A Power-Aware Low-Latency Cache Management Architecture for Mobile Computing Environments

G. Anandharaj
Lecturer, Sengunthar Engineering College, Tiruchengode
Tamil Nadu, India

Dr. R. Anitha
Professor and Director, K.S. Rangasamy College of Technology, Tiruchengode
Tamil Nadu, India

Summary
Caching frequently accessed data items on the client side is an effective technique to improve performance in a mobile environment. Classical cache management strategies may not be suitable for mobile environments due to the disconnection and mobility of the mobile clients. In this paper, we propose a novel Cache Management Architecture (CMA) for mobile hosts. The goal of our architecture is to reduce the caching overhead and provide optimal replacement policy. It aims to improve the network utilization, reduce the search latency, bandwidth and energy consumption. The architecture comprises of the following algorithms: cache placement algorithm, cache discovery algorithm and, cache replacement algorithm. By simulation results, we will show that our proposed architecture achieves lower latency and packet loss, reduced network bandwidth consumption, reduced data server workload.
Keyworks

1. Introduction

In a mobile computing system, the geographical area is divided into small regions, called cells. Each cell has a *base station* (BS) and a number of *mobile terminals* (MTs). Inter cell and intra-cell communications are managed by the BSs. The MTs communicate with the BS by wireless links. An MT can move within a cell or between cells while retaining its network connection. An MT can either connect to a BS through a wireless communication channel or disconnect from the BS by operating in the *doze* (power save) mode. [1]

The mobile computing platform can be effectively described under the *client/server* paradigm. A data item is the basic unit for update and query. MTs only issue simple requests to read the most recent copy of a data item. There may be one or more processes running on an MT. These processes are referred to as clients. In order to serve a request sent from a client, the BS needs to communicate with the database server to retrieve the data items.

Caching frequently accessed data items on the client side is an effective technique to improve performance in a mobile environment. Classical cache management strategies may not be suitable for mobile environments due to the disconnection and mobility of the mobile clients. In general caching results in

1. Enhanced QoS at the clients – i.e., lower jitter, latency and packet loss,
2. Reduced network bandwidth consumption, and
3. Reduced data server/source workload.

Caching plays a key role in mobile computing because of, its ability to alleviate the performance and availability limitations of weakly-connected and disconnected operation. But evaluating alternative caching strategies for mobile computing is problematic. Cache management in mobile environment, in general, includes the following issues to be addressed: [5]

1. The cache discovery algorithm that is used to efficiently discover, select, and deliver the requested data item(s) from neighboring nodes.

2. Cache admission control – this is to decide on what data items can be cached to improve the performance of the caching system.

3. The design of cache replacement algorithm – when the cache space is sufficient for storing one new item, the client places the item in the cache. Otherwise, the possibility of replacing other cached item(s) with the new item is considered.

4. The cache consistency algorithm which ensures that updates are propagated to the copies elsewhere, and no stale data items are present.

None of the existing works give a combined and complete solution for efficient cache placement, discovery and replacement.

We propose a novel Cache Management Architecture (CMA) for mobile hosts. The goal of our architecture is to reduce the caching overhead and provide optimal replacement policy. It aims to improve the network utilization, reduce the search latency, bandwidth and

Manuscript received October 5, 2008
Manuscript revised October 20, 2008

energy consumption. The architecture comprises of the following algorithms:

- Cache placement algorithm
- Cache discovery algorithm and
- Cache replacement algorithm

This paper is organized as follows. Section 2 gives the detailed related work done. Section 3 presents the system model for our architecture. Section 4 presents the cache placement algorithm, followed by the cache discovery and cache replacement algorithms in section 5 and section 6 respectively. Section 7 gives the experimental results and section 8 concludes the paper.

2. Related Work

Barbara and Imielinski [2] proposed a cache solution which is suitable for mobile environments. In this approach, the server periodically broadcasts an *invalidation report (IR)* in which the changed data items are indicated. Rather than querying the server directly regarding the validation of cached copies, the clients can listen to these IRs over the wireless channel, and use them to validate their local cache. The IR-based solution is attractive because it can scale to any number of clients who listen to the IR. However, the IR-based solution has some major drawbacks such as long query latency and low bandwidth utilization.

In [3], Guohong Cao addressed an UIR-based approach. In this approach, a small fraction of the essential information (called updated invalidation report (UIR)) related to cache invalidation is replicated several times within an IR interval, and hence the client can answer a query without waiting until the next IR. However, if there is a cache miss, the client still needs to wait for the data to be delivered.

