International Journal of Soft Computing 4 (3): 142-147, 2009
ISSN: 1816-9503

An Improved Architecture for Complete Cache Management in Mobile Computing Environments

[1]G. Anandharaj and [2]R. Anitha
[1]Sengunthar Engineering College, [2]K.S. Rangasamy College of Technology,
Tiruchengode, Tamil Nadu, India

Abstract: Caching plays a key role in mobile computing because of, its ability to alleviate the performance and availability limitations of weakly-connected and disconnected operations. Caching frequently accessed data objects at the local buffer of a mobile is an efficient way to reduce query delay, save bandwidth and improve system performance. Classical cache management strategies may not be suitable for mobile environments due to the disconnection and mobility of the mobile clients. Cache management in mobile environment, in general, includes cache placement, cache discovery, cache consistency and cache replacement techniques. In this study, we design a combined cache management architecture which includes all the above techniques. By simulation results, it has been shown that our proposed architecture achieves lower latency and packet loss, reduced network bandwidth consumption and reduced data server workload.

Key words: Architecture, complete cache, mobile computing, environments

INTRODUCTION

The mobile computing platform can be effectively described under the client/server paradigm. A data item is the basic unit for update and query. Mobile nodes only issue simple requests to read the most recent copy of a data item. There may be one or more processes running on a mobile node, referred to as clients. In order to serve a request sent from a client, the base station 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. Average data access latency is reduced as several data access requests can be satisfied from the local cache (Cao, 2003), thereby obviating the need for data transmission over the scarce wireless links. However, frequent disconnections and mobility of the clients make cache consistency a challenging problem Classical cache management strategies may not be suitable for mobile environments due to the disconnection and mobility of the mobile clients.

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 (Chand *et al*., 2007):

- The cache discovery algorithm that is used to efficiently discover, select and deliver the requested data item (s) from neighboring nodes
- Cache admission control-this is to decide on what data items can be cached to improve the performance of the caching system
- The cache consistency algorithm which ensures that updates are propagated to the copies elsewhere and no stale data items are present
- 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

There is no prior research, which combines all the above techniques and provides a complete solution for cache management.

This study is an extension of our previous research. We propose a Combined and Complete Cache Management (CCCM) Architecture for mobile hosts. The goal of our architecture is to reduce the caching overhead and provide optimal consistency and replacement. It aims to improve the network utilization, reduce the search latency, bandwidth and energy consumption. The architecture comprises of the following algorithms:

Corresponding Author: G. Anandharaj, Sengunthar Engineering College, Tiruchengode, Tamil Nadu, India

- Cache placement algorithm
- Cache discovery algorithm
- Cache consistency algorithm
- Cache replacement algorithm

MATERIALS AND METHODS

Barbará and Imieliński (1994) have proposed a cache solution, which is suitable for mobile environments. In their 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.

Castro *et al.* (1997) have proposed a new hybrid adaptive caching technique, which combines page and object caching to reduce the miss rate in client caches dramatically.

Vakali (2002) has presented a study of applying a history based approach to the Web-based proxy cache replacement process. Trace-driven simulation was employed to evaluate and comment on the performance of the proposed cache replacement techniques.

Ari *et al.* (2002) have proposed the use of machine learning algorithms to rate and select the current best policies or mixtures of policies via weight updates based on their recent success, allowing each adaptive cache node to tune itself based on the workload it observes. ACME is used to manage the replacement policies within distributed caches to further improve the hit rates over static caching techniques.

Cao (2002) has proposed a power-aware cache management. Based on a novel prefetch-access ratio concept, the proposed scheme can dynamically optimize performance or power based on the available resources and performance requirements. Simulation results have shown that their solution not only improves the cache hit ratio, the throughput and the bandwidth utilization, but also reduces the query delay and the power consumption.

Cao (2003) has addressed an UIR-based approach. In his 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.

Tang *et al.* (2008) have focused on developing efficient caching techniques in ad-hoc networks with memory limitations.

SYSTEM MODEL

Network model: In a mobile computing environment, 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 (Yin *et al.*, 1995).

Consider a mobile environment with n cells C_1, C_2,... 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,... S_m.

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.

