AN EFFICIENT CACHE REPLACEMENT STRATEGY FOR THE HYBRID CACHE CONSISTENCY APPROACH IN A MOBILE ENVIRONMENT

by

ALINE MICHEL ZEITUNLIAN

B.S., Computer Science, Haigazian University, 1997

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

Department of Computer Science and Mathematics
LEBANESE AMERICAN UNIVERSITY
June 2008
Thesis Approval Form

Student Name: Aline Zeitunlian I.D. #: 200202896


Program: Master of Science

Division/Dept: Computer Science and Mathematics

School: School of Arts and Sciences

Approved by: Ramzi A. Haraty

Thesis Advisor:

Member: Faisal Abu Khzam

Member: AbdulNasser Kassar

Date: July 8, 2008
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:  Aline Michel Zeitounian

Signature:  

Date:  9 July 2008
I grant to the LEBANESE AMERICAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University’s own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.
To my parents
Acknowledgment

I would like to thank my advisor Dr. Ramzi Haraty Advisor for his guidance throughout my Thesis work. A thanks is also to Dr. Faisal Abu Khzam and Dr. Abdul Nasser Kassar for being on my thesis committee.

I would like to express my sincere gratitude to the Lebanese American University whose financial support during my graduate studies made it all possible.

Finally, I would like to thank my friends and family for their long support.
Abstract

Caching was suggested as a solution for reducing bandwidth utilization and minimizing query latency in mobile environments. Over the years, different caching approaches have been proposed, some relying on the server to broadcast reports periodically informing of the updated data while others allowed the clients to request for the data whenever needed. Until recently a hybrid cache consistency scheme Scalable Asynchronous Cache Consistency Scheme (SACCS) was proposed, which combined the two different approaches benefits’ and is proved to be more efficient and more scalable. Nevertheless, caching has its limitations too, due to the limited cache size and the limited bandwidth, which makes the implementation of cache replacement strategy an important aspect for improving the cache consistency algorithms. In this thesis, we proposed a new cache replacement strategy, the Least Unified Value strategy to replace the Least Recently Used that SACCS was based on. This thesis studies the advantages and the drawbacks of the new proposed strategy, comparing it with different categories of cache replacement strategies.
Contents

1. Introduction 1
   1.1 Scope of the Thesis 2
   1.2 Organization of the Thesis 3

2. Literature Review 4
   2.1 Mobile Computing Environment 4
   2.2 Mobile Computing Environment Architecture/System Model 4
   2.3 The Characteristics of Mobile Computing Environments: Mobility and Disconnection 5
   2.4 Mobile Data Caching 6
      2.4.1 Data Dissemination 6
      2.4.2 Cache Invalidation 7
   2.5 Data Consistency 17
      2.5.1 Cache Consistency Maintenance Algorithms 18
      2.5.2 Cache Replacement Strategies 19
      2.5.3 Web Caching 22

3. A Hybrid Cache Consistency in Mobile Environments 25
   3.1 Scalable Asynchronous Cache Consistency Scheme (SACCS) 25
      3.1.1 Data Structure and Message Formats 26
      3.1.2 The SACCS Cache Management 28
      3.1.3 The Algorithm and its Description 29
      3.1.4 Efficiency of SACCS 32

4. The Cache Replacement Policies 34
4.1 SACCs and the Cache Replacement Policies

4.1.1 A Value-based Function: Least-Unified Value

4.1.2 The New Algorithm for SACCs and its Description

4.1.3 The Other Cache Replacement Strategies

4.1.3.1 A Recency-Based Strategy: Least Recently Used

4.1.3.2 A Frequency-Based Strategy: Least Frequently Used

4.1.3.3 A Recency/Frequency Based Strategy: Least Recently/Frequently Used

4.1.3.4 A Randomized Strategy: Random

5. Performance Evaluation

5.1 Environment

5.2 Simulation Results

6. Conclusion

References

Appendix Simulation Output
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Wireless Data Communication System Architecture</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>a) Validity Period, b) AVI model</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Reducing the Query Latency by Replicating UIRs</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>State Diagram of a Cache Entry</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>MSS Main Pseudocode</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>MU Pseudocode</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>New SACCS Algorithm</td>
<td>38</td>
</tr>
<tr>
<td>5.1</td>
<td>Total Hit for Simulation</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Number of Hits vs. Time</td>
<td>46</td>
</tr>
<tr>
<td>5.3</td>
<td>Number of Misses vs. Time</td>
<td>46</td>
</tr>
<tr>
<td>5.4</td>
<td>Miss Ratio vs. Time</td>
<td>47</td>
</tr>
<tr>
<td>5.5</td>
<td>Hit Ratio vs. Time</td>
<td>48</td>
</tr>
<tr>
<td>5.6</td>
<td>Total Delay vs. Time</td>
<td>49</td>
</tr>
<tr>
<td>5.7</td>
<td>Average Delay Vs. Time</td>
<td>49</td>
</tr>
<tr>
<td>5.8</td>
<td>Bytes per Query</td>
<td>50</td>
</tr>
<tr>
<td>5.9</td>
<td>Data Download per Query</td>
<td>51</td>
</tr>
<tr>
<td>A.1</td>
<td>LRU Simulation Output</td>
<td>57</td>
</tr>
<tr>
<td>A.2</td>
<td>LFU Simulation Output</td>
<td>59</td>
</tr>
<tr>
<td>A.3</td>
<td>LRFU Simulation Output</td>
<td>61</td>
</tr>
<tr>
<td>A.4</td>
<td>LUV Simulation Output</td>
<td>63</td>
</tr>
<tr>
<td>A.5</td>
<td>Random Simulation Output</td>
<td>65</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1 Communication Messages in SACCS</td>
<td>28</td>
</tr>
<tr>
<td>Table 5.1 Total Query, Total Hit, Total Miss</td>
<td>45</td>
</tr>
<tr>
<td>Table 5.2 Bytes per Query</td>
<td>50</td>
</tr>
<tr>
<td>Table 5.3 Data Download per Query</td>
<td>51</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In mobile computing environments, where low powered devices are used to access and query databases over relatively low-bandwidth wireless channels, caching frequently accessed data objects will reduce bandwidth usage and delays perceived by users.

Caching is a technique that is used in different computing environments (computer architecture and computer databases) to improve the speed gap of storage. Nevertheless, in mobile environments, it is even more challenging because one needs to consider the mobility of the users and the disconnected modes, which arise due to the battery power saving measures or the unpredictable disconnection of wireless networks. However, having a copy of the data in the cache is not sufficient; the cache should provide the users a fresh data on each hit. If the data is changed at the origin server then the cache data is stale. For this issue, broadcasting was assumed to be an effective method for data dissemination. By broadcasting the information of the consistent data or invalidation reports, it is possible to share data information with a random number of mobile units on the network, by consuming as minimum bandwidth as possible.

Several methods for data distribution had been suggested to guarantee the cache consistency in mobile environments. Some suggested stateless servers to maintain the mobile environment (Barbara, 1994), (Cao, 2002), (Jing, 1997), others stateful servers (Kahol, 1994). Stateless servers do not need to remember which data object is cached in
which mobile unit; they rather broadcast invalidation reports periodically. In the stateful
servers approach on the other hand, the server knows and maintains the state of each
mobile unit and broadcast invalidation reports of only the data objects that have been
accessed. Combining each approaches positive features, the Scalable Asynchronous
Cache Consistency Scheme (SACCS) maintenance scheme was proposed. It was based
on the Least-Recently-Used (LRU) cache replacement strategy which used recency as a
basis to determine which data to evict (Wang, 2003), (Wang, 2004). SACCS is a
scalable and efficient method.

To support these maintenance algorithms, it is important to suggest an efficient
cache replacement policy, for after all mobile units have limited disk storage and not all
data objects can be cached.

1.1 Scope of the Thesis

In this thesis, we examine five different categories of cache replacement strategies that
were used for web caching and find out an efficient strategy for mobile environments.
Besides the LRU, we propose the least-unified algorithm (LUV) (Bahn, 2003) to be used
with the SACCS cache consistency maintenance scheme and compare it with the other
cache replacement strategy categories. LUV is a cache replacement technique that
associates a value to each object in the cache and when needed replaces it with the
object with the smallest value. This policy considers the reference potential and the
retrieval cost of the data object per unit size.
1.2 Organization of the Thesis

This thesis is organized in six chapters. In chapter 2 we provide background information about mobile computing caching consistency and a literature review of the approaches used to maintain the cache consistency of mobile environments. It includes a major part of the work proposed of methods related to caching and cache consistency, the cache invalidation strategies, the cache replacement policies. Chapter 3 describes the SACCS maintenance approach in detail. In chapter 4 we present SACCS and LUV the new cache replacement strategy proposed and the other cache replacement techniques that we compared LUV with. Chapter 5 presents the experimental results of the LUV cache strategy with SACCS and the comparison with the remaining four different class strategies. Finally, in chapter 6 we provide a conclusion and discuss the future work.
Chapter 2

Literature Review

2.1 Mobile Computing Environment

With the development of wireless communications a new model of distributed computing was introduced. This new environment differs from the other client/server based distributed environments, by the following three features a) users are able to connect from different access points and may continue to be connected while on the move, b) the bandwidth determines the reliability of communication and c) the battery power of the mobile device plays a determining role and affects on the performance and reliability of the system. Hence, due to these features, it faces more challenges, making the maintenance of the database more difficult.