To increase the cache hit ratio and reduce the bandwidth consumption, clients intelligently prefetch the data that are most likely used in the future. However, prefetch consumes power. In [4], the client marks some invalid cache entries as non-prefetch and it will not prefetch these items. For each cached item, the client records the prefetch-access ratio (PAR), which is the number of prefetches divided by the number of accesses. When power consumption becomes an issue, the client marks those cache items which have PAR > B as *non-prefetch*, where B > 1 is a system tuning factor. With a small B, more energy can be saved, but the cache hit ratio may be reduced. On the other hand, with a large B, the cache hit ratio can be improved, but at a cost of more energy consumption. Note that when choosing the value of B, the uplink data request cost should also be considered. The main drawback of this approach is the appropriate selection of B, such that cache hit ratio should be improved and energy consumption should be reduced.

In [6], a new hybrid adaptive caching technique, which combines page and object caching to reduce the miss rate in client caches dramatically is presented. HAC, is a novel technique for managing the client cache in a distributed, persistent object storage system.

In [7], an overview of a series of web cache replacement algorithms based on the idea of preserving a history record for cached Web objects is presented. The number of references to Web objects over a certain time period is a critical parameter for the cache content replacement.

In [8], the preliminary design of an adaptive caching scheme using multiple experts, called ACME is described. ACME is used to manage the replacement policies within distributed caches to further improve the hit rates over static caching techniques.

In [9], developing efficient caching techniques in ad hoc networks with memory limitations are focused. Research into data storage, access, and dissemination techniques in ad hoc networks is not new. In particular, these mechanisms have been investigated.

In [10], the feasibility of having a global replacement algorithm is proposed. In that efficient cache hierarchy, several global replacement policies and their behavior with several benchmarks using a cycle accurate simulator are discussed.

3. System Model

3.1. Network model

Consider a mobile environment with n cells $C_1, C_2, \ldots C_n$. For each cell C_i, DS_i is the database server that can keep pieces of information that may be accessed by other systems. We assume that the database is updated only by the server. A client is a system, which invokes queries for data. Each cell C_i contains a set of clients $S_1, S_2, \ldots S_m$.

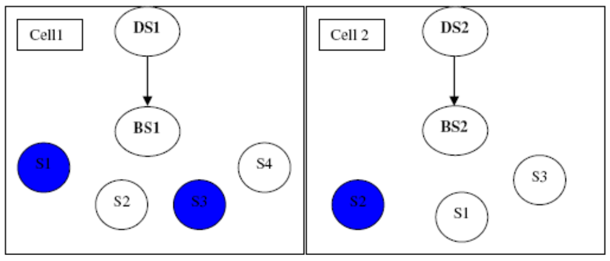

Fig 1 Mobile Network Model

Each client S_j of the cell C_i can issue the query through the base station BS_i which is directly connected to the database server DS_i . A database server (simply, server hereafter) can contain more than one database and can indirectly communicate with all mobile clients in the same cell through the Base station BS_i . A database can be cached in one or more clients in a cell. In Figure 1, we show an example of a network, which include clients, base station and database servers. The cached clients are denoted as blue circles.

3.2 Overview of the Architecture

Cache Placement Algorithm: In this algorithm, data caches are placed into some clients based on their weight vector which comprise the following parameters:

- Available Bandwidth
- CPU Speed
- Access Latency
- Cache Hit Ratio

Active nodes belonging to the neighborhood of a given client form a cooperative cache system for this client, since the cost for communication with them is low both in terms of energy consumption and message exchanges. For a data miss in the local cache, the client first searches the data item in its zone before forwarding the request to the next client that lies on a path towards server.

Cache Discovery Algorithm: In this algorithm, when a data request is initiated at a client, it first looks for the data item in its own cache. If there is a local cache miss, the client will send broadcasts request packet to the set of active client. When an active client receives the request and has the data item in its local cache, it will send an *ack* packet to the requester to acknowledge that it has the data item.

Cache Replacement Algorithm: In the cache replacement algorithm, we propose to develop a Least Relevant *Value* (LR*V*) based cache replacement policy, where data with the lowest LRV are removed from the cache. The LRV is based on the following factors:

- Access probability: It is based on the previous access rate of a data item for a host
- Distance: It is measured as the number of hops between the requesting client and the responding client.
- Size :A data item with larger data size should be chosen for replacement, because the cache can accommodate more data items and satisfy more access requests

In the subsequent sections, we present the detailed design of each of these algorithms.

