WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
Wirel. Commun. Mob. Comput. 2013; **13**:1337–1352
Published online 23 August 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/wcm.1182

RESEARCH ARTICLE

OUR: Optimal Update-based Replacement policy for cache in wireless data access networks with optimal effective hits and bandwidth requirements

Mursalin Akon, Mohammad Towhidul Islam, Xuemin (Sherman) Shen* and Ajit Singh

Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1

ABSTRACT

In mobile wireless data access networks, remote data access is expensive in terms of bandwidth consumption. An efficient caching scheme can reduce the amount of data transmission, hence, bandwidth consumption. However, an update event makes the associated cached data objects obsolete and useless for many applications. Data access frequency and update play a crucial role in deciding which data objects should be cached. Seemingly, frequently accessed but infrequently updated objects should have higher preference while preserving in the cache. Other objects should have lower preference or be evicted, or should not be cached at all, to accommodate higher-preference objects. In this paper, we proposed *Optimal Update-based Replacement*, a replacement or eviction scheme, for cache management in wireless data networks. To facilitate the replacement scheme, we also presented two enhanced cache access schemes, named *Update-based Poll-Each-Read* and *Update-based Call-Back*. The proposed cache management schemes were supported with strong theoretical analysis. Both analysis and extensive simulation results were given to demonstrate that the proposed schemes guarantee optimal amount of data transmission by increasing the number of effective hits and outperform the popular Least Frequently Used scheme in terms of both effective hits and communication cost. Copyright © 2011 John Wiley & Sons, Ltd.

KEYWORDS

wireless data access; cache; replacement scheme; access scheme; data update

***Correspondence**

Xuemin (Sherman) Shen, Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1.

E-mail: xshen@bbcr.uwaterloo.ca

1. INTRODUCTION

Extraordinary advances in computing electronics and wireless communications promise flexibility to our daily lives. Traditional cellular devices have been mostly used for voice communication, whereas digital organizer devices have been for storing details about personal contacts and schedules. In contrast, modern mobile devices, such as smartphones, personal digital assistants, and other handheld computers, and deployed high-bandwidth 3G (and expected 3.5G and 4G) cellular networks and wireless LANs create the platform for ubiquitous mobile computing. These technologies have made many necessary and entertaining mobile IP-based applications possible. Mobile IP telephony, mobile TV, video on demand, video conference, tele-medicine, instant messaging, mobile online banking, stock market tracking, online multiplayer games are examples of such applications. Online social networking, such as Facebook [1], Qzone [2], MySpace [3], and Twitter [4]; video sharing sites, such as YouTube [5] and Youku [6]; and image and file hosts, such as SkyDrive [7], Flickr [8], Picasa [9], Photobucket [10], and Dropbox [11] have changed the way people communicate and store their multimedia and other documents. These applications and services consume so much bandwidth, burdening the communication infrastructures, that the service providers have no choice but to look for alternate methodologies and user incentive mechanisms [12–14]. As prices of advanced mobile devices become more affordable and more users subscribe for wireless data access services, the problem of bandwidth is simply going to be worse.

In a mobile information retrieval system, databases and files are hosted at remote servers, conventionally connected on the wired networks. Each database or file server hosts a number of objects made available to be accessed by the mobile users. Obviously, whenever a mobile user accesses

a data object from the remote server, all the communications have to pass through the wireless network. Despite the advancements in wireless technologies, wireless bandwidth is the most scarce and the most expensive resource in a wireless data network. A client has to be very economic about the bandwidth consumption and make the best effort towards higher utilization. Many applications adaptively adjust the quality of service depending on the network state. For example, a mobile Internet browser may retrieve images whose quality is adjusted to the available bandwidth. However, developing such network-aware applications is not trivial [15]. Many applications can reduce bandwidth consumption by caching recurrently accessed objects locally. Note that, cache-oriented solutions are not orthogonal for developing network-aware applications, rather, in most cases, cache can be deployed irrespective of network awareness.

Caching contributes in three ways to improve the performance of the data access applications and the network systems. Firstly, the average access latency is reduced as many data objects are delivered from the local cache, instead of fetching from the remote server. Secondly, without cache at each access, an object has to be fetched from the server and the object has to be passed through the network from the server to the client; thus, caching reduces the network load. Reduced network load decreases the cost of data access, and it is a very important design goal for most of the wireless applications. Thirdly, as the server gets fewer request from a client, the server becomes more scalable without additional computing and network resources. An interesting side effect of employing cache is that, by cutting down the number of communication transmissions, caching not only salvage on expensive wireless access but also saves power and prolongs battery life.[†]

A cache mechanism needs to address several aspects, and two of these aspects are related to our problem domain—(1) a cache *access* and (2) a *replacement* scheme. A cache access scheme describes two important jobs of a cache mechanism—(1) how a client and a server utilize the cache and (2) how consistency between the original data items at the server and copies at the client caches is maintained. In a distributed environment, the second task is often very complicated and termed as cache *consistency* or *invalidation* scheme, if investigated from data consistency or data invalidation perspective, respectively.

For many applications, the existence of a more recent update renders all older copies of the data object invalid. These applications must have access to the most recent updated data. This kind of consistency is called the *latest value* consistency [17,18], and a cache, satisfying the latest value consistency, is said to be *strongly consistent* [17,19]. With the latest value consistency requirement, when a data object is updated, all the cached copy becomes obsolete and cannot be used to serve application requests. In this case, a client has to retrieve the data item from the server. Several strongly consistent cache consistency schemes for wireless data access have been proposed, such as *Invalidation Report (IR)* schemes [17,20–27], *Poll-Each-Read* [19], and *Call-Back* [19,26,28].

The other aspect of a cache mechanism, the replacement scheme, comes into play when the cache is full and an accessed object has to be accommodated. Here, one or more cached objects may have to be evicted to make room for the newly arrived object. Most research works have considered *Least Recently Used (LRU)* [17,19–22,25–27,29,30] or *Least Frequently Used (LFU)* [31] replacement schemes for wireless data access. Note that, a cache mechanism may perform differently with different combinations of access and replacement schemes; therefore, a system developer has to be prudent in choosing the appropriate replacement and access schemes.

