# Peer-to-Peer Cooperative Caching in a Hybrid Data Delivery Environment*

Chi-Yin Chow Hong Va Leong Alvin Chan

*Department of Computing, Hong Kong Polytechnic University, Hong Kong*
E-mail: {*cscychow, cshleong, cstschan*}*@comp.polyu.edu.hk*

## Abstract

*In a conventional mobile environment, mobile clients retrieve data items from database servers via mobile support stations, by downloading the items over scalable broadcast channels (push-based), requesting them over shared point-to-point channels (pull-based), or making use of both types of channels to retrieve them (hybrid). Caching is a key technique for improving data retrieval performance of mobile clients, regardless of the data delivery mechanism. The emergence of robust peer-to-peer technologies now brings to reality what we call "cooperative caching" in which mobile clients can access data items from the cache in their neighboring peers, thereby adding a new dimension for data caching. This paper studies the system performance of a cooperative caching scheme, called COCA, in a hybrid data delivery environment, and proposes a cooperative cache replacement scheme, called CORE, for mobile systems. The performance of COCA and CORE schemes is evaluated through a number of simulated experiments. The experiment results show that COCA effectively improves the system performance in push-based and hybrid data delivery environments, especially in a heavily-loaded environment. The results also indicate that CORE can further improve on the access latency and reduce the number of expensive server requests, which consume scarce pull-based bandwidth.*

## 1. Introduction

A mobile system is composed of a wireless network, connecting mobile hosts (MHs) to mobile support stations (MSSs), in which the MHs are clients equipped with portable devices while the MSSs play the role of servers in the mobile system. In a conventional mobile environment, mobile clients retrieve data items from database servers via the MSSs, either by downloading them over scalable broadcast channels (push-based data delivery), or by requesting them over shared point-to-point channels (pull-based data delivery), or through the utilization of both types of channels to retrieve them (hybrid data delivery). With the recent widespread deployment of new peer-to-peer wireless communication technologies, such as IEEE 802.11 and Bluetooth, and coupled with the fact that the computation power and storage capacity of mobile devices have been improving at a fast pace, there is a new information sharing alternative, known as **CO**operative **CA**ching (COCA). In COCA, the MHs can communicate among themselves to share information rather than having to rely on the communication link to the MSSs. COCA can bring about several distinctive benefits to a mobile system: improving the access latency, sharing the server workload and alleviating the point-to-point channel congestion. However, COCA could increase the communication overheads among MHs.

Bandwidth and data allocation problems in a hybrid data delivery model have been extensively studied recently [2, 4, 6, 10, 13]. To our best knowledge, there is no study on cooperative caching in hybrid data delivery environments. It is worthy to evaluate the performance of COCA in a hybrid data delivery environment, and based on the COCA model, a **CO**operative cache **RE**placement scheme, namely CORE, is designed to further improve the system performance.

Recently, cooperative caching in mobile environments has been drawing increasing attention [5, 9, 12]. However, these research efforts do not consider a hybrid data delivery environment. In this paper, cooperative caching and cooperative cache replacement schemes are adapted for a hybrid data delivery environment, and their performance is evaluated through a number of simulated experiments.

The rest of this paper is organized as follows. Section 2 presents the background of mobile data delivery models. Section 3 delineates the COCA model. Section 4 describes the CORE model. Section 5 defines the simulation model of COCA and CORE. Section 6 studies the performance of COCA and CORE through a number of simulated experiments. Finally, Section 7 offers brief concluding remarks.

## 2. Background

The major challenge to designing an efficient data delivery model for a mobile system is associated with the inherent limitations of mobile environments: limited battery power, restricted bandwidth of wireless communication, user mobility and asymmetric communication. The conventional means of data delivery in mobile systems can

* Research supported in part by the Research Grant Council and the Hong Kong Polytechnic University under grant numbers PolyU 5084/01E and H-ZJ86.

be classified into three models: pull-based, push-based and hybrid data delivery.

A pull-based data delivery model, which is also known as a point-to-point model, is similar to the conventional client/server model in wired networks. An MH sends a request to the MSS via an uplink channel, when it encounters a cache miss. The MSS processes the request and sends the required data item back to the MH via a downlink channel. Thus, the MSS would potentially be a bottleneck in the system, as it serves an enormous number of MHs.

