Proc. of the 3rd Intern'l Conference on Intelligent Multimedia Computing and Networking (IMMCN), Sep. 2003.

# Intelligent Cache Capacity Allocation and Relocation Schemes in a Mobile Proxy

Jiun-Long Huang and Ming-Syan Chen
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC
E-mail: jlhuang@arbor.ee.ntu.edu.tw, mschen@cc.ee.ntu.edu.tw

## Abstract

*In a mobile environment, since a mobile user is able to freely move around these cells, an inter service area handoff occurs when a user moves from the service area of one proxy to that of another. Some cache misses may be incurred by inter service area handoffs since the data objects interest may not be cached in the new proxies. To address this problem, we propose in this paper a novel cache capacity allocation scheme to combine the advantages of the prior cache capacity allocation schemes. A cache relocation scheme is then proposed on the basis of the designed cache capacity allocation scheme to address the problem caused by user mobility.*

***Keyword:*** *Cache capacity allocation, cache relocation, mobile proxy, mobile computing*

## 1 Introduction

The rapid growth of wireless communication has resulted in strong demand of WWW access for mobile devices, and hence attracted much research attention on system design to support WWW access in wireless and mobile environments [2][6]. The characteristics of mobile environments and the long access latency caused by the low speed wireless communication call for the design of new mobile proxies [4][5].

Figure 1 shows a typical mobile environment with mobile proxies. In a cellular network architecture, the whole service area of a mobile environment is divided into several cells. A mobile proxy is in charge of servicing several adjacent cells, and these cells form the service area of the mobile proxy. These mobile proxies are connected by a high speed, fixed network, and are able to access Internet via gateways. In a mobile environment, since a mobile user is able to freely move around these cells, an *inter service area handoff* occurs when a user moves from the service area of one proxy to that of another. Some cache misses may be incurred by inter service area handoffs since the data objects of interest may not be cached in the new proxies. The problem of cache relocation deals with how to move the cached data objects from the original proxy into the new one when an inter service handoff occurs.

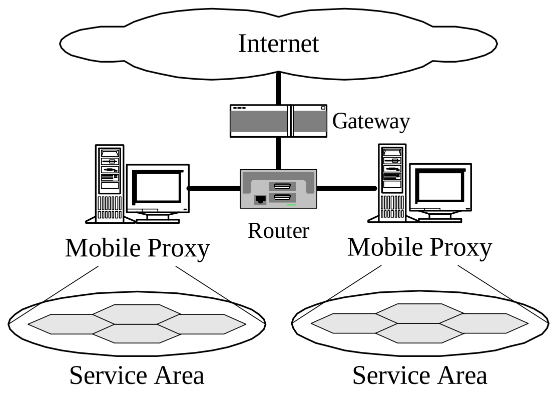

**Figure 1. Network architecture**

In addition, the problem of cache capacity allocation deals with how to organize the cache capacity of a proxy to fulfill the given requirements. Two categories of cache capacity allocation schemes, global and personal cache capacity allocation schemes, have been employed in the literature. Although global cache capacity allocation is able to optimize the overall performance, it may cause the problem of fairness especially to the users with different access interest from others. In addition, it is difficult to design a cache relocation for global cache capacity allocation, and therefore, the system performance degrades due to the cache misses incurred by inter service area handoffs. On the other hand, authors in [4] and [5] had proposed a personal cache capacity scheme by allocating a personal cache, which is exclusively used by its owner, for each user in order to facilitate cache relocation. However, due to the characteristics of personal cache capacity allocation, the overall cache capacity of the proxy might not be fully utilized.

Consequently, we develop in this paper a Hybrid Cache capacity Allocation scheme (referred to as scheme HCA) to combine the respective merits of both schemes. Explicitly, the whole cache capacity is first divided into two pools, a global pool and a personal pool. The global pool stores data objects of high interest from the system's respect. To avoid the fairness problem in global cache capacity allocation, each user is assigned a personal cache allocated in the personal pool to store data objects which are hot from the system's respect but of the user's interest. Moreover, personal cache of each user is shared with others, and each data object is cached at most once in a proxy in order to avoid the problem of personal cache capacity allocation. Each user is also assigned a CacheInfo which records the data objects of his or her interest. These data objects recorded in a CacheInfo are the candidates to be relocated when an inter service area handoff is incurred by the owner of the CacheInfo. Therefore, with the aid of CacheInfoes, we devise a cache relocation scheme on the basis of scheme HCA which is able to cooperate with a moving path prediction algorithm to eliminate possible cache misses incurred by inter area service handoffs.

