

2.10.2 Interest groups and acquaintances

Database coordination is proposed as a solution to the problems of the peer-to-peer databases (Fausto, 2004). It consists of two main concepts which are interest groups and acquaintances. An interest group is composed of a group of nodes about certain topics in a form of a tree where the root is the most general node and as one goes down to the children the nodes become more specific. An example of an interest group is shown in figure 2.9:

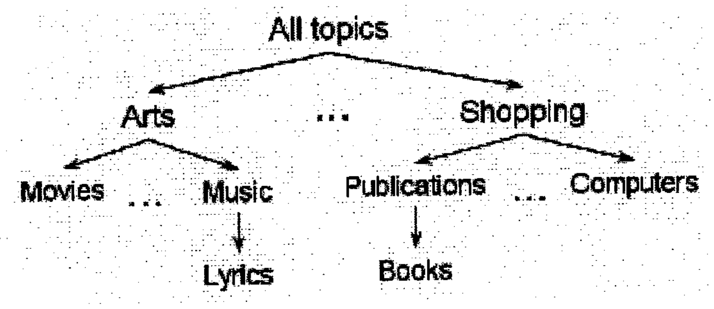


Figure 2.9 Interest group hierarchy (Fausto, 2004)

An acquaintance is a node that is known by another node. If a node is an acquaintance for another node, it is not necessary that the latter is also an acquaintance for the former. If a node has an acquaintance, it is called an acquainted node for the latter.

Chapter 3

Scalable Asynchronous Cache Consistency Scheme

For mobile computing environment, there are two types of cache consistency maintenance algorithms: stateless and stateful. In the stateless approach, the server is not informed about the client's cache content so it periodically broadcasts a data invalidation report (IR) to all mobile users (MUs). In such algorithms, mobile support station (MSS) does not maintain any state information about its mobile user caches (MUCs), thus allowing simple database management for the server cache (SC) but poor scalability and ability to support user disconnectedness and mobility. On the other hand, the stateful approach is used with large database systems at the cost of complex server database management. The communication in these approaches is reliable between MUs and the mobile support station (MSS) for IR broadcast, which means an MU has to send back an acknowledgement for each IR received from the server broadcast. Thus, if a mobile user is disconnected, it does not receive any IR broadcast and the server, which does not get any acknowledgement, has to resend the IR again.

This chapter describes a novel cache consistency maintenance algorithm, called Scalable Asynchronous Cache Consistency Scheme (SACCS), which combines the positive features of both stateless and stateful approaches (Wang, 2003). Under unreliable communication environments, SACCS provides small stale cache hit probability; A stale cache hit for a data entry occurs at a connected MU when the MU misses the IR of the data entry and when the update time for the data entry is during its TTL (time-to-live period).

The scalable asynchronous cache consistency scheme (SACCS) was proposed to maintain the mobile user cache (MUC) consistency for systems with read-only transactions. Maintaining minimum state information, SACCS has the positive features of both the stateless and stateful approaches. The advantage of SACCS over the asynchronous stateful algorithm, where the mobile support station (MSS) has to identify all data objects for every MUC, is that the MSS needs to recognize only the valid state of MUC data objects in SACCS. On the other hand, the advantage of SACCS over the stateless approach is that in SACCS, the server does not need to periodically broadcast IRs to all MUs, thus reducing the number of IR messages sent through the network. In addition, the broadcast channel efficiency progresses in SACCS, which has uncertain and ID-only states in MUC in order to handle sleep-wakeup patterns (or disconnection-reconnection patterns) and mobility of all MUs.

3.1 SACCS Key Features

SACCS consists of four key features which make it a highly efficient, scalable and low complexity algorithm. Flag bits are used at SC and MUC, an ID is used for every data entry in MUC after its invalidation. All valid data entries in MUC become in an uncertain state when an MU reconnects (or wakes up), and each cached data entry has a TTL (time-to-live).

When an MU retrieves a data object from the MSS, the flag bit in SC is set so the MSS has to broadcast the IR to all MUs. On the other hand, if the flag bit is not set, the MSS does not have to send an IR to the MUs. In this case, bandwidth utilization is reduced by avoiding unnecessary IRs. When a data entry in an MUC becomes invalid, it is removed and only its ID is kept (set to ID-only state), making the management of sleep-wakeup pattern simple. When an MU wakes up, all valid data items in its cache are set to uncertain state. TTL is estimated by an MSS for each data item depending on its update history. If the last update time of a data item added to its TLL equals to the current time, the data item is set to an uncertain state. This process helps in preventing a stale data object from being accessed in an MUC for a long time due to an incorrect IR arrival to the MU (IR loss).

3.2 Data Structures and Communication Messages

The SACCS uses a data structure in SC composed of dx,tx,lx,fx where dx represents the data object, tx the last update time for the data object, lx the estimated time-to-live (TTL) and fx a flag bit (Wang, 2003).

The data structure in MUC is composed of dx,tsx,llx,sx where tsx is the time stamp indicating the last update time for the cached data object dx, llx is an associated TTL and sx a two-bit flag representing four data entry states: 0 (valid dx), 1 (uncertain dx), 2 (uncertain dx with a waiting query), and 3 (ID-only state) (Wang, 2003). The movement of a data entry from one state to another is shown in figure 3.1:

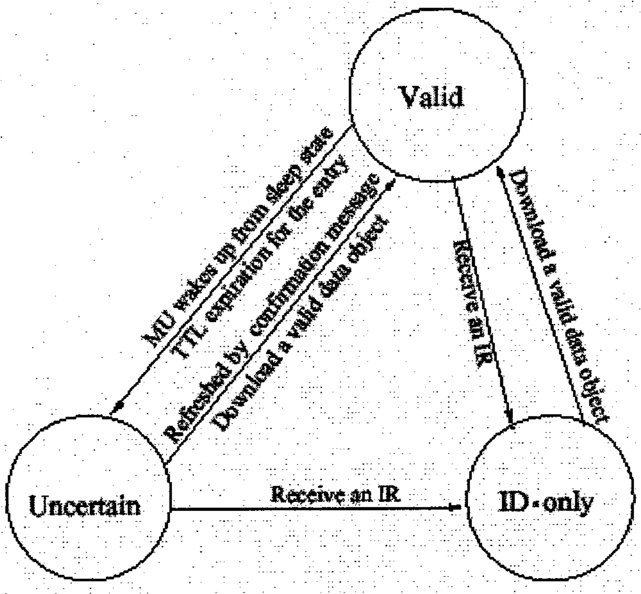
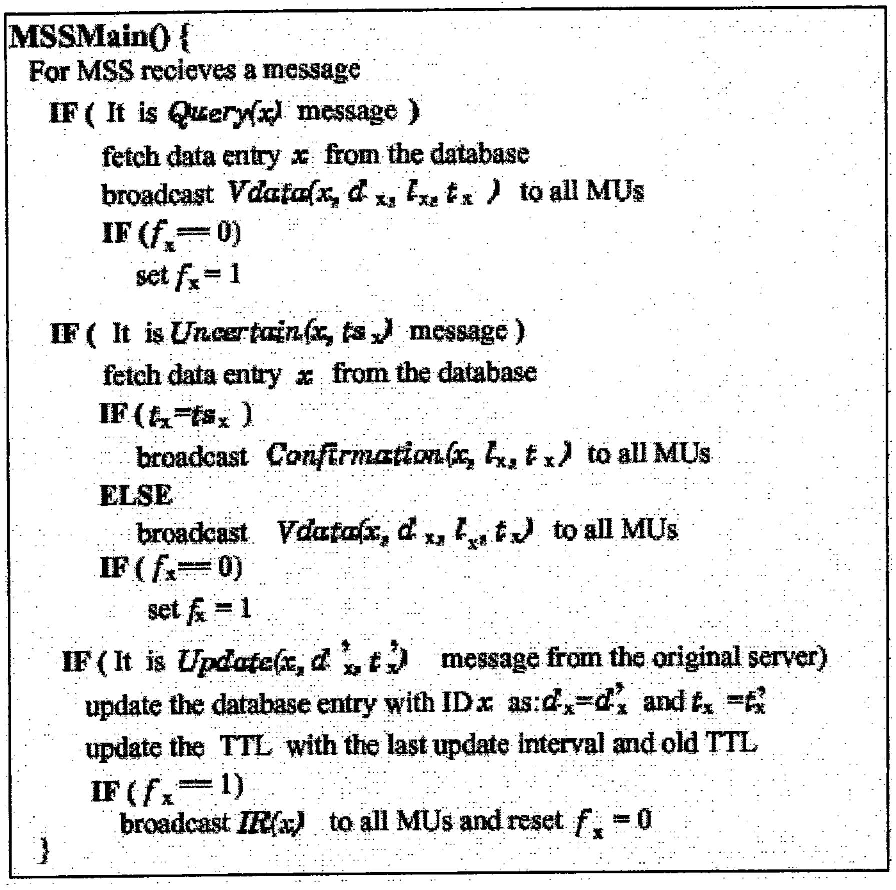

Valid
MU wakes up from sleep state
TTL expiration for the entry
Refreshed by confirmation message
Download a valid data object
Receive an IR
Download a valid data object
Uncertain
Receive an IR
ID-only

Figure 3.1 Cache entry state diagram (Wang, 2003)

An MSS and its MUs communicate with each other by exchanging messages. Incorporating the data structures mentioned previously, the communication messages between the MSS and all the MUs are described in table 3.1:

Table 3.1 Communication messages in SACCS (Wang, 2003)

Name	Sender	Receiver	Comments
$Update(x, d'_x, t'_x)$	original servers	MSS	indicating d_x has been updated to d'_x at time t'_x
$Vdata(x, d_x, l_x, t_x)$	MSS	MUs	broadcast valid data object d_x with update time at t_x and TTL=l_x
$IR(x)$	MSS	MUs	indicating cached d_x is invalid
$Confirmation(x, l_x, t_x)$	MSS	MUs	indicating d_x is valid if $ts_x = t_x$; the associated TTL=l_x
$Query(x)$	MUs	MSS	query for data object d_x
$Uncertain(x, ts_x)$	MUs	MSS	verifying if d_x in uncertain state with update time ts_x is valid



```
MSSMain() {
  For MSS recieves a message
    IF ( It is $Query(x)$ message )
        fetch data entry $x$ from the database
        broadcast $Vdata(x, d_x, l_x, t_x)$ to all MUs
        IF ($f_x == 0$)
           set $f_x = 1$
    IF ( It is $Uncertain(x, ts_x)$ message )
        fetch data entry $x$ from the database
        IF ( $t_x = ts_x$ )
           broadcast $Confirmation(x, l_x, t_x)$ to all MUs
        ELSE
           broadcast $Vdata(x, d_x, l_x, t_x)$ to all MUs
        IF ( $f_x == 0$)
            set $f_x = 1$
     IF ( It is $Update(x, d'_x, t'_x)$ message from the original server)
       update the database entry with ID $x$ as: $d_x = d'_x$ and $t_x = t'_x$
       update the TTL with the last update interval and old TTL
       IF ( $f_x == 1$)
          broadcast $IR(x)$ to all MUs and reset $f_x = 0$
 }
```

Figure 3.2 MSSMain() pseudocode (Wang, 2003)

The MSSMain(), whose pseudocode is presented in the figure above, describes how the MSS deals with the update of data items by the original server and the queries sent by the MUs. In the MSSMain() procedure, if the MSS receives a request for a data item from an MU, it gets the data entry from its database and broadcasts it to all MUs. If the flag of the data entry is 0, then it is set to 1. When the message received from an MU is an uncertain message for a data item, a comparison is done between the time stamps of the data item in MSS and in the MU. If the time stamps are of equal values (i.e. the uncertain data item is valid), then the MSS broadcasts a confirmation message concerning this data item or else it broadcasts a valid data item. If the flag of the data entry is 0 then it is set to 1. In addition, if a data item is sent in an update message by the original server to MSS, then the data item is updated in the

MSS database with a new estimated TTL. In this case, if the flag of the data entry is 1, then an invalidation report for this data entry is broadcast to all MUs and the flag is reset to 0.

```
MUMain() {
  For MU recieves a message
    IF ( It is a Request for $d_x$ )
      IF ($d_x$ is valid in cache list )
        answer the request with cached data object $d_x$
        move the entry into cache list head
      ELSE IF ($d_x$ is in uncertain state )
        send Uncertain($x$, $ts_x$) message to MSS
        add ID $x$ to query waiting list
        set $s_x = 2$ and move the entry into cache list head
      ELSE IF (the entry $x$ is ID-only entry in cache )
        send Query($x$) to MSS
        remove the entry in cache
        add ID $x$ to query waiting list
      ELSE
        send Query($x$) to MSS
        add ID $x$ to query waiting list
    IF ( It is Vdata($x$, $d_x$, $l_x$, $t_x$) message )
      IF ($x$ is in query waiting list )
        answer the request with $d_x$
        remove the uncertain entry $x$ if it exists in cache
        add the valid entry $x$ at cache list head
      ELSE
        IF (entry $x$ is ID-only entry in cache )
          download $d_x$ to orginal entry location in cache
        ELSE IF (entry $x$ is uncertain entry in cache )
          IF ($ts_x < t_x$ )
            download $d_x$ to orginal entry location in cache
            set $ts_x = t_x$, $ll_x = l_x$ and $s_x = 0$
          ELSE
            set the entry to valid entry ( $s_x = 0$)
    IF ( It is IR($x$) message )
      IF ($d_x$ is valid or uncertain in cache)
        delete $d_x$ and set $s_x = 3$
    IF ( It is Confirmation($x$, $l_x$, $t_x$) message)
      IF ( the entry $x$ is uncertain entry in cache list)
        IF ($ts_x = t_x$)
          set $s_x = 0$ and $ll_x = l_x$
          IF ( ID $x$ is in query waiting list)
            answer the request with $d_x$
        ELSE
          delete $d_x$ and set $s_x = 3$
  IF( MU wakes up from the sleep state )
    set all valid ($s = 0$) entry into uncertain state ( $s = 1$)
  IF ( The TTL expires for entry $x$ )
    set the valid entry $x$ into uncertain state ( $s_x = 1$)
}
```

Figure 3.3 MUMain() pseudocode (Wang, 2003)

The MUMain(), whose pseudocode is presented in the figure above, describes how the MU deals its own queries and broadcast messages. In the MUMain() procedure, when the message received is a request for a data item, the data in the cache is checked; if it is valid, the request is answered immediately and the data entry is moved to the head of the cache list. If it is in an uncertain state, the MU sends an uncertain message to MSS. If it has ID-only entry in the MU cache, a request message for that data entry is sent to the MSS. When the message received is a valid data (Vdata) message, if there is a request for that data item, it is answered and the data item is cached. If the data item is in an uncertain state, it is refreshed. If the data is an ID-only entry, the data item is saved as a valid data in the MU cache. When an IR message for a data entry is received, the data entry is removed from the MU cache and ID-only entry is kept. When the message received is a confirmation message for a data entry, a comparison is made between the time stamps of the data in MSS and MU. If their values are equal then the data entry is refreshed in the MU cache or else the data entry is removed and set into ID-only state. In addition, when an MU wakes up, all the data entries in its cache are set into an uncertain state. And when the TTL of any valid data entry in the MU cache expires, the data is set into an uncertain state.

3.5 MUC consistency maintenance and efficiency in SACCS

The MUC consistency is maintained in SACCS by the use of a flag bit for each data entry. If an MU retrieves a data entry from MSS, the flag is set to 1, indicating that this data entry is found in a certain MU. In this case, whenever MSS receives an update message for that data entry, it broadcasts a corresponding IR to all MUs and the flag bit is reset to 0. When an IR for a data is broadcasted, the data is set to ID-only state or invalidated in a connected MU. If the MU is disconnected, it does not receive any IR. When it gets reconnected, all the data items in its cache are set to an uncertain state. In this way, SACCS preserves cache consistency between MSS and MUs.

In addition, SACCS is a scalable and efficient cache consistency algorithm. In fact, the associated flag bit with each data object helps in reducing the number of IRs to broadcast. When a data is updated in MSS, IR for the data with a flag bit set to 1 only is to be broadcasted. Thus, the bandwidth consumption is reduced.

3.6 Additional techniques

The SACCS described in the previous chapter, is a scalable and efficient cache consistency maintenance algorithm, which maintains data consistency between SC and MUCs. The cache replacement algorithm adopted in SACCS to manage the mobile user caches is the LRU (Least Recently Used) also described in the previous chapter.

Different cache replacement algorithms are proposed in order to show their impact on SACCS. Additional cache replacement algorithms used are the following:

- LFU – Least Frequently Used
- MFU – Most Frequently Used
- MRU – Most Recently Used
- FIFO – First In First Out

Each of these cache replacement algorithms will be described to show how it works when used with SACCS. A simulation of SACCS is done with every algorithm and the variation of the movement of an accessed data object in the mobile user cache list is shown in the simulation code. In fact, with the LRU algorithm used, the data entry at the cache list tail is removed to make space for new cached data entries if the cache is full. And when a data entry is accessed or hit, it is moved to the head of the cache list. Thus, the performance of the system is affected by the difference in the number of data hits in the cache list and in other results discussed later.

3.6.1 LFU (Least Frequently Used)

The LFU is defined as the following: when a mobile user cache is full, the least frequently used data object is removed to make space for a new cached data object. In the case where LFU is used with SACCS, the number of accesses for each data entry in the mobile user cache list is counted. The data entry with the minimum number of accesses is moved to the tail of the list.

3.6.2 MFU (Most Frequently Used)

The MFU is defined as the following: when a mobile user cache is full, the most frequently used data object is removed to make space for a new cached data object. In the case where the MFU is used with SACCS, the number of accesses for each data entry in the mobile user cache list is counted. The data entry with the maximum number of accesses is moved to the tail of the list.

3.6.3 MRU (Most Recently Used)

The MRU is defined as the following: when a mobile user cache is full, the most recently used data object is removed to make space for a new cached data object. When the MRU is used with SACCS, a data object is moved to the tail of the cache list when it is hit or newly cached.

3.6.4 FIFO (First In First Out)

The FIFO is that the first data entry stored in a mobile user cache is removed to make space for a new cached data item if the cache is full. When the FIFO is used with SACCS, the oldest data entry exists at the tail of the mobile user cache list. In fact, whenever a new data item is cached, it is added at the head of the cache list if the cache is not full.