A push-based data delivery model is also known as a data broadcast model, in which the MSS periodically broadcasts data items to the MHs via a broadcast channel. Since there is no uplink channel, the MHs only listen to the broadcast channel and "catch" the required data items when they appear in the broadcast channel. This model is scalable to an immense number of MHs. However, since the data items are broadcast sequentially, the access latency gets longer with increasing number of data items being broadcast.

To overcome the downside of pull- and push-based models, a hybrid data delivery model, which encompasses both models, is proposed [2, 4, 6, 11, 13]. In the hybrid data delivery model, part of the bandwidth is reserved for broadcasting data items to the MHs, and the rest is dedicated to point-to-point communication.

# 3. COCA Model

In a conventional hybrid data delivery model, if an MH cannot obtain the required data item in its local cache (called a *local cache miss*), it tunes to the broadcast channel and catches the required data item when it appears. However, if the required data item is not found in the broadcast channel or the latency of access to it is beyond a predefined latency threshold, the MH sends a request to the MSS. The MSS then delivers the item to the MH. In COCA, when an MH suffers from a local cache miss, it first looks up the required data item from its neighboring peers' cache, while it continues to listen to the broadcast channel for the required data item. When the required data item is turned in by any peer before it appears in the broadcast channel, a *global cache hit* is recorded. However, if the MH fails to obtain the data item from its neighboring peers and the broadcast channel, it sends a request to the MSS via the point-to-point channel. The MSS then handles this request as in the conventional model.

## 3.1. Bandwidth Allocation

In a pull-based data delivery model, all the channels are dedicated to point-to-point communication. On the other hand, in a push-based data delivery model, all of them are reserved for data broadcast. A hybrid data delivery model consists of both types of channels. In COCA, a static channel allocation scheme is used, and the channel allocation method is based on a ratio, namely *PushBW*, which indicates the number of channels that are allocated for data broadcast. For example, if there is a total number of 20 channels and *PushBW*=30%, then six channels are reserved for data broadcast and the remaining channels are devoted to point-to-point communication.

## 3.2. Data Allocation

Data allocation is also known as broadcast scheduling. The simplest broadcast scheduling is the *flat disk* scheme [1], in which each data item is broadcast to the MH only once in a broadcast cycle. Therefore, the expected access latency is half of the length of the broadcast cycle. There is another broadcast scheduling scheme, called the *broadcast disk* scheme [1], that is designed for a mobile environment with a skewed access pattern. In the broadcast disk scheme, there are several disks with different spinning speeds and the hottest data items are allocated in the fastest disk. The coldest data items are allocated in the slowest disk. Thus, the expected access latency to the hot data items is shortened. In [1], it shows that the broadcast disk scheme outperforms the flat disk scheme in an environment where the client access pattern is skewed.

In a hybrid data delivery model, an MH can access the hot data items (up to a total size of *DiskSize*) via the broadcast channel. However, to access the remaining cold data items, it still has to send a request to the MSS and the MSS sends the requested data items back to the MH via the point-to-point channel.

The selection of data items to be allocated in the broadcast channel is based on their access probabilities, $p$, which can be estimated by their observed access frequency, $f$. The MSS records the relative access frequency of each data item to estimate its access probability. For data item $i$, $f_i$ is initially set to zero and the last access time, $t_i$, is set to the time of initialization, and then $f_i$ is updated as it is requested by an MH via the uplink channel based on an equation:

$$f_i^{new} = \omega \times \frac{1}{\mathsf{now} - t_i} + (1 - \omega) \times f_i^{old}, \qquad (1)$$

where $\omega$ ($0 \leq \omega \leq 1$) is a parameter to weight the importance of the most recent access. Then, $t_i$ is set to the current time (now). The data allocation is performed at the beginning of each analysis period (*AnalysisPeriod*), based on the weighted access frequency of the data items, which is approximately proportional to their access probabilities.

Since the access frequency collected by the MSS belongs to "past" information, it can only be used to predict the trend of the access pattern. The hottest data items are likely to be cached by most MHs, so they need not to be broadcast frequently. Thus, a parameter *Offset* [1] is used to determine the number of the hottest data items that should be shifted from the disk with the highest spinning speed to the one with the lowest spinning speed.