2.2 Mobile Computing Environment Architecture/System Model

The mobile computing environment consists of fixed hosts (servers) also called Mobile Support Stations (MSS) and mobile hosts (clients) called Mobile Units (MUs). An MU can communicate with a fixed host or server via an MSS over a wireless channel. The wireless channel is logically separated into two sub-channels: uplink channel and download channel. MUs use the uplink channel to submit queries to the server via an MSS, while the MSSs use the download channel to disseminate information or to
forward the answers from the server to the target client. Each MSS is responsible for all the MUs within a given geographical or logical area, which is known as a cell. Therefore, when an MU leaves a cell serviced by an MSS, a handoff protocol transfers the responsibility to support this MU to the MSS of the new cell. This is shown in Figure 2.1.

![Wireless data communication system architecture](image)

**Figure 2.1** Wireless data communication system architecture (Wang, 2004)

### 2.3 The Characteristics of Mobile Computing Environments: Mobility and Disconnection

The characteristics of a mobile computing environment are: mobility (the movement of the mobile unit through the cells) and disconnection from the network.
Disconnection can occur at different times, due to resource limitations. We can define the following three cases a) predictable due to low battery or weak communication, b) intentional to conserve energy and operate in sleep mode, or c) accidental when arriving to an area out of reach of a MSS.

Therefore, based on the characteristics of a mobile environment mentioned earlier, an MU can reconnect to a different MSS after a disconnection for an indefinite time.

2.4 Mobile Data Caching

Data caching in mobile environments is an efficient solution that improves the performance of mobile access and reduces network traffic in a limited bandwidth mobile environment. By storing a copy of data close to the user of the data, we ensure faster data access and save limited bandwidth. However, all these raise different concerns, like a) how would the data be distributed having in mind the power consumption, low bandwidth and disconnections, b) the consistency of the cache, how to ensure the existence of a valid data in the cache.

2.4.1 Data Dissemination

Delivering of data from a server or a set of servers to a large group of clients is called data dissemination. There exist two different systems, the push-based and the pull-based (Barbara, 1999).
In a push-based system, the server needs to send information to the clients without waiting for the clients requests. The server decides when to send the data, which can be done either periodically or aperiodically and each of them has its own advantages.

In a pull-based system, clients send messages to the server to request for data objects, and later the server in its turn sends the requested information back to the clients.

Broadcasting was considered an efficient way to distribute information considering the limitation of the mobile environments’ bandwidth channels. A desirable goal is to minimize the number of uplink requests.

2.4.2 Cache Invalidation

Sending validation checks which is used for wired distributed system is inefficient for mobile environments. Taking into consideration the limited bandwidth, another solution was sending invalidation reports that notify clients about the cached items changes. The invalidation reports are very direct ways of transmitting the important information of updated data, yet a client may miss invalidation reports (IRs) when disconnected during broadcast and this might have its disadvantages. There were many cache invalidation strategies proposed.

Barbara and Imielinski have proposed three different strategies that use cache invalidation reports with stateful servers. They are the Broadcasting Timestamp (TS),
Amnesic Terminals (AT) and Signatures (SIG) (Barbara, 1994). All of them are synchronous.

In all of them the server periodically broadcasts the report which reflects the changing database state within the predefined window frame. The interval for broadcasting is $L$ seconds and the broadcast window is $w$. The mobile unit waits and pays attention to the broadcast to invalidate its cache accordingly. A client discards its entire cache if it disconnects from the network and misses $k$ consecutive reports, not knowing whether its items have been updated since its disconnection. By discarding the entire cache due to disconnection a major part of the caching benefits are lost, especially if the cached objects are still valid. While user query requests are answered only after receiving an invalidation report causing an access delay. The broadcast-based solution is a good solution because it fits to infinite number of clients that listen to the broadcast report.

A major challenge of broadcast-based solutions is how to organize the information to be broadcasted. They require a large report that would be informative and effective to invalidate the cache. However, large reports mean long latency, which is not good especially that the broadcast bandwidth is limited. Therefore, other schemes had been proposed which extended the broadcasting IR scheme by optimizing the size of the invalidation report IR (Jing, 1997). Jing et al. used in their proposed scheme binary bits for every data objects in the database. If more than half of the objects have been updated then the cache has to be discarded. This scheme uses a report that arranges and traces the updated records and the size of the report is related to the database size. However, since
this structure contains more update history information, it results in larger invalidation reports when even only a few things have changed.

Wu et al. proposed the MU to send back to the server the identifiers of all cached data objects along with their timestamps after a long disconnection. The server identifies the changed data object and returns a validity report, which is not very efficient, as it wastes bandwidth and creates unnecessary uplink requests and discards the local cache after long disconnections. To reduce this overhead of validity checking, a possible approach is to do it at a group level, but the disadvantage of this method is that, if any data object is validated then the whole group must be invalidated. Wu et al. had suggested grouping with Cold Update-set Retention (GCORE). With this grouping characteristic, all the data objects are identified that have been updated since the last IR and after the mobile unit had disconnected. Upon waking up, a mobile unit checks its validity with the server at a group level to save uplink costs. In GCORE the cold update set objects are retained if the updated objects have been included in the most recent invalidation report broadcasted (Wu, 1996). The hot update set objects are those data objects that have frequently been updated meanwhile the cold update set of objects are those data objects that are less frequently updated. In fact, the objects that are referenced by queries and are cached in the mobile unit could belong to the hot update set which may be invalidated frequently. Therefore, objects in a group that belong to the cold update set are most likely to be retained. GCORE is energy-efficient because it incurs both low uplink cost for validity checking and low downlink costs as it retains more cached objects.
Since the three methods suggested by Barbara did not utilize the bandwidth efficiently, Jing's proposed bit-sequence scheme required much larger invalidation reports, and Wu's scheme required uplink bandwidth and update history window of the past broadcast intervals, another suggestion was the adaptive cache invalidation algorithm, which adjusts the periodicity of IRs according to the query rate and client disconnection time, while spending little uplink cost (Hu, 1997). The adaptive invalidation method improved mobile caching and reduced the bandwidth usage costs without degrading the system throughput. It is important to mention that uplink transmissions require more power from the clients than downlink receptions, implying more battery power consumption.

Kahol et al. have proposed a scheme that minimizes the overhead for MUs to validate their caches when reconnected, using stateless servers and minimizing the need for bandwidth. They did this by using asynchronous invalidation messages. During disconnection invalidation messages from servers are buffered in the home location cache of the MU and when the MU reconnects later those messages are redelivered (Kahol, 2001). Each mobile host maintains its Home Location Cache (HLC). It has an entry for each data object cached in the MU, and therefore needs to maintain the timestamp indicating the time when was the data object last invalidated. Before responding to a query it checks whether the data is consistent or not, for this reason callbacks from a MSS are used. When an update message is received the MSS checks the set of MUs that are using this data from the HLCs and sends the invalidation report to each of them. The MU receiving this invalidation message marks the specific data object in its local cache to be invalid. Therefore, when a request message for a data object
arrives at the MU, it first checks the validity of the object in its local cache. If the data object is valid then it uses the local cache data saving latency, bandwidth and battery power, else an uplink request is sent to the MSS to require the data. The MSS in its turn requests the server for the data object. When it is received the MSS adds an entry to the HLC and forwards it to the MU. In sleep mode the mobile unit cannot receive any invalidation messages sent by its HLC. By using timestamps the MU can decide which invalidations it needs to retransmit to the mobile host. Those timestamps are maintained in the cache as the cache timestamp, which indicate the timestamp of the last message received by the MH from its MU. Those timestamps are needed to discard the invalidations that it no longer needs to keep or to know the invalidation it needs to resend. So when an MU wakes-up it sends a probe message to its HLC. Accordingly the HLC sends an invalidation report, in order for the MU to determine which items changed while it was in sleep state or disconnected. Note that an MU defers all query requests until it receives an invalidation report from its HLC.

An important consideration in the design of cache invalidation scheme is to minimize the bandwidth required for broadcasting the invalidation reports as well as the reports’ size. Yuen et al. based their work on invalidation reports by an estimate of the lifespan of a data called the absolute validity interval (AVI) (Yuen, 2000). Considering the real-time properties, updates captured by mobile units require to maintain the valid data objects. The updates might arrive periodically or at irregular intervals. Therefore, every object is assigned an AVI which may not necessarily be equal to the actual lifespan of the values of the data object but has to be a reasonable approximation. Usually this approximation is based on the previous update intervals of the data object.
This AVI is used to guess whether the data object’s value is valid or not. However, since the validity period is only recognized after the next update arrival, there are two periods, when the data object is invalid called the False Valid Period (FVP) and valid called the False Invalid Period (FIP). FIP is the overestimation of the validity of the data item and the FVP is the underestimation of the validity period.