3.7 MUC Management

When the SACCS is used with one of the proposed cache replacement algorithms, data entries are removed from the tail of the list to make space for new entries cached if the mobile user cache is full. Only data entries with no awaiting queries can be removed from the tail of the list when needed. Data objects with uncertain or ID-only state are refreshed in their original places in the cache list.

A simulation of SACCS is made, using each of the previous cache replacement algorithms showing their impact on the results of SACCS and the performance of the system. Statistical results are presented in order to show the advantages and disadvantages of each of these cache replacement algorithms when used with SACCS.

Chapter 4

Experimental Results

The simulation of SACCS is running under fixed parameters and conditions such as fixed number of documents exchanged and mobile users, fixed mobile cache sizes, specified periods of time, etc for each of LRU, which is the based cache replacement algorithm for SACCS mobile user cache management, and the proposed cache replacement algorithms: LFU, MRU, MFU, and FIFO. Using these cache replacement algorithms with SACCS affects the system performance with different factors such as the simulation results of the cache miss ratio, the total miss, the total hit in MUCs and the delay of data during certain periods of time. A comparison between the cache replacement algorithms results on SACCS is done to show the advantages and disadvantages of every cache replacement algorithm used. The statistical results obtained are based on eight simulation time units with an interval of 50000 microsec of simulation time. The simulation results of the LRU, LFU, MFU, MRU, and FIFO are found respectively in the appendixes A-1, A-2, A-3, A-4, and A-5.

4.1 Cache Miss Ratio

The cache miss ratio is the ratio of the number of unfound data items in the cache over the number of all requested data. The cache miss ratio simulation results are presented in figure 4.1, where the statistical results of each cache replacement algorithm miss ratio versus specific periods of simulation time are shown.

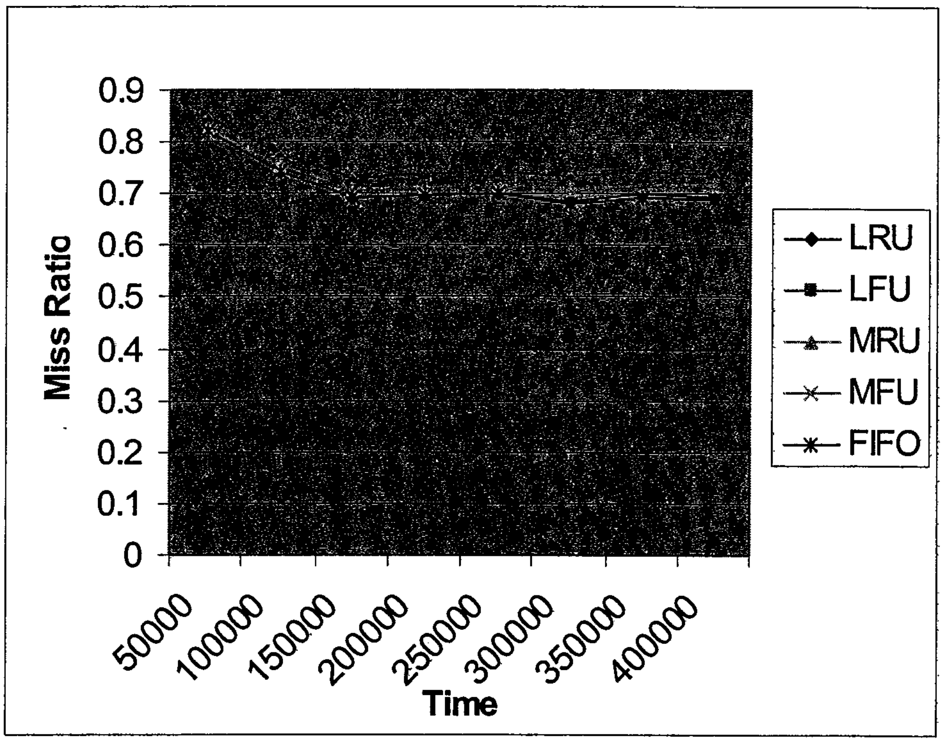


Figure 4.1 Miss ratio versus simulation time

Among all the cache replacement algorithms, it is shown that the FIFO has the lowest miss ratio with the average of 0.69053, while the MFU has the highest miss ratio with average of 0.750494. With the SACCS, the first in first out replacement strategy ensures the highest cache hit ratio (i.e. the lowest cache miss ratio) among the other strategies so it handles the mobile users' queries for data faster than the other strategies. Thus, using the FIFO with SACCS improves the performance of the system because it has a lower miss ratio result than the based LRU with a miss ratio average of 0.696047.

4.2 Total Delay

The total delay is defined as the period of time between the time the request is issued and the time the result is received by the mobile user application. The total delay simulation results are presented in figure 4.2, where the statistical results of each cache replacement algorithm total delay time versus specific periods of simulation time are shown.

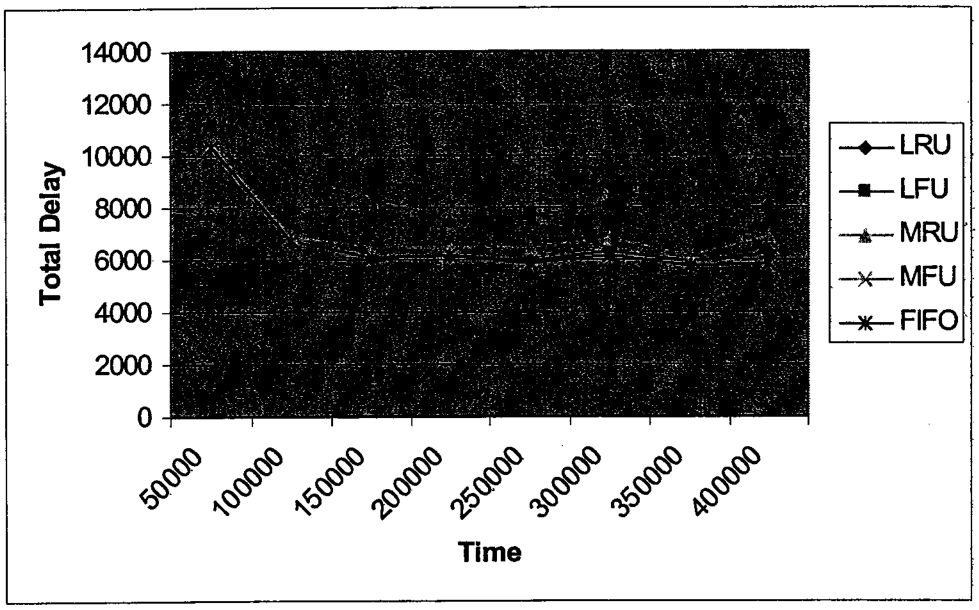


Figure 4.2 Total delay versus simulation time

Among all the cache replacement algorithms, it is shown that the FIFO has the lowest total delay with the total average of 14893.7, while the MFU has the highest total delay with the total average of 21953. When used with the SACCS, the first in first out replacement strategy ensures the lowest period of delay time between a request and its answer among the other strategies because of different factors. The miss ratio is one of the factors that affects the delay time. When a requested data is invalid or not found in the mobile user cache, it is time-consuming for the MU to get the data from the server. And in this case, the FIFO is previously shown to have the lowest miss ratio among other replacement algorithms, which makes the total delay lower. Thus, using the FIFO with SACCS improves the performance of the system because it has a lower total delay result than the based LRU with a total delay average of 15590.5.

4.3 Total Hit

The total hit is defined as the number of data items hit or accessed in the mobile user cache. The total hit simulation results are presented in figure 4.3, where the statistical results of each cache replacement algorithm total hit versus specific periods of simulation time are shown.

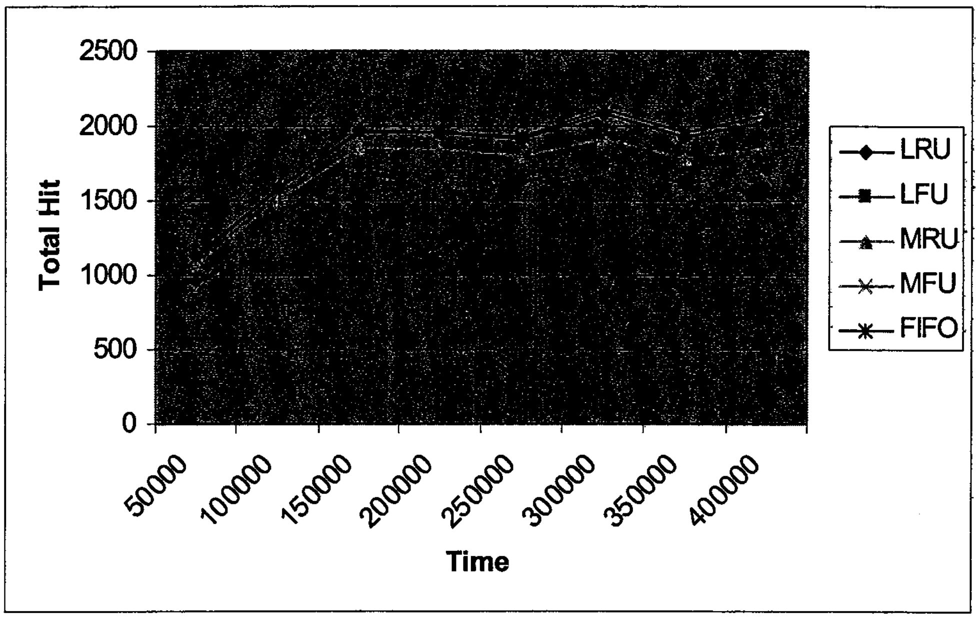


Figure 4.3 Total hit versus simulation time

Among all the cache replacement algorithms, it is shown that the FIFO has the highest total hit with the total average of 5184, while the MFU has the lowest total hit with the total average of 4166. The total hit affects the system performance because it affects the delay time of a user request for data and the network traffic. In fact, as the number of data hit gets higher, users' queries are answered more quickly because fetching the data from the user cache is much faster than getting it from the server database. And the network traffic is reduced because the number of users' requests for data from the server database is lower as the data hit in the users' caches is higher. Consequently, using the FIFO with SACCS improves the performance of the system because it has a higher total hit result than the based LRU with a total hit average of 5091.