### 3.3. Indexing the Broadcast Channel

In a hybrid data delivery model, the MSS periodically broadcasts the index of the data items being broadcast in a broadcast cycle to the MHs, so that the MHs are able to determine whether the required data items can be obtained via the broadcast channel or the point-to-point channel based on the index. The index contains the identifier of each data item being broadcast and its broadcast latency, which is defined by subtracting the current broadcast slot from the slot containing the data item. The index is broadcast to the MH every *IndexInterval*. When an MH cannot find the identifier of the required data item in the index or the latency of accessing it is longer than a latency threshold, *Threshold*, the MH switches to retrieve the data item from the MSS via the point-to-point channel. The threshold technique is also adopted in [2, 6].

### 3.4. Cooperative Caching

We assume that, in COCA, each mobile device is equipped with two wireless network interface cards. One operated in an infrastructure mode is for point-to-point and broadcast communication with the MSS, whereas another one operated in an ad hoc mode is for peer-to-peer communication. When an MH encounters a local cache miss, it listens to the broadcast channel, and sends a request to its neighboring peers via the peer-to-peer channel. If there is no peer turning in the required data item, and the data item is not allocated in the broadcast channel or the latency of access to the data item is greater than *Threshold*, the MH sends the request to the MSS and obtains the required data item via the point-to-point channel.

## 4. CORE Model

In this section, we describe the CORE mechanism in COCA. CORE allows the MH to collaborate with its neighboring peers to replace the *least valuable* data item with respect to the MH itself and its neighboring peers. The proposed CORE exhibits three properties. First, the most valuable data items are always retained in the local cache. Second, in a local cache, a data item which has not been accessed for a long time, is replaced eventually. Third, in a global cache, a data item which "spawns" replica is first replaced to increase the effective cache size.

In CORE, the LRU (Least Recently Used) cache replacement policy is used. To retain valuable data items in the local cache, only a number of *ReplaceCandidate* least valuable data items are selected to participate in CORE. The CORE information is stored in a bit-vector. This technique is also known as bit-sequence naming that was used for cache invalidation in mobile environments [8]. The length of the bit-vector is the same as the number of data items stored in the database. A unique identifier, which is expressed as an integer from zero to the length of the bit-vector minus one, is assigned to each data item. Thus, each data item can be represented by a bit at the position of its identifier in the bit-vector. There are three types of bit-vectors: initial, peer and replication.

**Initial Bit-Vector.** When an MH encounters a local cache miss and its local cache is fully occupied, the MH generates a bit-vector for the *ReplaceCandidate* least recently used data items by setting the corresponding bits in the bit-vector. Then, the MH sends a request with the initial bit-vector to its neighboring peers.

**Peer Bit-Vector.** An MH which receives the request with the initial bit-vector from a neighboring peer generates a temporary bit-vector by setting the corresponding bit of each data item cached in the local cache. Then, the MH produces a peer bit-vector by superimposing the temporary bit-vector and the initial bit-vector. The peer bit-vector is called a *non-zero bit-vector*, if at least one bit is set in the bit-vector. If the MH caches the required data item, it sends the data item back to the requesting peer with the non-zero peer bit-vector; otherwise, it only sends the non-zero peer bit-vector back to the requesting peer. The MH does not reply to the requesting MH, if it does not cache the required data item and the peer bit-vector is not non-zero.

**Replication Bit-Vector.** After the MH collects the peer bit-vector from its neighboring peers, it produces a replication bit-vector by superimposing all the peer bit-vectors. A data item is a replica when the corresponding bit is set in the replication bit-vector. If the replication bit-vector is non-zero, the MH drops the replica of the least recently accessed data item; otherwise, the MH drops the least recently used data item from the local cache.

There could be a starvation problem with the above arrangement: a data item without any replica is always retained in the local cache, even though it will not be accessed again. If most data items cached in the local cache of an MH belong to such type of data item, the MH client cache will be populated with useless data items. As a result, the local cache hit ratio will be degraded. To solve this problem, a time-to-live (TTL) counter is associated with each data item in the local cache. The TTL counter value is initially set to *ReplaceDelay*. When the least recently used data item is not replaced because it does not have any replica, its counter is decremented. The MH skips the CORE mechanism, if the counter value of the least valuable data item is equal to zero, and the MH simply drops this data item from its local cache. The counter is reset to *ReplaceDelay*, when the data item is accessed by the MH again.