![Diagram of AVI concept]

Figure 2.2 a) Validity period, b) AVI model (Yuen, 2000)

The cache invalidation called Invalidation by Absolute Validity Interval (IAVI) which was proposed by Yuen et al. considers that no invalidation notification is needed to invalidate the cached objects in the mobile units assuming a cached object to be invalid if its AVI expired. The AVI value is continuously adjusted to minimize the values of False Valid Period (FVP) and False Invalid Period (FIP). See Figure 2.2 for illustration. This invalidation scheme differs from the previously mentioned schemes in the fact that those schemes depend on actual updates occurring within a window frame and the
database size, while Yuen's proposed IAVI is based on real-time properties. When a data is accessed by an MU the update timestamp and the AVI verify the validity of the item. If the access time is greater than the last update time and its AVI then the cached item is invalidated. With this self-invalidation mechanism, the size of invalidation reports is reduced significantly.

Later, Wong et al., by taking semantics into consideration, devised a new optimistic concurrency control algorithm OCC-AVI to process mobile read-only transactions with deadline requirements (Wong, 2004). Basing their proposal on the same concept, of using the estimated AVI information for data objects and without waiting for IR, the cache is invalidated when the mobile client reconnects to the network. Their contribution was that OCC-AVI is an algorithm that improves performance and guarantees temporal consistency in addition to cache coherency and data consistency. The OCC-AVI reduces IR size, transaction response time and restart rate.

Cao had addressed the long query latency problem with the UIR-based approach (Cao, 2002). In this approach, a small portion of the important information called updated invalidation report is replicated several times within an IR interval and therefore the user can answer a query without waiting until the next IR. By replicating the IRs \( m \) times, the IR is repeated every \((l/m)\)th of the IR interval. As a result, the client will need to wait at most time \((l/m)\)th of the normal IR interval before answering a query. This solution will reduce the latency to \((l/m)\)th of the latency in the previous methods. But again, replicating the complete IR \( m \) times, is not very efficient as it may consume a
large amount of broadcast bandwidth especially that the IR will contain a big part of update history information. Hence, to save broadcast bandwidth, after the first IR, it is better to include the \( m-I \) updated invalidation reports (UIR) within an IR interval, in this way minimizing the size of the IR compared to an IR. It is possible for an MU to verify its own cache using the UIR, on condition that the MU had downloaded the most recent IR. This is illustrated in Figure 2.3, where \( T_{i,k} \) represents the time of the \( k \)th UIR after the \( i \)th IR. Let us say an update has occurred sometime in the interval of \( T_{i-1,1} \) and \( T_i \). If the mobile unit gets a query between \( T_{i-1,1} \) and \( T_{i-1,2} \), in the IR-based approach it cannot answer the query until \( T_i \), but in the UIR-based approach it can answer the query at \( T_{i-1,2} \). Note that, in case the data object is not available in the cache and there is a cache miss, the client still needs to fetch data from the server. This implies an increase in the query latency.

![Figure 2.3 Reducing the query latency by replicating UIRs (Cao, 2002)](image-url)
Nevertheless, this did not solve the problem of the client waiting for the data to be delivered if there was a cache miss. Cao also suggested that the server uses counters to identify hot data objects and to broadcast the information of the updates, helping the clients to get back the data objects which will be accessed in the near future intelligently. Although to improve cache hit ratio and reduce bandwidth consumption users can prefetch the data that are most likely needed in the near future based on intelligent decisions. Prefetching is power consuming (Cao, 2002). Cao proposed a new prefetch access ratio (PAR), to be used for a power-aware cache management, which can dynamically optimize the performance or power consumption, based on the available resources and performance requirements. This suggestion not only helps in improving the cache hit ratio, the bandwidth utilization, the throughput, but also reduces the query delay and the power consumption (Cao, 2002).

It is noticeable that the IR-based cache management is a useful and proficient approach since the server periodically broadcasts invalidation reports in which the changed data objects are indicated. Therefore, instead of validating the cached copies by querying the server directly, the clients can validate their local cache waiting and using the IRs broadcasted over the wireless channel. However, despite usefulness of this method this has its weaknesses too such as the long query. Another weakness of IR-based management is that the IR of the server includes even the data objects that are not cached by any unit, and hence wastes a significant amount of bandwidth. In addition to that, there may be the same data cached in several caches and each of them will query server to get the same updated data value, wasting a lot of the wireless bandwidth and battery energy. In (Cao, 2002) the adaptive prefetch approach was implemented. For
each data object in the cache, the access number is recorded, as well as its number of prefetches over a period of time. Dividing the recorded number of prefetches with the recorded number of accesses, the prefetch access ratio (PAR) is calculated. If the PAR is less than 1, prefetching the data object is beneficial as it may be used again later. When power consumption is an issue, the client marks the cache objects which have PAR greater than a certain value $\beta$, a specific system tuning factors, as non-prefetch, $\beta$ being greater than 1. This tuning factor is changed according to the requirements of power consumption. As a conclusion for Cao’s proposals, to resolve these issues Cao suggested using counters that help identify hot data objects, which could be needed and used in the near future, helping the server broadcast only the updated hot data objects and for the clients to retrieve them. His new proposed technique helps reduce unnecessary uplinks and even downlink broadcasts (Cao, 2003), (Cao, 2004).

In all the previous works, broadcasting was considered an efficient method for delivering data in a mobile computing and that updates can change the data at the server at any time. But no one considered the fact that since there is no control on changes that are carried while broadcasting data objects, then users might access inconsistent data values. For read-only mobile transactions, disseminating consistent data objects means reading data that are committed at the last broadcast time. A solution for this problem was the basic multi-version broadcast method. Another solution was the data re-broadcast scheme called Update First with Ordering (UFO) designed for mobile transactions in which operations are unordered. This scheme is able to provide the mobile transaction with the most updated values of data objects, at the same time maintaining the serializability of the execution between update and mobile transactions.
Therefore, Lam et al. proposed a broadcast method called Ordered Update First with Order (OUFO) which is an extension of the Update First with Ordering (UFO), to guarantee that the users will receive the newest data, at the same time, reducing the access delay (Lam, 2000).

Shao and Lu discussed the different disadvantages of periodic and non-periodic invalidation reports and suggested a variant of periodic updating report broadcast method, which can reduce the overhead of broadcasting reports and improves the cache hit ratio. Their strategy is based on the updating frequency of data objects. It broadcasts the updating messages instead of invalidation messages. This is carried out by using a substitution strategy based on semantically cached data objects and the historical report after the mobile units reconnects (Shao, 2003).

Madhukar and Alhaji presented an adaptive energy efficient cache invalidation scheme (AEECIS) that changes the data dissemination strategy based on the current conditions. This method improves the mobile caching and reduces the bandwidth for query processing. The server has three modes slow, fast and super-fast modes. According to the thresholds specified for time and number of requesters’ the server transmits IRs in any of the three modes (Madhukar, 2006).

2.5 Data Consistency

The most important point in data caching is to guarantee the consistency of the used data.
2.5.1 Cache Consistency Maintenance Algorithms

To maintain cache consistency three different types of algorithms were suggested.

In stateless approaches, an MSS has no knowledge of MUs cache contents. The MSS periodically sends invalidation reports to the MUs. While at an MU, a data object request cannot be served and carried out until the next IR. The advantage for stateless approaches is that they are easy to manage, since there is no need to maintain any state information about the MU. However, these methods have their drawback since they are not scalable to large databases because of the increased number of IR messages. Another drawback is that the access latency on average is always longer than half of the broadcast period. And the last disadvantage is that at reconnection after a long disconnection the cache entries are deleted, even the valid data objects.

In stateful approaches, an MSS keeps the state of each object for every MU cache and broadcasts IRs only for those objects. These approaches have their advantages and drawbacks as well. Since invalidation reports are broadcasted only when the data object is changed, it will reduce the unnecessary IRs but on the other hand, it makes the database more difficult to be maintained.

As for Scalable Asynchronous Cache Consistency Scheme (SACCS), which is a hybrid approach, the MSS identifies only the data objects that might be valid in MU caches. It does not broadcast IRs periodically. The uncertain and ID-only states of an MU allow the handling of sleep-wakeup patterns and mobility. All these improve the broadcast channel efficiency. In the proposed SACCS approach the LRU (Least recently used) cache replacement strategy was used (Wang, 2003), (Wang, 2004), (Shen, 2005).
2.5.2 Cache Replacement Strategies

A cache replacement strategy is an algorithm that decides which object to evict from the cache when no space is available to store additional objects. It is based on several factors: recency (to distinguish between objects that were accessed recently), frequency (to distinguish between objects that would be referenced soon), cost (for fetching) and size.

To determine the effectiveness of a replacement strategy, it is necessary to measure certain metrics such as the cache hit ratio, which is the number of hits on the local storage divided by number of operations, byte hit ratio which is the number of bytes saved from transmission by using the cache over the total number of bytes referenced, delays the time between the instant the request is issued and the instant when the result is received by the application, stretch the ratio of the access latency of a request to its service time, where service time is defined as the ratio of the requested object’s size to broadcast bandwidth.