4. Cache Placement Algorithm

The cache placement algorithm is described below:

For each client of the cell C_j, $\mathrm{j} = 1,2.......\mathrm{n}$, let

BW_i - Available bandwidth

SP_i - CPU speed

AL_i - Access Latency

CRi - Cache Hit Ratio $where\ \mathrm{i} = 1,2.....\mathrm{m}$

1. The weight of the client can be calculated as

$$W_i = (BW_i + SP_i + CR_i) \div AL_i \qquad (1)$$

2. Form the vector $W = \{S_i, W_i\}$, which denotes the client ids and their corresponding weight values, sorted on the descending order.

3. Denote the set of active nodes $S_k, (0 <= \mathrm{k} < \mathrm{m})$, which satisfies the following condition $W_k > \beta$, where β is the minimum threshold value for the weight .

4. Each database server DS_j caches the databases into the active nodes set S_k .

5. Cache Discovery

The mobile clients that belongs to the active node set then a cooperative cache system for other clients, since the cost for communicating with them is low both in terms of energy consumption and message exchange.

For each request, one of the following three cases holds:
Case 1: Local hit. When a copy of the requested data item is stored in the cache of the requester. If the data item is valid, it is retrieved to serve the query and no cooperation is necessary.

Case 2: Active hit. When the requested data item is stored in the cache of one or more active node neighbors of the requester.

Case3: Global hit. Data item is retrieved from the database server.

5.1. Cache Discovery Algorithm

A cache discovery algorithm is needed to determine if and where the requested item is cached when the requester does not have knowledge of the destination.

(i) Once a set of active clients is formed, the server broadcasts the vector $\{S_k, d_{kj}\}$ to all clients, where d_{kj}, $j = 1,2......$ is the index of the cached items placed in the active client $S_k, k = 1,2....$

(ii) When a data request is initiated at a client, it first looks for the data item in its own cache (local hit). If there is a local cache miss, the client broadcasts request packet to the set of active clients.

(iii) When an active client receives the request packet and has the data item in its local cache (i.e., a active hit), it will send an *ack* packet to the requester to acknowledge that it has the data item. The ack packet will contain the following fields: time stamp Ts and weight value W. The time stamp field helps to choose the latest copy of the searched item and the weight value field helps to choose the best client node.

(iv)When the query client receive *ack* packets from the active clients, it selects the best active client Sbest with max (T_s, W) and sends a *confirm* packet to the client Sbest. The *ack* packets for the same item received from other clients are discarded

(v) When the client Sbest receives a *confirm* packet, it responds back with the actual data value to the requested query node.

6. Cache Replacement

A cache replacement policy is required when a client wants to cache a data item, but the cache is full, and thus it needs to victimize a suitable subset of data items to evict from the cache. Cache replacement policies have been extensively studied in operating systems, virtual memory management and database buffer management.

- The data item size may not be fixed, the used replacement policy must handle data items of varying sizes.
- The data item's transfer time might depend on the item's size and the distance between the requesting client and the data source (or cache). Consequently, the cache hit ratio might not be the most accurate measurement of a cache replacement policy's quality.
- The replacement algorithm should also consider cache consistency, that is, data items that tend to be inconsistent earlier should be replaced earlier.

6.1 Cache Replacement Algorithm

We have developed Least Relevant *Value* (LR*V*) based cache replacement policy, where data with the lowest LRV are removed from the cache. The LRV is based on the following factors

Access probability*:* It is based on the previous access rate of a data item for a host. An item with lower access probability should be chosen for replacement. At a host, the access probability A*i* for data item d*i* is given as

$$A_i = a_i \div \sum_{k=1}^{N} a_k \qquad (2)$$

Where a_i is the mean access rate to data item d_i. a_i can be estimated by employing *sliding window* method of last K access times [29]. Keep a sliding window of K most recent access timestamps $(ts^1{}_i, ts^2{}_i, ...ts^k{}_i)$ for data item d_i in the cache. The access rate is updated using the following formula:

$$a_i = K \div (t_c - ts^k{}_i) \qquad (3)$$

where tc is the current time and ts k_i is the timestamp of oldest access to item d_i in the sliding window. K can be as small as two or three to achieve the best performance [29].

Distance: Distance (dt) is measured as the number of hops between the requesting client and the responding client (data source or cache). This policy incorporates the distance as an important parameter in selecting a victim for replacement. This is because caching data items which are further away, save bandwidth and reduce latency for subsequent requests.

Size (sz): A data item with larger data size should be chosen for replacement, because the cache can accommodate more data items and satisfy more access requests.

Based on the above factors, a *function* F_i for a data item d*i* is computed using the following expression:

$$F_i = (A_i \,.\, \mathrm{dt}_i) \div sz_i \qquad (4)$$

The idea is to remove the data item with least value of F_i

7. Experimental Results

7.1 Simulation Setup

This section deals with the experimental performance evaluation of our algorithms through simulations. In order to test our protocol, The NS2 simulation software [13] is used. NS2 is a general-purpose simulation tool that provides discrete event simulation of user defined networks.