In this paper, we study two strongly consistent cache access and an update-oriented replacement schemes for wireless data networks, spanned over more than one wireless cell. Our main contributions are of three folds: we introduce *Update-based Poll-Each-Read (UPER)* and *Update-based Call-Back (UCB)* access schemes, which are tailored to provide both access and update histories to the replacement schemes; we propose *Optimal Update-based Replacement* (OUR), a replacement scheme, based on the idea of giving preference to frequently accessed but infrequently updated data objects while storing objects in the cache; and we analytically prove that our replacement scheme, combined with the access schemes, ensures the worst-case optimal performance. As a result, this research provides the upper boundary for the worst-case performance of any caching scheme and a benchmark for average-case performance comparison. The design goals of the proposed scheme are to increase the effective hit ratio and to reduce transmission cost (i.e., bandwidth consumption) over the wireless cellular networks. Simulations are performed to validate our proposals and claims.

The remainder of the paper is organized as follows. We present our system model, related works, and performance metrics in Section 2. The UPER and the UCB cache access and our cache replacement schemes are introduced in Section 3. Quantitative analysis with an analytical model is provided in Section 4. Performance evaluations and comparisons are presented in Section 5. We conclude the paper in Section 6.

2. BACKGROUND

In this section, we discuss our system model followed by important related works. We fold up the section by describing the design goals and performance metrics.

[†]It is worth noting that transmission over wireless data network consumes a good amount of power [16]. However, reducing power consumption is not the main focus, but is an attractive by-product of this research.

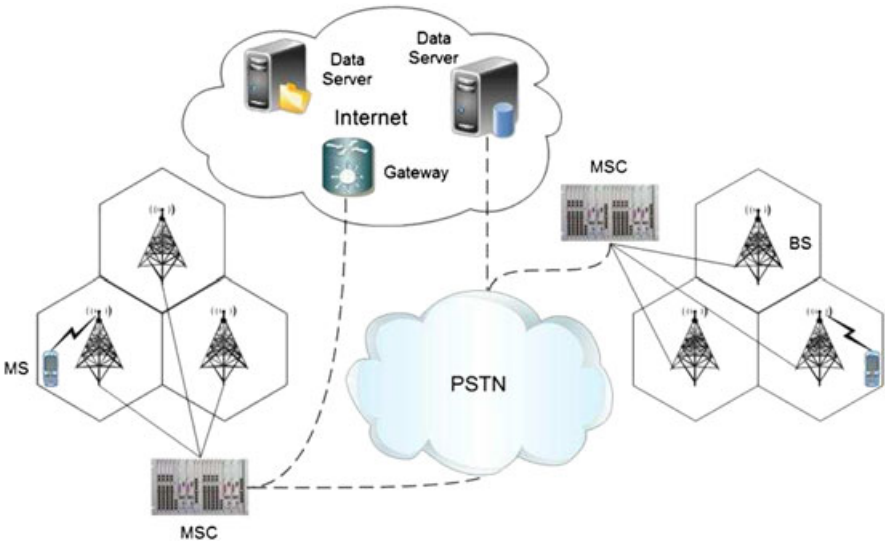


Figure 1. A wireless cellular network for data access. MSC, mobile switching center; MS, mobile station; BS, base station; PSTN, public switched telephone network.

2.1. System model

Our system model is based on the wireless data networks, already available in the consumer market [32]. In these networks, service areas are divided into a number of *location areas* (LAs). An LA is further partitioned into a number of *cells*. Each cell has a *base station* (BS). Many *mobile stations* (MSs) reside in a service area, and each of them connects to the closest BS. All the BSs within one LA are connected to a *mobile switching center* (MSC). All the MSCs are finally connected to the *public switched telephone network*. Such a network is connected to the Internet in different ways, such as through proprietary networks of the wireless carriers or the public switched telephone network. A mobile terminal accesses data servers residing either in the service provider's network or in the Internet through its associated BS. Figure 1 shows a wireless cellular network for data access. Typically, wireless links exist only between MSs and corresponding BSs, and in this wireless cellular network, MSs host client applications.[‡] A client on an MS may use a cache to improve its performance. Each client manages its own cache. However, the data servers also participate in improving the performance of the client caches. Note that, in our model, we keep the wireless cellular network infrastructures, that is, the network of BSs, MSCs, and gateways, unchanged. We are motivated to take this route because modifications to the wireless cellular network infrastructures are extremely expensive. Besides, if caches are implemented at the BSs or MSCs, whenever an MS switches from one BS to another or one MSC to another, the cache or cache-related data is also required to be moved, resulting in significant overhead. When each MS maintains its own cache, the performance of cache at an MS is not influenced by intracell or intercell or LA mobility. In this research, we consider that proper transmission schedules, channel condition tracking, error and flow control are performed by lower communication layers, and a client focuses on lowering *soft bandwidth*—the amount of data the client requests to and from the lower layers to transmit and receive, respectively.

2.2. Related works

Significant works have been done on caching in wired environment. As bandwidth in wired environment is abandoned, many researches concentrate on faster data retrieval by using cache rather than focusing on bandwidth consumption. In [33,34], a scheme to prefetch and cache streaming multimedia segments to be played in the near future are proposed. The application behavior in streaming multimedia is mostly deterministic, where data is accessed in sequence. In [35], alternates to distributed hash table are proposed to cache indices among a group of peers in a distributed manner. These techniques enable fast and efficient query processing in structured peer-to-peer networks. However, in this application, cache buffer is considered to be ample, and cache overflow is not considered to happen. In [36,37], a collaborative scheme is presented to share file/disk caches among network workstations. These works assume availability of direct unicast and broadcast communications among cache entities. However, such features are not available in our system model; hence, the presented collaborations are not directly applicable. In [38], a cache management scheme for wireless networks is proposed.

[‡]In the rest of the paper, we use the terms MS, client (application), and user interchangeably.

This scheme performs better or worse than LFU depending on access frequencies and patterns. Moreover, this scheme is not supported by any analytical model.