Xu and Hu’s proposed replacement policy considered several factors that affect cache performance, like probability, update frequency, data size, retrieval delay and cache validation cost. The Minimum Stretch integrated with Access rates, Update frequencies, and cache validation Delay (Min_SAUD) policy takes into consideration the cost of guaranteeing the cache consistency before using each cached data object. This policy has been shown to be the most advantageous in terms of the stretch performance measure (Xu, 2001). Their assumption was based on systems that use the
on-demand broadcast for data dissemination, i.e. the mobile unit asks and the server sends the requested data objects through the broadcast channel periodically from where the clients retrieve the requested data. According to them a push-based broadcast, which periodically broadcasts a set of data objects following a certain access pattern is a special case of on-demand broadcasting with a zero uplink cost. The cache replacement policy Min_SAUD used a gain function that considered the low access probability, short data retrieval delay, high update frequency, large data size and the cost of cache invalidation to determine the data objects that will be evicted.

Yin et al. presented a generalized target-driven cache replacement policy for mobile environments. Their derived two value functions satisfied two specific targets; minimize the query delay and minimize the downlink traffic. The distinctive feature of the value function is its generalized function that can be used for various performance metrics by making the necessary changes. As a result, the cache hit-ratio may not be the best measurement for evaluating the quality of a cache replacement algorithm (Yin, 2003). The only drawback of this algorithm is that there is no method to set the optimization target.

In Shen’s (Shen, 2004), (Shen, 2005) and Seifert’s works (Seifert, 2002), suggested for mobile hybrid data delivery environments, cache consistency algorithms that integrated cache replacement and prefetching algorithms to efficiently maintain the read-only transactions data requirements. The first presented the Greedy Dual Utility cache replacement policy and the second Multi-version integrated caching and prefetching policy.
Shen et al. derived a utility function to evaluate the utility of each data object in terms of energy saving. It is an analytical model that considers all the different events that might arise in a mobile computing environment (data request, data update, connection/disconnection, and mobility handoff) and might affect the energy consumption. The GreedyDual (GD) Least Utility (LU) caching mechanism consists of two algorithms: GD-LU cache replacement algorithm and GD-LU passive prefetching algorithm. The GD-LU cache replacement algorithm checks the validity of the data and if it is valid then calculates the new utility value. If it is uncertain then it checks if the data is valid or not. If the data is not available in the cache then it retrieves it. When the data is confirmed then it returns to the application. If the data is received and there is not enough space, the cache replacement algorithm selects those data that have the minimum utility value and replaces them with the new incoming data. The GD-LU passive prefetching was proposed to overcome the blind prefetching degradation of the cache performance, since a blind prefetching may result in wasting of energy and even replacement of some recently accessed data. According to the proposed passive prefetching considers a data object whose metadata are saved in the metadata queue more possible to be accessed again and therefore its prefetching more valuable. Defining a threshold and computing a relative utility it could be determined whether the data object is to be admitted to the cache or not.

The Multi-version Integrated Caching and Prefetching (MICP) proposed by Seifert et al. takes into consideration the dynamically and infrequently changing cached and/or disseminated data object versions' cost/benefit ratios to determine the cache replacement and prefetching data objects. It also considers the replacement issue of a
newly created or outdated re-cacheable object version, and divides the MUs cache into two partitions the re-cacheable (REC) and non-re-cacheable object versions (NON-REC).

A different replacement strategy than the conventional ones was suggested by Santhosh et al. It was based on semantic, which saved a lot in the synchronization-related communication overhead (Santhosh, 2005). They took into consideration intra-file and inter-file relationships. Based on the observation that file access patterns are not random and that there is a semantic relationship usually, they tried to transform this relationship information into an eviction index that would help determine the victims of a cache replacement policy.

2.5.3 Web Caching

A widespread example of a mobile environment is the web. Web caching, like mobile data caching, aims to reduce network traffic, server load, and access delays. Web cache is again impacted by the replacement strategy. The important factors of web objects that can influence the replacement process are: recency (time of the last reference to the object), frequency (number of requests to an object), size, cost of fetching the object, modification time, and expiration time (heuristic).

Rabinovich and Spatscheck presented an overview of web caching and replications (Rabinovich, 2002). In their book, they presented a) all approaches for cache consistency: validation and invalidation, the overheads of each approach, b) the cache
replacement policies and the metrics that need to be taken into consideration, and c) prefetching that can improve the access latency.

Web caching was classified differently throughout literature. The first classification of replacement strategies was given by Aggarwal et al. who proposed three categories: direct extensions of traditional strategies, key-based and function-based (Aggarwal, 1999). In direct-extension category, traditional policies such as LRU or FIFO are extended to handle data objects of non-homogeneous size. In the key-based policies keys are used to prioritize and sort objects. In this way prioritizing some replacement factors over others; however, such prioritization may not always be ideal. Function-based replacement policy has received considerable attention. The idea in function-based replacement policies to employ a function of the different factors such as time since last access and entry time of the data object in the cache.

Later, Podlipnig and Boszormenyi classified the web cache replacement strategies with the following categories: recency-based strategies, frequency-based strategies, recency/frequency-based strategies, function-based strategies, and randomized strategies (Podlipnig, 2003). The first class of strategies is the recency-based strategies, which use the recency as the basic factor; includes LRU and its extensions. The frequency-based strategies use the frequency as the basic factor; it includes LFU strategy and its extensions. Another class is the recency/frequency-based strategies that take into consideration both recency and frequency, LRFU is a strategy belonging to this class. The function-based strategies are based on a function, which calculates a data object’s value and when replacing chooses the object with the minimum value. There are several strategies belonging to this class, greedy-dual strategies being among them. The
last class of cache replacement policies is the randomized strategies, which use randomized decisions.

The recency-based or frequency-based strategies are almost similar conceptually. However, they differ in the factors that they take into consideration. Therefore, each strategy is to be considered as a separate category. Similarly, the recency/frequency-based strategies, which mix the two factors recency and frequency should be considered as a separate class of cache replacement strategies. Meanwhile, the function-based strategies which calculate a value for the data object, basing it on a special function which includes several factors of interest like the size of the object, the last request time for the object, the number of requests for the object, the access latency, the cost for fetching it, etc, can be considered as belonging to any of the previous three mentioned categories but at the same time they should be considered as a separate class. Finally, the randomized strategies being nondeterministic approaches can definitely be considered as a class of their own. Certainly, each class of strategies has its own advantages and disadvantages.
Chapter 3

A Hybrid Cache Consistency in Mobile Environments

3.1 Scalable Asynchronous Cache Consistency Scheme (SACCS)

The Scalable Asynchronous Cache Consistency Scheme was proposed by Wang et al. to maintain an MU’s cache consistency (Wang, 2003). This hybrid approach, that is a combination of stateful and stateless approaches, maintains minimum state information.

According to SACCS, the MSS is only responsible for identifying the data objects of the database that might be valid in the MU caches, unlike the stateful algorithm which requires the MSS to remember for every single MUs cache all the data objects, making the management of the database simpler.

To save downlink bandwidth usage, SACCS reduces the IR messages sent through broadcast; thus SACCS does not use periodic IR broadcasting to invalidate the data in the MU caches, while the stateless approach needs to broadcast IRs periodically.

In addition to these two features, SACCS added two different states for data objects in MU caches, they are uncertain and ID-only, that allow handling of random sleep-wakeup patterns and mobility.

To achieve the above mentioned features, SACCS uses a) flag bits at the MSS and MU caches, b) an identifier (ID) in MU cache for each entry after its invalidation, and c) an uncertain state in the MU cache after disconnection.
In the cache consistency maintenance algorithm proposed by Wang et al., they used the LRU (Least Recently Used) replacement algorithm (Wang, 2003) (Wang, 2004). This consistency scheme was for systems with read-only transactions.

### 3.1.1 Data Structure and Message Formats

For each data object, identified by \( x \), the data structure, suggested by Wang et al., used both for the MSS and MU cache is given with the following:

In MSS: \((d_x, t_x, f_x)\) where \( d_x \) is the data object; \( t_x \) is the last update time for the data object; and \( f_x \) is a flag bit.

In MU: \((d_x, t_x, s_x)\) where \( d_x \) is the data object; \( t_x \) is the time stamp indicating the last updated time for the cached data object; and \( s_x \) is a two-bit flag identifying four data entry states: *valid, uncertain, uncertain with a waiting query* and *ID-only*, respectively.

Therefore, in SACCS, a data object is in one of the three states. They are the valid state, the uncertain state and the ID-only state. The valid state is the state when a data is cached in a mobile unit. The uncertain state is defined for data objects that have been cached in a mobile unit, and the mobile unit had reconnected after a random period of disconnection. The ID-only state is the state that indicates that the data object cached in the mobile unit was updated and it is not a valid one. When an uncertain data object is validated and the object is downloaded then it changes its state to valid, similarly when an ID-only state data object is downloaded, the state is changed to valid. Whenever an invalidation report is broadcasted, in whichever of the two states
(uncertain or valid) the data object is, it changes to ID-only state. All these state changes are illustrated in the state diagram in Figure 3.1.

