Update-Based Cache Access and Replacement in Wireless Data Access

Hui Chen, *Student Member*, *IEEE*, Yang Xiao, *Senior Member*, *IEEE*, and Xuemin (Sherman) Shen, *Senior Member*, *IEEE*

Abstract—Cache has been applied for wireless data access with different replacement policies in wireless networks. Most of the current cache replacement schemes are access-based replacement policies since they are based on object access frequency/recency information. Access-based replacement policies either ignore or do not focus on update information. However, update information is extremely important since it can make access information almost useless. In this paper, we consider two fundamental and strongly consistent access algorithms: Poll-Per-Read (PER) and Call-Back (CB). We propose a server-based PER (SB-PER) cache access mechanism in which the server makes replacement decisions and a client-based CB cache access mechanism in which clients make replacement decisions. Both mechanisms have been designed to be suitable for using both update frequency and access frequency. We further propose two update-based replacement policies, least access-to-update ratio (LA2U) and least access-to-update difference (LAUD). We provide a thorough performance analysis via extensive simulations for evaluating these algorithms in terms of access rate, update rate, cache size, database size, object size, etc. Our study shows that although effective hit ratio is a better metric than cache hit ratio, it is a worse metric than transmission cost, and a higher effective hit ratio does not always mean a lower cost. In addition, the proposed SB-PER mechanism is better than the original PER algorithm in terms of effective hit ratio and cost, and the update-based policies outperform access-based policies in most cases.

Index Terms—Cache replacement policy, access, update, wireless network.

1 Introduction

Wireless data access suffers from inherent constraints such as bandwidth and battery power limitation and inevitable user mobility, resulting in expensive communication cost, large access latency, and frequent disconnections from networks. Wireless data access typically follows client-server models, where databases are hosted at remote servers with clients (i.e., mobile terminals) accessing data objects stored at the databases through mobile/wireless networks. The servers update the objects when being requested. Caches at mobile terminals (MTs), called client caching, can save bandwidth usage and power consumption.

A *cache access algorithm* describes how a client-server system uses the cache. It is also called a cache consistency/coherency algorithm or a cache invalidation scheme if the data consistency/invalidation schemes are emphasized. For many applications, a strongly consistent cache data access algorithm has to be adopted, requiring that obsolete data have to be evicted from the cache or made distinguishable from valid data. In this paper, we are studying strongly consistent cache data access in a mobile/wireless environment.

- *H. Chen is with the Department of Computer Science, The University of Memphis, 373 Dunn Hall, Memphis, TN 38152. E-mail: hchen2@memphis.edu.*
- *Y. Xiao is with the Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35487. E-mail: yangxiao@ieee.org.*
- *X. (Sherman) Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. E-mail: xshen@bbcr.uwaterloo.ca.*

Manuscript received 4 Apr. 2005; revised 11 Dec. 2005; accepted 8 Feb. 2006; published online 16 Oct. 2006.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0116-0405.

Invalidation Report (IR) schemes have been proposed and studied in a mobile environment with consideration of disconnections of mobile terminations in [2], [3]. In an IR scheme, the server periodically broadcasts a message, called the *invalidation report*, to all its clients. The message contains sufficient information on the update history of data objects so that clients know whether the cached data objects are valid or not. Several related major issues, such as the sizes of IR, energy-savings, disconnections of MTs from networks, access latency, and communication cost, have been addressed in recent studies [4], [5], [6], [7], [8], [10], [11], [12], [16], [20], [24]. The approaches include adjusting broadcasting intervals, organizing contents of IR, and/or reducing uplink transmissions. Since most of them require broadcasting within entire networks and are suitable to be implemented in lower layers of mobile/wireless network protocol stacks, it is believed that the IR schemes should perform well under the following assumptions: 1) The channel is a broadcasting channel, 2) all the clients are within one wireless cell in a cellular network or wireless LAN (WLAN), and 3) the server is also local to the wireless cell, such as the base station in a cellular network or the access point in the WLAN. However, in a realistic wireless network, the above assumptions are hardly true. In other words, the server is more likely located in a remote site and subscribed clients may be all over the cellular network or even in remote wireless Internet access points. For example, if all the clients are in different wireless cells, the broadcasting of IRs becomes very expensive since IRs contain too much information that is irrelevant to a particular client in a particular wireless cell. Furthermore, IRs are needed to be sent separately to different clients located at different wireless cells. Using broadcasting to distribute IRs to save cost becomes less favorable. Therefore, the IR schemes are not practical approaches in realistic wireless applications [13].

The effective hit ratios and communication costs of two fundamental cache access algorithms, poll-each-read (PER) and call-back (CB), for client-server data access models have been studied and compared in a mobile/wireless environment [13]. The PER and CB algorithms were originally introduced for efficient network file systems [9], [14], [23]. In PER, when a client attempts to read data objects from a remote server, the client asks the server whether it is valid or not, and the server sends the data object only if it is invalid. PER is a stateless cache access algorithm in which the server does not keep track of what data objects are updated, and the client enforces replacement policies. In CB, the server informs the clients that the data objects in their caches are invalid.

A *cache replacement policy* is executed when the cache is full. In this case, if an accessed data entry is to be cached, some other cached data entries have to be evicted to accommodate the newly accessed object. In the client-server model for mobile/wireless data access, the update process plays an important role because it makes cached data obsolete. Thus, a replacement policy may have to utilize update information besides access information. Most replacement policies, such as the Least-Recently-Used (LRU) and Least-Frequently-Used (LFU) replacement policies [13] widely used in other systems such as operating systems and computer architectures, have been introduced into mobile/wireless network client caching. These replacement policies basically use access time (recency) or access frequency [15] to determine which data object is to be replaced. However, the studies in [13] only consider LRU without the effects of the update process in the replacement policies. Two replacement policies have been proposed to utilize update information in [21] and [22] to study IR schemes and to reduce only *stretch* [1], which is defined as normalized access latency, and it is a totally different goal from the communication cost and the effective hit ratio which we study in this paper.

In this paper, we are interested in strongly consistent cache access algorithms and update-based replacement policies which are applicable to more realistic wireless networks with more than one wireless cell. In such systems, IR schemes are inefficient due to different locations of the clients who subscribe the service, and those clients may be a very small percentage of the total number of users in the wireless system. The cache access algorithms have to consider update information besides access information. We propose a server-based PER and a revised version of CB with two update-based and frequency-based replacement policies, the *least access-to-update ratio* (LA2U) and the *least access-to-update difference* (LAUD). Since bandwidth is costly in mobile/wireless networks, we focus on reducing not only the access latency, but also the usage of the bandwidth in terms of the communication cost. In PER, some invalidated objects still occupy cache space if these objects are not accessed. This is the fundamental problem of PER. On the other hand, SB-PER can fix this problem. We also propose an analytical model for SB-PER and compare it with PER using both analytical and simulation results. In addition, we provide an in-depth analysis and performance evaluation via extensive simulations. It should be mentioned that our proposed algorithms are not limited to those in wireless networks. They should also be applicable to wired networks, such as Internet client-server applications, WEB caching, etc., because the proposed algorithms improve not only the communication cost, but also the effective hit ratio.

Since our research work is to introduce new cache access algorithms in order to reduce transmission cost over wireless links, the server-based PER requires more powerful servers. We believe that a server can be easily upgraded with a more powerful machine or a cluster of machines, i.e., a virtual server or a server farm. Furthermore, the server-based PER improves effective hit ratio and communication cost, i.e., fewer objects are fetched from the server. We believe that the server-based PER can be extended to systems with a large number of users.

The rest of the paper is organized as follows: We introduce some applications of wireless data access in Section 2. We propose SB-PER and the revised CB in Section 3. We present two update-based replacement policies in Section 4. An analytical model for SB-PER is proposed in Section 5. Performance evaluation and comparison are presented in Section 6. Finally, we conclude the paper in Section 7.

2 Applications in Wireless Data Access

An MT can be a small computer with a relatively powerful operating system and hardware, especially for 3G cell phones. Subscribers can run data applications on MTs to access various data. Data sources, e.g., database servers, are not necessarily located within the cellular network, e.g., remote Internet services. In the following sections, we introduce some potential applications, such as the Wireless Application Protocol (WAP) application, the Real-time Stock Price Monitor, and a business card application. Our algorithms should also be useful for many other potential application services.

2.1 WAP Application

The WAP Forum was formed to standardize the WAP for cellular networks. WAP is aimed at enabling easy, fast delivery of relevant information and services to mobile users. All the WAP applications are running in client-server models. The WAP application server or other entities in the network can be informed of the capability of the handheld devices through the user agent profile [19] defined in WAP. Hence, the WAP application server can deliver the content in a way that handheld devices can handle.

Cache mechanism has been proposed for WAP applications [18]. It can be used to better utilize the bandwidth of the network, reduce the power consumption of the mobile terminals, and decrease the latency time for WAP applications. The communication links between MTs and base stations are wireless; therefore, the communication cost becomes a very important issue since wireless bandwidth is limited and precious; on the other hand, the communication links between base stations and a server are wired links, which assume that there is enough bandwidth. Handheld devices maintain their own caches. The contents delivered by a Web server (for example, stock information) can be cached in the cache of a handheld device. We call a unit of content cached in the cache a data object. The handheld device tries to use the data objects in the cache first. If the device uses the data objects in the cache, the bandwidth of the network is saved and the latency time is reduced.

2.2 Real-Time Stock Price Monitor

A real-time stock price monitor can be used to check stock prices, and a user can make a transaction using the MT based on the stock information obtained from a server. Stock prices are dynamically changed, and the user can obtain stock prices in real-time, hourly, daily, or any time unit basis. If a stock transaction is involved, large access latency may not be appropriate. In such a case, IR schemes may not be appropriate.

In this application, the data sources may or may not be located outside of the cellular networks. Stocks are updated and accessed at different rates. Some stocks may be traded in a huge volume. Their prices may be updated very frequently. On the other hand, the prices of other stocks may not be updated so frequently.

2.3 Business Card Application

In [13], a business card application is proposed and implemented in wireless telecommunication networks. The application requires a remote database maintained at a remote server. Each record in the database stands for a correspondence. All records are maintained at the server. A user may view, modify, and add records through an MT. The records can be cached in MTs to reduce the communication cost and latency. The business card application also helps a user maintain a consistent phone book at any networked terminal. When the user changes his/her terminal, he/she is not required to transfer the phone book from the old terminal to the new terminal [13]. The application server may maintain a public phone book like the yellow pages. Then, the user's record may refer to the records in the public phone book. When any record in the public phone book is changed, it can be reflected to the users. Those users can then always access the latest records.