In our simulation, the channel capacity of mobile hosts is set to the same value: 2 Mbps. The distributed coordination function (DCF) of IEEE 802.11 for wireless LANs as the MAC layer protocol is used. It has the functionality to notify the network layer about link breakage. In the simulation, mobile nodes move in a 600 meter x 600 meter rectangular region for 50 seconds simulation time. Initial locations and movements of the nodes are obtained using the random waypoint (RWP) model of NS2. All nodes have the same transmission range of 250 meters. We have the simulation setup as per Fig.1. We divided the area into 6 cells. Each cell consists of 6 clients.

The simulation parameters are summarized in Table 1.

Number Of Nodes	36
No. of Cells	6
Clients per Cell	6
Slot Duration	2 msec
Routing Protocol	AODV
Speed of mobile	5 m/s
Traffic Model	CBR

In all the experiments, we have used the following evaluation criteria and compared our CMA architecture with the traditional LRU scheme.

7.2 Simulation Results

A. The average downlink traffic under different query generate time

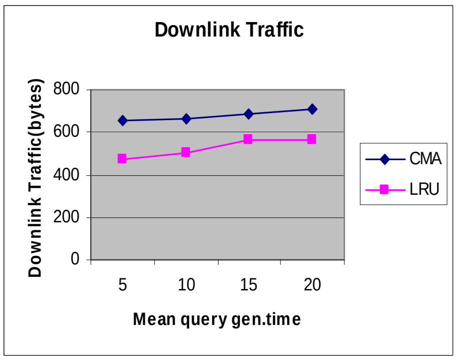

Fig. 2 Query generation timeVs Downlink Throughput

Fig. 2 shows the relationship between the downlink traffic and the query generate time *T*query. As can be seen, the average downlink traffic increases when *T*query increases. Note that if several clients request for the same data item during the same interval, the cached host broadcasts the data item once. As less broadcasting data is shared, the average downlink traffic increases. Not surprisingly, CMA outperforms LRU.

B. The average delay under different query generate time

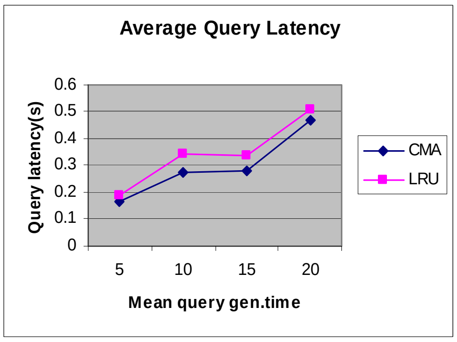

Fig. 3 Query generation timeVs Query Latency

Fig. 3 shows the average query latency as a function of Tquery. Each client generates queries according to the mean query generate time. The generated queries are served one by one. If the queried data is in the local cache, the client can serve the query locally; otherwise the client has to request the data from the active clients. As we can see from Fig.3, the delay of CMA is much less than that of LRU. This is due to the reason that CMA uses the cache space more effectively and the number of queries sent to the server can be reduced

C. End-to-End Delay under different traffic rates

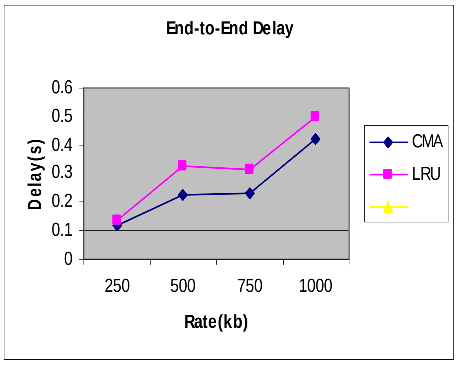

Fig. 4 Traffic Rate Vs Delay

Fig 4 shows the average end-to-end delay for different traffic rates. From the figure we can see that CMA has less delay , when compared with LRU.

D. The average Throughput under different cache sizes

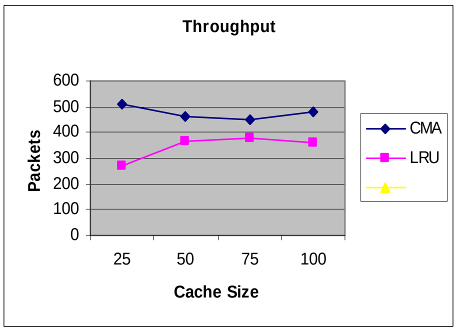

Fig. 5 Cache Size Vs Throughput

Fig. 5 shows the average throughput received for different cache sizes. From the figure we can see that CMA has more through put, when compared with LRU.