COMBINED AND COMPLETE CACHE MANAGEMENT (CCCM) ARCHITECTURE

Cache placement algorithm: In this algorithm, data caches are placed into some clients known as active nodes, based on their weight vector which comprises the following parameters:

- Available bandwidth
- CPU speed
- Access latency
- Cache hit ratio

Active nodes, which are neighbors 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 local neighbors before forwarding the request to the next client that lies on a path towards server.

As per our network model, in our system, there are n cells. In each cell there are m clients.

For each client of the cell C_j, j = 1,2.....n, let
BW_i = Available bandwidth
SP_i = CPU speed
AL_i = Access Latency
Cri = Cache Hit Ratio
where, i = 1,2.....m

1. The weight of the client can be calculated as
$$W_i = (BW_i + SP_i + CR_i) \div AL_i \quad (1)$$
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, $(0 < = k < m)$, which satisfies the following condition $W_k > \beta$, where, β the minimum threshold value for the weight is.
4. Each database server DS_j caches the databases into the active nodes set S_k.

Cache discovery: 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 broadcast request packet to the set of active clients. 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.

The mobile clients that belong to the active node set then form 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 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.

Case 3: Global hit: Data item is retrieved from the database server.

Cache discovery algorithm: A cache discovery algorithm is required to determine, where the requested item is cached when the requester does not know the destination.

Once a set of active clients are 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.....

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.

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.

When the query client receives ack packet 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 packet for the same item received from other clients are discarded.

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

Cache consistency algorithm: Cache consistency is required to ensure the validity of data. It involves consistency between a data source and the cache copies stored by the cache nodes. Cache consistency maintenance algorithms can be divided into 2 main categories: stateful (Kahol *et al*., 2001) and stateless (Cao, 2003), based on whether the cache status is maintained on the data source node. Existing consistency maintenance algorithms for MANET (Huang *et al*., 2006, 2007) are mainly stateless.

In this study, we propose a stateful cache consistency maintenance algorithm based on an Adaptive Push approach. Each node maintains a timestamp value to indicate the expiry of the cached items. Here the data source node decides the set of active nodes, to which updations are to be made, based on their cache status. For the selected active caching nodes, data update information is broadcasted based on the adaptive push approach.

Using adaptive push, each source node informs the caching nodes of data updates. The design is based on the following objectives:

- If there are any pending queries to be served, update the necessary cache copies
- Send the update information, before the timestamp expires so that there should not be any other updates.
    - If the server receives a data update on a data set DS_j, then it decides the set $\{Su_i\}$ of active caching nodes to update, based on the following:
- If the no of pending queries Q_{pj} on DS_j, is more than the minimum query threshold Q_m

- The timestamp T_s at the node S_i is about to expire
  - After the data source node has selected the set of updating cache nodes, it needs a specific mechanism to transmit the data updates to the selected caching nodes
  - The server sends the UPDATE message to the nearest caching node in the set $\{Su_i\}$
  - Upon receiving the UPDATE message, the caching node acknowledges the UPDATE message by sending an ACK message and forwards the UPDATE message to the next nearest caching node in the set $\{Su_i\}$
  - This process is repeated until all the nodes in the set $\{Su_i\}$ receive the UPDATE message and send ACK
  - From the gathered ACK messages, the server knows the ids and timestamps of all the nodes. Then, it propagates the updated data along with the modified new timestamp value nTs, to all the receiving nodes of $\{Su_i\}$
  - If the sender of the UPDATE message did not receive an ACK message within a time T_a, it removes the corresponding node id from the set

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

Cache replacement algorithm: In the cache replacement algorithm, we propose to develop a Least Relevant Value (LRV) 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.