The environment to host the business card application is iSMS [13], a platform that integrates an IP network with the Short Message Services (SMS) in the mobile telephone network. Wireless data services are provided through the iSMS gateway whose role is similar to the WAP gateway. The business card application follows the vCard standard to format a business card. A vCard contains a field for a single piece of structured data, including the family name, given name, additional names, honorific prefixes, and honorific suffixes. It also contains a field for a telephone number which may include more than one phone number. The business card application also defines a field for a calendar which is absent in the vCard standard. Even though the size of the vCard stored in the application server may vary, the vCard is always tailored to be of fixed size when it is sent to a wireless terminal. For more information about iSMS and business card application, please refer to [13] and the references therein. Although the above applications are for cellular networks, they can also be applied to other types of wireless networks, for example, wireless LANs, and even wired networks, such as the Internet.

3 Cache Access Algorithms

We first introduce two traditional cache access algorithms, PER and CB, in Section 3.1. Traditionally, PER and CB do not store the access and update information, and they are only suitable for the time (recency)-based algorithms. Therefore, we propose a server-based PER and a revision of CB in Sections 3.2 and 3.3, respectively, in which access information and update information are maintained. The proposed schemes are suitable for frequency-based algorithms. For example, to maintain update and access frequencies, two different counters, access counter for each client object and update counter for each object, are maintained.

3.1 PER and CB

In PER, a client sends a request message to the server to check whether a data object is still valid when the client requests to access the object. The server sends a confirmation message to the client if the object is valid; otherwise, the server sends the object to the client. The server can update objects when needed without notifying the clients. PER uses a stateless server, i.e., the server does not keep track of what data objects are updated.

On the other hand, CB has different access and update procedures. A client checks whether an object is in the cache when the client requests to access the object. If the object is not in the client cache, the client retrieves the object from the server; otherwise, the client uses the cached object. When the server wants to update an object, it informs all the clients to evict the object from the cache. The CB server is a stateful server in the sense that the server needs to maintain the client information. The objects cached at a client are always valid and a client does not need to contact the server when an access happens unless the data object is not in the cache.

3.2 Server-Based Poll-Each-Read

The server-based PER (SB-PER) algorithm, shown in Fig. 1a, requires a user profile for each client maintained at the server. The history of the accesses on each object from a client is stored in its user profile. The user profile also stores what objects are cached at the client. This can be implemented by storing the identities of cached objects in the user profile. The server also has an object profile in which the update history on each object is stored.

When a client wants to access an object, it always sends a request to the server (Step 1.2). The server hence knows which client initiates the access and which object is to be used. It stores the access history of the object in the user profile of the corresponding client (Step 1.3). Different from the meaning of the corresponding request message in traditional PER, this message may not check the validity of the cached object at the client. It is used to inform the server that an access on an object is requested (Step 1.2).

When an update occurs (Step 4.1), the server removes the corresponding object from the user profile of the corresponding client at the server or makes it invalid. It also renews the update history for the object (Step 4.2) in the object profile.

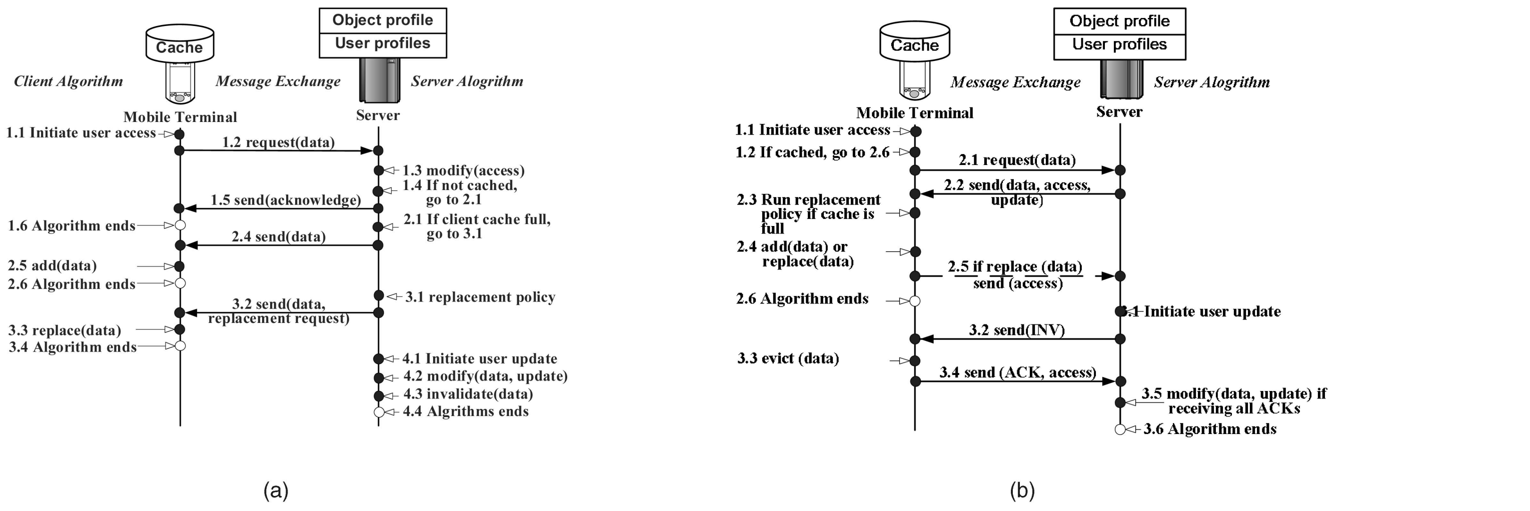


Fig. 1. Update-based cache access algorithms: (a) the server-based poll-each-read algorithm and (b) the revised Call-Back algorithm.

Therefore, the server can check the user and object profiles for not only access and update history, but also the cache status of objects. The server makes the following decisions when an access occurs:

1. If the object is cached and is valid, the client uses the cached object (Step 1.5 and Step 1.6). An object is valid if there have been no updates to this object since it was cached.
2. If the object is not cached and the cache is not full, the server sends the object to the client, and the client caches it (Steps 2.4, 2.5, and 2.6).
3. If the object is not cached and the cache is full, the server chooses a previously updated cached object to be evicted and this updated cached object is also called an invalid object. If there is not such an object as the above, the server chooses a cached object to be evicted according to the replacement policy (Step 3.2 and Step 3.3).

Note that, in SB-PER, the server makes replacement decisions and knows which objects are obsolete in the caches as well as the access and update history. In the original PER algorithm, clients decide which object will be evicted from the cache. However, clients are not aware of update information because the server does not contact the client when an update occurs. Furthermore, only the server has all the knowledge of global access information and global update information. In other words, the server is the most knowledgeable entity that can make the smartest decision. In the original PER, a client can only have partial access information without global access information and any update information, so its decision should be less globally optimal. Therefore, the proposed server-based PER should outperform the original PER if the knowledge mentioned above can be fully utilized.

It seems that it may be unnecessary to make PER server-based. The server can maintain user profiles for each client to keep update history. When an access is requested, the server sends the update history back to the client with the object or the reply message. However, such an approach not only introduces additional traffic overhead, but also causes response delay to the clients. Furthermore, the information of the client side may not be global information.

3.3 A Revised Call-Back

The proposed CB algorithm is shown in Fig. 1b, which differs from the original CB algorithm by inclusion of maintenance and exchange of access and update history between clients and the server.

The server maintains an update history of all objects. It has an object profile to store the update history. There are two ways to maintain access history. We can either allow clients to maintain access history of all objects they have ever accessed, or let the server maintain both access and update history. In the former case, each client has an object profile. It keeps the access history of all objects the client has ever accessed. In the latter case, the server has a user profile for each individual client. In the following discussion, to avoid storing too much access history information at the client which is scarce of resources, the server stores global access histories, and the client only maintains access and update histories of cached objects. In such a case, the server updates the user profile to renew the access history in the following two situations:

1. When the object is not cached, the client requests the object from the server (Step 2.1).
2. When the cache is full and a cache replacement happens (Step 2.3), the client sends the access history of the replaced object to the server (Step 2.5).

The server has to send access and update history with the object to the client when an object is read from the server (Step 2.2).

The server could perceive all the access history happening at the client only when an update is happening to the object and the server receives the acknowledgment with access history. The server may, hence, not be appropriate to make the replacement decision due to inaccurate access history. In contrast, the client knows the accurate access and update history using the proposed algorithm. Therefore, the client is the best entity to make the decision, and the update-based replacement policy is more appropriate to be deployed at the client.

4 Update-Based Replacement Policies

When an object is accessed at a client, the client may cache the object. If the cache is full, a cached object may be evicted so that the new object can be cached. It may not be beneficial that the replacement decision relies solely on access history. Given two objects having different access rates, the object with a higher access rate should be cached if only access history is taken into account. However, if the object has also a much higher update rate, it may become invalid right after it has been cached. Therefore, caching an object with a high update rate (no mater whether the access rate is higher or lower) may only bring up the message cost and may not benefit the cache hit. Therefore, we propose two update-based replacement policies by using access and update frequencies. In other words, they are frequency-based schemes and are called the Least Access-to-Update Ratio and the Least Access-to-Update Difference.

4.1 Least Access-to-Update Ratio (LA2U)

When an accessed object satisfies the replacement policy, it will be cached. The larger the access rate of the object is, the more frequently the object will be accessed. On the other hand, the cached object then becomes invalid. If the access rates of all objects are the same, similarly to the LFU, a replacement policy can be proposed to cache the least frequently updated objects. Therefore, access and update rates are two opposite factors. It would be best if a replacement policy caches the more-frequently-accessed but less-frequently-updated objects. This can be done by designing a replacement policy using a measurement which is proportional to the access rate but inverse-proportional to the update rate. The simplest form gives the least access-to-update ratio replacement policy.

The proposed least access-to-update ratio replacement policy (LA2U) chooses the object with the least access-to-update ratio not to be cached among the cached objects and the object being accessed.

The access-to-update ratio r_{ij}^{A2U} of object j regarding client i is defined as $r_{ij}^{A2U} = f_{ij}^a / f_j^a$, where f_{ij}^a is the access frequency of object j at client i and f_j^u is the update frequency of object j at the server. Note that f_j^u does not have a subscript i, but f_{ij}^a does since f_j^u is a global statistic.

In the user and object profiles, access and update counters are maintained for each object. The access and update frequencies can be measured by $f_{ij}^a = n_{ij}^a / t$ and $f_j^u = n_j^u / t$, where n_{ij}^a is the accumulated number of accesses on object j at client i up to time t and n_j^u is the accumulated number of updates on object j at the server. We denote μ_{ij} and λ_j as the access and update rates of object j at the client i and at the server, respectively. Obviously, we expect f_{ij}^a to approach access rate μ_{ij} and f_j^u to approach λ_j when t is significantly large, i.e., $\mu_{ij} = \lim_{t\to\infty} f_{ij}^a$ and $\lambda_j = \lim_{t\to\infty} f_j^u$. Therefore, we have $r_{ij}^{A2U} = n_{ij}^a / n_j^u$.