The rest of this paper is organized as follows. The related work is described in Section 2. The design of the proposed cache capacity allocation scheme is presented in Section 3. The proposed cache relocation scheme is described in Section 4. Finally, Section 5 concludes this paper.

## 2 Related Work

### 2.1 Cache Capacity Allocation

The problem of cache capacity allocation deals with how to organize the cache capacity of a proxy to fulfill the given requirements. Cache capacity allocation schemes used in prior studies can be categorized as the following two categories.

*Global cache capacity allocation:* Global cache capacity allocation is a common assumption in most research studies of proxies [1][8]. In global cache capacity allocation, the whole cache capacity is shared by all users. Data objects which are hot in global view will be cached, and those which are not hot enough will be replaced. The advantage of global cache capacity allocation is that it optimizes the overall performance, and hence, is able to achieve good overall system performance. However, the fairness issue is a problem. Consider a user with distinct access behavior. The data objects of the user's interest are of high likelihood to be replaced since global cache capacity allocation only concerns overall system performance. Hence, the user's average access time is longer than others'. In addition, it is difficult to design cache relocation schemes in global cache capacity allocation since global cache capacity allocation only concerns the overall cache utilization. Therefore, several cache misses may occur due to inter area service handoffs.

*Personal cache capacity allocation:* In personal cache capacity allocation, each user is allocated with a cache capacity dedicated to him or her with a predetermined size [4][5]. The advantage of personal cache capacity allocation is that it not only avoids the fairness problem caused by global cache capacity allocation, but also facilitates cache relocation to be described later in this subsection. However, the problems of personal cache capacity allocation are twofold. First, many copies of one data object may be stored in the proxy, hence reducing the storage utilization and the scalability of the proxy. Second, each personal space is of equal size and is exclusive for its owner. This arrangement wastes the storage if the required size of the personal cache of a user is smaller than that of allocated one. On the other hand, the personal cache is not efficient enough if the required size of the personal cache of a user is larger than that of allocated one.

### 2.2 Cache Relocation

When an inter service handoff occurs, some cache misses may occur since the data objects of the user's interest may not be cached in the new proxy. The problem of cache relocation deals with how to move the cached data objects from the original proxy into the new one when an inter service handoff occurs.

The issue of cache relocation was addressed in [5]. The system designed in [5] employed personal cache capacity allocation and a cache relocation scheme was proposed on the basis of personal cache capacity allocation. To facilitate the proposed cache relocation scheme, a learning automata was designed [5] to capture the users' moving patterns and to predict the users' next movements among cells. When sensing that an inter service handoff is likely to be triggered by a user, the system uses the learning automata to determine the probabilities of all adjacent proxies where the user may move. Then, the system moves the objects cached in the user's personal proxy to these possible new proxies according to the obtained probabilities. The amount of the cached objects that the system moves to each proxy is in proportion to the probability that the user moves into the proxy.

## 3 Design of Scheme HCA

We propose in this section scheme, to combine the advantages of global and personal cache capacity allocation schemes. The characteristics of scheme HCA are as follows.

- Each mobile user is allocated a personal cache to store interesting data objects for this mobile user.
- Each allocated personal cache is not for its owner's exclusive use. That is, data objects cached in personal spaces can be accessed by other users, and the unused space in each personal cache can be lent to other users if necessary.
- Each data object is stored at most once in the proxy at the same time.

### 3.1 Organization of the Cache Capacity

In scheme HCA, the whole cache capacity (denoted as $C_{Total}$) is logically divided into two portions, a global pool and a personal pool, with capacity $C_{Global}$ and $C_{Pers.}$, respectively. Each cached object may be in the PERSONAL, GLOBAL or UNPOINTED state. A cache replacement policy [1][8] is chosen as the underlying cache replacement policy. Since most cache replacement policies employ an *evict* function to determine the cache priorities of all data objects, each cached data object here is also associated with an access information block to store the access information of the object used by the underlying cache replacement policy to calculate cache priority of the data object. An access information block associated with a data object is called a *system-wide* access information block since it stores the access information from the system's point of view. The CacheInfo, which is stored in the personal pool, is assigned to each user using the proxy to store meta data of its owner. In addition, a personal pool also stores data objects in the PERSONAL or UNPOINTED states. Similarly, a global pool is the place to store data objects in the GLOBAL states.