4.4 Total Miss

The total miss is defined as the number of data items invalid or missed in the mobile user cache. The total miss simulation results are presented in figure 4.4, where the statistical results of each cache replacement algorithm total miss versus specific periods of simulation time are shown.

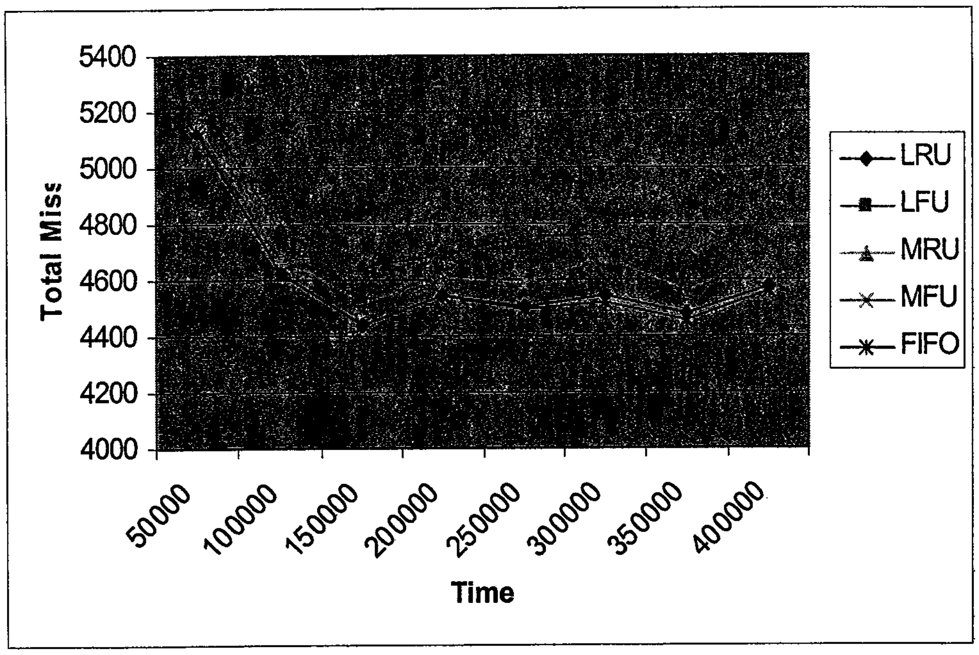


Figure 4.4 Total miss versus simulation time

Among all the cache replacement algorithms, it is shown that the FIFO has the lowest total miss with the total average of 11565, while the MFU has the highest total miss with the total average of 12522. Because the total miss factor is inversely proportional to the total hit factor, the performance of the system is reduced as the total miss gets higher. In fact, when a requested data is missed in the user cache, a delay exists for the data should be brought from the server database through the network to the user. A data miss costs the system a delay time and additional network use. Consequently, using the FIFO with SACCS improves the performance of the system because it has a lower total miss result than the based LRU with a total miss average of 11656.

4.5 Performance Evaluation

As already shown in the previous charts of cache miss ratio, total delay, total hit, and total miss, the FIFO has the best results among the cache replacement algorithms: LRU, LFU, MRU, and MFU. The FIFO has the lowest miss ratio, the lowest total delay, the highest total hit, and the lowest total miss among all proposed replacement algorithms. All these factors affect the system performance of SACCS by affecting the speed of handling the users' requests for data and the network traffic. Thus, the system performance of SACCS with the based cache replacement algorithm LRU can be improved by using the FIFO for MUC management instead of the LRU.

Chapter 5

Conclusion and Future Work

Mobile computing is one of the most important technologies in the computer field. Data caching is imposed when data is exchanged between the mobile users and the server in order to improve the performance of the system. Maintaining cache consistency is one of the major problems arising in mobile computing. A cache consistency maintenance algorithm called Scalable Asynchronous Cache Consistency Scheme (SACCS), gathering both stateless and stateful approaches, was proposed with the Least-Recently-Used (LRU) replacement algorithm to manage mobile user caches. A detailed description for the SACCS is done.

In this work, different cache replacement algorithms are proposed to be used with SACCS and compared with the LRU already used. The impact of the proposed cache replacement algorithms LFU, MRU, MFU, and FIFO on the system performance is studied. A simulation of SACCS with each of these algorithms is done, producing results evaluating the system performance. The simulation results, such as the miss ratio, total delay, total hit, and total miss are compared showing the variation of the performance of the system. Statistical results point out the advantages and disadvantages of each proposed cache replacement algorithm compared to the based LRU.

As a result, the FIFO used with SACCS is shown to be the cache replacement algorithm with the best simulation results compared to all the cache replacement algorithms used. The FIFO has the lowest miss ratio, the lowest total delay time, the highest total hit, and the lowest total miss. In this case, the system performance of

SACCS is improved when the FIFO is used instead of the based LRU for MUC management.

In future work, the performance of the system of SACCS can be improved in different ways. The MSS cache management algorithm can be enhanced to transfer data effectively and answer the users' request for data more quickly. Also the data consistency between the server and the mobile users could be ensured by combining the SACCS with one of the proposed cache invalidation schemes. A further study should be made to choose which cache invalidation scheme combined with the SACCS leads to more satisfactory results concerning system performance and other system factors. A simulation of the combination of the two schemes could be made to show and compare the new results of SACCS with the old ones. In addition, a further study could be made for the IR management in SACCS and how it could be improved in order to have better bandwidth utilization and ensure data consistency with a better system performance. Finally, the SACCS can also be enhanced to be used for read-write systems.

References

Mazumdar, S.Mazumdar and P.K.Chrysanthis (1999). *Achieving Consistency in Mobile Databases through Localization in PRO-MOTION.* [Online]. Available: http://www.csse.monash.edu.au/projects/MobileComponents/mdds99/published/02810082.pdf

Hou, Hou, W.C., C.F. Wang, and M. Su (2005). Composing Optimal Invalidation Reports for Mobile Databases. *Journal of Digital Information Management*, 3 No.2, 126-132.

Seetha, A.J.Seetha and A.Kannan (2001). Maintaining data consistency in mobile database broadcasts. [Online]. Available: http://www.gisdevelopment.net/technology/mobilemapping/techmp001pf.htm

Qi Lu, Q.M. Satyanarayanan (1995). *Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions.* [Online]. Available: http://www.cs.cmu.edu/~coda/docdir/hotos95-iot.pdf

Wang, Z.Wang, S.K.Das, H.Che and M.Kumar (2003). *Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments.* [Online]. Available: http://www.cse.uta.edu/research/publications/Downloads/CSE-2003-9.pdf

Chrysanthis, P.K.Chrysanthis and E.Pitoura (2000). *Mobile and Wireless Database Access for Pervasive Computing.* [Online]. Available: http://www.cs.uoi.gr/~pitoura/icde00-tutorial.ppt

Lam, K-Y.Lam, E.Chan, H-W.Leung and M-W.Au (2000). *Broadcasting Consistent Data to Mobile Clients with Local Cache.* [Online]. Available: http://www.cs.cityu.edu.hk/~rtmm/comad2000.pdf

Yao, J-F.Yao and M.Dunham (1998, August). *Caching Management of Mobile DBMS.* [Online]. Available: http://engr.smu.edu/~mhd/pubs/99/icae.doc

Kumar, V.Kumar (2006). *Mobile Database Systems.* [Online]. Available: http://www.sice.umkc.edu/~kumarv/co-tutorial.ppt

Cao1, G.Cao (2002). *Adaptive Power-Aware Cache Management for Mobile Computing Systems.* [Online]. Available: http://www2002.org/CDROM/poster/88.pdf

Wu, K-L.Wu, P.S.Yu and M-S.Chen (1996). *Energy-Efficient Caching for Wireless Mobile Computing.* [Online]. Available: http://www.doi.ieeecomputersociety.org/10.1109/ICDE.1996.492181

Chuang, P-J.Chuang and Ch-Y.Hsu (2004). *An Efficient Cache Invalidation Strategy in Mobile Environments*. [Online]. Available: http://www.doi.ieeecomputersociety.org/10.1109/AINA.2004.1283799

Cao2, G.Cao (2002). *On Improving the Performance of Cache Invalidation in Mobile Environments*. [Online]. Available: http://www.doi.ieeecomputersociety.org/10.1109/PERSER.2004.16

Kahol, A,Kahol, S.Khunara, S.K.S.Gupta and P.K.Srimani (2001, July). *A Strategy to Manage Cache Consistency in a Disconnected Distributed Environment*. [Online]. Available: http://www.doi.ieeecomputersociety.org/10.1109/71.940744