The replacement policy compares the ratios among all the objects cached at the client with the accessed object. If the object with the least access-to-update ratio is not the access object, it will be evicted and replaced with the object requested; otherwise, the accessed object will not be cached and there will be no change in the cache.

The actual implementation is designed to also consider the situation when n_j^u is zero. It ensures that LA2U is equivalent to the LFU when there is no update at all.

4.2 Least Access-to-Update Difference (LAUD)

There is more than one object at the server. Two objects with the same access-to-update ratio but different access rates have different effects on the total effective hit ratio. Therefore, we propose the least access-to-update difference replacement policy (LAUD) in which the replacement policy uses the difference of access and update rates other than the access-to-update ratio.

LAUD is the same as the LA2U except that it uses access-to-update difference rather than access-to-update ratio. The access-to-update difference d_{ij}^{AUD} of object j regarding client i is defined as follows: $d_{ij}^{AUD} = f_{ij}^a - f_j^u = (n_{ij}^a - n_j^u)/t$. Because the algorithm has the accurate access and update history and both access and update histories are measured from the beginning to the time t, we can simply use the following measure $d_{ij}^{AUD} = n_{ij}^a - n_j^u$ instead.

4.3 Remarks

The proposed replacement policies are actually adaptive algorithms based on measurements. The time t can be defined as a moving time window, i.e., the accesses and updates will only be accumulated during the time window of length t from a previous time to current time. The time window length t can be adjusted to capture the variation of the popularity of different objects varying with time. In addition, the access-to-update ratio and the access-to-update difference are measured for each object. Even though more computational power from the server may be required, the proposed replacement policies can capture that different objects have different popularity in terms of access and update rates.

5 An Analytical Model for SB-PER

In [13], analytic models for both PER and CB are derived. In this section, we derive an analytic model for the proposed SB-PER scheme and compare PER and SB-PER under the LRU replacement policy with both analytical results and simulation results in terms of effective hit ratio and communication cost. In the analytical model, the following assumptions are used which are the same as in [13]:

- The server database has a total of N objects, and all objects have the same size.
- A client can hold K objects in its local cache ($K \leq N$).
- Accesses to a data object $O_j (j = 1, \ldots, N)$ follow a Poisson distribution with rate μ_j, where $\mu_j = \mu$ for $j = 1, \ldots, N$.
- Updates to a data object $O_j (k = 1, \ldots, N)$ follow a Poisson distribution with rate λ_j, where $\lambda_j = \lambda$ for $j = 1, \ldots, N$.
- The replacement policy LRU is adopted.

Let p_{CB} denote the effective hit ratio of the CB scheme. From [13], we have

$$p_{CB} = \frac{1}{N(\mu+\lambda)^N} \sum_{k=0}^{K-1} \sum_{l=k+1}^{N} \left[\binom{N}{l} \lambda^{N-l} \mu^l \right]. \tag{1}$$

Let $p_{SB\text{-}PER}$ denote the effective hit ratio of SB-PER. We claim that $p_{SB\text{-}PER}$ should be the same as p_{CB}. This is because, for CB, when an update even happens, the server sends an invalidation message to clients so that clients have accurate validation information of objects in the cache to make decisions. In SB-PER, when an update happens, although the server does not send an invalidation message to the client, the server makes decisions and knows accurate validation information of objects in the cache. In other words, for both CB and SB-PER, the entity that makes decisions knows accurate validation information of objects in the cache. Therefore, we have

$$p_{SB\text{-}PER} = p_{CB} = \frac{1}{N(\mu+\lambda)^N} \sum_{k=0}^{K-1} \sum_{l=k+1}^{N} \left[\binom{N}{l} \lambda^{N-l} \mu^{l} \right]. \quad (2)$$

Let c_{req}, c_{ack}, and c_{obj} denote communication costs for the request message, the acknowledgment message, and the data object, respectively. Let $p_{SB\text{-}PER}$ and $c_{SB\text{-}PER}$ denote the effective hit ratio and communication cost per access for SB-PER. We have

$$c_{SB\text{-}PER} = c_{req} + p_{SB\text{-}PER} c_{ack} + (1 - p_{SB\text{-}PER}) c_{obj}. \quad (3)$$

Let p_{PER} and c_{PER} denote the effective hit ratio and the communication cost per access for PER, respectively. From [13], we have

$$p_{PER} = \sum_{k=0}^{K-1} \left\{ \binom{N-1}{k} \left[\frac{k!}{\prod_{m=0}^{k} \left(N - k + \frac{\lambda}{\mu} + m \right)} \right] \right\}, \quad (4)$$

$$c_{PER} = c_{req} + p_{PER} c_{ack} + (1 - p_{PER}) c_{obj}. \quad (5)$$

Compared with CB, SB-PER has an advantage of saving transmission cost of sending invalidation messages. Compared with PER, SB-PER has an advantage that the entity who makes decisions knows accurate validation information of objects in the cache. However, for PER, the decision maker (i.e., the client) does not have accurate validation information of objects in the cache so that some invalidated objects still occupy cache space if their objects are not accessed. This is the fundamental problem of PER, which the proposed SB-PER can address.

Fig. 2 shows the effective hit ratios and communication costs of PER and SB-PER obtained through both analytical models and simulations with different λ/μ values, where $c_{req} = 60$, $c_{ack} = 60$, $c_{obj} = 120$, and $\mu = 1$. It can be seen that SB-PER is better than PER in terms of both effective hit ratio and communication cost. Furthermore, both the analytical and simulation results match very well. A detailed description of the simulation approaches is given in Section 6. Similar results have been obtained when c_{obj} is much larger than the message sizes.

SB-PER also performs better than PER even though only LRU is adopted. If both proposed updated polices are applied, SB-PER performance can be further improved, as shown in the next section. Another advantage of SB-PER is that a client does not need to record any access/update information so that we refer it as to a thin-client.

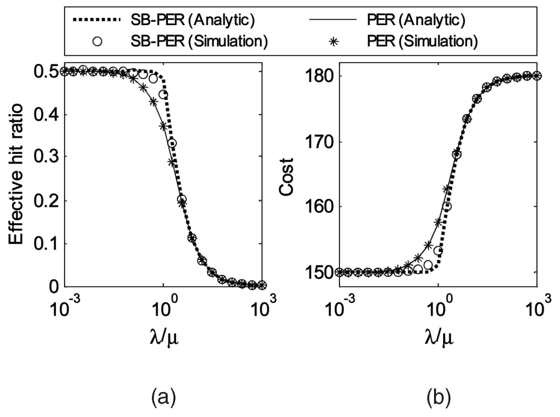


Fig. 2. Comparing PER and SB-PER. (a) Effective hit ratio versus λ/μ. (b) Cost versus λ/μ.

6 Performance Evaluation

In this section, we use discrete event simulation to evaluate the performance of the proposed cache access algorithms and replacement policies. Our simulation programs are written in C++. In most cases of our simulation experiments, objects are divided into two classes, denoted as class A and class B. Classes A and B contain N_A objects and N_B objects, respectively, where $N = N_A + N_B$. In each class, the access rates are the same and the update rates are the same for all the objects. Let μ_A and μ_B denote the access rate of each object in class A and class B, respectively, λ_A and λ_B denote the update rate of each object in class A and class B, respectively, and K denote the cache size at the client in terms of the number of objects. In this section, CB stands for the revised CB described earlier. Note that N is a set that a user could access in a period of time. It is often not the size of the database because a mobile client usually accesses only a small portion of data due to its function limitation.

In the following section, we present performance metrics and assumptions. Section 6.2 compares SB-PER with the original PER. Section 6.3 studies the effects of K and N, and Section 6.4 studies the effects of update rate and access rate. Section 6.5 compares the LA2U and the LAUD replacement policies. We extend our work to a more general network traffic type in Section 6.6. Finally, in Section 6.7, we provide some remarks on SB-PER and CB.

6.1 Performance Metrics and Simulation Assumptions

The server has N objects and the client is equipped with a cache which can hold up to K objects. Assume that the access/update events follow a Poisson distribution. An access always happens at a client and an update always occurs at the server. The rates of the access and the update to the object j are μ_j and λ_j, respectively. We have the following measures:

1. n_j^a: Total number of accesses to object j at the client.
2. n_j^u: Total number of updates to object j at the server.
3. n_{miss}: Total number of accesses that result in the object delivery. It happens when the accessed object is absent (i.e, a cache miss) or invalid in the cache no matter whether SB-PER or CB is used.

4. n_{inv}: Total number of invalidation messages. When CB is exercised, an invalidation message will be delivered to the client when the object being updated is cached.
5. n_{rep}: Total number of cache replacements.

The above measures are used to calculate the effective hit ratios and the communication costs for both SB-PER and CB per client. The effective hit ratio is defined as the ratio of the total number of valid cache hits over the total number of accesses, where an object is valid. We denote the effective hit ratios in SB-PER and CB as $p_{SB\text{-}PER}$ and p_{CB}, respectively. The communication costs per access in SB-PER and CB are denoted as $c_{SB\text{-}PER}$ and c_{CB}, respectively. Let n_a be the total number of accesses regardless of the objects. Similarly to [13], we have

$$p_{SB\text{-}PER} = p_{CB} = 1 - \frac{n_{miss}}{n_a}, \text{ where } n_a = \sum_{i=1}^{N} n_i^a. \tag{6}$$

When SB-PER is executed, the client sends a request message to the server whenever an access happens. As shown in Fig. 1a, Step 1.2 includes the message. We denote the traffic cost of the message as c_{req}. When a cache miss happens, a copy of the object needs to be sent to the client (Step 3.2); otherwise, an acknowledgment is sent (Step 1.5). The costs of sending an acknowledgment and the object are c_{ack} and c_{obj}, respectively. Note that the server makes replacement decisions when the client cache is full. The decision is delivered to the client with the data object (Step 3.2). We assume it can be stored in the message header and there is no extra cost. A similar analysis can be applied to CB. Note that c_u and c_a are the extra costs of messages containing the update and access frequencies, respectively (Steps 2.2, 2.5, and 3.4 of Fig. 1b), and p_{rep} is the probability that a cache replacement is needed when an access happens and the cache is full. We have

$$\begin{aligned} c_{SB\text{-}PER} &= \frac{n_a c_{req} + (n_a - n_{miss})c_{ack} + n_{miss}c_{obj} + n_{rep}c_{rep}}{n_a} \\ &= c_{req} + p_{SB\text{-}PER}c_{ack} + (1 - p_{SB\text{-}PER})c_{obj}, \end{aligned} \tag{7}$$

$$\begin{aligned} c_{CB} &= \frac{1}{n_a}\big[n_{miss}(c_{req} + c_{obj} + c_u) \\ &\quad + n_{inv}(c_{inv} + c_{ack} + c_a) + n_{rep}c_a\big] \\ &= (1 - p_{CB})(c_{req} + c_{obj} + c_u) \\ &\quad + p_{inv}(c_{inv} + c_{ack} + c_a) + p_{rep}c_a. \end{aligned} \tag{8}$$

We also have $n_{rep} \leq n_{miss} \leq n_a$. In the simulations, we assume the costs of request message (Step 1.2 in Fig. 1a and Step 2.1 in Fig. 1b), acknowledgment (Step 1.5 in Fig. 1a Step 3.4 in Fig. 1b), and invalidation message (Step 3.2 in Fig. 1b) are the same: $c_{req} = c_{ack} = c_{inv} = 60$. The extra costs of the replacement decision, access, and update counter are the same as well: $c_a = c_u = 4$. We assume the costs of sending objects are the same and $c_{obj} = 120$.