In wireless data access, update process plays an important role because it makes the locally cached data objects invalid. Thus, it is beneficial for a replacement scheme to utilize update and access information together. The studies in [19] investigates LRU as the replacement scheme without considering the effects of the update process. Although other researches [39,40] have proposed to utilize update information in replacement schemes, the main concentration was to reduce the *stretch*[§] [41] of the IR. It is important to understand that IR schemes perform well under the following assumptions: (1) the channel is a broadcast channel, (2) all the clients are in one wireless cell, and (3) the server is local to the wireless cell, that is, located at the BS. For example, if clients are in different wireless cells, the IR schemes become very expensive, because IR reports include too much information that is irrelevant to clients in different wireless cells [19]. In a practical wireless cellular network, not all of the mentioned assumptions are satisfied. In other words, the server is more likely to be at a remote site, subscribed clients are scattered all over the wireless cellular network or even in remote networks, and the entire network is not covered by a single broadcast channel. Furthermore, a suitable implementation of an IR scheme requires cross layer support for efficiency; hence, IR schemes are not realistic in practical wireless cellular networks. LRU [17,20–22], LFU [31], and Most Recently Used [42] cache replacement schemes are extensively studied particularly in the domain of operating systems, databases, and Web caches. It has been shown that for online data objects, LFU provides superior performance [43].

2.3. Performance metrics

The goals of our replacement scheme is to increase the effective hit ratio and to reduce the communication cost. When an access takes place, either a *cache hit* or a *cache miss* happens. A cache miss is an event where the accessed data object is not in cache. A cache hit is an event where a cached data item is accessed. Cache hit can be further classified into *valid cache hit* and *invalid cache hit*. Given that the cached object is the most recent version, that is, the object has not been updated because it has been cached, a valid cache hit takes place. Otherwise, the access results in an invalid cache hit. *Effective hit ratio* is the ratio of a valid cache hit over all accesses. While measuring *cost*, we focus on bandwidth consumption—the amount of data transmitted to and from an MS to satisfy data requests from the applications.

3. PROPOSED SCHEME

In this section, we first present the notations and primitives used in this paper. Then, we introduce the UPER and UCB cache access schemes. We describe the proposed update-oriented replacement scheme at the end of this section.

3.1. Notations

Before describing the cache scheme, we introduce several key terms, used in the following discussions.[¶] Let the number of distinct and equal size objects hosted by the server be N. The hosted objects are identified as O_i, where $i = 1, \ldots, N$. In case the objects are not of the same size, we consider that they are broken down into fixed-size smaller pieces.[‖] In our problem domain, object updates happen at the server only. Let the maximum number of objects an MS can locally cache be K.

Let $\mu_{i,j}(t)$ be the access frequency of data item j at client i and $\lambda_j(t)$ be the update frequency of data item j at the server up to time t. Note that $\mu_{i,j}(t)$ is maintained for each client, and $\lambda_j(t)$ is an aggregated measurement and does not involve any client. We denote $\mu_{i,j}$ and λ_j as the expected access rate at client i and the expected update rate at the server for data item j, respectively. When t is sufficiently large, $\mu_{i,j}(t)$ approaches to access rate $\mu_{i,j}$ and $\lambda_j(t)$ to λ_j. Formally, $\mu_{i,j} = \lim_{t\to\infty} \mu_{i,j}(t)$ and $\lambda_j = \lim_{t\to\infty} \lambda_j(t)$. We denote $\mu = \sum_{i=1}^{N} \mu_i$ as total access rate. Similarly, we denote $\lambda = \sum_{i=1}^{N} \lambda_i$ as the total update rate.

3.2. Primitives

The proposed UPER and an UCB cache access schemes are spawned from the Poll-Each-Read and Call-Back access schemes, respectively, to integrate both update and access information when the cache is accessed or updated. Subsequently, our replacement scheme employs the update and access information to ensure the optimum performance. To describe our access schemes, we use the following primitives:

add(data): Add *data* object to the local cache.
evict(id): Evict the local object with identification—*id*.
replace(data): Overwrite local older copy of *data* object with a newer one.
modify(profile): Depending on schemes in place, access-related and update-related profiles per objects and per client are maintained. Each profile keeps access and/or update frequencies of the hosted objects and may

[§] Stretch is defined as the ratio of average response time to service time, where service time is the response time as if there exists no other job in the system.

[¶] We continue to use the same notations introduced in previous research publications.

[‖] In literature, this idea is often reflected through *depth* of a cache block.

contain information about availability of objects at different clients. This primitive modifies the given *profile* with the most recent information. A modification updates a small part, often related to a single object or user, of the entire profile.

find_id(replacement_scheme): Find the identification of the object to evict, using the given replacement scheme, identified by *replacement_scheme*.

3.3. Working principle of the access schemes

3.3.1. Update-based Poll-Each-Read scheme.

Different steps of UPER access scheme are shown in Figures 2 and 3. In this scheme, the server maintains access profile for each object per client, as well as update profile for each object. Whenever an access or update takes place, the server updates the corresponding profile. Thus, the server becomes the most knowledgeable entity in the system. If a locally cached data object is requested, the MS sends a request to the server to verify the validity of the cached object. If the cached copy is the most updated one, the server simply replies back with an acknowledgement (Figure 2(a)). Otherwise, the server forwards the most recently updated copy (Figure 2(b)). Contrary to the previous two scenarios, if the request is for an object that does not exist in the local cache, the mobile terminal sends a request for the object. If the mobile terminal has a space to accommodate a new object (i.e., cache is not full), the server simply forwards a copy (Figure 3(a)). Otherwise, the server makes a decision about a replacement and forwards the decision with a copy of the requested object (Figure 3(b)). In this scheme, when the server receives an update request, it replaces the object with the most recent copy and adjusts the update profile for that object.