Fausto, F.Giunchiglia and I.Zaihrayeu (2004). *Coordinating Mobile Databases*. [Online]. Available: http://www.p2pkm.org/2004/Camera_Ready/1568938498.pdf

Choo, Choo (2001). *Unix And C/C++ Runtime Memory Management For Programmers*. [Online]. Available: http://users.actcom.co.il/~choo/lupg/tutorials/unix-memory/unix-memory.html

Lupu, E.C.Lupu and M.Sloman (1999, November). *Conflicts in Policy-Based Distributed Systems Management*. [Online]. Available: http://www.doc.ic.ac.uk/~mss/emil/tse.pdf

Barr, R.Barr (1999, March). *Mobile distributed systems*. [Online]. Available: http://www.rimonbarr.com/repositiry/cs714/mobile.ppt

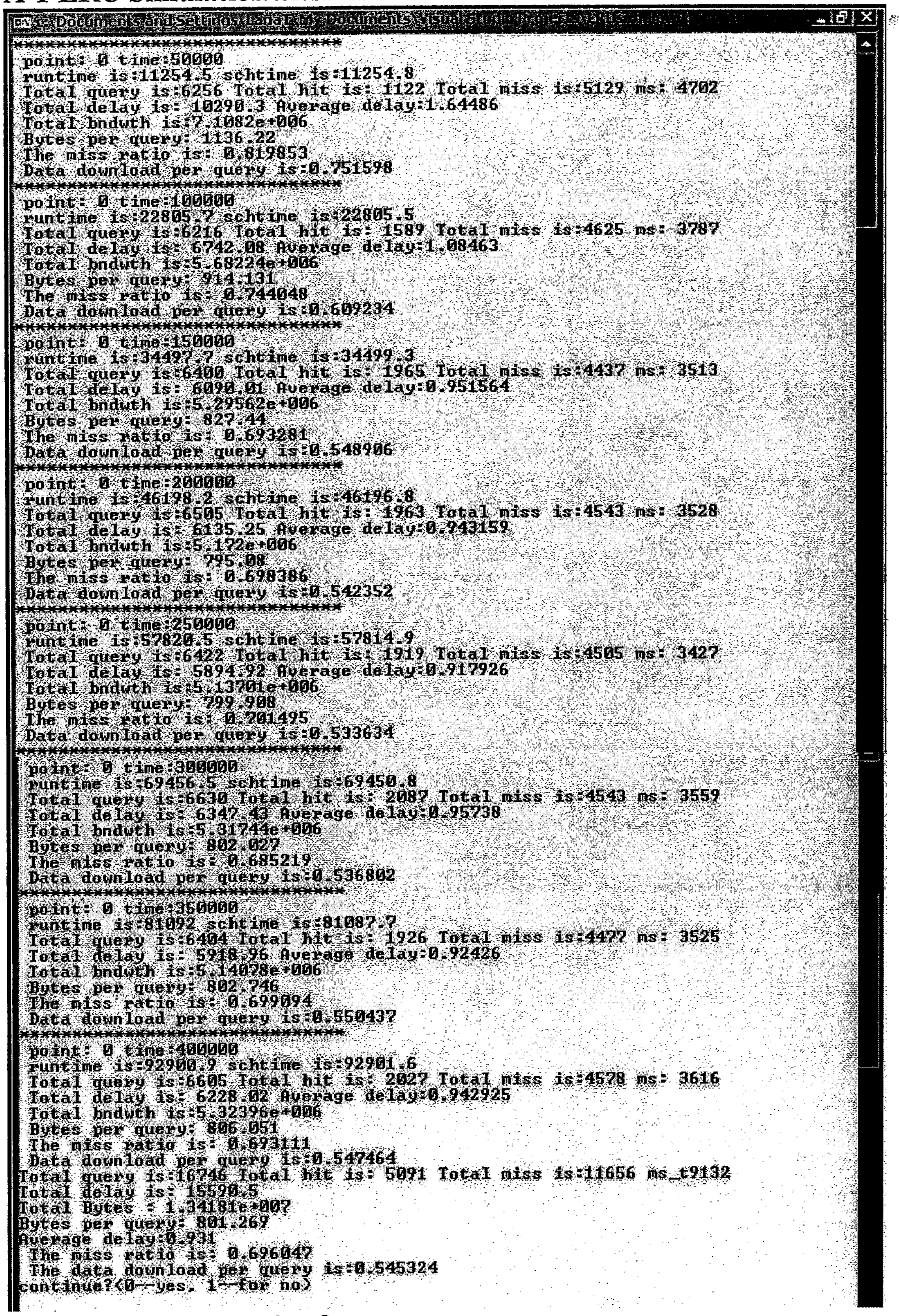
Kilgore, R.A.Kilgore (2002). *MULTI-LANGUAGE, OPENSOURCE MODELING USING THE MICROSOFT .NET ARCHITECTURE*. [Online]. Available: http://www.informs-sim.org/wsc02papers/080.pdf

Dobbs, Dr. Dobbs (2006). Source Code. *C/C++ Journal*. [Online]. Available: ftp://66.77.27.238/sourcecode/cuj/

Yin, L.Yin, G.Cao and Y.Cai (2006). *A Generalized Target-Driven Cache Replacement Policy for Mobile Environments*. [Online]. Available: http://www.cs.iastate.edu/~yingcai/GCP.pdf

Appendix SACCS Simulation Results

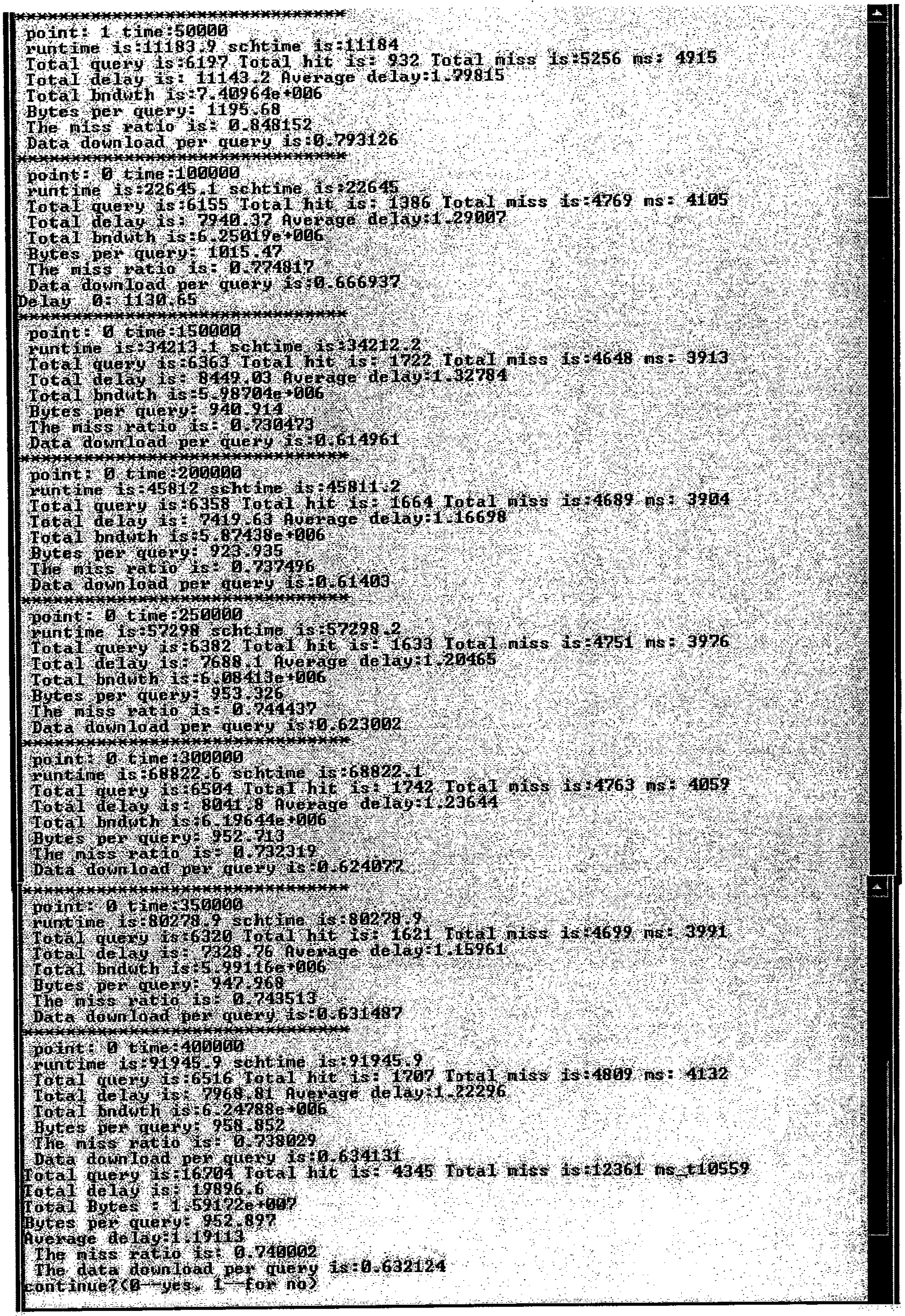
A-1 LRU Simulation Results



```
********************************
 point: 0 time:50000
 runtime is:11254.5 schtime is:11254.8
 Total query is:6256 Total hit is: 1122 Total miss is:5129 ms: 4702
 Total delay is: 10290.3 Average delay:1.64486
 Total bndwth is:7.1082e+006
 Bytes per query: 1136.22
 The miss ratio is: 0.819853
 Data download per query is:0.751598
********************************
 point: 0 time:100000
 runtime is:22805.7 schtime is:22805.5
 Total query is:6216 Total hit is: 1589 Total miss is:4625 ms: 3787
 Total delay is: 6742.08 Average delay:1.08463
 Total bndwth is:5.68224e+006
 Bytes per query: 914.131
 The miss ratio is: 0.744048
 Data download per query is:0.609234
********************************
 point: 0 time:150000
 runtime is:34497.7 schtime is:34499.3
 Total query is:6400 Total hit is: 1965 Total miss is:4437 ms: 3513
 Total delay is: 6090.01 Average delay:0.951564
 Total bndwth is:5.29562e+006
 Bytes per query: 827.44
 The miss ratio is: 0.693281
 Data download per query is:0.548906
********************************
 point: 0 time:200000
 runtime is:46198.2 schtime is:46196.8
 Total query is:6505 Total hit is: 1963 Total miss is:4543 ms: 3528
 Total delay is: 6135.25 Average delay:0.943159
 Total bndwth is:5.172e+006
 Bytes per query: 795.08
 The miss ratio is: 0.698386
 Data download per query is:0.542352
********************************
 point: 0 time:250000
 runtime is:57820.5 schtime is:57814.9
 Total query is:6422 Total hit is: 1919 Total miss is:4505 ms: 3427
 Total delay is: 5894.92 Average delay:0.917926
 Total bndwth is:5.13701e+006
 Bytes per query: 799.908
 The miss ratio is: 0.701495
 Data download per query is:0.533634
********************************
 point: 0 time:300000
 runtime is:69456.5 schtime is:69450.8
 Total query is:6630 Total hit is: 2087 Total miss is:4543 ms: 3559
 Total delay is: 6347.43 Average delay:0.95738
 Total bndwth is:5.31744e+006
 Bytes per query: 802.027
 The miss ratio is: 0.685219
 Data download per query is:0.536802
********************************
 point: 0 time:350000
 runtime is:81092 schtime is:81087.7
 Total query is:6404 Total hit is: 1926 Total miss is:4477 ms: 3525
 Total delay is: 5918.96 Average delay:0.92426
 Total bndwth is:5.14078e+006
 Bytes per query: 802.746
 The miss ratio is: 0.699094
 Data download per query is:0.550437
********************************
 point: 0 time:400000
 runtime is:92900.9 schtime is:92901.6
 Total query is:6605 Total hit is: 2027 Total miss is:4578 ms: 3616
 Total delay is: 6228.02 Average delay:0.942925
 Total bndwth is:5.32396e+006
 Bytes per query: 806.051
 The miss ratio is: 0.693111
 Data download per query is:0.547464
Total query is:16746 Total hit is: 5091 Total miss is:11656 ms_t9132
Total delay is: 15590.5
Total Bytes = 1.34181e+007
Bytes per query: 801.269
Average delay:0.931
 The miss ratio is: 0.696047
 The data download per query is:0.545324
continue?(0--yes, 1--for no)
```