## 5. Simulation Model

In this section, we present the simulation model, which is depicted in Figure 1, for evaluating the performance of COCA and CORE. The simulation model is implemented in C++ using CSIM. It consists of a client model and a server model. The simulated mobile environment is composed of an MSS and 100 MHs. In the simulation, the database in the MSS contains 3000 equal-sized (1 KB) data items.

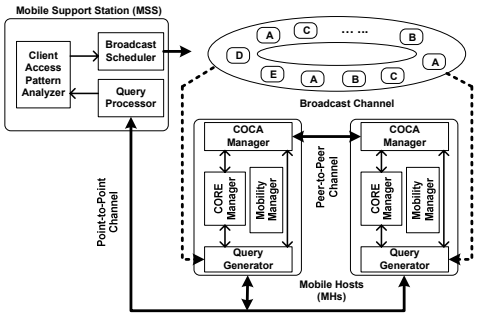



**Figure 1. The simulation model.**

In the simulation model, there are *NumChannel* communication channels between the MSS and the MHs with a total bandwidth of $BW_{Wireless}$, and peer-to-peer channels with an individual bandwidth of $BW_{P2P}$ for peer-to-peer communication. An MH retrieves the data items from the MSS either via the broadcast channels or the point-to-point channels, and communicates with its neighboring peers via peer-to-peer channels. The MHs' devices adopt the same type of wireless network interface cards, with a transmission range of 50 m.

### 5.1. Client Model

The client model consists of four components: query generator, mobility manager, COCA manager and CORE manager.

**Query Generator.** The query generator is a separate autonomous process to generate queries. The time interval between two consecutive queries generated by the query generator follows an exponential distribution with a mean of one second. It generates accesses to the data items following a Zipf distribution with a skewness parameter $\theta$. If $\theta$ is set to zero, MHs access the data items uniformly. By increasing $\theta$, we are able to model a more skewed access to the data items. The MHs access data items within an access range (*AccessRange*) in the database, and each MH has its own randomly selected hot spot within *AccessRange*.

**Mobility Manager.** The mobility manager is another separate autonomous process to control the movement pattern of each MH. The MHs move according to a "random waypoint" model [3]. The MHs can move freely in a 500 m $\times$ 500 m (*Area*) space, which constitutes the service area of the mobile system. At the beginning, the MHs are randomly distributed in *Area*. Each MH randomly chooses its own destination in *Area* with a randomly determined speed $s$ from a uniform distribution $U(v_{min}, v_{max})$. It then travels with the constant speed $s$. When it reaches the destination, it comes to a standstill for one second to determine its next destination. It then moves towards its new destination with another random speed $s' \sim U(v_{min}, v_{max})$. All MHs repeat this movement behavior during the simulation.

**COCA Manager.** The COCA manager is dedicated to handle peer-to-peer communication among the MHs. When an MH encounters a local cache miss, the COCA manager gets the initial bit-vector from the CORE manager, combines it into the request, and sends the combined request to its neighboring peers. It then collects the replies from the peers. If some peers turn in the required data item, it forwards the data item to the query generator. If the reply comes along with a peer bit-vector, the bit-vector is extracted from the reply, and then transferred to the CORE manager.

**CORE Manager.** The major functions provided by the CORE manager are generating the initial and peer bit-vectors, producing the replication bit-vector by superimposing all the peer bit-vectors forwarded from the COCA manager, maintaining the TTL counter for each data item cached in the local cache and selecting a data item to be replaced.

In addition, to model a realistic client behavior in a mobile environment [7], an MH may switch its device off for a period of time (*SleepTime*) to conserve power, with a uniform probability ($P_{Sleep}$). When the MH falls into the sleep mode, it is disconnected from the network, i.e., no peer can communicate with a sleeping MH.

### 5.2. Server Model

There is a single MSS in the system. The server is composed of three major components: broadcast scheduler, query processor and client access pattern analyzer.

**Broadcast Scheduler.** The broadcast scheduler arranges the appropriate data items to be broadcast based on the client access pattern that is provided by the client access pattern analyzer. It periodically broadcasts the data items and the index information to all MHs residing in the service area. The *DiskSize* hottest data items are selected by the broadcast scheduler. For the broadcast disk scheme, three broadcast disks are used, and the relative frequencies for them are 4, 2, and 1.