3.3.2. Update-based Call-Back scheme.

In UCB, beside the server, each MS also maintains access and updates information for each cached object. Although the server continues maintaining both the global profiles, often, they do not reflect the most recent information. Some of the steps in this scheme is shown in Figure 4. In this scheme, when an MS receives a request for an object and finds that a copy locally exists, the object is served immediately without consulting the server. Additionally, the locally maintained access profile for that object is updated. On the other hand, when the requested object is not cached locally, the MS forwards the request for the missing object to the server. At the reception of such a message, the server replies back with the requested object with its access and update information. Now, if the MS has a space to accommodate the new object in the cache, the object and its profiles are simply added (Figure 4(a)). Otherwise, an eviction decision is made. The decision

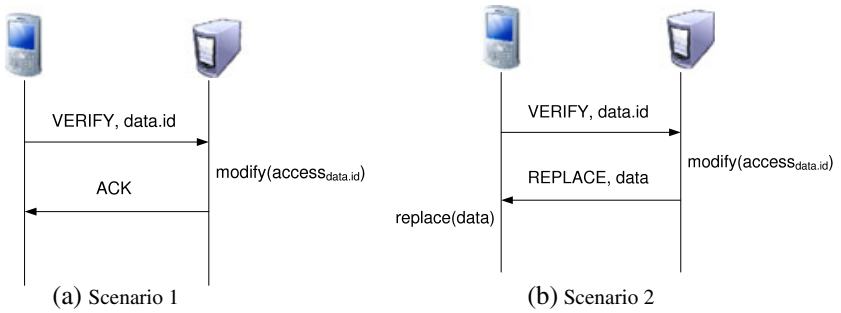


Figure 2. Steps for Update-based Poll-Each-Read access scheme in different scenarios when requested object is locally cached: (a) Scenario 1 and (b) Scenario 2.

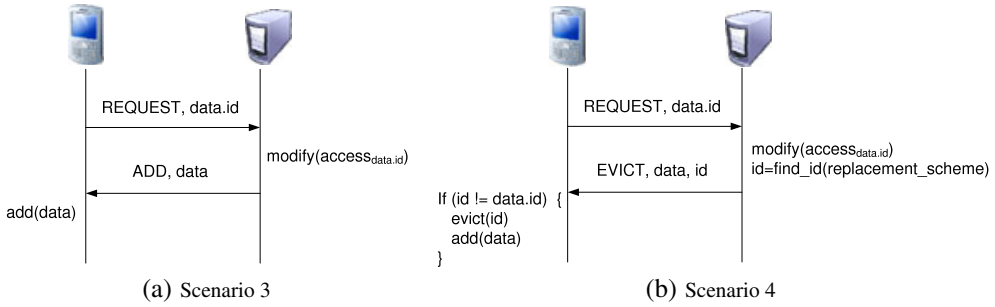


Figure 3. Steps for Update-based Poll-Each-Read access scheme in different scenarios when requested object is not locally cached: (a) Scenario 3 and (b) Scenario 4.

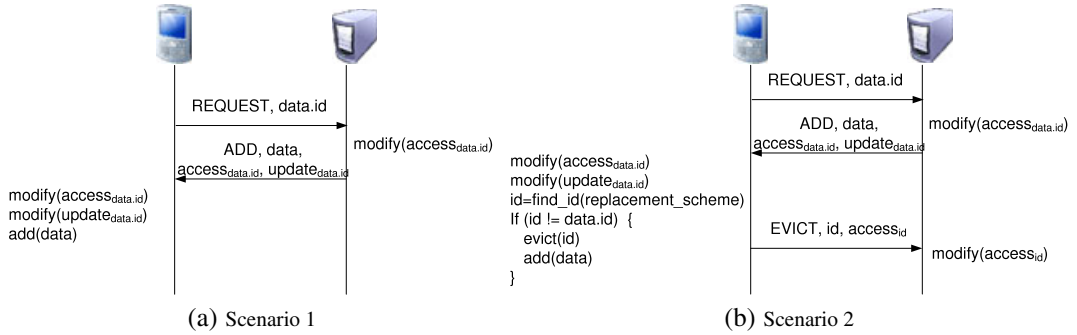


Figure 4. Steps for Update-based Call-Back access scheme in different scenarios at a cache miss: (a) Scenario 1 and (b) Scenario 2.

is forwarded to the server with the access profile of the to-be-evicted object so that the server adjusts the central profile accordingly. The MS continues to discard the evicted object and its related profiles (Figure 4(b)).

3.4. Optimal Update-based Replacement scheme

In this subsection, we present our cache replacement scheme. We name it Optimal Update-based Replacement (OUR) scheme, which uses both update and access frequencies to achieve superior-guaranteed performance. We define the *performance factor (PF)* from the perspective of client i for object j up to time t as follows:

$$PF_{i,j}(t) = \frac{\mu_{i,j}(t)^2}{\mu_{i,j}(t) + \lambda_j(t)} \tag{1}$$

Similarly, long term PF is defined as

$$PF_{i,j} = \lim_{t\to\infty} \frac{\mu_{i,j}(t)^2}{\mu_{i,j}(t) + \lambda_j(t)} = \frac{\mu_{i,j}^2}{\mu_{i,j} + \lambda_j} \tag{2}$$

Detail semantics about PF will be discussed in the next section. OUR scheme works by computing the PF for all the objects using the access and update histories collected through the cache access scheme. The PF of the most recently accessed object is simultaneously computed. If the new object has a higher PF than that of any cached one, the object with the lowest PF is evicted, and the new object is stored in the cache. Otherwise, the most recently accessed object is not cached, and the cache is kept intact without any change. Algorithm 1 presents the replacement scheme using the notations introduced in Section 4.

Algorithm 1: Replacement scheme at client i where the most recently accessed object is O_a

```
// Cached object with the lowest PF
MinPF ← +∞
foreach O_j ∈ (C ∪ {O_a}) do
    PF_{i,j} ← μ²_{i,j} / (μ_{i,j} + λ_j)
    if MinPF > PF_{i,j} then
        MinPF ← PF_{i,j}
        e ← j
// The new object has better PF
if e ≠ a then
    // So, keep the fetched object
    C ← (C\{O_e}) ∪ {O_a}
// Otherwise, leave the cache
   unchanged
```

4. QUANTITATIVE ANALYSIS

In this section, we analytically prove that OUR ensures the worst-case optimal cache performance in terms of cache hits and communication cost.

We number all replacements in the order they take place in time, for example, the earliest replacement be numbered as the first replacement, the next one as the second replacement, and so on. We use $C(t)$ and $C(t)'$ to denote the set consisting of all cached objects before and after the tth replacement. For our analysis, we define two performance metrics—*effective hits* and *cost*. Note that, effective hits and cost are computed considering all accesses over a long time.