Figure A.1 LRU Simulation Output

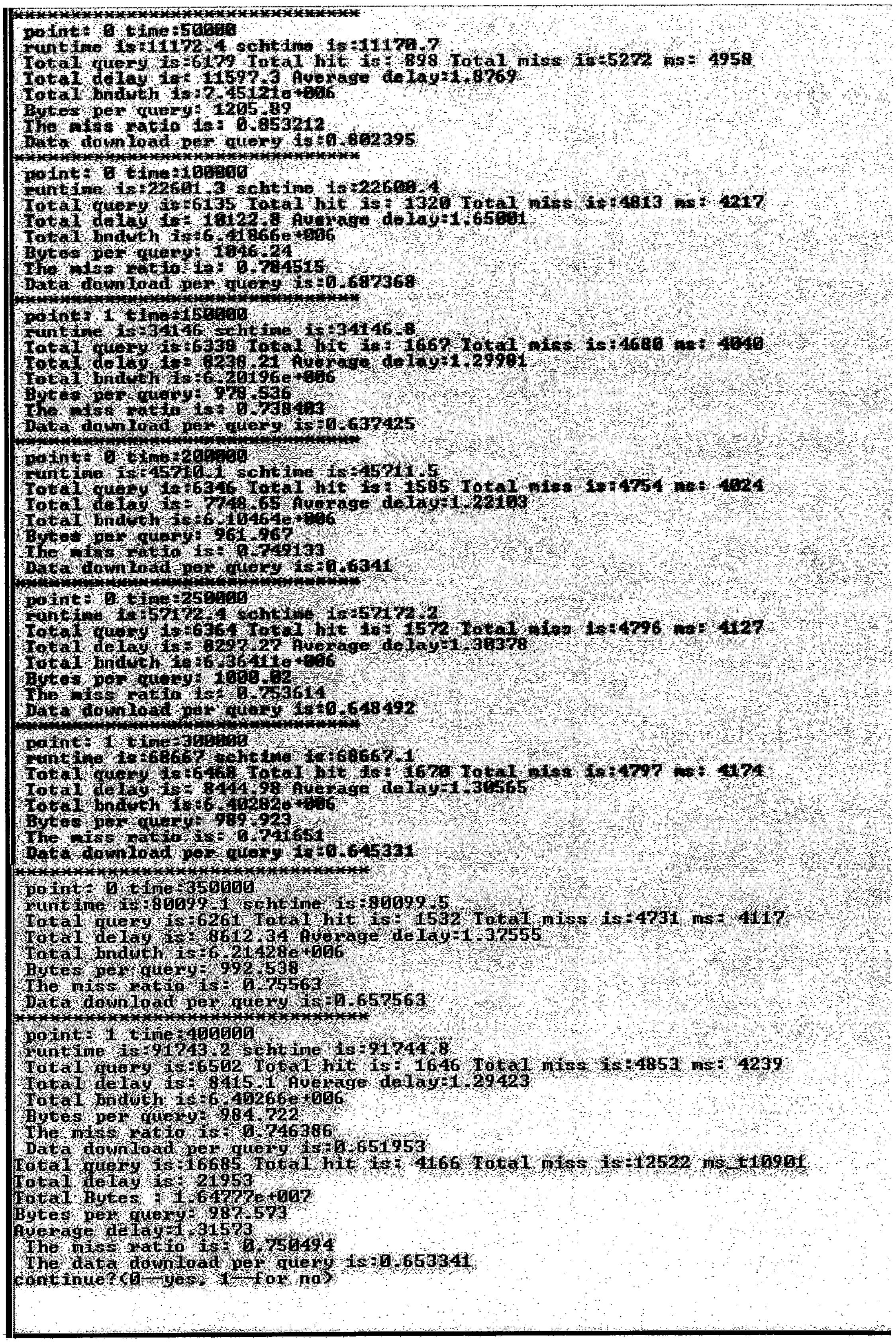
A-2 LFU Simulation Results



```
********************************
 point: 1 time:50000
 runtime is:11183.9 schtime is:11184
 Total query is:6197 Total hit is: 932 Total miss is:5256 ms: 4915
 Total delay is: 11143.2 Average delay:1.79815
 Total bndwth is:7.40964e+006
 Bytes per query: 1195.68
 The miss ratio is: 0.848152
 Data download per query is:0.793126
********************************
 point: 0 time:100000
 runtime is:22645.1 schtime is:22645
 Total query is:6155 Total hit is: 1386 Total miss is:4769 ms: 4105
 Total delay is: 7940.37 Average delay:1.29007
 Total bndwth is:6.25019e+006
 Bytes per query: 1015.47
 The miss ratio is: 0.774817
 Data download per query is:0.666937
Delay  0: 1130.65
********************************
 point: 0 time:150000
 runtime is:34213.1 schtime is:34212.2
 Total query is:6363 Total hit is: 1722 Total miss is:4648 ms: 3913
 Total delay is: 8449.03 Average delay:1.32784
 Total bndwth is:5.98704e+006
 Bytes per query: 940.914
 The miss ratio is: 0.730473
 Data download per query is:0.614961
********************************
 point: 0 time:200000
 runtime is:45812 schtime is:45811.2
 Total query is:6358 Total hit is: 1664 Total miss is:4689 ms: 3904
 Total delay is: 7419.63 Average delay:1.16698
 Total bndwth is:5.87438e+006
 Bytes per query: 923.935
 The miss ratio is: 0.737496
 Data download per query is:0.61403
********************************
 point: 0 time:250000
 runtime is:57298 schtime is:57298.2
 Total query is:6382 Total hit is: 1633 Total miss is:4751 ms: 3976
 Total delay is: 7688.1 Average delay:1.20465
 Total bndwth is:6.08413e+006
 Bytes per query: 953.326
 The miss ratio is: 0.744437
 Data download per query is:0.623002
********************************
 point: 0 time:300000
 runtime is:68822.6 schtime is:68822.1
 Total query is:6504 Total hit is: 1742 Total miss is:4763 ms: 4059
 Total delay is: 8041.8 Average delay:1.23644
 Total bndwth is:6.19644e+006
 Bytes per query: 952.713
 The miss ratio is: 0.732319
 Data download per query is:0.624077
********************************
 point: 0 time:350000
 runtime is:80278.9 schtime is:80278.9
 Total query is:6320 Total hit is: 1621 Total miss is:4699 ms: 3991
 Total delay is: 7328.76 Average delay:1.15961
 Total bndwth is:5.99116e+006
 Bytes per query: 947.968
 The miss ratio is: 0.743513
 Data download per query is:0.631487
********************************
 point: 0 time:400000
 runtime is:91945.9 schtime is:91945.9
 Total query is:6516 Total hit is: 1707 Total miss is:4809 ms: 4132
 Total delay is: 7968.81 Average delay:1.22296
 Total bndwth is:6.24788e+006
 Bytes per query: 958.852
 The miss ratio is: 0.738029
 Data download per query is:0.634131
Total query is:16704 Total hit is: 4345 Total miss is:12361 ms_t10559
Total delay is: 19896.6
Total Bytes : 1.59172e+007
Bytes per query: 952.897
Average delay:1.19113
 The miss ratio is: 0.740002
 The data download per query is:0.632124
continue?(0--yes, 1--for no)
```