**Query Processor.** The query processor receives and processes the request sent by all the MHs with a first-come-first-serve policy. An infinite queue is used to buffer the request from the MHs when the processor is busy. For each request, the query processor forwards the request information, i.e., the identifier of the data items, to the client access pattern analyzer.

**Client Access Pattern Analyzer.** The client access pattern analyzer collects the data access information from the query processor and estimates the access frequency of each data item by using Equation 1. It sends a sorted list of the *DiskSize* hottest data items in a descending order of the access frequency to the broadcast scheduler every *AnalysisPeriod*.

## 6. Simulation Experiments

In our simulation, we compare the performance of COCA with a conventional caching approach that does not involve any cooperation among MHs. This serves as a base case for comparison. LRU cache replacement policy is applied on the base case (non-COCA or NC) and standard COCA (CC). Likewise, the performance of COCA

with CORE (CR) is evaluated by comparing it with standard COCA. Also, we consider two broadcast scheduling schemes: flat disk and broadcast disk, to provide extensive evaluation on the performance of COCA and CORE. All simulation results are recorded after the system reaches a stable state. Each MH generates 20,000 queries, where 2,000 are treated as warm-up queries in order to avoid a transient effect. We conduct the experiments by varying two parameters: number of MHs in the system and cache size. The performance metrics include the access latency, the server request ratio and the average number of messages sent by an MH.

### 6.1. Effects of Number of MHs

Our first experiment studies the effect of system workload on the system performance by varying the number of MHs in the system from 100 to 500.

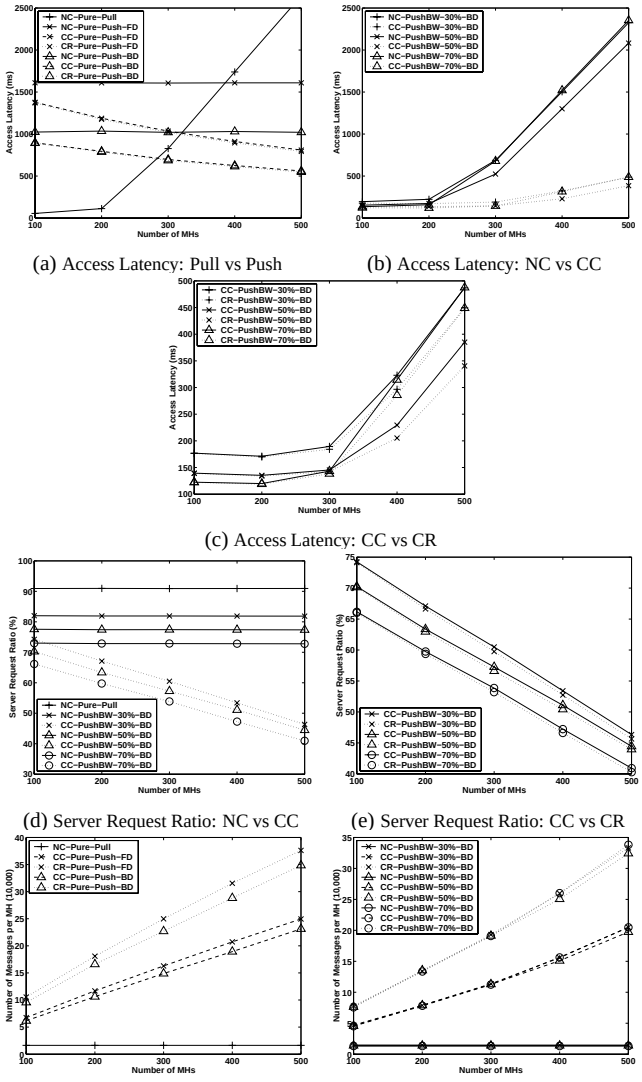
Figure 2(a), 2(b) and 2(c) show that COCA schemes outperform all non-COCA schemes, except the non-COCA pure-pull scheme. However, COCA schemes can perform better than the pure-pull scheme in heavily-loaded environments, in which the number of MHs is more than 330. The results show that COCA is a scalable scheme that improves the access latency with increasing number of MHs. This is because the system workload is shared by all MHs in the system. When an MH has more neighboring peers, the chance of the peers turning in the required data item increases. In heavily-loaded environments, CORE further shortens the access latency, since it increases the data availability in the system that enhances the global cache hit ratio.