We denote the probability of guaranteed effective hits after the tth replacement of UPER and UCB as $P_{\text{UPER}}(t)$ and $P_{\text{UCB}}(t)$, respectively, under any replacement scheme. The costs of UPER and UCB resulting from the tth replacement are denoted as $C_{\text{UPER}}(t)$ and $C_{\text{UCB}}(t)$, respectively, under any replacement scheme. Similarly, we denote $P_{\text{UPER+OUR}}(t)$, $P_{\text{UCB+OUR}}(t)$, $C_{\text{UPER+OUR}}(t)$, and $C_{\text{UCB+OUR}}(t)$ as the corresponding metrics at the tth replacement when OUR replacement scheme is exercised. Let $O_{\text{a}}(t)$ and $O_{\text{e}}(t)$ denote the accessed and replaced objects at the tth replacement, respectively. Let $O_{\text{OUR}}(t)$ denote the replaced object when OUR scheme is used. Notations C_{req}, C_{ack}, and C_{obj} are used to denote the transmission cost of a request (or verification) message, an

acknowledge message,, and an object, respectively. Here, we omit the costs of transmitting replacement decisions or cost of piggy backing profile information of an object, as they incur very negligible overheads. Based on the behavior of OUR scheme, we have the following theorem.

Theorem 1. *OUR scheme maximizes the probability of guaranteed effective hits at each replacement for both UPER and UCB access schemes.*

Proof. A replacement happens when the cache is full and a new object is fetched by the cache manager. At such an event, an object not available in the cache is introduced. The replacement scheme makes the final decision about preserving the object in the local cache. Our goal is to show that the probability of guaranteed effective hits after tth replacement obtained by replacing $O_{\mathrm{OUR}}(t)$ with $O_{\mathrm{a}}(t)$ is larger than or equal to the probability obtained by replacing $O_{\mathrm{e}}(t)$ with $O_{\mathrm{a}}(t)$. At the tth replacement, the following conditions must hold:

$$O_{\mathrm{a}}(t) \notin C(t) \tag{3}$$

$$O_{\mathrm{e}}(t) \in C(t) \cup \{O_{\mathrm{a}}(t)\} \tag{4}$$

$$C(t)' = (C(t) \cup \{O_{\mathrm{a}}(t)\}) \setminus \{O_{\mathrm{e}}(t)\} \tag{5}$$

Similarly, for OUR scheme,

$$O_{\mathrm{OUR}}(t) \in C(t) \cup \{O_{\mathrm{a}}(t)\} \tag{6}$$

$$C(t)' = (C(t) \cup \{O_{\mathrm{a}}(t)\}) \setminus \{O_{\mathrm{OUR}}(t)\} \tag{7}$$

At any access, the probability of the accessed object being O_i, $1 \leqslant i \leqslant N$, is

$$p_{\mathrm{a},i} = \frac{\mu_i}{\mu} \tag{8}$$

The probability of O_i being accessed before an update takes place is

$$\overline{p_{\mathrm{u},i}} = \frac{\mu_i}{\mu_i + \lambda_i} \tag{9}$$

Accesses to different data items are independent, that is, they are disjoint events. Therefore, from Equation (5) to Equation (9), no matter what replacement scheme is used, we have

$$\begin{aligned} P_{\mathrm{UPER}}(t) &= P_{\mathrm{UCB}}(t) \\ &= \sum_{\forall i \mid O_i \in C(t)'} p_{\mathrm{a},i}\, \overline{p_{\mathrm{u},i}} \\ &= \sum_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\} \setminus \{O_{\mathrm{e}}(t)\}} p_{\mathrm{a},i}\, \overline{p_{\mathrm{u},i}} \\ &= \sum_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\} \setminus \{O_{\mathrm{e}}(t)\}} \frac{\mu_i}{\mu} \frac{\mu_i}{\mu_i + \lambda_i} \\ &= \sum_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\} \setminus \{O_{\mathrm{e}}(t)\}} \frac{\mu_i^2}{\mu(\mu_i + \lambda_i)} \\ &= \sum_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\}} \frac{\mu_i^2}{\mu(\mu_i + \lambda_i)} - \frac{\mu_{\mathrm{e}}^2}{\mu(\mu_{\mathrm{e}} + \lambda_{\mathrm{e}})} \end{aligned} \tag{10}$$

So, the equality when the OUR is exercised can be driven as follow:

$$\begin{aligned} P_{\mathrm{UPER+OUR}}(t) &= P_{\mathrm{UCB+OUR}}(t) \\ &= \sum_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\}} \frac{\mu_i^2}{\mu(\mu_i + \lambda_i)} \\ &\quad - \frac{\mu_{\mathrm{OUR}}^2}{\mu}(\mu_{\mathrm{OUR}} + \lambda_{\mathrm{OUR}}) \end{aligned} \tag{11}$$

According to the definition of the OUR scheme, we have

$$\begin{aligned} \frac{\mu_{\mathrm{OUR}}^2}{\mu_{\mathrm{OUR}} + \lambda_{\mathrm{OUR}}} &\equiv \min_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\}} \frac{\mu_i^2}{\mu_i + \lambda_i} \\ &\leqslant \frac{\mu_{\mathrm{e}}^2}{\mu_i + \lambda_{\mathrm{e}}} \end{aligned} \tag{12}$$

Therefore, we have

$$\begin{aligned} P_{\mathrm{UPER+OUR}}(t) &= P_{\mathrm{UCB+OUR}}(t) \\ &= \sum_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\}} \frac{\mu_i^2}{\mu(\mu_i + \lambda_i)} \\ &\quad - \min_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\}} \frac{\mu_i^2}{\mu(\mu_i + \lambda_i)} \\ &\geqslant \sum_{\forall i \mid O_i \in C(t) \cup \{O_{\mathrm{a}}(t)\}} \frac{\mu_i^2}{\mu(\mu_i + \lambda_i)} \\ &\quad - \frac{\mu_{\mathrm{e}}^2}{\mu(\mu_{\mathrm{e}} + \lambda_{\mathrm{e}})} \\ &= P_{\mathrm{UPER}}(t) = P_{\mathrm{UCB}}(t) \end{aligned} \tag{13}$$