Figure A.2 LFU Simulation Output

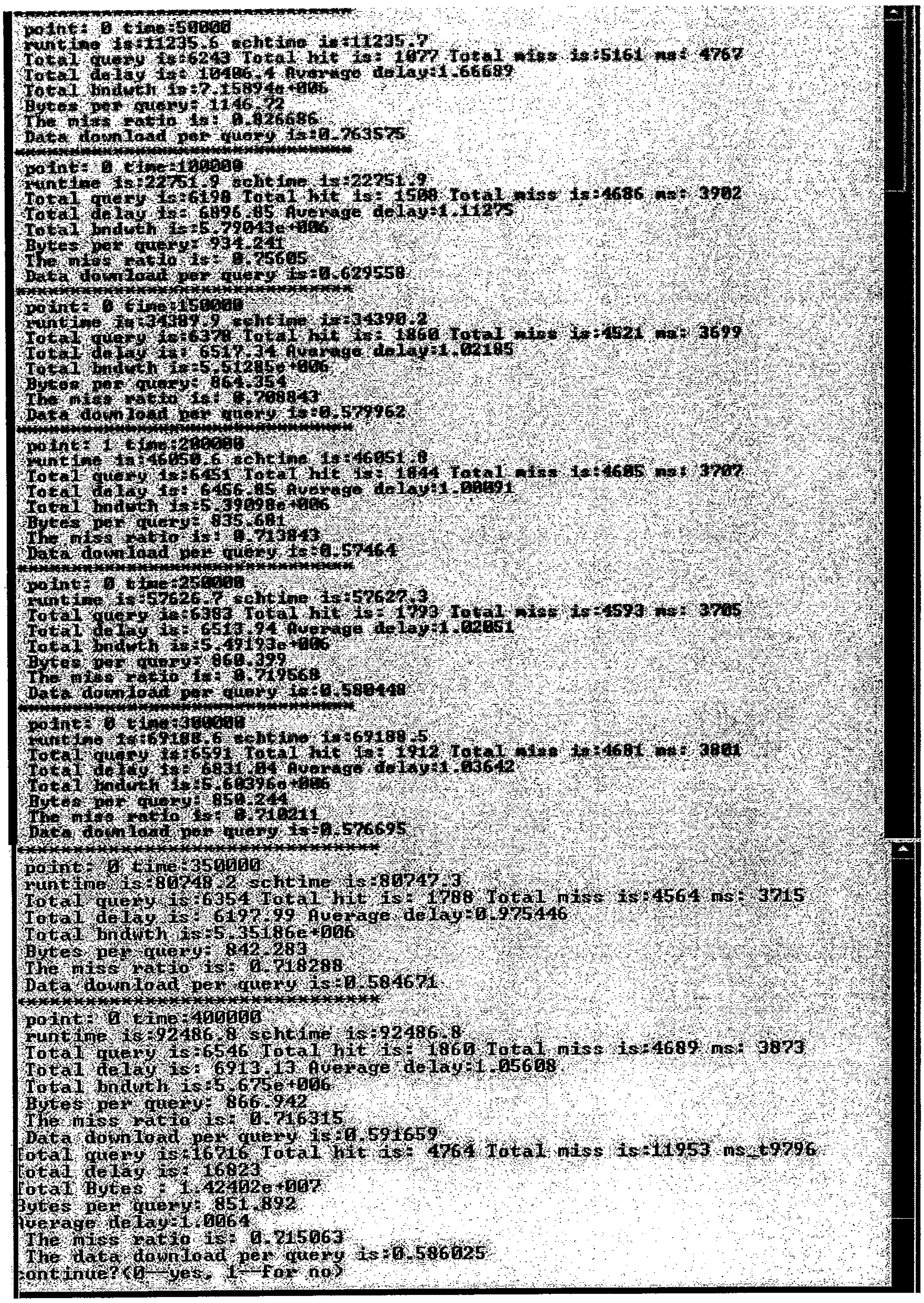
A-3 MFU Simulation Results



```
********************************
 point: 0 time:50000
 runtime is:11172.4 schtime is:11170.7
 Total query is:6179 Total hit is: 898 Total miss is:5272 ms: 4958
 Total delay is: 11597.3 Average delay:1.8769
 Total bndwth is:7.45121e+006
 Bytes per query: 1205.89
 The miss ratio is: 0.853212
 Data download per query is:0.802395
********************************
 point: 0 time:100000
 runtime is:22601.3 schtime is:22600.4
 Total query is:6135 Total hit is: 1320 Total miss is:4813 ms: 4217
 Total delay is: 10122.8 Average delay:1.65001
 Total bndwth is:6.41866e+006
 Bytes per query: 1046.24
 The miss ratio is: 0.784515
 Data download per query is:0.687368
********************************
 point: 1 time:150000
 runtime is:34146 schtime is:34146.8
 Total query is:6338 Total hit is: 1667 Total miss is:4680 ms: 4040
 Total delay is: 8238.21 Average delay:1.29981
 Total bndwth is:6.20196e+006
 Bytes per query: 978.536
 The miss ratio is: 0.738403
 Data download per query is:0.637425
********************************
 point: 0 time:200000
 runtime is:45710.1 schtime is:45711.5
 Total query is:6346 Total hit is: 1585 Total miss is:4754 ms: 4024
 Total delay is: 7748.65 Average delay:1.22103
 Total bndwth is:6.10464e+006
 Bytes per query: 961.967
 The miss ratio is: 0.749133
 Data download per query is:0.6341
********************************
 point: 0 time:250000
 runtime is:57172.4 schtime is:57172.2
 Total query is:6364 Total hit is: 1572 Total miss is:4796 ms: 4127
 Total delay is: 8297.27 Average delay:1.30378
 Total bndwth is:6.36411e+006
 Bytes per query: 1000.02
 The miss ratio is: 0.753614
 Data download per query is:0.648492
********************************
 point: 1 time:300000
 runtime is:68667 schtime is:68667.1
 Total query is:6468 Total hit is: 1670 Total miss is:4797 ms: 4174
 Total delay is: 8444.98 Average delay:1.30565
 Total bndwth is:6.40282e+006
 Bytes per query: 989.923
 The miss ratio is: 0.741651
 Data download per query is:0.645331
********************************
 point: 0 time:350000
 runtime is:80099.1 schtime is:80099.5
 Total query is:6261 Total hit is: 1532 Total miss is:4731 ms: 4117
 Total delay is: 8612.34 Average delay:1.37555
 Total bndwth is:6.21428e+006
 Bytes per query: 992.538
 The miss ratio is: 0.75563
 Data download per query is:0.657563
********************************
 point: 1 time:400000
 runtime is:91743.2 schtime is:91744.8
 Total query is:6502 Total hit is: 1646 Total miss is:4853 ms: 4239
 Total delay is: 8415.1 Average delay:1.29423
 Total bndwth is:6.40266e+006
 Bytes per query: 984.722
 The miss ratio is: 0.746386
 Data download per query is:0.651953
Total query is:16685 Total hit is: 4166 Total miss is:12522 ms_t10901
Total delay is: 21953
Total Bytes : 1.64777e+007
Bytes per query: 987.573
Average delay:1.31573
 The miss ratio is: 0.750494
 The data download per query is:0.653341
continue?(0--yes, 1--for no)
```