![State diagram of a cache entry](image)

Figure 3.1 State diagram of a cache entry (Wang, 2004)

As for the communication messages, they are the Update, VData, IR, Confirmation and Query. Update message is sent from the original server to the MSS, to indicate that the data object has been updated. VData is sent from the MSS to all MUs, to transmit the valid data object with the time the object was updated. An MSS sends IR messages from the MSS to all MUs to indicate the cache data are not valid anymore. The Confirmation message is sent to assure that the cached data object is valid. MUs can send query messages to an MSS and if they are uncertain then they send the uncertain
message to validate their data object’s value and timestamp. All the communication messages are defined as in Table 3.1.

Table 3.1 Communication Messages in SACCs (Wang, 2003)

<table>
<thead>
<tr>
<th>Name</th>
<th>Sender</th>
<th>Receiver</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update ((x,d'_x,t'_x))</td>
<td>Original servers</td>
<td>MSS</td>
<td>Indicating (d_x) has been updated to (d'_x) at time (t'_x)</td>
</tr>
<tr>
<td>VData ((x,d_x,t_x))</td>
<td>MSS</td>
<td>MUs</td>
<td>Broadcast valid data object (d_x) with update time at (t_x)</td>
</tr>
<tr>
<td>IR((x))</td>
<td>MSS</td>
<td>MUs</td>
<td>Indicating cached (d_x) is invalid</td>
</tr>
<tr>
<td>Confirmation ((x,t_x))</td>
<td>MSS</td>
<td>MUs</td>
<td>Indicating (d_x) is valid if (ts_x = t_x)</td>
</tr>
<tr>
<td>Query ((x))</td>
<td>MUs</td>
<td>MSS</td>
<td>Query for data object (d_x)</td>
</tr>
<tr>
<td>Uncertain ((x, ts_x))</td>
<td>MUs</td>
<td>MSS</td>
<td>Verifying if (d_x) in uncertain state with update time (ts_x) is valid</td>
</tr>
</tbody>
</table>

3.1.2 The SACCs Cache Management

In the server, each data object is coupled with a flag bit, which changes when the data is retrieved. This flag bit is changed whenever an MU retrieves the data object, as a sign that there may be a valid copy found in the MU cache. Consequently, when this data object is updated, the server immediately broadcasts its IR and resets the flag bit to indicate that the data object found in the cache is not a valid copy anymore. Therefore, until the flag is reset no update requires broadcast of IR.
At an IR broadcast, an MU is either in an awake (connected to the MSS) or in a sleep state (disconnected). If the MU is awake then the state is changed from valid to ID-only. If the MU is disconnected, then the invalidation reports are ignored and the data objects are unaffected. However, when an MU wakes up after a disconnection, all valid state cached data objects are changed to uncertain state.

Since Wang et al. had based their approach using the LRU (Least Recently Used) cache replacement strategy, every time a data object is cached or is already found in the cache, the object is moved to the head of the cache list. In case the cache is full and a new data object needs to be cached, to accommodate the new data object, and to make enough space only data entries with uncertain, ID-only, or valid states are deleted from the tail of the cache list. In case the cache needs to be refreshed, to validate the data of the cache, all data objects with uncertain or ID-only state are allocated their original places and if there is not enough space, then data entries found at the tail are removed.

3.1.3 The Algorithm and its Description

The SACCS algorithm is described in Figure 3.2 and Figure 3.3.
MSSMain() { 
    IF ( MSS gets Query(x) message )
        fetch data entry x from the database
        broadcast Vdata(x, d_x, t_x) to all MUs
        IF ( f_x == 0 )
            set f_x = 1
    IF ( MSS gets Uncertain(x; ts_x) message )
        fetch data entry x from the database
        IF ( t_x == ts_x )
            broadcast Confirmation(x, t_x) to all MUs
        ELSE
            broadcast Vdata(x, d_x, t_x) to all MUs
        IF ( f_x == 0 )
            set f_x = 1
    IF ( MSS gets Update(x, d'_x, t'_x) from original server )
        update the database entry with ID x as: d'_x = d_x and t'_x = t_x
        IF ( f_x == 1 )
            broadcast IR(x) to all MUs and reset f_x = 0
}

Figure 3.2 MSS Main pseudocode (Wang, 2003)

The MSSMain() is the procedure that handles the MU queries and the data objects updates. When an MSS gets a query for object x, it fetches the data and broadcasts the valid data to all MUs, setting the flag bit to 1. While if the message received is an uncertain message, it fetches the object x and checks if the last update time is equal to the time of the item x in the cache, if yes then it broadcasts a confirmation to all MUs, otherwise it broadcasts the valid data to all MUs. When an MSS gets an update message then it updates the item x. If the flag bit is set then an invalidation report is broadcasted to all MUs and the flag bit is reset.
MUMain() {
  IF (MU receives 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 (MU receives a Vdata(x, $d_x$, 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 ($t_s < t_x$)
          download $d_x$ to orginal entry location in cache
          set ts$_x = t_x$ and $s_x = 0$
        ELSE
          set the entry to valid entry ($s_x = 0$)
      IF (MU receives a IR(x) message)
        IF ($d_x$ is valid or uncertain in cache)
          delete $d_x$ and set $s_x = 3$
      IF (MU receives a Confirmation(x, t$_x$) message)
        IF (the entry x is uncertain entry in cache list)
          IF ($t_s = t_x$)
            set $s_x = 0$
          ELSE 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 ($s = 1$)
}

Figure 3.3 MU pseudocode (Wang, 2003)
The MU executes the MUMain() procedure continuously to receive its query and broadcast messages. When an MU gets a query message requesting a data object, first it needs to check its local cache, if the requested object is valid then the request is answered at once; otherwise, if it is in an uncertain state then a message is sent to the MSS to get the valid data. In case the data entry is in an ID-only state or not in the cache, then a query message is sent to the MSS to get the new value of the data object. When an MU gets a VData message if the data in the cache is in an uncertain state, then the entry is refreshed else if there is a query waiting for the data, then the MU responds to the query request and caches the object. Otherwise if it was an ID-only state, then the data object is downloaded. If an invalidation report (IR) message is received then the data entries are set to ID-only state. In case the MU cache receives a confirmation message and there is an uncertain entry corresponding to it, then the entry is refreshed.

3.1.4 Efficiency of SACCS

The SACCS algorithm is efficient and scalable too. Since data flags are used in the database, only those entries whose flags are set will need to broadcast invalidation reports to indicate that the data object is updated. Consequently, the bandwidth consumption is reduced, as invalidation reports bandwidth consumption frequency is less than the uplink query-confirmation or data object update commands.

A second positive feature in SACCS is the introduction of uncertain and ID-only states. When a data object is retrieved by any MU, it brings those uncertain or ID-only states back to valid state. Similarly, with a single confirmation message, all entries of the
data object in the uncertain state in connected MUs will change to valid or ID-only state. In addition to that, the uncertain state allows an MU to keep all valid data objects when it wakes up after a random sleep time. All these minimize the traffic of the uplink and download channels. As a result this is good for scalability, and makes it efficient for big number of MUs.

Having only a flag bit in the database, the database management is easier too. The overhead is minimal, since it only needs either to check the single bit that indicates the data object, or needs to set/reset this flag bit.
Chapter 4

The Cache Replacement Policies

4.1 SACCS and the Cache Replacement Policies

In general, cache replacement strategies affect on hit rates, however they are not the limiting factor for caching. These strategies have certain targets which are defined by the metric and are based on the workload.

While Wang et al.'s proposed cache consistency maintenance scheme SACCS has been based on the conventional and most popular cache replacement technique Least Recently Used (LRU), in this thesis; we examine the SACCS using a value-based function, the Least Unified Value (LUV). The LUV emphasizes the reference information of the object as well as takes into consideration the fetch cost of the object and its size.

SACCS's efficiency lies in its features that support to reduce the uplink bandwidth usage, therefore a good replacement strategy would be definitely a strategy that increases this impact even more.
4.1.1 A Value-based Function: Least-Unified Value

The Least Unified Value replacement algorithm is based on time of all past references and hence considers the number of references (Bahn, 2002). Its advantages are that it uses complete reference history and can optimize any performance measure. The only weakness of this strategy lies in how to consider the parameter tuning.

The LUV replacement strategy is a value-based strategy, which calculates for every data item $i$ a value $V(i)$, which is defined by the following formula

$$V(i) = W(i)p(i)$$

where $W(i)$ is the relative cost to fetch the object from its original server and is calculated by the ratio of cost to fetch $i$ from the server ($c(i)$) and the size of the object $i$ ($s(i)$).

$$W(i) = \frac{c(i)}{s(i)}$$

while $p(i)$ is the "probability" that object $i$ is referenced in the future and is calculated as

$$p(i) = \sum_{k=1}^{f} (t_c - t_k)$$

$t_c$ stands for the current time and $t_k$ stands for the oldest request time in a sliding window of $k$ request times. $F(x)$ should be a decreasing factor, to give more weight to more recent references. For example, if a data object $i$ is referenced at time $t_1$, $t_2$, and $t_3$, then $p(i)$ is calculated at current time $t_c$ by $F(t_c-t_1) + F(t_c-t_2) + F(t_c-t_3)$. 