From Equation (13), at each replacement, the probability of effective hits obtained by using OUR is larger than or equal to the same metric by using any other replacement scheme. It is concluded that the eviction choice made by OUR ensures optimal worst-case effective hits at each replacement. □

Corollary 1. *OUR minimizes the cost of data access at each replacement for both the UPER and the UCB access schemes.*

Proof. When the UPER cache access scheme is exercised and an access causes a cache hit, the client and the server only exchange a request and an acknowledgement message (Figure 2(a)). If the access causes a cache miss or an invalid cache hit, the client sends the server a request message, and the server replies to the client with the data object

(Figures 2(b) and 3(a)) and the replacement decision if the cache is full (Figure 3(b)). Therefore, for UPER,

$$\begin{aligned} C_{\mathrm{UPER}}(t) &= P_{\mathrm{UPER}}(t)(C_{\mathrm{req}} + C_{\mathrm{ack}}) \\ &\quad + (1 - P_{\mathrm{UPER}}(t))(C_{\mathrm{req}} + C_{\mathrm{obj}}) \\ &= C_{\mathrm{req}} + C_{\mathrm{obj}} + P_{\mathrm{UPER}}(t)(C_{\mathrm{ack}} - C_{\mathrm{obj}}) \end{aligned}$$

For OUR, the previous equality is expressed as follows:

$$\begin{aligned} C_{\mathrm{UPER+OUR}}(t) &= C_{\mathrm{req}} + C_{\mathrm{obj}} \\ &\quad + P_{\mathrm{UPER+OUR}}(t)(C_{\mathrm{ack}} - C_{\mathrm{obj}}) \end{aligned}$$

Without loss of generality, the cost of transmitting an acknowledgment is much smaller than the cost of transmitting an entire object. So, $C_{\mathrm{ack}} \ll C_{\mathrm{obj}}$, that is, $C_{\mathrm{ack}} - C_{\mathrm{obj}} \ll 0$. It has been indicated in Theorem 1 that $P_{\mathrm{UPER}}(t)$ is maximized while using OUR. Therefore, $C_{\mathrm{UPER}}(t)$ is minimized while the OUR scheme is used.

In UCB scheme, there is no message exchange when the access causes an effective cache hit. Thus, the cost resulting from the replacement at the tth access is

$$C_{\mathrm{UCB}}(t) = (1 - P_{\mathrm{UCB}}(t))(C_{\mathrm{req}} + C_{\mathrm{obj}}) \qquad (14)$$

and for OUR replacement,

$$C_{\mathrm{UCB+OUR}}(t) = (1 - P_{\mathrm{UCB+OUR}}(t))(C_{\mathrm{req}} + C_{\mathrm{obj}}) \qquad (15)$$

Again, $C_{\mathrm{UCB}}(t)$ is minimized while OUR is used because $P_{\mathrm{UCB+OUR}}(t) \geq P_{\mathrm{UCB}}(t)$. □

With these properties, we have the following theorem:

Theorem 2. *OUR scheme gives long-term optimal-guaranteed effective cache hits for both UPER and UCB access schemes.*

Proof. Suppose another scheme, called *OPT*, make a different decision than OUR at time t_1 as shown in Figure 5. At this point, object O_{a_1} is accessed, OUR replaces object O_{OUR_1}, and OPT replaces object O_{OPT_1}. By definition, $PF_{\mathrm{OUR}_1} < PF_{\mathrm{OPT}_1}$. From Theorem 1, it is evident that OUR performs better than OPT scheme before any further replacement takes place. However, at some later time t_2, when O_{a_2} is accessed, OPT makes another replacement where object O_{OPT_2} is replaced to accommodate the accessed object. To prove the theorem, we assume that because of this replacement decision, OPT outperforms OUR scheme. Now, we prove the theorem by contradiction.

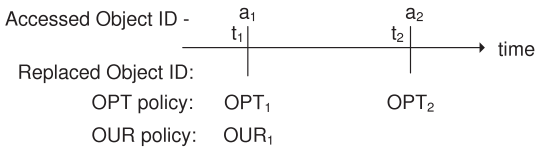


Figure 5. Replacement in Optimal Update-based Replacement (OUR) and other schemes.

As OUR outperforms OPT between the period t_1 and t_2, to outperform OUR in the long run, OPT has to have a better performance than OUR through the replacement at t_2. From Equation (10), we know that objects with higher PF results in higher effective hit rate. As a result, PF_{a_2} has to be higher than PF_{OPT_2}. Denote the content of the cache before t_1 be C. Of course, $O_{\mathrm{OPT}_2} \in \left(C \backslash \{O_{\mathrm{OPT}_1}\}\right) \cup \{O_{\mathrm{a}_1}\}$. If $O_{\mathrm{OPT}_2} \in C \backslash \{O_{\mathrm{OPT}_1}\}$, OUR scheme would have also replaced an object with the lowest PF from the cache, but that contradicts with our initial assumption. Then, O_{OPT_2} and O_{a_1} have to be the the same, resulting in $PF_{\mathrm{a}_2} > PF_{\mathrm{a}_1}$. Again, OUR scheme would have replaced O_{a_1} or another object with lower PF to accommodate O_{a_2}, which contradicts our initial assumption.

Hence, the OPT scheme cannot exist. Using the same argument, it can be shown that there exists no sequence of replacements, which results in a higher guaranteed effective hit rate than OUR. □

5. PERFORMANCE EVALUATION

We have developed an extensive discrete event simulator in C++ to evaluate the performance of the proposed OUR replacement scheme. We compare OUR with the LFU replacement scheme [44] because of the reasons described in Section 2. Our simulation setup is described in Subsection 5.1. We discuss the results from the simulations in the rest of this section. We also present results from theoretical analysis wherever applicable.