In Figure 2(d), it shows that COCA schemes significantly reduce the server request ratio in both lightly- and heavily-loaded environments. This clearly demonstrates the effectiveness of the COCA schemes in alleviating the congestion of the point-to-point channel. As more requests of an MH can be satisfied by its neighboring peers, the probability of enlisting the MSS for help can be reduced. Likewise, CORE schemes further reduce the server request ratio, as depicted in Figure 2(e).

A drawback of COCA and CORE schemes is that each MH incurs additional overheads for communicating with its neighboring peers. The communication overheads are expressed as the average number of messages sent by each MH. The number of messages of non-COCA schemes is not affected by the increasing number of MHs, as depicted in Figure 2(f) and 2(g), because there is no communication among the peers in non-COCA schemes. However, COCA and CORE schemes substantially increase the number of messages with increasing number of MHs. When there are more MHs in the system, an MH has more neighboring peers. Thus, an MH has to send more requests and more replies to its neighboring peers as the chance of caching the required data item requested by the peers increases. Figure 2(g) shows that CORE schemes incur more messages than COCA schemes. In CORE, an MH not only replies to a requesting peer, when it caches the required data item, but also it has to reply to the peer, if the peer bit-vector is non-zero. Therefore, the MHs adopting the CORE scheme have to send more messages to their neighboring peers than those based on the standard COCA scheme.

**Table 1. Default parameter settings.**

| Parameters | Default Values | Parameters | Default Values |
|---|---|---|---|
| *NumChannel* | 20 | *Offset* | *ClientCacheSize* |
| *PushBW* | 0, 30, 50, 70, 100 percent | *Threshold* | 300 items |
| $BW_{Wireless}$ | Downlink 10 mbps; Uplink 100 kbps | *ClientCacheSize* | 50 items |
| $BW_{P2P}$ | 1 mbps | *Speed* $(v_{min} \sim v_{max})$ | $1 \sim 5\,m/s$ |
| *AccessRange* | 1000 items | $\omega$ | 0.25 |
| *DiskSize* | 600 items | $\theta$ | 0.5 |
| *AnalysisPeriod* | 10 Broadcast cycles | $P_{Sleep}$ | 0.1 |
| *ReplaceDelay* | 10 | *SleepTime* | 10.0 s |
| *ReplaceCandidate* | 20 % of *ClientCacheSize* | *IndexInterval* | 100 items |




(a) Access Latency: Pull vs Push (b) Access Latency: NC vs CC

(c) Access Latency: CC vs CR

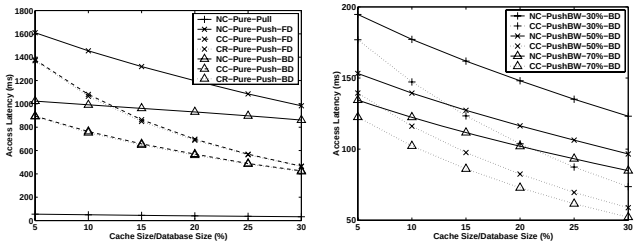
(d) Server Request Ratio: NC vs CC (e) Server Request Ratio: CC vs CR

(f) No. of Messages per MH: Pull vs Push (g) No. of Messages per MH: Hybrid

**Figure 2. Performance studies on various number of MHs.**

### 6.2. Effects of Cache Size

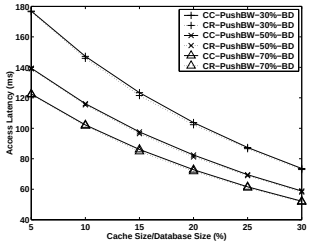
In this series of simulated experiment, we examine the influence of system performance on various ratio of cache



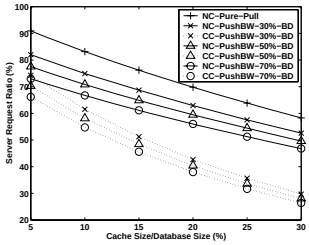

(a) Access Latency: Pull vs Push



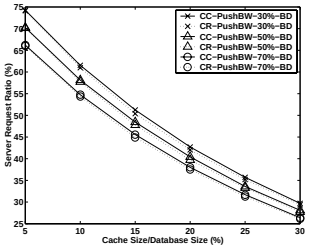

(b) Access Latency: NC vs CC




(c) Access Latency: CC vs CR




(d) Server Request Ratio: NC vs CC




(e) Server Request Ratio: CC vs CR

**Figure 3. Performance studies on various cache sizes.**

size to database size, from 5 percent to 30 percent.