35
A possibility for the function is \( F(x) = \frac{1}{2} \cdot 2^x \) \((0 \leq \lambda \leq 1)\). Adding that \( \lambda \) converging to 1 will reduce it to LRU where only the last reference time is considered, while \( \lambda \) converging to 0 will reduce it to a weighted LFU, counting the number of previous references.

To predict the future references, one can choose either to save the reference history of all data objects available in the cache (in-cache-history) or to save the reference history of data objects even after eviction. In this way, whenever the object is returned to the cache the information could be used.

### 4.1.2 The New Algorithm for SACCS and its Description

The algorithm first checks if the data is available in the cache, if yes i.e. the data object is in valid state then updating the time of the last reference, it calculates the new value. Adding this new value to the heap, it restores the heap property. If the data is available in cache with uncertain state, it calculates the new value at the new referenced time, and then sends a message to check the validity of the data. In case the data is in ID-only state or not available in the cache it fetches the data and its value, adds the time of the last reference and calculates the new value, inserting it to the heap, it adjusts the heap. If there is not enough space to download the data then it finds the data object with the lowest value and replaces it with the new data object, adds the value of the newly introduced data object and restores the heap.
The algorithm for the new proposed cache replacement strategy is shown in Figure 4.1. Note that \( d_x \) stands for the data object, \( M_x \) is the message for the data object, \( d_y \) stands for the data object that will be replaced, \( LUV \) is the value calculated for the data object, \( L \) is the minimum value.
Case 1: \( d_\kappa \) is in cache and valid

then calculate the LUV value

return \( d_\kappa \) to the application.

Case 2: \( d_\kappa \) is in cache and uncertain

then calculate the new LUV value

send uncertain message to the server.

Case 3: \( d_\kappa \) is not cached or ID-only

Send cache missing message to the server.

Wait for message \( M_\kappa \) to appear at downlink channel

If \( M_\kappa \) is confirmation then set the state of \( d_\kappa \) as valid

Return \( d_\kappa \) to the application,

If \( M_\kappa \) is the data item \( d_\kappa \) then

While there is not enough space for \( d_\kappa \)

Find min value \( L = \) Minimum LUV value for data object \( y \) belonging to the cache

Evict the \( d_y \) such that LUV Value of \( y = L \);

Keep value of the evicted data object

End while

Bring \( M_\kappa \) into cache

Calculate its LUV value

Return \( M_\kappa \) to the application.

Figure 4.1 New SACCS Algorithm
Like many other replacement algorithms that base their decision on the ordering of data objects by a given criterion, LUV uses the heap structure to maintain the ordering of data items according to their LUV values. The root of the heap certainly has the lowest value. Therefore, when the cache replacement takes place, the object found at the root of the heap is the object to be deleted. The delete action is repeated until we get enough space is for the incoming data object.

Every deletion and insertion operation has a complexity of $O(\log_2 N)$. So if we have $N$ number of cached items and $M$ victim set size, the time complexity for every cache replacement operation is $O(M\log_2 N)$. To find the object it only needs $O(1)$. In addition, for every object’s whose value is updated, the heap needs to be adjusted, to allocate the new entry with the correct position. The time complexity for every adjustment operation is $O(\log_2 N)$.

### 4.1.3 The Other Cache Replacement Strategies

To examine the efficiency of our proposed new strategy LUV, we compared it with four different cache replacement algorithms, each of them belonging to a different classification of cache replacement strategies; recency-based, frequency-based, recency and frequency based and random.
4.1.3.1 A Recency-Based Strategy: Least Recently Used (LRU)

The recency-based strategies in general are used to replace objects that were used least recently. Their implementation is fairly easy.

The LRU cache replacement strategy that Wang et al. have already based on their hybrid cache consistency replacement SACCS considered the reference time of the data object. This algorithm optimizes for byte hit ratio, it is easy, with time complexity $O(1)$ and simple to implement. Its weakness lies in the fixed performance measure, as it considers partial aspects of reference history. This is a conventional cache replacement strategy that is very popular and used in several applications.

4.1.3.2 A Frequency-Based Strategy: Least Frequently Used (LFU)

The frequency-based strategies which replace the data objects that were used least frequently are again used in many applications and just like the recency-based strategies are easy to be implemented. This algorithm is similar to the LRU, optimizes for byte hit ratio. It has time complexity $O(\log_2 N)$. Another disadvantage for frequency based replacement is that many data objects may have the same frequency count and then there is a need for a tie breaker.

In the LFU cache replacement policy, the frequency of references or accesses for each data entry in the mobile user cache list is counted. The tail of the list contains the data with the minimum number of accesses.
4.1.3.3 A Recency/Frequency Based Strategy: Least Recently/Frequently Used (LRFU)

The LRFU Least Recently/Frequently Used policy inherits the benefits of the two policies (LRU and LFU) and results in a policy that is better than both. To each data object it assigns a value. This value combines recency and frequency, in this way it computes the probability of referencing the object in the near future soon. Every time the data object is referenced a weighing function \( F(x) \) is calculated which considers the data objects reference time span from the past to the current. Usually, this is a decreasing function. Therefore, the more recently the data object is referenced, the bigger weight this data will have.

Combining recency and frequency is advantageous, as they resolve the issues faced in recency-based and frequency-based strategies mentioned earlier. However, to combine recency and frequency the strategy may become more complex. (Lee, 1999).

4.1.3.4 A Randomized Strategy: Random

The randomized strategies are different from the previously mentioned strategies in the sense they are nondeterministic approaches. On one hand, they try to reduce the complexity of the replacement strategy but they are awkward to evaluate.

The random strategy uses randomized decisions to remove and replace an object from the cache (Podlipnig, 2003). The advantages of this strategy are that it does not need special data structure for inserting or deleting object and that it is simple to
implement. The disadvantage is that it cannot be evaluated and different simulation runs will give different results.
Chapter 5

Performance Evaluation

5.1 Environment

We tested the performance of our model by means of a simulated environment in C++. For our simulation we considered a single cell environment with 100 MUs as clients and each MU with identical cache size 300. We had also considered 1000 data objects of five types of access of random object sizes (bytes) and variable average update interval (sec).

The sleep wakeup process is modeled as two-state Markov chain with MUs alternating between sleep and awake states.

Each MU has a sleep-wakeup period randomly picked from the set of values (500, 1000, 1500, 2000, 2500) sec. The sleep ratio is picked from (0.1, 0.3, 0.5, 0.7, 0.9) and the request arrival rate from (1/10, 1/60, 1/110, 1/160, 1/210).

When an MU is in the sleep state, all requests are ignored. The query delay is counted as 0, when a requested data object is available at the MU. Otherwise, the query delay is counted as the time interval between the query response and query initiation. An uplink is counted when a query is retrieved from the Base Station through an uplink channel.
A zipf-like distribution for MU access pattern is used in the simulation with $z$ equal to 1. The update process for a data object and the arrival requests follow a Poisson distribution.

The channel is used for downlink and uplink data transmission with a bandwidth 1250 bps. Uplink message size is assumed 64 bytes and downlink message size as 64 bytes.

As for the function parameters used for the LUV, we considered $\lambda$ to be equal to 0.5 and considered variable and random fetching cost and size ratios.

5.2 Simulation Results

The performance of SACCs based on LUV value-based cache replacement policy is evaluated and compared to SACCS based on LRU, LFU, LRFU and Random representative cache replacement policies of the remaining other four categories of strategies.

Our evaluation measures the following metrics: hit ratio and miss ratio, total hit and total miss, average delay, total bandwidth, bytes per query, data download per query.
When an MU receives a query, if the queried data object is valid in the cache, a cache hit is counted, and no uplink is needed for the query. The higher the hit ratio is the fewer the uplink per query.

![TOTAL HIT Graph](image)

Figure 5.1 Total Hit for Simulation

<table>
<thead>
<tr>
<th>TOTAL QUERY</th>
<th>TOTAL HIT</th>
<th>TOTAL MISS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>16751</td>
<td>11666</td>
</tr>
<tr>
<td>LFU</td>
<td>16688</td>
<td>12519</td>
</tr>
<tr>
<td>LRFU</td>
<td>16695</td>
<td>12206</td>
</tr>
<tr>
<td>LUV</td>
<td>16726</td>
<td>11599</td>
</tr>
<tr>
<td>Random</td>
<td>16567</td>
<td>11871</td>
</tr>
</tbody>
</table>

Table 5.1 Total Query, Total Hit, Total Miss

As it is shown in Table 5.1, the total hit for LUV is the highest. Figure 5.1 shows that the LUV cache replacement policy improves performance since the data requested is available in the cache, it will reduce the IR message broadcasts, avoiding the unnecessary traffic and retaining the valid data objects of the MU.
Figure 5.2 Number of Hits vs. Time

Figure 5.3 Number of Misses vs. Time
In the following figures, the results of the number of misses and number of hits (Figure 5.2 and Figure 5.3) and the miss and hit ratios (Figure 5.4 and Figure 5.5) are depicted for the five cache replacement strategies, over eight simulation time units with an interval of 50000 msec of simulation time. The miss (hit) ratio is the ratio of the number of unfound (found) data items in the cache over the number of all requested data. The worst hit ratios performance is for LFU, while LRU and LUV have the best hit ratio performances interchangeably. However, looking at the average the LUV outperforms the LRU.