6.2 Comparison of SB-PER and PER

We compare the proposed SB-PER with the original client-based PER using LFU. Note that PER is not suitable to use the update-based replacement policies, such as LA2U and LAUD, which are also frequency-based replacement policies. Both of them use measured access and update frequencies. To use these two replacement policies, the access and update histories have to be stored at either the client or the server. When PER is exercised, the client makes replacement decisions, but it does not know the update histories in PER since all updates occur only at the server and the server does not inform the client. Therefore, PER is not suitable to use update histories in its replacement policies. On the contrary, PER is able to record all access histories since all accesses happen at the client. PER can use the replacement policies which use the access histories only. Therefore, our comparison between SB-PER and PER in this section is limited to LFU.

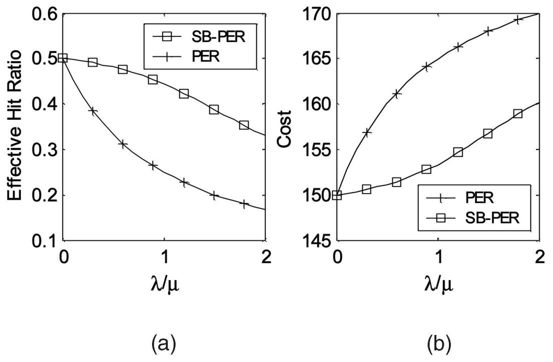


Fig. 3. Metrics over λ/μ. (a) Effective hit ratio versus λ/μ. (b) Cost versus λ/μ.

In Fig. 3, we assume that the update rates of objects are the same, and the access rates of objects are the same. Let $N = 40$, $K/N = 1/2$, and $\mu = 1.0$, and only one class of objects is used. Fig. 3 compares effective hit ratios and costs of SB-PER and PER. We have the following interesting observations:

- For both SB-PER and PER, the effective hit ratio decreases and the cost increases when λ/μ increases. For a larger λ/μ value, there are more updates on objects so that the cached objects are obsolete more frequently and, therefore, more cache misses happen; as a result, the effective hit ratio is lower and the cost is higher.
- SB-PER has a better effective hit ratio than PER when $\lambda/\mu > 0$. When $\lambda/\mu = 0$, both algorithms have the same effective hit ratio since there is no update at all. When $\lambda/\mu > 0$ and SB-PER is exercised, obsolete objects will be chosen to be replaced at first. But, for PER, the client cannot realize if an object is obsolete. This is why SB-PER outperforms PER. In other words, we expect that the valid objects in the cache in SB-PER are more than (or equal to) those in PER. This will be verified in Fig. 4.
- The cost is higher in SB-PER than in PER when λ/μ is larger than 0. The same reason in the above bullet for the effective hit ratio explains this observation.

Fig. 4 shows the number of valid objects in the cache and the α over time, where $\alpha = C/K$ and C is the number of accessed objects that are invalid, referred to as invalid cache hit. We also refer α as the percentage of the harmful

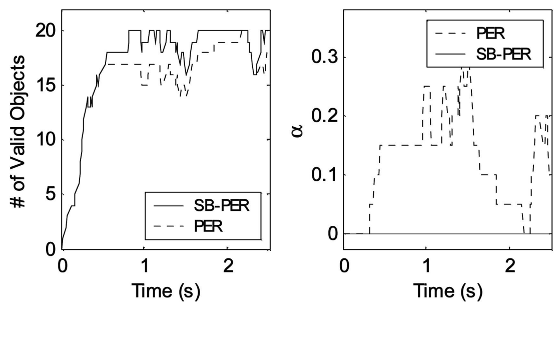


Fig. 4. Number of valid objects and α over time. (a) Number of valid objects versus time. (b) α versus time.

wasted-cache. The following parameters are adopted for Fig. 4: $N = 40$, $K = 20$. Furthermore, the access rates (μ) are equal to 1.0 for all the objects, and all the update rates (λ) are equal to 0.5. Fig. 4a shows that the number of the valid objects in SB-PER is larger than (or equal to) that in PER. Fig. 4b shows that α in SB-PER is zero, whereas α in PER even reaches 30 percent, i.e., 30 percent cache space is wasted for causing invalid cache hits.

The two experiments shown in Fig. 3 and Fig. 4 are actually homogenous cases since all access rates or the update rates are the same for all the objects. In Fig. 5, we show heterogeneous cases where the access rates or the update rates are not the same. We have two classes A and B as described before, and parameters are chosen as follows: $N_A = N/2$, $N_B = N/2$, $\mu_A = \mu_B = 1.0$, and $K = N/2 = 20$. We vary $\lambda_B - \lambda_A$ from 0.0 to 2.0 while maintaining $\lambda_A + \lambda_B = 2.0$. In other words, λ_A varies from 1.0 to 0.0, while λ_B varies from 1.0 to 2.0. The purpose of this experiment is to study the effects of different distributions of updates among two classes on the performance, given that the total update rate is fixed.

Fig. 5 also shows the average numbers of valid objects of class A and class B, as well as their difference when SB-PER or PER is exercised. We have the following observations:

- In Fig. 5a, when $\lambda_B - \lambda_A$ increases, for both SB-PER and PER, the average number of valid objects of class A increases since the update ratio of class A objects decreases, but that of class B decreases since the update ratio of class B objects increases.
- In Fig. 5b, when $\lambda_B - \lambda_A$ increases, for both SB-PER and PER, the difference between the average number of valid objects of class A and that of class B increases. Interestingly, the figure shows that the difference in SB-PER is larger than that in PER since class A objects have a smaller update rate and SB-PER tends to cache more class A objects.

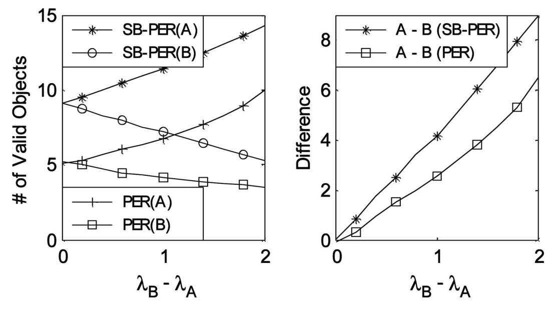


Fig. 5. Average numbers of valid objects and their difference. (a) Number of valid objects versus $\lambda_B - \lambda_A$. (b) Difference versus $\lambda_B - \lambda_A$.

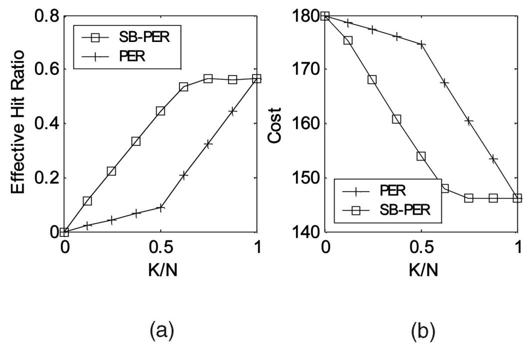


Fig. 6. Metrics over K/N. (a) Effective hit ratio versus K/N. (b) Cost versus K/N.

In Figs. 6, 7, 8, and 9, several experiments for more heterogeneous cases are presented and not only do the update rates of two classes significantly differ, but also the access rates are not the same. The parameters for the experiments are chosen as follows: $N_A = N/2$, $N_B = N/2$, $K = N/2$, $\mu_A = 1.0$, $\lambda_A = 0.0$, $\mu_B = 1.1$, and $\lambda_B = 5.5$. Fig. 6 shows effective hit ratio and cost over K/N when $N = 40$. We have the following observations under current parameter setting:

- In Fig. 6, the effective hit ratios of both SB-PER and PER increase, while the costs of both SB-PER and PER decrease when K/N increases. When $K/N = 0$ or $K/N = 1$, SB-PER and PER have the same effective hit ratio and the cost. Since $K/N = 0$ means that the cache size is zero, each access will cause data fetch from the server for both SB-PER and PER. When $K/N = 1$, no object is replaced since all the objects can be held in the cache. Hence, the performance of SB-PER and that of PER are the same. SB-PER outperforms PER in terms of both the effective hit ratio and the cost when $0 < K/N < 1$. When K/N increases from 0 to 1, the objects start to

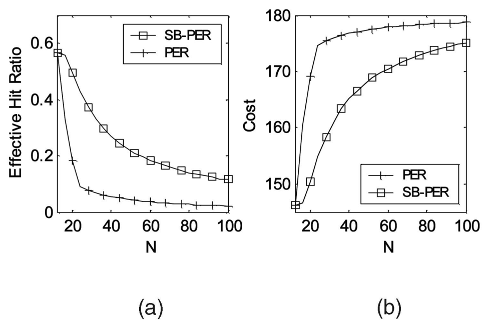


Fig. 7. Metrics over N when $K = 12$. (a) Effective hit ratio versus N. (b) Cost versus N.

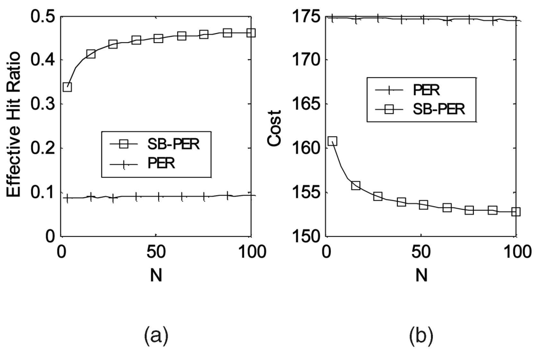


Fig. 8. Metrics over N when $K/N = 1/2$. (a) Effective hit ratio versus N. (b) Cost versus N.