We have developed Least Relevant Value (LRV) 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:

$$A_i = \frac{a_i}{\sum_{k=1}^{N} a_k} \tag{2}$$

where, a_i is he mean access rate to data item d_i. a_i can be estimated by employing sliding window method of last K access times. 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 Eq. 3:

$$a_i = \frac{K}{(t_c - t^k_{s\,i})} \tag{3}$$

where:

t_c = The current time

$t^k_{s\,i}$ = The timestamp of oldest access to item d_i in the sliding window

K = Small as 2 or 3 to achieve the best performance

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, saves bandwidth and reduces 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 di with distance dt_i is computed using the following expression:

$$F_i = (A_i \,.dt_i)/\, sz_i \tag{4}$$

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

RESULTS AND DISCUSSION

Simulation setup: This study deals with the experimental performance evaluation of our algorithms through simulations. In order to test our protocol, the NS_2 simulation software is used. NS_2 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 as 2 Mbps. The MAC protocol used is 802.11 for WLAN. It has the functionality to notify the network layer about link breakage. In the simulation, mobile nodes move in a 600×600 m rectangular region for 50 sec simulation time. Initial locations and movements of the nodes are obtained using the Random Waypoint (RWP) model of NS_2. All nodes have the same transmission range of 250 m. We divided the area into 6 cells. Each cell consists of 6 clients. The simulation parameters are summarized in Table 1.

In all the experiments, we used the following evaluation criteria. We compare our CCCM architecture with the traditional LRU scheme (Cao, 2003).

Simulation results: A. The average downlink traffic under different query generate time Fig. 1 shows the relationship between the downlink traffic and the query generation time Tquery. As can be shown, the average downlink traffic increases when Tquery 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, CCCM outperforms LRU.

The average delay under different query generate time: Figure 2 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 shown in Fig. 2, the delay of CCCM is much less than that of LRU. This is due to the reason that CCCM use the cache space more effectively and the number of queries sent to the server can be reduced.

End-to-end delay under different traffic rates: Figure 3 shows the average end-to-end delay for different traffic rates. Figure 3, shows that CCCM has less delay, when compared with LRU.

Table 1: Simulation parameters

No. nodes	36
No. cells	6
Clients/cell	6
Slot duration	2 ms
Routing protocol	AODV
Speed of mobile	5 ms
Traffic model	CBR

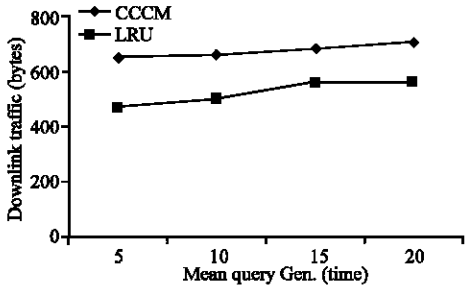

Fig. 1: Query generation time vs. downlink throughput

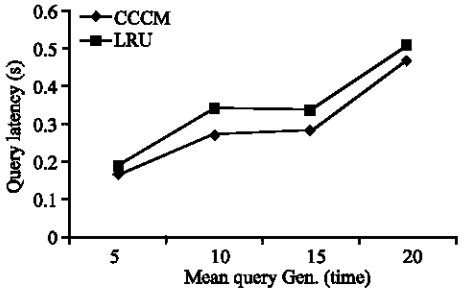

Fig. 2: Query generation time vs. query latency

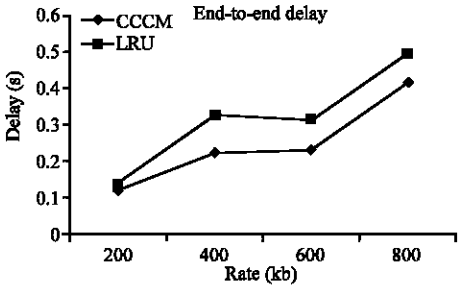

Fig. 3: Traffic rate vs. delay

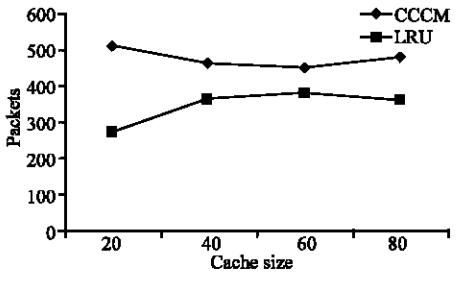

Fig. 4: Cache size vs. throughput

The average throughput under different cache sizes: Figure 4 shows the average throughput received for different cache sizes. Figure 4, shows that CCCM has more through put, when compared with LRU.