![Miss Ratio vs. Time](image)

**Figure 5.4 Miss Ratio vs. Time**
A delay is the period of time between the time a request is issued and the time the result is received by the mobile application user. The average access delay is an important measurement of system performance. A shorter delay implies better performance. The total delay results of our simulation are presented in Figure 5.6 and the average delay results in Figure 5.7.

It is obvious that the tradeoff between energy cost and access latency is a hard one, we can decrease the uplink and download messages or improve the access latency to decrease the energy cost.
Figure 5.6 Total Delay vs. Time

Figure 5.7 Average Delay Vs. Time
In Figure 5.8 based on the Table 5.2 results show that LUV results in less bytes/query, outperforming the other four cache replacement strategies. This is due to the fact that LUV not only considers the most recent data information but also future references according to their fetch cost. The decision of evicting data objects with low fetching costs is a smart way to save power consumption at a later stage.

Figure 5.8 Bytes per Query

Table 5.2 Bytes per Query

<table>
<thead>
<tr>
<th></th>
<th>BYTES/QUERY</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>793.541</td>
</tr>
<tr>
<td>LFU</td>
<td>978.204</td>
</tr>
<tr>
<td>LRFU</td>
<td>920.849</td>
</tr>
<tr>
<td>LUV</td>
<td>791.037</td>
</tr>
<tr>
<td>Random</td>
<td>859.57</td>
</tr>
</tbody>
</table>
In Figure 5.9 based on the results in the Table 5.3 LUV has the lowest ratio for data download/query. This means the value-based cache replacement strategy LUV is quite efficient and its selection of the victims set has saved unnecessary downloads. Since the algorithm favors the data objects that have low fetch cost values, it has saved fetching costs, which implies that it is less power consuming.

![Data Download/Query Diagram](image)

Figure 5.9 Data Download per Query

Table 5.3 Data Download per Query

<table>
<thead>
<tr>
<th></th>
<th>Data Download/Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>0.541938</td>
</tr>
<tr>
<td>LFU</td>
<td>0.647951</td>
</tr>
<tr>
<td>LRFU</td>
<td>0.609404</td>
</tr>
<tr>
<td>LUV</td>
<td>0.534617</td>
</tr>
<tr>
<td>Random</td>
<td>0.585622</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusion

Caching is a good solution for mobile environments that are characterized with several constraints such as low bandwidth for uplink, irregular connections, and limited client resources. Its main aim is to minimize query latency or access delay and bandwidth usage, especially download traffic. However, caching has some limitations that were addressed in this work.

Throughout the literature, several approaches had been suggested to maintain cache consistency. In the stateful approaches the server knows what data was cached in which mobile unit while in stateless approaches the server is unaware of these information. The disadvantages of the stateful approaches are that the database maintenance is more difficult while the drawback for the stateless approaches is their scalability. An efficient and scalable hybrid caching maintenance approach SACCS has been suggested, which is based on LRU. Since caches are limited in size the cache replacement strategies play a central role in cache management.

In this thesis, we proposed the value-based function LUV for cache replacement algorithm implemented with SACCS. The LUV strategy, which considers several factors such as the reference probability, the cost of fetching of a data object and its size, was shown to be an efficient strategy. A good replacement policy is one that is used as infrequently as possible to generate the same hit rates. By selecting the set of victims
that have a potential of being referenced in the near future and taking into consideration at the same time the cost of fetching the data the LUV is considered a good strategy. All these parameters are based on the complete history of the data object.

The proposed strategy was compared with other strategies that belong to different categories of cache replacement strategies. As a result, we found out that it is more efficient than the LRU strategy, which was a recent strategy used by Wang et al. We used a fixed parameter $\lambda$ to calculate the function value for a data object. For future work we propose to find an adaptive function for the $\lambda$ parameter, according to the query rate and client disconnection. Also, in the function, we need to consider the update frequency of the data object.
References


International Conference on Distributed Computing Systems Workshops (ICDCSW’03). pp.1-6.


Appendix SACCs Simulation Results

A-1 LRU Simulation Results

```
point: 0 time:50000
runtime is:11254.5 scht ime is:11254.8
Total query is:6255 Total hit is: 1122 Total miss is:5129 ms: 4695
Total delay is: 10258 Average delay:1.6397
Total bndt uth is:7.09465e+006
Bytes per query: 1134.05
The miss ratio is: 0.019053
Data download per query is:0.75048

point: 0 time:100000
runtime is:22807.1 scht ime is:22807.6
Total query is:6220 Total hit is: 1591 Total miss is:4625 ms: 3768
Total delay is: 6673.46 Average delay:1.07612
Total bndt uth is:5.64939e+006
Bytes per query: 900.262
The miss ratio is: 0.743569
Data download per query is:0.605788

point: 0 time:150000
runtime is:34497.3 scht ime is:34497.3
Total query is:6398 Total hit is: 1962 Total miss is:4441 ms: 3492
Total delay is: 6013.1 Average delay:0.939841
Total bndt uth is:5.25491e+006
Bytes per query: 821.336
The miss ratio is: 0.694123
Data download per query is:0.545796

point: 0 time:200000
runtime is:46198.6 scht ime is:46196.8
Total query is:6504 Total hit is: 1964 Total miss is:4540 ms: 3504
Total delay is: 6002.4 Average delay:0.935178
Total bndt uth is:5.23084e+006
Bytes per query: 788.875
The miss ratio is: 0.698032
Data download per query is:0.538745

point: 0 time:250000
runtime is:57821.5 scht ime is:57814.9
Total query is:6422 Total hit is: 1920 Total miss is:4504 ms: 3409
Total delay is: 5852.31 Average delay:0.911291
Total bndt uth is:5.10272e+006
Bytes per query: 794.569
The miss ratio is: 0.701339
Data download per query is:0.530332
```
```plaintext
point: 0 time:2538000
runtime is:57821.5 schtme is:57814.9
Total query is:6422 Total hit is: 1720 Total miss is:4504 ms: 3409
Total delay is: 5852.31 Average delay:0.911291
Total bnduth is:5.10272e+006
Bytes per query: 794.569
The miss ratio is: 0.701339
Data download per query is:0.530832

point: 0 time:300000
runtime is:69457.3 schtme is:69450.8
Total query is:6631 Total hit is: 2088 Total miss is:4543 ms: 3526
Total delay is: 6200.65 Average delay:0.935101
Total bnduth is:5.24798e+006
Bytes per query: 791.432
The miss ratio is: 0.685115
Data download per query is:0.531745

point: 0 time:350000
runtime is:81089.7 schtme is:81087.7
Total query is:6404 Total hit is: 1922 Total miss is:4481 ms: 3504
Total delay is: 5844.34 Average delay:0.912607
Total bnduth is:5.09614e+006
Bytes per query: 795.775
The miss ratio is: 0.699719
Data download per query is:0.547158

point: 0 time:400000
runtime is:92898.3 schtme is:92895.8
Total query is:6607 Total hit is: 2024 Total miss is:4585 ms: 3598
Total delay is: 6121.15 Average delay:0.926464
Total bnduth is:5.27476e+006
Bytes per query: 790.359
The miss ratio is: 0.693961
Data download per query is:0.544574
Total query is:16751 Total hit is: 5006 Total miss is:11666 ms: 9070
Total delay is: 15345.9
Total Bytes: 1.32926e+007
Bytes per query: 793.541
Average delay:0.916117
The miss ratio is: 0.696436
The data download per query is:0.541938
continue? (0--yes, 1--for no)
```

Figure A.1 LRU Simulation Output
A-2 LFU Simulation Results

******************************************************************************
point: 0 time:2500000
runtime is:57177.4 schtime is:57177.4
Total query is:6372 Total hit is: 1579 Total miss is:4795 ms: 4102
Total delay is: 8097.07 Average delay:1.27053
Total bndvth is:6.31657e+006
Bytes per query: 991.145
The miss ratio is: 0.752393
Data download per query is:0.643653
******************************************************************************
point: 0 time:300000
runtime is:68670.9 schtime is:68671.4
Total query is:6469 Total hit is: 1672 Total miss is:4797 ms: 4159
Total delay is: 8421.65 Average delay:1.30185
Total bndvth is:6.38091e+006
Bytes per query: 987.519
The miss ratio is: 0.741537
Data download per query is:0.642912
******************************************************************************
point: 0 time:350000
runtime is:80110.5 schtime is:80107.7
Total query is:6265 Total hit is: 1539 Total miss is:4728 ms: 4091
Total delay is: 8509.93 Average delay:1.35833
Total bndvth is:6.16188e+006
Bytes per query: 983.589
The miss ratio is: 0.754669
Data download per query is:0.652993
******************************************************************************
point: 1 time:400000
runtime is:91751.9 schtime is:91752.3
Total query is:6501 Total hit is: 1649 Total miss is:4849 ms: 4198
Total delay is: 8050.14 Average delay:1.23829
Total bndvth is:6.32857e+006
Bytes per query: 974.587
The miss ratio is: 0.745805
Data download per query is:0.645747
Total query is:16688 Total hit is: 4172 Total miss is:12519 ms_t10813
Total delay is: 21750.9
Total Bytes : 1.63243e+007
Bytes per query: 978.294
Average delay:1.30339
The miss ratio is: 0.75018
The data download per query is:0.647951
continue? (0--yes, 1--for no)
Figure A.2 LFU Simulation Output
A-3 LRFU Simulation Results