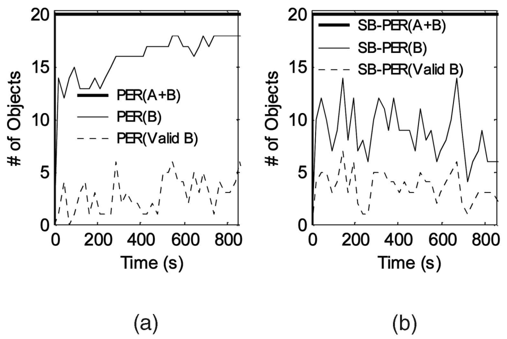


Fig. 9. Number of objects over simulation time. (a) Number of objects versus time for PER. (b) Number of objects versus time for SB-PER.

be cached. Unlike PER, invalid objects in SB-PER may be replaced.

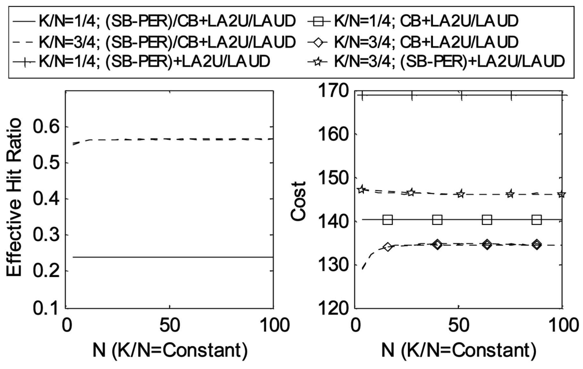
- For PER, the cached objects always stay in the cache because the server does not inform the client of the update information. For SB-PER, the cached objects also always stay in the cache even though some objects are obsolete. If the cache is not full for both schemes, no object will be evicted. We expect that there are more class B objects than class A objects in the cache because $\mu_B > \mu_A$ for both schemes. Therefore, the two schemes should have the same performance under this condition. From Fig. 6, we observe that, in all cases of K/N, the cache will be full at least sometimes.
- SB-PER is better than PER since, in SB-PER, there are invalid objects in the cache at the client and the server makes decisions to replace those invalid objects when the cache is full. On the other hand, the client in PER has no information of invalidity of objects in the cache.
- For PER, the effective hit ratio (cost) increases (decreases) slower when K/N is smaller than that when K/N is large. When $K/N \leq N_B/N$, LFU favors class B objects because $\mu_A < \mu_B$. However, class B objects are more costly since they have larger update rates so that they are easier to be invalidated. When $K/N > N_B/N$, the cache size is large enough to cache more class A objects so that better effective hit ratio and cost can be achieved since class A objects cannot be invalidated. In other words, more class A objects can be cached when K/N is large.
- The effective hit ratio and the cost of SB-PER becomes flat when K/N is larger than a threshold. The effective hit ratio (cost) increases (decreases) as K/N increases when K/N is smaller than the threshold. This observation indicates that, when K/N reaches a certain value, further increasing cache size K will not improve the effective hit ratio or cost. Since class A objects are less frequently updated objects, SB-PER favors class A objects when K/N increases. However, when K/N is large enough, all the class A objects are cached. On the other hand, class B objects are more-frequently-accessed objects. When K/N increases, more class A and B objects are cached so that higher effective hit ratio and lower cost are achieved. However, more B objects in the cache means that more of them will get invalidated. Therefore, on average, only a few of class B objects are in the cache, especially when K/N reaches the threshold.

Fig. 7 shows the effective hit ratio and cost over N when $K = 12$. We have the following observations:

- When $N = K$, the effective hit ratios and costs in SB-PER and PER are the same, respectively, with the same reason stated above.
- When $K < N$, the effective hit ratio and the cost in SB-PER are better than those in PER for the similar reasons explained before.
- For both SB-PER and PER, as N increases, the effective hit ratio decreases and the cost increases.

Fig. 8 shows the effective hit ratio and cost over N when $K/N = 1/2$. We have the following observations:

- SB-PER has a much higher effective hit ratio and lower cost than PER. PER favors caching more-frequently-accessed objects, and the client is not aware of updates at the server so that the obsolete cached objects may be still in the cache and the client cannot distinguish obsolete objects from valid objects. For PER, an obsolete object in the client cache is never replaced by a different object unless when the cache is full and the replacement policy program decides to replace it. Otherwise, the obsolete object will stay in the cache forever. For SB-PER, an obsolete object in the client cache can be replaced by a different object even before the replacement policy program makes decision since, if the server finds there is an obsolete object, it will decide to replace it by the accessed object without going through the replacement policy program. On the other hand, for PER, a valid object may be replaced even though there is one invalid object in the cache. In this experiment, LFU is used by both SB-PER and PER. LFU tends to cache class B objects. In the long run, $\mathrm{PER} + \mathrm{LFU}$ tends to cache more class B objects, which become obsolete easily since their update rates are higher. On the other hand, for SB-PER, the number of class B objects is smaller since spaces of some invalid objects are available.
- The effective hit ratio and cost of PER are almost flat with N when $K/N = 1/2$.



(a) (b)

Fig. 10. Effective hit ratio and cost over N when K/N is fixed (without LFU). (a) Effective hit ratio versus N. (b) Cost versus N.

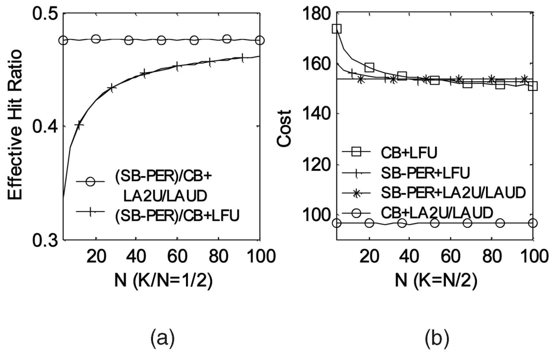
Fig. 9 shows the numbers of objects over simulation time for both PER and SB-PER. Fig. 9a and Fig. 9b both show that the total number of objects is 20, which is the cache size. The number of cached class B objects in PER is larger than that in SB-PER, and the number of cached valid class B objects for PER is smaller than or relatively the same as that for SB-PER. Since more cached class B objects mean worse performance, SB-PER therefore outperforms PER under the current parameter setting. Furthermore, the figure implies that the number of cached class A objects in PER is smaller than that in SB-PER.

6.3 Effects of K and N

In Sections 6.3 and 6.4, parameters are chosen as follows: $N_A = N/2$, $N_B = N/2$, $\mu_A = 1.0$, $\lambda_A = 0.0$, $\mu_B = 1.1$, $\lambda_B = 5.5$, $K = N/2$, and $N = 40$ unless stated otherwise. In the following, $\mathrm{CB} + \mathrm{LAUD/LA2U}$ and $\mathrm{SB\text{-}PER} + \mathrm{LAUD/LA2U}$ are also referred to as the updated-based cache algorithms.

Fig. 10 shows how the effective hit ratio and cost vary with N when K/N is fixed. We have the following observations under the above parameter setting except that K/N ratio is chosen differently:

- In Fig. 10a and Fig. 10b, the effective hit ratios and costs are flat when N is relatively large and K/N is fixed. It indicates these two metrics largely depend on only K/N when N is large. Therefore, K/N is a good parameter instead of cache size to compare different schemes.
- In Fig. 10a and Fig. 10b, there are four schemes under two different K/N ratios: the $\mathrm{SB\text{-}PER} + \mathrm{LA2U}$, $\mathrm{SB\text{-}PER} + \mathrm{LAUD}$, $\mathrm{CB} + \mathrm{LA2U}$, and $\mathrm{CB} + \mathrm{LAUD}$ schemes under either $K/N = 1/4$ or $K/N = 3/4$.
- Fig. 10a also shows that with larger K/N ratio, better/higher effective hit ratio is obtained. There is no difference in terms of effective hit ratio between SB-PER and CB as well as between LA2U and LAUD.
- Fig. 10b also shows that, with a larger K/N ratio, a better cost is obtained. Furthermore, CB has a lower cost than SB-PER. There is no cost difference between LA2U and LAUD in the same cache access scheme. Note that CB and SB-PER have the same effective hit ratio in Fig. 10a, but different costs in Fig. 10b.



(a) (b)

Fig. 11. Effective hit ratio and cost over N when K/N is fixed (with LFU). (a) Effective hit ratio versus N. (b) Cost versus N.

Fig. 11 compares the effective hit ratios and the costs of the six schemes over N when $K/N = 1/2$, where the six schemes are

$$\mathrm{SB\text{-}PER} + \mathrm{LFU},\ \mathrm{SB\text{-}PER} + \mathrm{LA2U},\ \mathrm{SB\text{-}PER} + \mathrm{LAUD},$$
$$\mathrm{CB} + \mathrm{LFU},\ \mathrm{CB} + \mathrm{LA2U},\ \text{and}\ \mathrm{CB} + \mathrm{LAUD}.$$

We have the following observations regarding to K and N:

- Fig. 11a shows that LA2U/LAUD has better effective hit ratio than LFU. In other words, the update-based replacement policies can improve the performance in terms of the effective hit ratio due to the following reasons: LFU tends to cache class B objects more than class A objects since $\mu_A < \mu_B$. However, LA2U tends to cache class A objects more than class B objects since $\mu_A/\lambda_A > \mu_B/\lambda_B$, and LAUD tends to cache class A objects more than class B objects since $\mu_A - \lambda_A > \mu_B - \lambda_B$. Therefore, the update-based cache algorithms cache class A objects more than class B objects. On the other hand, class B objects have a higher update ratio than the class A objects, but have a similar access ratio as the class B objects. Caching class A objects is preferred since the effective hit ratio will be improved by caching less frequently updated objects. Therefore, the updated cache replacement policies are better than LFU.
- Fig. 11a also shows the effective hit ratio of LFU increases as N increases when K/N is fixed. A similar observation is also shown in Fig. 8a. On the other hand, the effective hit ratio of LA2U/LAUD is flat with N since K/N is fixed, and this observation is also shown in Fig. 10a.
- Fig. 11b shows that $\mathrm{CB} + \mathrm{LA2U}$ and $\mathrm{CB} + \mathrm{LAUD}$ have the smallest costs among the six schemes. The cost of $\mathrm{CB} + \mathrm{LFU}$ decreases when N increases and K/N is fixed, and a similar observation is shown in Fig. 8b. Fig. 11b also shows that $\mathrm{CB} + \mathrm{LFU}$ has worse cost than $\mathrm{SB\text{-}PER} + \mathrm{LFU}$ when N is relatively small.