Figure 2 shows the relationship of CacheInfoes, data objects in the global or personal pools. The CacheInfo for each user $i$ consists of $n_{Ptr.}$ pointers pointing to $n_{Ptr.}$ data objects which are of top $n_{Ptr.}$ personal cache priorities from user $i$'s respective. These pointed data objects are the candidates to be relocated when user $i$ triggers an inter service area handoff. The proposed cache relocation scheme employing CacheInfoes is described in Section 4. If a pointer points to one data object, the state of a pointer is set to be the state of the pointed data object. Otherwise, the state of the pointer is UNUSED. Suppose that the size of a CacheInfo is $size(Info)$ and each user is *assigned* a personal cache with capacity $size(Pers.Cache)$. Then, the

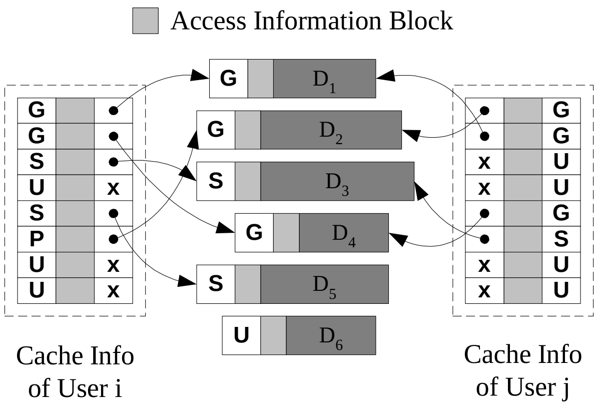

**Figure 2. Cache Info**

maximum number of users for using the service of a proxy with a personal pool with capacity $C_{Pers.}$ at the same time can be formulated as

$$MaxUserNo = \frac{C_{Pers.}}{size(Info) + size(Pers.Cache)}.$$

Let $U_{Shared}$ and $U_{Pers.}$ be the summations of the sizes of the data objects in the SHARED and PERSONAL states, respectively. We also let $U_{Shared}(i)$ be the size of personal cache *used* by user $i$. Then, $U_{Shared}(i)$ can be formulated as

$$U_{Pers.}(i) = \sum_{\substack{\forall D_j \text{ in the SHARED} \\ \text{state and pointed by} \\ \text{one user } i\text{'s pointer}}} \left( \frac{size(D_j)}{\text{No. of pointers pointing } D_j} \right).$$

Consider the example in Figure 2. The size of personal cache used by user 1 (i.e., $User_{Pers.}(1)$) is equal to $\frac{size(D_3)}{2} + size(D_5)$ since $D_3$ is pointed by user 1's and user 2's pointers.

Note that the size of personal cache *used* by a user may be larger than that *assigned* to the user, especially when the number of users in service is smaller than $MaxUserNo$. In addition, each pointer is associated with an access information block to store the access information of the pointed object from user $i$'s respect used by the underlying cache replacement policy. An access information block in user $i$'s CacheInfo is called a *personal* access information block of user $i$ since it stores the access information of the pointed object from user $i$'s respect.

### 3.2 State Transition and Primitive Operations

Figure 3 shows the state transition diagram of each data object. The state of each data object which is newly cached

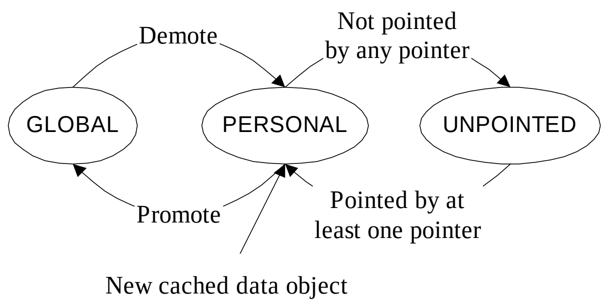

**Figure 3. State transition diagram of each data object**

in the proxy is set to be the PERSONAL state. The state of the data object will be promoted to the GLOBAL state when the cache priority of a data object in the PERSONAL state is high enough. Similarly, the state of a data object in the GLOBAL state will be demoted to the PERSONAL state if the cache priority of the data object is not high enough. On the other hand, the state of a data object in the PERSONAL state will be changed to the UNPOINTED state if the data object is not pointed by any pointer. The state of a data object in the UNPOINTED state will be changed to the PERSONAL state if the data object is pointed by at least one pointer. Note that only data objects in the UNPOINTED states will be considered by the underlying cache replacement policy as the candidates to be replaced.

We now describe some primitive operations to facilitate the design of the proposed cache replacement policy.