Figure A.3 MFU Simulation Output

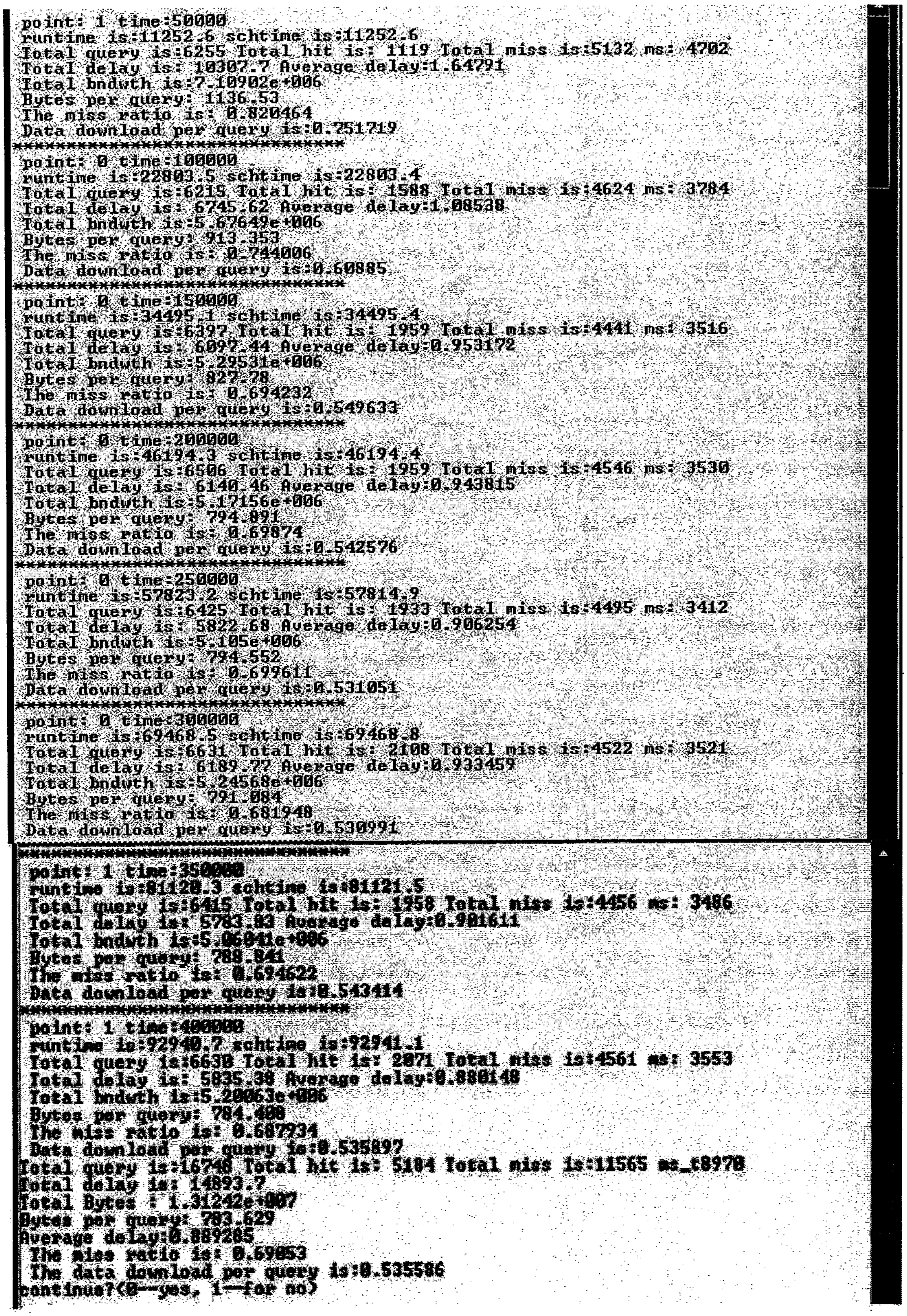
A-4 MRU Simulation Results



```
*********************************
point: 0 time:50000
runtime is:11235.6 schtime is:11235.7
Total query is:6243 Total hit is: 1077 Total miss is:5161 ms: 4767
Total delay is: 10406.4 Average delay:1.66689
Total bndwth is:7.15894e+006
Bytes per query: 1146.72
The miss ratio is: 0.826686
Data download per query is:0.763575
*********************************
point: 0 time:100000
runtime is:22751.9 schtime is:22751.9
Total query is:6198 Total hit is: 1508 Total miss is:4686 ms: 3902
Total delay is: 6896.85 Average delay:1.11275
Total bndwth is:5.79043e+006
Bytes per query: 934.241
The miss ratio is: 0.75605
Data download per query is:0.629558
*********************************
point: 0 time:150000
runtime is:34389.9 schtime is:34390.2
Total query is:6378 Total hit is: 1860 Total miss is:4521 ms: 3699
Total delay is: 6517.34 Average delay:1.02185
Total bndwth is:5.51285e+006
Bytes per query: 864.354
The miss ratio is: 0.708843
Data download per query is:0.579962
*********************************
point: 1 time:200000
runtime is:46050.6 schtime is:46051.0
Total query is:6451 Total hit is: 1844 Total miss is:4605 ms: 3707
Total delay is: 6456.85 Average delay:1.00091
Total bndwth is:5.39098e+006
Bytes per query: 835.681
The miss ratio is: 0.713843
Data download per query is:0.57464
*********************************
point: 0 time:250000
runtime is:57626.7 schtime is:57627.3
Total query is:6383 Total hit is: 1793 Total miss is:4593 ms: 3705
Total delay is: 6513.94 Average delay:1.02051
Total bndwth is:5.49193e+006
Bytes per query: 860.399
The miss ratio is: 0.719568
Data download per query is:0.580440
*********************************
point: 0 time:300000
runtime is:69188.6 schtime is:69188.5
Total query is:6591 Total hit is: 1912 Total miss is:4681 ms: 3801
Total delay is: 6831.04 Average delay:1.03642
Total bndwth is:5.60396e+006
Bytes per query: 850.244
The miss ratio is: 0.710211
Data download per query is:0.576695
*********************************
point: 0 time:350000
runtime is:80748.2 schtime is:80747.3
Total query is:6354 Total hit is: 1788 Total miss is:4564 ms: 3715
Total delay is: 6197.99 Average delay:0.975446
Total bndwth is:5.35186e+006
Bytes per query: 842.283
The miss ratio is: 0.718288
Data download per query is:0.584671
*********************************
point: 0 time:400000
runtime is:92486.8 schtime is:92486.8
Total query is:6546 Total hit is: 1860 Total miss is:4689 ms: 3873
Total delay is: 6913.13 Average delay:1.05608
Total bndwth is:5.675e+006
Bytes per query: 866.942
The miss ratio is: 0.716315
Data download per query is:0.591659
otal query is:16716 Total hit is: 4764 Total miss is:11953 ms_t9796
otal delay is: 16823
otal Bytes : 1.42402e+007
ytes per query: 851.892
verage delay:1.0064
The miss ratio is: 0.715063
The data download per query is:0.586025
ontinue?(0--yes, 1--for no)
```

Figure A.4 MRU Simulation Output

A-5 FIFO Simulation Results



```
point: 1 time:50000
runtime is:11252.6 schtime is:11252.6
Total query is:6255 Total hit is: 1119 Total miss is:5132 ms: 4702
Total delay is: 10307.7 Average delay:1.64791
Total bndwth is:7.10902e+006
Bytes per query: 1136.53
The miss ratio is: 0.820464
Data download per query is:0.751719
******************************
point: 0 time:100000
runtime is:22803.5 schtime is:22803.4
Total query is:6215 Total hit is: 1588 Total miss is:4624 ms: 3784
Total delay is: 6745.62 Average delay:1.08538
Total bndwth is:5.67649e+006
Bytes per query: 913.353
The miss ratio is: 0.744006
Data download per query is:0.60885
******************************
point: 0 time:150000
runtime is:34495.1 schtime is:34495.4
Total query is:6397 Total hit is: 1959 Total miss is:4441 ms: 3516
Total delay is: 6097.44 Average delay:0.953172
Total bndwth is:5.29531e+006
Bytes per query: 827.78
The miss ratio is: 0.694232
Data download per query is:0.549633
******************************
point: 0 time:200000
runtime is:46194.3 schtime is:46194.4
Total query is:6506 Total hit is: 1959 Total miss is:4546 ms: 3530
Total delay is: 6140.46 Average delay:0.943815
Total bndwth is:5.17156e+006
Bytes per query: 794.891
The miss ratio is: 0.69874
Data download per query is:0.542576
******************************
point: 0 time:250000
runtime is:57823.2 schtime is:57814.9
Total query is:6425 Total hit is: 1933 Total miss is:4495 ms: 3412
Total delay is: 5822.68 Average delay:0.906254
Total bndwth is:5.105e+006
Bytes per query: 794.552
The miss ratio is: 0.699611
Data download per query is:0.531051
******************************
point: 0 time:300000
runtime is:69468.5 schtime is:69468.8
Total query is:6631 Total hit is: 2108 Total miss is:4522 ms: 3521
Total delay is: 6189.77 Average delay:0.933459
Total bndwth is:5.24568e+006
Bytes per query: 791.084
The miss ratio is: 0.681948
Data download per query is:0.530991
******************************
point: 1 time:350000
runtime is:81120.3 schtime is:81121.5
Total query is:6415 Total hit is: 1958 Total miss is:4456 ms: 3486
Total delay is: 5783.83 Average delay:0.901611
Total bndwth is:5.06041e+006
Bytes per query: 788.841
The miss ratio is: 0.694622
Data download per query is:0.543414
******************************
point: 1 time:400000
runtime is:92940.7 schtime is:92941.1
Total query is:6630 Total hit is: 2071 Total miss is:4561 ms: 3553
Total delay is: 5835.38 Average delay:0.880148
Total bndwth is:5.20063e+006
Bytes per query: 784.408
The miss ratio is: 0.687934
Data download per query is:0.535897
Total query is:16748 Total hit is: 5184 Total miss is:11565 ms_t8970
Total delay is: 14893.7
Total Bytes : 1.31242e+007
Bytes per query: 783.629
Average delay:0.889285
 The miss ratio is: 0.69053
 The data download per query is:0.535586
continue?(0--yes, 1--for no)
```

Figure A.5 FIFO Simulation Output