Fig. 12 shows effective hit ratio and cost over K/N when $N = 40$. We have the following observations under current parameter setting:

- Fig. 12a shows that a K/N threshold exists. The effective hit ratio increases as K/N increases when

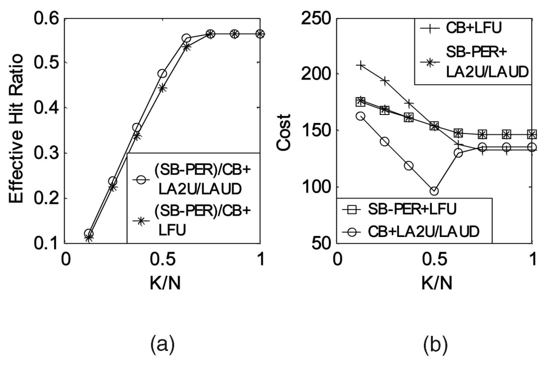


Fig. 12. Effective hit ratio and cost over K/N when $N = 40$. (a) Effective hit ratio versus K/N. (b) Cost versus K/N.

K/N is smaller than the threshold. The effective hit ratios become flat when K/N is greater than the threshold. Similarly, Fig. 12b shows that a K/N threshold exists. The cost becomes flat when K/N is larger than the threshold. This observation indicates that, when K/N reaches a certain value, further increasing cache size K will not improve effective hit ratio or cost. A similar observation has been obtained from Fig. 6.

- Fig. 12a also shows that updated-based replacement policies are slightly better than the LFU under the current parameter setting.
- A more interesting observation is stated as follows: In Fig. 12b, the costs of $\mathrm{CB} + \mathrm{LA2U}$ and $\mathrm{CB} + \mathrm{LAUD}$ decrease when K/N increases and $K/N \leq 1/2$, but, when K/N further increases and $K/N > 1/2$, the costs increases. According to previous discussions, we know that $\mathrm{CB} + \mathrm{LA2U}$ and $\mathrm{CB} + \mathrm{LAUD}$ tend to cache more class A objects. Since there is no update for class A objects, the cost decreases when more class A objects are cached. However, when $K/N \geq 1/2$, the algorithms will sometimes cache class B objects since the cache is large enough to hold all A class objects and some B class objects. Caching class B objects will introduce invalidation messages which are sent from the server to the client when updates happen to cached objects so that the cost increases. One conclusion from this observation is that a larger cache size does not always perform better in terms of the cost. Cache size should be relatively large to get better performance, but overlarge cache size may degrade the performance.
- Fig. 12b also shows that the costs of $\mathrm{SB\text{-}PER} + \mathrm{LA2U}$ and $\mathrm{SB\text{-}PER} + \mathrm{LAUD}$ decrease as K/N increases and are gradually becoming flat when $K/N > 1/2$. The reasons are similar to those in the previous bullet: Class B objects will be cached when $K/N > 1/2$. However, compared with CB, SB-PER will not introduce an invalidation message when updates happen to cached objects so that it will not cause more cost when more-frequently-updated objects are cached. On the contrary, more or less, it will improve effective cache hit and cost when K/N increases. Caching more-frequently-used class B objects, however, will increase effective cache hits only a little.

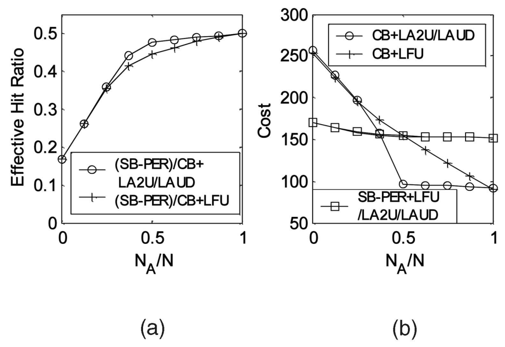


Fig. 13. Metrics over N_A/N. (a) Effective hit ratio versus $\mathrm{N_A/N}$. (b) Cost versus $\mathrm{N_A/N}$.

- Effective hit ratios for $\mathrm{CB} + \mathrm{LA2U}$ and $\mathrm{CB} + \mathrm{LAUD}$ always increase as K/N increases, but the costs of these two schemes do not always decrease as K/N increases. This means that the effective hit ratio and cost are sometimes conflicting objectives. Lower cost normally means a higher effective hit ratio, but a higher effective hit ratio does not always mean lower cost for $\mathrm{CB} + \mathrm{LA2U}$ and $\mathrm{CB} + \mathrm{LAUD}$. In general, in a distributed system, update and access are distributed and independent. The update process is very important and has to be incorporated in a cache access algorithm and a cache replacement policy to ensure better performance.
- Effective hit ratio and cost are two different metrics. That a scheme has a higher effective hit ratio does not mean that it has a lower cost. But, a scheme having lower cost means that it has a higher effective hit ratio at most cases. In such a sense, cost is a much better performance metric than effective hit ratio. This is also observed in other figures, such as Fig. 13.

6.4 Effects of Update Rate and Access Rate

Experiments are performed to study the effects of the update rate and the access rate for the following six schemes: $\mathrm{SB\text{-}PER} + \mathrm{LA2U}$, $\mathrm{SB\text{-}PER} + \mathrm{LAUD}$, $\mathrm{SB\text{-}PER} + \mathrm{LFU}$, $\mathrm{CB} + \mathrm{LA2U}$, $\mathrm{CB} + \mathrm{LAUD}$, and $\mathrm{CB} + \mathrm{LFU}$. SB-PER and CB are cache access schemes, and LAUD, LA2U, and LFU are replacement policies.

In the experiments, we vary the numbers of class A objects and class B objects, N_A and N_B, respectively, while the total number of objects is fixed, i.e., $N = N_A + N_B$, and other parameters are the same as before. Fig. 13 shows how the effective hit ratio and the cost change with N_A/N. We have the following observations:

- Fig. 13a shows that the effective hit ratio increases as N_A/N increases for all six schemes. When N_A/N increases, there are more less-frequently-updated objects in the cache. Therefore, the effective hit ratio increases as N_A/N increases.
- Fig. 13a also shows that update-based replace policies (LA2U and LAUD) are a little better effective hit ratio than LFU. A similar result is also shown in Fig. 11a. Interestingly, when N_A/N is small, the effective hit ratios of LA2U, LAUD, and LFU are similar.

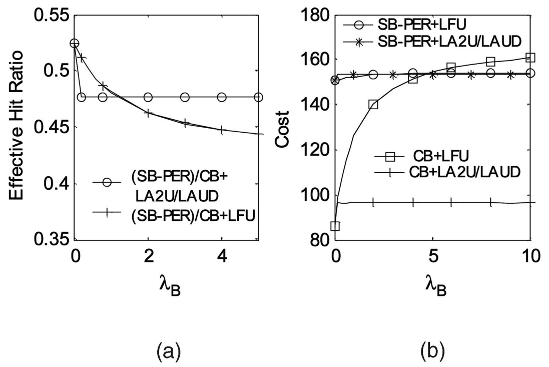


Fig. 14. Metrics over λ_B. (a) Effective hit ratio versus λ_{B}. (b) Cost versus λ_{B}.

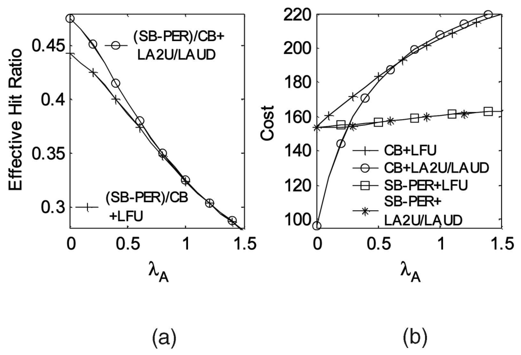


Fig. 15. Metrics over λ_A. (a) Effective hit ratio versus λ_{A}. (b) Cost versus λ_{A}.

- Fig. 13b shows that the cost decreases for all the schemes as N_A/N increases since more class A objects are cached and class A objects have a smaller update rate. It also shows that $\mathrm{SB\text{-}PER} + \mathrm{LFU}$, $\mathrm{SB\text{-}PER} + \mathrm{LA2U}$, and $\mathrm{SB\text{-}PER} + \mathrm{LAUD}$ have the similar costs.
- Fig. 13b shows that SB-PER has a lower cost than CB when N_A/N is small and has a higher cost when N_A/N is large. When N_A/N is small, more objects are more-frequently-updated so that the cost is high for CB since the server sends invalidation messages to the client when updates happen to cached objects, and these invalidation messages are very costly. When N_A/N is large, more class A objects are cached so that the cost goes down.
- Fig. 13b also shows that the cost of $\mathrm{CB} + \mathrm{LA2U/LAUD}$ decreases less steeply when $N_A/N > 1/2$ as N_A/N increases than when $N_A/N < 1/2$. LA2U and LAUD favor more class A objects. CB favors class A objects too since it favors caching less-frequently-updated objects. When $N_A/N < 1/2$, cache can hold all class A objects. When $N_A/N < 1/2$, the cache holds almost all of the class A objects and a portion of class B objects. When N_A/N increases, class B objects will be squeezed out of the cache by class A objects. When N_A/N increases to a value, the cache cannot hold all of the class A objects and no more class B objects will be squeezed out of the cache. When $N_A/N > 1/2$, the cache is likely full of class A objects. Therefore, the cost does not drop significantly as N_A/N increases.

We vary the update rates λ_B and λ_A while $N_A = N_B = N/2$ and show how effective hit ratio and cost change with λ_B and λ_A, in Fig. 14 and Fig. 15, respectively.