**DropPointer** Operation DropPointer is used to drop the pointer with the least personal cache priority among pointers in a specified user's CacheInfo. Given a user $i$, operation DropPointer first calculates the personal cache priorities of the data objects pointed by user $i$'s pointers from user $i$'s respect. Suppose that the data object $D_k$ is with the least personal cache priority. Then, the state of the pointer in user $i$'s CacheInfo pointing $D_k$ is set to be the UNUSED state. The state of $D_k$ will be set to be the UNPOINTED state if $D_k$ is not pointed by any pointer. Finally, $U_{Global}$, $U_{Pers.}$ and $U_{Pers.}(j)$ for each user $j$ with a pointer in the SHARED state pointing to $D_k$ are recalculated.

**InsertPointer** Given a pointer $p$, a data object $D_k$ and a user $i$, operation InsertPointer will insert $p$ into user $i$'s CacheInfo. First, $p$ is set to point $D_k$. If the state of $D_k$ is UNPOINTED, the state of $p$ and $D_k$ are set to be the PERSONAL states. Otherwise, the state of $p$ is set to be the state of $D_k$. Then, pointer $p$ is inserted into user $i$'s CacheInfo. If there is at least one pointer in the UNUSED state in user $i$'s CacheInfo, one of these pointers is removed. Otherwise, operation DropPointer is activated to drop one of the pointers in user $i$'s CacheInfo.

**Demote** Operation Demote is invoked to demote the state of the specified data object, say $D_k$, from the GLOBAL state to the PERSONAL state. To demote $D_k$, the states of $D_k$ and all pointers pointing to $D_k$ are first set to be the PERSONAL states. Then, $U_{Global}$, $U_{Pers.}$ and $U_{Pers.}(j)$ for each user $j$ with a pointer pointing to $D_k$ are recalculated. Operation Promote terminates if $U_{Pers.} \leq C_{Pers.}$.

If $U_{Pers.} > C_{Pers.}$, operator Demote first finds the user, say user $i$, with the largest $U_{Pers.}(i)$. Then, operator DropPointer is invoked to drop one of the pointers in user $i$'s CacheInfo. $U_{Global}$, $U_{Pers.}$ and $U_{Pers.}(j)$ for each user $j$ with a pointer pointing to $D_k$ are recalculated. The steps mentioned in this paragraph repeat until $U_{Pers.} \leq C_{Pers.}$.

**Promote** Operation Promote is used to promote the state of the specified data object, say $D_k$, from the PERSONAL state to the GLOBAL state. To promote $D_k$, the states of $D_k$ and all pointers pointing to $D_k$ are first set to be the GLOBAL states. Then, $U_{Global}$, $U_{Pers.}$ and $U_{Pers.}(j)$ for each user $j$ with a pointer pointing to $D_k$ are recalculated. Operation Promote terminates if $U_{Global} \leq C_{Global}$.

If $U_{Global} > C_{Global}$, operator Promote first finds the data object, say $D_x$ with the least global cache priority among all data objects in the GLOBAL states, and demotes $D_x$ by operation Demote. $U_{Global}$, $U_{Pers.}$ and $U_{Pers.}(j)$ for each user $j$ with a pointer pointing to $D_k$ are then recalculated. The steps mentioned in this paragraph repeat until $U_{Global} \leq C_{Global}$.

## 3.3 Cache Replacement Policy

We now describe the proposed cache replacement policy by considering a scenario that a user $i$ issues a request to data object $D_k$. Note that the proposed cache replacement policy for scheme HCA employs a priori cache replacement policy to calculate the cache priorities of all data objects.

### Cache Hit

When a cache hit occurs, the system-wide access information of $D_k$ is updated. When there is a pointer in user $i$'s CacheInfo pointing to $D_k$, the personal access information of $D_k$ in user $i$'s CacheInfo is also updated. In addition, if the state of $D_k$ is PERSONAL, the system will try to promote the $D_k$ to GLOBAL state by operation Promote. On the other hand, if there is no pointer in user $i$'s CacheInfo pointing to $D_k$, operation InsertPointer will be invoked to insert a pointer pointing to $D_k$ and with the same state as $D_k$ into user $i$'s CacheInfo.

**Cache Miss**

When a cache miss occurs, the proxy first retrieves $D_k$ from the corresponding data server, returns $D_k$ to user $i$, and sets the state of $D_k$ as PERSONAL. Operation InsertPointer is then invoked to insert a pointer pointing to $D_k$ and in the PERSONAL state into user $i$'s CacheInfo. The procedure to handle cache misses terminates if $U_{Global} + U_{Pers.} \leq C_{Total}$.