******************************************************************************
point: 0 time:50000
runtime is:11446.2 schtme is:11444.4
Total query is:6424 Total hit is: 1086 Total miss is:5330 ms: 4977
Total delay is: 11352.6 Average delay:1.76722
Total bndth is:7.4841?e+006
Bytes per query: 1165.03
The miss ratio is: 0.827701
Data download per query is:0.774751
******************************************************************************
point: 0 time:100000
runtime is:22546.5 schtme is:22545.4
Total query is:6266 Total hit is: 1456 Total miss is:4817 ms: 4130
Total delay is: 8040.34 Average delay:1.28445
Total bndth is:6.27709e+006
Bytes per query: 1001.77
The miss ratio is: 0.760752
Data download per query is:0.659113
******************************************************************************
point: 0 time:150000
runtime is:34047.2 schtme is:34047.2
Total query is:6496 Total hit is: 1683 Total miss is:4811 ms: 4009
Total delay is: 7219.71 Average delay:1.11141
Total bndth is:6.04638e+006
Bytes per query: 933.557
The miss ratio is: 0.74061
Data download per query is:0.617149
******************************************************************************
point: 0 time:200000
runtime is:45812 schtme is:45813
Total query is:6509 Total hit is: 1840 Total miss is:4666 ms: 3943
Total delay is: 7099 Average delay:1.09064
Total bndth is:5.96999e+006
Bytes per query: 917.19
The miss ratio is: 0.716854
Data download per query is:0.605777
******************************************************************************
point: 0 time:250000
runtime is:57287.9 schtme is:57287.9
Total query is:6428 Total hit is: 1694 Total miss is:4736 ms: 3095
Total delay is: 7444.62 Average delay:1.15815
Total bndth is:5.93123e+006
Bytes per query: 922.717
The miss ratio is: 0.736777
Data download per query is:0.605943
******************************************************************************
Figure A.3 LRFU Simulation Output
A-4 LUV Simulation Results

******************************************************************************
point: 0 time:50000
runtime is:11180.9 sctime is:11181
Total query is:6427 Total hit is: 1165 Total miss is:5253 ms: 4886
Total delay is: 10978.5 Average delay:1.70819
Total bnduth is:7.38429e+006
Bytes per query: 1148.95
The miss ratio is: 0.81733
Data download per query is:0.76023
******************************************************************************
point: 0 time:100000
runtime is:22608.1 sctime is:22608.2
Total query is:6299 Total hit is: 1514 Total miss is:4792 ms: 4053
Total delay is: 7665.43 Average delay:1.21693
Total bnduth is:6.12218e+006
Bytes per query: 971.928
The miss ratio is: 0.760756
Data download per query is:0.643435
******************************************************************************
point: 0 time:150000
runtime is:34134.7 sctime is:34126
Total query is:6513 Total hit is: 1740 Total miss is:4772 ms: 3947
Total delay is: 6962.02 Average delay:1.06906
Total bnduth is:5.91767e+006
Bytes per query: 908.593
The miss ratio is: 0.732688
Data download per query is:0.606019
******************************************************************************
point: 0 time:200000
runtime is:45911.9 sctime is:45912.7
Total query is:6526 Total hit is: 1873 Total miss is:4646 ms: 3843
Total delay is: 6779.22 Average delay:1.03976
Total bnduth is:5.7619e+006
Bytes per query: 883.726
The miss ratio is: 0.712577
Data download per query is:0.589417
******************************************************************************
point: 0 time:250000
runtime is:57425.8 sctime is:57425.5
Total query is:6444 Total hit is: 1784 Total miss is:4660 ms: 3773
Total delay is: 7135.37 Average delay:1.10729
Total bnduth is:5.72938e+006
Bytes per query: 889.184
The miss ratio is: 0.723153
Data download per query is:0.505506
******************************************************************************
<table>
<thead>
<tr>
<th>point: 0 time:250000</th>
<th>runtime is:57425 schttime is:57425.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total query is:6444 Total hit is: 1784 Total miss is:4660 ms: 3773</td>
<td></td>
</tr>
<tr>
<td>Total delay is: 7135.37 Average delay:1.10729</td>
<td></td>
</tr>
<tr>
<td>Total bandwidth is:5.72938e+006 Bytes per query: 889.104</td>
<td></td>
</tr>
<tr>
<td>The miss ratio is: 0.723153</td>
<td></td>
</tr>
<tr>
<td>Data download per query is:0.585506</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>point: 0 time:300000</th>
<th>runtime is:68950.8 schttime is:68959.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total query is:6538 Total hit is: 1792 Total miss is:4748 ms: 3872</td>
<td></td>
</tr>
<tr>
<td>Total delay is: 7324.04 Average delay:1.12023</td>
<td></td>
</tr>
<tr>
<td>Total bandwidth is:5.86161e+006 Bytes per query: 896.545</td>
<td></td>
</tr>
<tr>
<td>The miss ratio is: 0.726216</td>
<td></td>
</tr>
<tr>
<td>Data download per query is:0.59223</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>point: 0 time:350000</th>
<th>runtime is:80420.2 schttime is:80418.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total query is:6429 Total hit is: 1802 Total miss is:4625 ms: 3761</td>
<td></td>
</tr>
<tr>
<td>Total delay is: 8555.3 Average delay:1.05075</td>
<td></td>
</tr>
<tr>
<td>Total bandwidth is:5.65897e+006 Bytes per query: 880.225</td>
<td></td>
</tr>
<tr>
<td>The miss ratio is: 0.719396</td>
<td></td>
</tr>
<tr>
<td>Data download per query is:0.585005</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>point: 0 time:400000</th>
<th>runtime is:92195 schttime is:92192.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total query is:6513 Total hit is: 1847 Total miss is:4666 ms: 3809</td>
<td></td>
</tr>
<tr>
<td>Total delay is: 7098.97 Average delay:1.08997</td>
<td></td>
</tr>
<tr>
<td>Total bandwidth is:5.76816e+006 Bytes per query: 885.637</td>
<td></td>
</tr>
<tr>
<td>The miss ratio is: 0.716413</td>
<td></td>
</tr>
<tr>
<td>Data download per query is:0.58403</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Bytes : 1.46448e+007</th>
<th>Total delay : 17643.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes per query: 876.774</td>
<td>Average delay:1.0563</td>
</tr>
<tr>
<td>The miss ratio is: 0.716927</td>
<td>The data download per query is:0.504626</td>
</tr>
</tbody>
</table>

continue? (0--yes, 1--for no)
A-5 Random Simulation Results

*******************************************************************************
point: 0 time:50000
runtime is:11294.2 schtime is:11294.8
Total query is:6349 Total hit is: 1001 Total miss is:5268 ms: 4894
Total delay is: 11975.6 Average delay:1.98638
Total bndwidth is:7.41947e+006
Bytes per query: 1168.6
The miss ratio is: 0.828477
Data download per query is:0.77083
Delay 1: 665.992
*******************************************************************************
point: 0 time:100000
runtime is:22664.5 schtime is:22664.3
Total query is:6460 Total hit is: 1620 Total miss is:4840 ms: 4088
Total delay is: 8838.24 Average delay:1.36815
Total bndwidth is:6.13479e+006
Bytes per query: 949.648
The miss ratio is: 0.749226
Data download per query is:0.632817
Delay 1: 855.022
*******************************************************************************
point: 0 time:150000
runtime is:34136.7 schtime is:34136.5
Total query is:6464 Total hit is: 1715 Total miss is:4785 ms: 3930
Total delay is: 8611.27 Average delay:1.33219
Total bndwidth is:5.86114e+006
Bytes per query: 906.729
The miss ratio is: 0.735613
Data download per query is:0.607903
*******************************************************************************
point: 0 time:200000
runtime is:45738.7 schtime is:45737.5
Total query is:6493 Total hit is: 1881 Total miss is:4612 ms: 3780
Total delay is: 8503.24 Average delay:1.32404
Total bndwidth is:5.52601e+006
Bytes per query: 852.394
The miss ratio is: 0.709857
Data download per query is:0.583063
Delay 1: 362.295
*******************************************************************************
point: 0 time:250000
runtime is:57441.3 schtime is:57440.2
Total query is:6444 Total hit is: 1896 Total miss is:4545 ms: 3615
Total delay is: 7416.32 Average delay:1.1509
Total bndwidth is:5.34392e+006
Bytes per query: 829.271
The miss ratio is: 0.706307
Data download per query is:0.560987
*******************************************************************************
Figure A.5 Random Simulation Output