- Fig. 14a shows the effective hit ratio of LFU decreases as λ_B increases. LFU tends to cache class B objects since class B objects are more-frequently-accessed objects. But, class B objects are also more-frequently-updated objects so that there are more invalid cache hits and, therefore, the effective hit ratio drops as λ_B increases.
- The effective hit ratio of LA2U/LAUD decreases as λ_B increases, but it quickly becomes flat. LA2U/LAUD sometimes cache class B objects, especially when λ_B is small. As λ_B increases, LA2U/LAUD tends to cache all class A objects so that the effective hit ratio becomes flat. The same reason holds for the costs of LA2U/LAUD in Fig. 14b.
- The effective hit ratio of LA2U/LAUD is smaller than that of LFU when λ_B is small, but larger than that of LFU when λ_B is large. In other words, there exists a threshold. When λ_B is smaller than the threshold, LFU has a higher effective hit ratio than LA2U/LAUD. On the contrary, when λ_B is larger than the threshold, LFU has a lower effective hit ratio than LA2U/LAUD. When λ_B is small, both updates of class A objects and class B objects are small and, in this case, the benefits of LA2U/LAUD do not show in terms of effective hit ratio. But, if we consider cost, as shown in Fig. 14b, LA2U/LAUD is always better than or equivalent to LFU.
- Fig. 14b shows that $\mathrm{CB} + \mathrm{LA2U}$ and $\mathrm{CB} + \mathrm{LAUD}$ are the best among all schemes in terms of cost for different λ_B values. Furthermore, LA2U/LAUD is always better than LFU in terms of cost for different λ_B values.
- Fig. 14b shows that the update rate can significantly increase the cost of $\mathrm{CB} + \mathrm{LFU}$. But, LA2U/LAUD has almost a constant cost except the first point where there is no update at all.
- Fig. 14b also shows that the cost of the $\mathrm{SB\text{-}PER} + \mathrm{LFU/LA2U/LAUD}$ is much higher than that of $\mathrm{CB} + \mathrm{LA2U/LAUD}$. Since only class B objects are updated, caching class B objects introduces invalidation costs in CB. But, we have $K = 20$ so that the cache is full of class A objects since $N_A = 20$. Therefore, CB will not introduce many invalidation messages because class A objects are not updated according to our parameter setting. Even though SB-PER does not have invalidation messages, SB-PER requires a request message at each access which is not present in CB. Therefore, SB-PER has a higher cost than CB under current parameter setting.
- Fig. 15a shows when λ_A increases, the effective hit ratios of all schemes decrease since a larger λ_A value means that more class A objects are invalidated easily. LA2U/LAUD is slightly better than LFU in terms of effective hit ratio under current parameter setting. For the same reasons, shown in Fig. 15b, the costs of all schemes increase when λ_A increases.

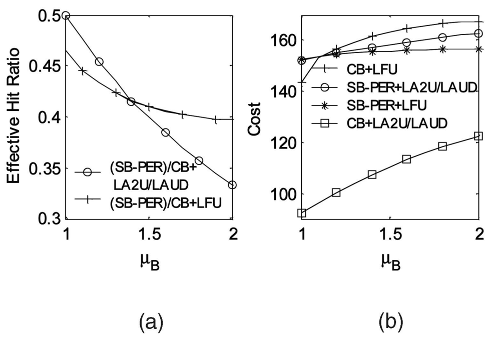


Fig. 16. Metrics over μ_B. (a) Effective hit ratio versus $\mu_{\rm B}$. (b) Cost versus $\mu_{\rm B}$.

- Fig. 15b shows that

$$\text{SB-PER} + \text{LFU}, \text{SB-PER} + \text{LA2U}, \text{and SB-PER} + \text{LAUD}$$

have almost the same cost. $\text{CB} + \text{LFU}$ is almost the worst in terms of cost. $\text{CB} + \text{LA2U}$ and $\text{CB} + \text{LAUD}$ outperform $\text{SB-PER} + \text{LFU/LA2U/LAUD}$ only when λ_A is small.

Fig. 16 shows how the effective hit ratio and the cost vary as μ_B increases, and we have the following observations:

- Fig. 16a shows that the effective hit ratios of LFU/LA2U/LAUD decrease when μ_B increases since increasing μ_B will increase the chances to cache class B objects. When μ_B increases, there are more accesses on class B objects, but class B objects have very large update rates. Therefore, caching more class B objects will decrease effective cache hit. Furthermore, caching more class B objects means caching fewer class A objects. Class A objects have no update. Effective cache hit ratios will hence be decreased. Therefore, under our current parameter setting, increasing μ_B will have negative effects on effective hit ratio.
- Fig. 16a also shows that there is a μ_B threshold. When μ_B is smaller than the threshold, LA2U/LAUD have lager effective hit ratios and, when μ_B is larger than the threshold, LFU has a larger effective hit ratio.
- Fig. 16b shows that $\text{CB} + \text{LA2U}$ and $\text{CB} + \text{LAUD}$ have the smallest cost. The costs of all schemes increase as μ_B increases.

The above observations indicate that the relationship between access and update rate matters. In general, a good replacement policy tends to cache objects with a high access rate and a low update rate. However, the effects of access and update are very complex.

6.5 Comparison of LA2U and LAUD

In this section, we compare two proposed replacement policies with the following parameters: $N_A = N/2$, $N_B = N/2$, $\mu_A = 1.0$, $\lambda_A = 0.1$, $\mu_B = 100.0$, $\lambda_B = 20.0$, and $K = N/2$. According to LA2U, $\mu_A/\lambda_A = 10.0 > \mu_B/\lambda_B = 5.0$, and any object from Class B in the cache tends to be replaced by the object from Class A. On the other hand, any object from Class A in the cache tends to be replaced by the object from Class B according to LAUD since $\mu_A - \lambda_A = 0.9 < \mu_B - \lambda_B = 80.0$.

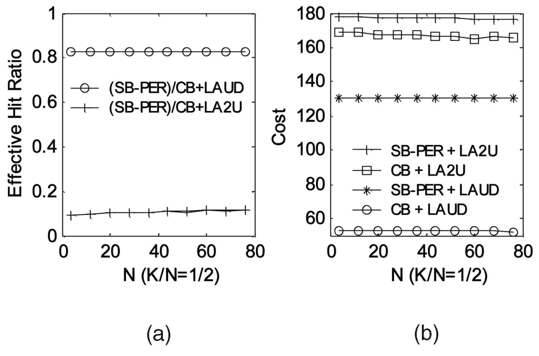


Fig. 17. Metrics over N $(K/N = 1/2)$. (a) Effective hit ratio versus N. (b) Cost versus N.

Fig. 17 shows effective hit ratio and cost over N when $K/N = 1/2$.

- Fig. 17a shows that $\text{SB-PER} + \text{LAUD}$ and $\text{CB} + \text{LAUD}$ have the same effective hit ratio when N increases, and $\text{SB-PER} + \text{LA2U}$ and $\text{CB} + \text{LA2U}$ have the same effective hit ratio when N increases. Furthermore, it shows that LAUD has a higher effective hit ratio than LA2U.
- Fig. 17b shows LA2U has a higher cost than LAUD, $\text{CB} + \text{LAUD}$ has the smallest cost, and $\text{SB-PER} + \text{LA2U}$ has the largest cost.

The parameter setting in Fig. 17 favors LAUD. Therefore, the better performance is expected for LAUD.

Fig. 18 shows effective hit ratio and cost over $\mu_B - \lambda_B$ under the following parameters: $N_A = N/2$, $N_B = N/2$, $\mu_A = 10.0$, $\lambda_A = 5.0$, $\mu_B/\lambda_B = \mu_A/\lambda_A = 2$, and $K = N/2$. $\mu_B - \lambda_B$ is changed from the values smaller than $\mu_A - \lambda_A$ to values larger than $\mu_A - \lambda_A$. It can be seen that LAUD outperforms LA2U under the current parameter setting. An interesting observation is that there is one point at which these two schemes have the same performance.

6.6 Access and Update in Gamma Distribution

Previous discussions have been based on the assumption that both the access and the update events follow Poisson distributions, i.e., the access event intervals and the update event intervals are drawn from exponential distributions. In this section, we show the similar results remain when we lose

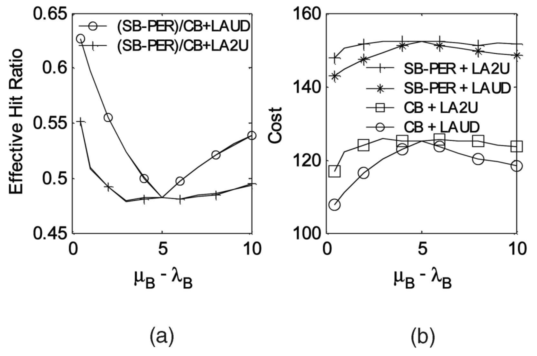


Fig. 18. Metrics over $\mu_B - \lambda_B$. (a) Effective hit ratio versus $\mu_{\rm B} - \lambda_{\rm B}$. (b) Cost versus $\mu_{\rm B} - \lambda_{\rm B}$.

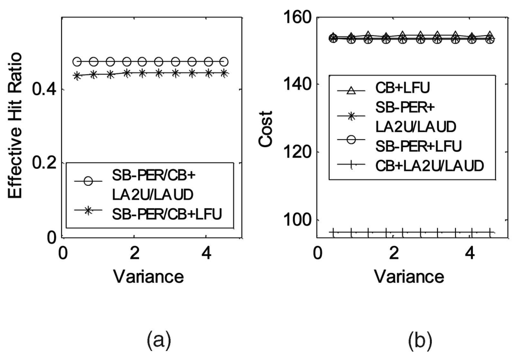


Fig. 19. Metrics over variance of Gamma distribution. (a) Effective hit ratio versus variance. (b) Cost versus variance.

this assumption. As an example, we compare the effective hit ratios and the communication costs of $\mathrm{CB/SB\text{-}PER} + \mathrm{LA2U/LAUD/LFU}$ when the access and update events intervals follow Gamma distributions, respectively.

Fig. 19 shows that, when the variance of the Gamma distributions varies, the effective hit ratio and communication costs are almost invariant for a given scheme. Therefore, we expect that our schemes will perform similarly for different traffic.

6.7 Remarks on SB-PER and CB

From Figs. 10, 11, 12, 13, 14, 15, 16, 17, and 18, CB and SB-PER, in general, have no significant difference on effective hit ratio when update-based replacement policies are exercised. However, it is not the case for cost. From Fig. 13 and Fig. 15, CB has a lower cost when the update rate is small, whereas SB-PER has a lower cost when the update rate is high. One possible reason is that SB-PER requires contacting the server at each read while CB does not. On the other hand, CB needs to send an invalidation message to clients whenever an update happens to a cached object. There is a trade-off between CB and SB-PER.

7 Conclusions

In this paper, we proposed the SB-PER and the revised CB cache access mechanisms. We further proposed two update-based replacement policies, LA2U and LAUD. We conducted extensive simulations and demonstrated that the update rate is a very important factor for designing a cache access scheme and replacement policy in wireless/mobile networks. In fact, this claim should also be applied to any distributed system using client-server model.

In PER, the client does not realize whether an object is updated/invalid or not. It is not a good choice for the client-server data access in distributed systems due to ignorance of the update process. Both PER and CB are not suitable for cache replacement policies using update histories, especially frequency-based replacement policies. The proposed SB-PER and the revised CB are able to use both update and access frequencies. Because the SB-PER algorithms can inform clients about the update information, more space of invalid objects is available in the cache than PER, and it has better performance than PER even using non-update-based replacement polices.

The proposed update-based replacement policies reduce the chance to cache more frequently updated objects. The effective hit ratio and communication costs can be improved when updates are heavy. Similar results can also be observed when CB is applied.