CONCLUSION

Cache management in mobile environment, in general, includes cache placement, cache discovery, cache consistency and cache replacement techniques. In this study, we have designed Combined and Complete Cache Management (CCCM) architecture for mobile hosts, which include all the above techniques. The architecture have improved the network utilization, reduced the search latency, bandwidth and energy consumption. By simulation results, we have shown that our proposed architecture achieves lower latency and packet loss, reduced network bandwidth consumption, reduced data server workload. We have not considered the failure of database server in our architecture. The effect of node mobility is also not considered. So, in our future research, we will extend this architecture with efficient recovery schemes and node mobility.

REFERENCES

Ari, I., A. Amer, R.B. Gramacy, E.L. Miller, S.A. Brandt and D.D.E. Long, 2002. ACME: Adaptive Caching Using Multiple Experts. In: Proc. Informatics, Paris, France, 14: 143-158. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.6895.

Barbará, D. and T. Imieliński, 1994. Sleepers and workaholics: Caching strategies in mobile environments. ACM SIGMOD Rec., 23 (2): 1-12. DOI: http://doi.acm.org/10.1145/191843.191844. http://portal.acm. org/citation.cfm?id=191844.

Cao, G., 2002. Adaptive power aware cache management for mobile computing systems. Elevnth Int. www Conference, Hawaii, USA, 7-11 May. ISBN: 1-880672-20-0. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.7718.

Cao, G., 2003. A scalable low-latency cache invalidation strategy for mobile environments. IEEE. Trans. Knowledge Data Eng., 15 (5): 1251-1265. DOI: 10.1109/TKDE.2003.1232276. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp? arnumber=1232276.

Castro, M., A. Adya, B. Liskov and A.C. Myers, 1997. HAC: Hybrid Adaptive Caching for Distributed Storage Systems. ACM SIGOPS Operat. Syst. Rev., 31 (5): 102-115. DOI: http://doi.acm.org/10.1145/269005. 266666. http://portal.acm.org/citation.cfm?doid=269005.266666.

Chand, N., R.C. Joshi and M. Misra, 2007. Cooperative caching in mobile ad hoc networks based on data utility. Mob. Inform. Syst., 3 (1): 19-37. http://portal.acm.org/citation.cfm?id=1376600.

Huang, Y.J. Cao and B. Jin, 2006. A predictive approach to achieving consistency in cooperative caching in MANET. ACM International Conference Proceeding Series, ACM New York, USA, 152 (50). DOI: http://doi.acm.org/10.1145/1146847.1146898. http://portal.acm.org/citation.cfm?id=1146898.

Huang, Y.C., J. Wang, Z. Jin, B. Feng and Yulin, 2007. Achieving flexible cache consistency for pervasive internet access. 5th Annual IEEE Int. Conf. Pervasive Comput. Communic., March, 19-23, IEEE Computer Society Washington, DC, USA, pp: 239-250. DOI: 10.1109/PERCOM.2007.6. http://portal.acm.org/citation.cfm?id=1263542.1263718&coll=GUIDE&dl=GUIDE.

Kahol, A.S. Khurana, S.K.S. Gupta and P.K. Srimani, 2001. A strategy to manage cache consistency in a disconnected distributed environment. IEEE Trans. Parallel Distributed Syst., 12 (7): 686-700. DOI:10.1109/71.940744.http://portal.acm.org/citation cfm?id=507295.

Tang, B., H. Gupta and S.R. Das, 2008. Benefit-Based Data Caching in Ad Hoc Networks. IEEE Trans. Mob. Comput., 7 (3): 289-304. DOI: 10.1109/TMC.2007.70770. http://portal.acm.org/citation.cfm?id=1340222.

Vakali, A., 2002. Proxy cache replacement Algorithms: A history-based approach. www, 4 (4): 277-297. DOI:10.1023/A:1015133818512.http://portal.acm.org/citation.cfm?id=598690.598766&dl=GUIDE&dl=GUIDE.

Yin, L., G. Cao and Y. Cai, 1995. A generalized target-driven cache replacement policy for mobile environments. J. Parall. Distrib. Comput., 65 (5): 583-594. DOI:10.1016/j.jpdc.2004.12.002.http://portal.acm.org/citation.cfm?id=1073768.