Figure 3(a), 3(b) and 3(c) show that all schemes exhibit better access latency with increasing cache size. This is because more required data items can be found in the local cache as the cache gets larger. COCA schemes outperform non-COCA schemes except non-COCA pure-pull scheme because this experiment is conducted in a lightly-loaded environment. In terms of the access latency, CORE schemes consistently perform better than COCA schemes. The reduction in the access latency of COCA and CORE schemes is larger than non-COCA schemes. For an MH adopting COCA and CORE schemes, other than achieving a higher local cache hit ratio as the cache size gets larger, it also enjoys a higher global cache hit ratio because the chance of some neighboring peers turning in the required data items increases with the larger cache size.

In Figure 3(d) and 3(e), it is obvious that the server request ratio reduces with the increasing cache size, as more requests can be satisfied by accessing the local cache. It is similar to the experiment conducted on the access latency, the reduction in the point-to-point server request ratios of COCA and CORE schemes is also higher than non-COCA schemes.

## 7. Conclusion

In this paper, we have evaluated the system performance of a cooperative caching scheme, called COCA, in a hybrid data delivery environment. We have also proposed a cooperative cache replacement scheme, called CORE, for mobile systems. The performance of COCA and CORE is evaluated through a number of simulated experiments, which show that they improve the access latency as well as the server request ratio. The CORE scheme further improves the system performance. We also measured the communication overheads, which is expressed as the average number of messages sent by an MH, incurred in COCA and CORE schemes. COCA performs very well in both lightly- and heavily-loaded environments. However, considering the increased communication overheads incurred in CORE, CORE is more appropriate for adoption in a heavily-loaded environment.

## References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proc. of the ACM SIGMOD*, pages 199–210, May 1995.

[2] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proc. of the ACM SIGMOD*, pages 183–194, May 1997.

[3] J. Broch, D. A. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of the 4th MOBICOM*, pages 85–97, October 1998.

[4] Y. Guo, M. C. Pinotti, and S. K. Das. A new hybrid broadcast scheduling algorithm for asymmetric communication systems. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(3):39–54, July 2001.

[5] T. Hara. Cooperative caching by mobile clients in push-based information systems. In *Proc. of the 11th CIKM*, pages 186–193, November 2002.

[6] Q. Hu, D. L. Lee, and W.-C. Lee. Performance evaluation of a wireless hierarchical data dissemination system. In *Proc. of the 5th MOBICOM*, pages 163–173, August 1999.

[7] T. Imielinski and B. R. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, 37(10):18–28, October 1994.

[8] J. Jing, A. Elmagarmid, A. S. Helal, and R. Alonso. Bit-sequences: an adaptive cache invalidation method in mobile client/server environments. *Mobile Networks and Applications*, 2(2):115–127, October 1997.

[9] W. H. O. Lau, M. Kumar, and S. Venkatesh. A cooperative cache architecture in support of caching multimedia objects in MANETs. In *Proc. of the 5th ACM WoWMoM*, pages 56–63, September 2002.

[10] H. V. Leong and A. Si. Database caching over the air-storage. *The Computer Journal*, 40(7):401–415, 1997.

[11] E. Mok, H. V. Leong, and A. Si. Transaction processing in an asymmetric mobile environment. In *Proc. of the 1st MDA*, pages 71–81, December 1999.

[12] F. Sailhan and V. Issarny. Cooperative caching in ad hoc networks. In *Proc. of the 4th MDM*, pages 13–28, January 2003.

[13] K. Stathatos, N. Roussopoulos, and J. S. Baras. Adaptive data broadcast in hybrid networks. In *Proc. of the 23rd VLDB*, pages 326–335, Athens, Greece, August 1997.