SB-PER is a stateful cache access algorithm, whereas PER is a stateless cache access algorithm. To fully utilize update and access history in replacement policies, a stateful SB-PER is better, whereas PER can only use full access histories, i.e., a client in PER must keep all access histories of objects.

SB-PER is a cache access algorithm ready to use both update and access information in its replacement policies. It maintains all access and update histories, which allow any kind of replacement policies no matter whether access, update, or both histories are required.

SB-PER and PER are extensively studied using LFU. The invalidation mechanism introduced in SB-PER makes it tend to cache more-frequently-accessed instead of less-frequently-updated objects. Therefore, SB-PER outperforms PER in both effective hit ratio and cost. When the update frequency is moderate, SB-PER dramatically improves the effective hit ratio and cost over PER. The understanding of the invalidation mechanism of SB-PER and the comparison of SB-PER and PER also help to understand the proposed update-based replacement policies.

The revised CB is a stateful cache access algorithm. The replacement policies are enforced at the client. It is suitable to adopt any replacement policy using access and update histories. The effective hit ratios and costs of SB-PER and CB with the update-based replacement policies, LA2U and LAUD, depend only on K/N, regardless of K and N.

A K threshold exists. The effective hit ratios of both SB-PER and CB are monotonically increasing with K when K is smaller than the threshold. The effective hit ratios are flat when K is larger than the threshold. In some cases, the cost of CB with a larger cache size is even worse than that with a smaller cache size. It indicates that a larger cache does not always mean a better performance.

Update-based replacement policies are better than LFU when the update rates of objects are moderately high and are not uniform. On the contrary, when access process is dominant, LFU is better. According to our simulation results, LAUD is better than LA2U in terms of effective hit ratio.

Another interesting result is that, normally, in the caching of the operating system and computer architecture, increasing the cache hit ratio is the only major objective. In [13], it is pointed out that, in wireless data access, the cache hit ratio is no longer a good metric, but the effective hit ratio is the one since some of the cache hits are invalid. In this paper, we found out that even the effective hit ratio is not necessarily a very good metric, but the communication cost is a better metric. A scheme that has a higher effective hit ratio does not mean that it has a lower communication cost, but a scheme having a lower cost communication does mean that it has a higher effective hit ratio at most cases. In this sense, the communication cost is a much better performance metric than effective hit ratio.

In summary, through extended simulations using carefully designed cases, we gain a deep understanding of various cache algorithms in terms of the access scheme, cache size, database size, access rate, update rate, and many more.

References

[1] S. Acharya and S. Muthukrishnan, "Scheduling On-Demand Broadcasts: New Metrics and Algorithms," *Proc. MobiCom,* pp. 43-54, 1998.

[2] D. Barbara and T. Imielinksi, "Sleepers and Workaholics: Caching Strategies for Mobile Environments," *Proc. ACM SIGMOD Conf. Management of Data,* pp. 1-12, May 1994.

[3] D. Barbara and T. Imielinksi, "Sleepers and Workaholics: Caching Strategies for Mobile Environments (Extended Version)," *VLDB J.,* vol. 4, no. 4, pp. 567-602, 1995.

[4] J. Cai and K.-L. Tan, "Energy-Efficient Selective Cache Invalidation," *Wireless Networks,* vol. 5, no. 6, pp. 489-502, 1999.

[5] G. Cao, "Proactive Power-Aware Cache Management for Mobile Computing Systems," *IEEE Trans. Computers,* vol. 51, no. 6, pp. 608-621, June 2002.

[6] G. Cao, "A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 5, Sept./Oct. 2003.

[7] B.Y.L. Chan, A. Si, and H.V. Leong, "Cache Management for Mobile Databases: Design and Evaluation," *Proc. 14th Int'l Conf. Data Eng. (ICDE '98),* pp. 54-63, Feb. 1998.

[8] C.C.F. Fong, J.C.S. Lui, and M.H. Wong, "Quantifying Complexity and Performance Gains of Distributed Caching in a Wireless Network Environment," *Proc. 13th Int'l Conf. Data Eng. (ICDE '97),* pp. 104-113, Oct. 1997.

[9] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West, "Scale and Performance in a Distributed File System," *ACM Trans. Computer Systems,* vol. 6, no. 1, pp. 51-58, Feb. 1988.

[10] Q.L. Hu and D.L. Lee, "Cache Algorithms Based on Adaptive Invalidation Reports for Mobile Environments," *Cluster Computing,* vol. 1, no. 1, pp. 39-48, Feb. 1998.

[11] J. Jing, A.K. Elmagarmid, A. Helal, and R. Alonso, "Bit-Sequences: A New Cache Invalidation Method in Mobile Environments," *Mobile Networks and Applications,* vol. 2, no. 2, pp. 115-127, 1997.

[12] A. Kahol, S. Khurana, S.K.S. Gupta, and P.K. Srimani, "A Strategy to Manage Cache Consistency in a Distributed Mobile Wireless Environment," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 7, pp. 686-700, July 2001.

[13] Y.-B. Lin, W.-R. Lai, and J.-J. Chen, "Effects of Cache Mechanism on Wireless Data Access," *IEEE Trans. Wireless Comm.,* vol. 2, no. 6, pp. 1240-1246, 2003.

[14] M. Nelson, B. Welch, and J. Ousterhout, "Caching in the Sprite Network File System," *ACM Trans. Computer Systems,* vol. 6, no. 1, pp. 134-154, Feb. 1988.

[15] J.T. Robinson and M.V. Devarakonda, "Data Cache Management Using Frequency-Based Replacement," *Proc. ACM Sigmetrics Conf.,* pp. 134-142, 1990.

[16] K.L. Tan, J. Cai, and B.C. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 8, pp. 789-807, Aug. 2001.

[17] WAP Forum, "Wireless Application Protocol Architecture Specification," technical report, WAP Forum, 2001.

[18] WAP Forum, "WAP Cache Operation Specification," technical report, WAP Forum, 2001.

[19] WAP Forum, "User Agent Profiling Specification," WAP Forum, 2001.

[20] K.-L. Wu, P.S. Yu, and M.-S. Chen, "Energy-Efficient Caching for Wireless Mobile Computing," *Proc. 12th Int'l Conf. Data Eng.,* pp. 336-343, Feb. 1996.

[21] J. Xu, Q. Hu, D.L. Lee, and W.-C. Lee, "SAIU: An Efficient Cache Replacement Policy for Wireless On-Demand Broadcasts," *Proc. Ninth ACM Int'l Conf. Information and Knowledge Management (CIKM '00),* pp. 46-53, Nov. 2000.

[22] J. Xu, Q. Hu, W.-C. Lee, and D.L. Lee, "Performance Evaluation of an Optimal Cache Replacement Policy for Wireless Data Dissemination," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 1, pp. 125-139, Jan. 2004.

[23] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Volume Leases for Consistency in Large-Scale Systems," *IEEE Trans. Knowledge and Data Eng.,* vol. 11, no. 4, pp. 563-576, Sept./Oct. 1999.

[24] J. Yuen, E. Chan, K.Y. Lam, and H.W. Leung, "Cache Invalidation Scheme for Mobile Computing Systems with Real-Time Data," *ACM SIGMOD Record,* vol. 29, no. 4, pp. 34-39, 2000.

Hui Chen received the MS degree in computer science from the University of Memphis, Tennessee, in 2003. He is currently a PhD candidate in computer science at the University of Memphis. His current research interests include the design and analysis of personal communication service networks, wireless LANs, and mobile distributed computer systems. He is a student member of the IEEE and the IEEE Computer Society.

Yang Xiao worked at Micro Linear as a MAC (Medium Access Control) architect involving the IEEE 802.11 standard enhancement work before he joined the Department of Computer Science at the University of Memphis in 2002. He is currently with the Department of Computer Science at the University of Alabama. He was a voting member of the IEEE 802.11 Working Group from 2001 to 2004. He currently serves as editor-in-chief for the *International Journal of Security and Networks* (IJSN) and for the *International Journal of Sensor Networks* (IJSNet). He serves as an associate editor or on the editorial board for the following refereed journals: the *International Journal of Communication Systems*, *Wireless Communications and Mobile Computing (WCMC)*, the *EURASIP Journal on Wireless Communications and Networking (WCN)*, and the *International Journal of Wireless and Mobile Computing (IJWMC)*. He has served as a guest editor for journal special issues of WCMC, WCN, IJSN, IJWMC, *Computer Communications*, and *IEEE Wireless Communications*. He serves as series coeditor for the book series on computer and network security published by World Scientific Publishing Co. He serves as a referee/reviewer for many funding agencies, in addition to being a panelist for the US National Science Foundation. He has edited or coedited 10 books, published 16 book chapters in the wireless networks and security fields, and published more than 140 papers in major journals and refereed conference proceedings related to these research areas. He received the Best Paper Award of the IEEE Consumer Communications and Networking Conference '06. He is a senior member of the IEEE, the IEEE Computer Society, the IEEE Communications Society, and the IEEE Vehicle Technology Society.

Xuemin (Sherman) Shen received the BSc (1982) degree from Dalian Maritime University, China, and the MSc (1987) and PhD (1990) degrees from Rutgers University, New Jersey, all in electrical engineering. From September 1990 to September 1993, he was first with Howard University, Washington D.C., and then with the University of Alberta, Edmonton, Canada. Since October 1993, he has been with the Department of Electrical and Computer Engineering, University of Waterloo, Canada, where he is a professor and the associate chair for graduate studies. Dr. Shen's research focuses on mobility and resource management in interconnected wireless/wired networks, UWB wireless communications systems, wireless security, and ad hoc and sensor networks. He is a coauthor of two books and has published more than 200 papers and book chapters in wireless communications and networks, control, and filtering. Dr. Shen was the technical program cochair for the IEEE Globecom '03 Symposium on Next Generation Networks and Internet, ISPAN '04, IEEE Broadnet '05, QShine '05, and is the special track chair of the 2005 IFIP Networking Conference. He serves as the associate editor for the *IEEE Transactions on Wireless Communications*, the *IEEE Transactions on Vehicular Technology*, *ACM/Wireless Networks*, *Computer Networks*, *Wireless Communications and Mobile Computing*, and the *International Journal of Computers and Applications*. He also serves as a guest editor for the *IEEE Journal on Selected Areas in Communications*, the *IEEE Transactions on Vehicular Technology*, *IEEE Wireless Communications*, and *IEEE Communications Magazine*. Dr. Shen received the Premier's Research Excellence Award (PREA) from the province of Ontario, Canada, for demonstrated excellence in scientific and academic contributions in 2003 and the Distinguished Performance Award from the faculty of engineering, University of Waterloo, for outstanding contributions in teaching, scholarship, and service in 2002. He is a registered professional engineer of Ontario, Canada. He is a senior member of the IEEE.