5.1. Simulation setup

For data access applications, different objects have different popularity. It has been observed that the access to different online objects follow *Zipf-like* distributions [19,44]. In our simulation, at an access (or update), object O_i is accessed (or updated) with the probability p_i, defined as

$$p_i = \left[i^{\alpha} \left(\sum_{j=1}^{N} \frac{1}{j^{\alpha}} \right) \right]^{-1}$$

where $\alpha \geq 0$ and is called the *Zipf ratio*. Note that, when $\alpha = 0$, $p_i = 1/N$, for all i, that is, all objects are chosen with the same probability of $1/N$. Let the Zipf ratio for access and update events be α_{a} and α_{u}, respectively. In our simulations, an object is ranked uniquely within the range from 1 to N to find its access and update probability. Notice that item O_i may have two distinct ranks to represent its access, and update probability and both of them may be different than the object ID, i.

Let $n_{\mathrm{a},j}$ and $n_{\mathrm{u},j}$ be the total number of accesses for all the clients and the total number of updates at the server

to object j, respectively. Let n_{miss} denote the total number of cache misses and invalid cache hits. Let n_{a} be the total number of accesses to all the objects from all the clients. For our simulations, we compute the effective hit ratios as

$$P_{\text{UPER}} = 1 - \frac{n_{\text{miss}}}{n_{\text{a}}} = P_{\text{UCB}} \tag{16}$$

where $n_{\text{a}} = \sum_{j=1}^{N} n_{\text{a},j}$ and then deduce effective hits as

$$\begin{aligned} EH_{\text{UPER}} &= P_{\text{UPER}} \times \mu \times T \\ &= P_{\text{UCB}} \times \mu \times T = EH_{\text{UCB}} \end{aligned} \tag{17}$$

where T is the simulation time. Finally, the computation of costs for UPER and UCB are shown in Equations (18) and (19), respectively.

$$\begin{aligned} C_{\text{UPER}} = \frac{1}{n_{\text{a}}}[&(n_{\text{a}} - n_{\text{miss}}) \times (C_{\text{req}} + C_{\text{ack}}) \\ &+ n_{\text{miss}} \times (C_{\text{req}} + C_{\text{obj}})] \end{aligned} \tag{18}$$

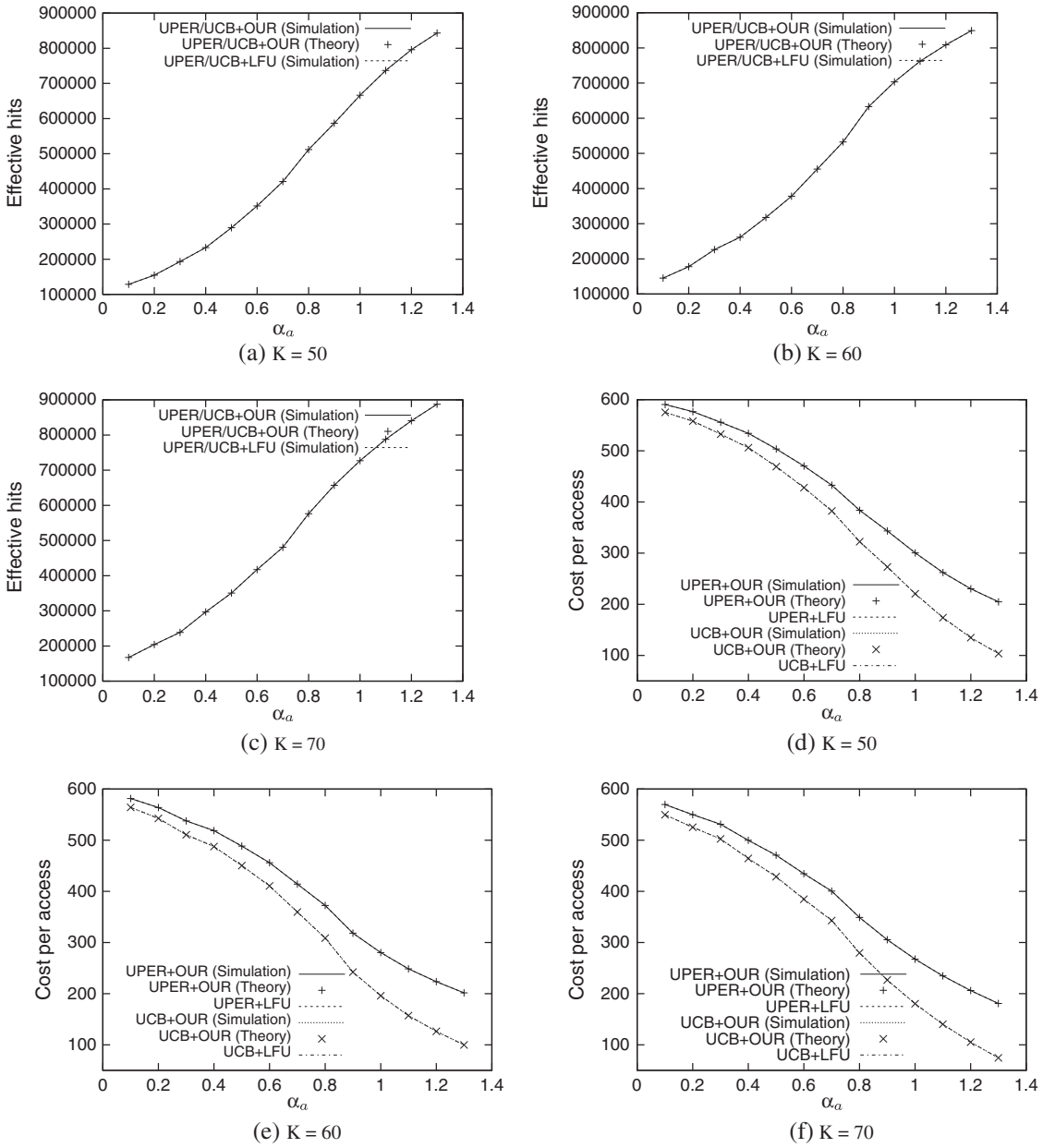


Figure 6. Performance of Optimal Update-based Replacement (OUR) and Least Frequently Used (LFU) for objects with the same update frequencies: (a) $K = 50$, (b) $K = 60$, (c) $K = 70$, (d) $K = 50$, (e) $K = 60$, and (f) $K = 70$. UPER, Update-based Poll-Each-Read; UCB, Update-based Call-Back.

$$C_{\mathrm{UCB}} = \frac{1}{n_{\mathrm{a}}}[n_{\mathrm{miss}}(C_{\mathrm{req}} + C_{\mathrm{obj}})] \tag{19}$$

Unless mentioned otherwise, we consider C_{req} and C_{ack} to have the same default value of C_{msg}, where $C_{\mathrm{msg}} = 60$. The default values for C_{obj} and μ are 600 and 1, respectively. It is worth to mention that all the objects in this experiment have equal sizes. Variable object sizes with different cache replacement policies are out of the scope of this paper. The default number of MSs in the network is 20.