If $U_{Global} + U_{Pers.} > C_{Total}$, some data objects should be replaced since storage space is not enough. If there are some data objects in the UNPOINTED states, the data object with the least system-wide cache priority among them is dropped in turn until $U_{Global} + U_{Pers.} \leq C_{Total}$. It is possible that $U_{Global} + U_{Pers.}$ is still larger than $C_{Total}$ even when all data objects in the UNPOINTED states are dropped. Suppose that user $j$ is of the largest $U_{Pers.}(j)$ among all users $\Big($i.e., $U_{Pers.}(j) = \max_{\forall\ user\ x}\{User_{Pers.}(x)\}\Big)$. Under this situation, operator DropPointer is invoked to drop one of user $j$'s pointer. The steps in this paragraph repeat until $U_{Global} + U_{Pers.} \leq C_{Total}$.

## 4 Design of Cache Relocation Scheme

We now design a cache relocation scheme on the basis of scheme HCA. When an inter area service handoff from proxy $P_a$ to proxy $P_b$ is triggered, proxy $P_b$ first allocates a CacheInfo for user $i$ and resets all pointers in the CacheInfo to be the UNUSED states. The data objects pointed by pointers in user $i$'s CacheInfo in proxy $P_a$ are considered to be moved to proxy $P_b$ in turn according to their personal cache priorities.

For each considered data object $D_k$, the pointer in user $i$'s CacheInfo in proxy $P_a$ pointing to $D_k$ is first set to be in the UNUSED state. If $D_k$ is not pointed by any pointer in proxy $P_a$, the state of $D_k$ in proxy $P_a$ is then set to be the UNPOINTED state. If $D_k$ has been cached in proxy $P_b$, operation InsertPointer is invoked to insert a pointer pointing to $D_k$ in user $i$'s CacheInfo in proxy $P_b$. If $D_k$ is not cached in proxy $P_b$, proxy $P_b$ automatically issues a request of $D_k$ for user $i$. Then, a cache miss on $D_k$ occurs and the procedure to handle cache misses is invoked. Finally, the space of user $i$'s CacheInfo in proxy $P_a$ is marked as unused when all pointers in user $i$ CacheInfo in proxy $P_a$ are in the UNUSED states.

In addition, the system can further employ a moving path prediction algorithm [5][7] to pre-relocate the cached objects when the users are likely to trigger an inter service area handoff to reduce the possible cache misses during the process of relocation.

## 5 Conclusion and Future Work

In this paper, we proposed scheme HCA which combines the advantages of global and personal cache capacity schemes. In addition, a cache relocation was proposed based on scheme HCA to eliminate cache misses incurred by inter service area handoffs.

In our future work, we will design a service admission control scheme since one proxy can only serve $MaxUserNo$ users at a time. In addition, a scheme to dynamically determine the ratio of the sizes of object and CacheInfo pools will also be designed. We shall extend our schemes to be applied to mobile transcoding proxies [3]. Finally, we shall conduct several experiments to evaluate the performance of scheme HCA.

## References

[1] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the World Wide Web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, 1999.

[2] C. Baquero, V. Fonte, F. Moura, and R. Oliveira. MobiScape: WWW Browsing under Disconnected and Semi-Connected Operation. In *Proceedings of First Portuguese WWW National Conference*, July 1995.

[3] C.-Y. Chang and M.-S. Chen. Exploring Aggregate Effect with Weighted Transcoding Graphs for Efficient Cache Replacement in Transcoding Proxies. In *Proceedings of the 18th IEEE International Conference on Data Engineering*, Feburary 2002.

[4] S. Hadjiefthymiades, V. Matthaiou, and L. Merakos. Supporting the WWW in Wireless Communications Through Mobile Agents. *ACM Wireless Networks*, 7(4):305–313, 2002.

[5] S. Hadjiefthymiades and L. Merakos. Using Proxy Cache Relocation to Accelerate Web Browsing in Wireless/Mobile Communications. In *Proceedings of the 10th International Conference on World Wide Web*, pages 26–35, May 2001.

[6] B. C. Housel, G. Samaras, and D. B. Lindquist. WebExpress: A Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment. *ACM Mobile Networks and Applications*, 3(4):419–431, 1998.

[7] W.-C. Peng and M.-S. Chen. Developing Data Allocation Schemes by Incremental Mining of User Moving Patterns in a Mobile Computing System. *IEEE Transactions on Knowledge and Data Engineering*, 15(1), February 2003.

[8] J. Shim, P. Scheuermann, and R. Vingralek. Proxy Cache Algorithm: Design, Implementation, and Performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4), July-August 1999.