8. Conclusion

In this paper, we have developed a novel Cache Management Architecture (CMA) for mobile hosts, to reduce the caching overhead and provide optimal replacement policy. It aims to improve the network utilization and to reduce the search latency, bandwidth and energy consumption. The architecture comprises of the following algorithms: cache placement algorithm, cache discovery algorithm and, cache replacement algorithm. In the cache placement algorithm, data caches are placed into some active clients based on their weight vector. In the cache discovery algorithm, when a query is initiated at a client, it will send broadcast request packet to the set of active clients. The active client which is having the highest weight value, process this request and send the requested data to the client. In the cache replacement algorithm, we have developed a Least Relevant *Value* (LR*V*) based cache replacement policy, where data with the lowest LRV are removed from the cache. By simulation results, we have shown that our proposed architecture achieves lower latency and packet loss, reduced network bandwidth consumption, reduced data server workload.

References

[1] Liangzhong Yin Guohong Cao Ying Cai "A generalized target-driven cache replacement policy for mobile environments" Applications and the Internet, Symposium on 27-31 Jan. 2003

[2] D. Barbara, T. Imielinski, "Sleepers and workaholics: caching strategies for mobile environments", ACM SIGMOD, 1994, pp. 1–12.

[3] Guohong Cao "A scalable low-latency cache invalidation strategy for mobile environments", IEEE Transactions on Knowledge and Data Engineering, 2003.

[4] Guohong Cao "Adaptive Power Aware Cache Management for Mobile Computing Systems"Proceedings of ICCPR2007: International Conference on Comprehensive Product Realization 2007 June 18-20, 2007, Beijing, China

[5] Narottam Chand, R.C. Joshi and Manoj Misra, "Cooperative Caching in Mobile Ad Hoc Networks Based on Data Utility," International Journal of Mobile Information Systems, Vol. 3, No. 1, pp. 19-37, 2007

[6] Miguel Castro Atul Adya Barbara Liskov Andrew C. Myers, "HAC: Hybrid Adaptive Caching for Distributed Storage Systems". Proceedings of the 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, October 1997.

[7] Athena Vakali, "Proxy Cache Replacement Algorithms: A History-Based Approach". Journal Title: World Wide Web. Date: 2001. Volume: 4. Issue: 4. p. 277 - 298.

[8] Ismail Ari, Ahmed Amer, Robert B. Gramacy, Ethan L. Miller, Scott A. Brandt, Darrell D. E. Long, "ACME: Adaptive Caching Using Multiple Experts", Workshop on Distributed Data and Structures, DOCIS 2002.

[9] Bin Tang Gupta, H. Das, S.,"Benefit-based Data Caching in Ad Hoc Networks", Computer Science Department, Stony Brook University, Stony Brook, NY 11790, Network Protocols, 2006.

[10] Mohamed Zahran, "Cache Replacement Policy Revisited". In. Proceedings of the 6th Workshop on Duplicating, Deconstructing, and Debunking, San Diego, CA, USA, June 2007.

[11] Hocine Grine et al. "Adaptive Query Processing in Mobile Environment" ACM International Conference Proceeding Series, Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing table of contents 2005.

[12] M. Satyanarayanan "Fundamental Challenges in Mobile Computing" Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing table of contents Philadelphia, Pennsylvania, United States Pages: 1 - 7 Year of Publication: 1996 ISBN:0-89791-800-2

[13] http://www.isi.edu/nsnam

G. Anandharaj received the B.Sc degree in computer science, M.C.A degree in computer applications from Bharathiar University, Coimbatore, and the M.Phil degree in computer science from Bharathidasan University, Tiruchirappalli ,India, in 1998, 2001 and 2006, respectively. He is currently pursuing a Ph.D degree in computer applications at the Anna University of Coimbatore, India. He is currently working as Assistant Professor in the Department of Master of Computer Applications, Sengunthar Engineering College, Tiruchengode, Tamilnadu, India. His research interests include mobile computing and wireless network technology. He is life member of the ISTE.

Dr. R. ANITHA is currently working as Director in the Department of Master of Computer Applications, K. S. Rangasamy College of Technology, Tiruchengode – 637 215, Tamilnadu, India. She has obtained her MCA Degree from Bharathidasan University, Tiruchirappalli, and Ph.D from Periyar University, Salem. She has vast experience in teaching as well as research. She has presented papers at several International and National Conferences and has published research articles in leading Journals. She is an active researcher and is usually associated with reputed Academic Forums and Associations of research interest.