5.2. Objects with no updates

Figure 6 shows the effective hit ratio and cost for OUR and LFU scheme when no update to any object takes place, that is, λ_j for all j is 0. In all the simulations, the server is populated with 500 objects. In this case, $OUR_{i,j}$ is determined by $\mu_{i,j}$, and consequently, while accommodating new objects in the cache, both OUR and LFU choose the same objects for eviction. Thus, both the schemes result in the same performance. In addition, we have the following observations:

- All combinations of access and replacement schemes enjoy higher effective hit ratio and lower cost when the objects are for read only and no update takes place (i.e., $\lambda = 0$). Static databases, audio, and video files sharing in wireless environment are examples of this kind of application.
- LFU performance is also optimal where no update takes place in the caching system.
- With increment of α_{a}, hit ratio also increases. The higher α_{a} is, the smaller set of objects is accessed more frequently, resulting in fewer misses.
- Larger cache helps in alleviating cache misses. However, this behavior is clearly visible when α_{a} is smaller. With larger α_{a}, most of the access are due to fewer objects; hence, increasing cache size results in very little additional benefit.

5.3. Impact of number of objects and cache size

In this subsection, we present the effect of the number of objects (N) and cache size (K) on cache performance. In Figure 7, the performance of UPER/UCB+OUR is compared with UPER/UCB+LFU for different object set sizes. The values for the parameters α_{a}, α_{u}, and K in these simulations are 0.20, 0.60, and 20, respectively. We have the following observations:

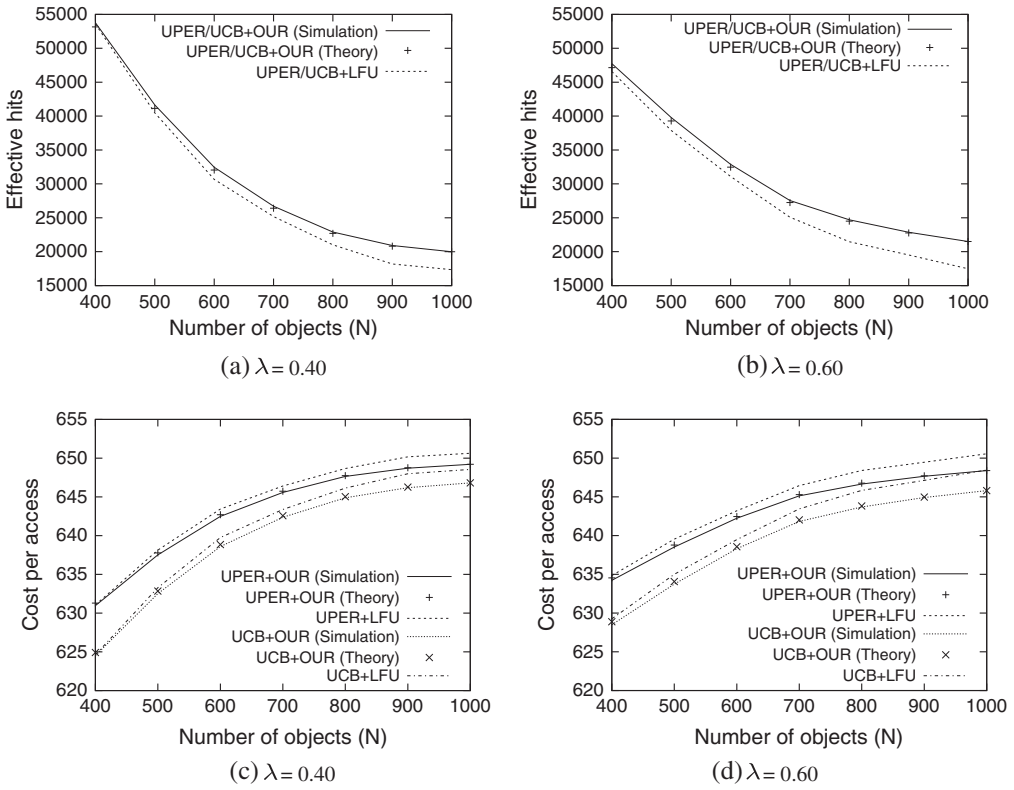


Figure 7. Performance of Optimal Update-based Replacement (OUR) and Least Frequently Used (LFU) for objects with different number of objects: (a) $\lambda = 0.40$, (b) $\lambda = 0.60$, (c) $\lambda = 0.40$, and (d) $\lambda = 0.60$. UPER, Update-based Poll-Each-Read; UCB, Update-based Call-Back.

- Effective hits for all combinations of access and replacement schemes decrease with the increment of database size. However, in all cases, OUR replacement scheme shows better results.
- Given a fixed size cache, as the object set size increases, the chance of hit reduces, irrespective of the replacement scheme. However, the gain of using OUR becomes prominent with larger set sizes.
- UCB+OUR scheme gives the best performance in terms of cost per access, and UCB+LFU closely follows. Provided that $C_{obj} \gg C_{msg}$, according to Equation (18), the difference between UCB+OUR and UCB+LFU is dominated by n_{miss}. Hence, with smaller miss rate, the gap between OUR and LFU scheme dilutes.
- With any of the replacement schemes, UCB access scheme cost less than UPER. However, OUR scheme reduces cost per access further as compared with LFU scheme.

Figure 8 shows different performance characteristics for different cache sizes. For these simulations, the values for the parameters α_a, α_u, and N are chosen as 0.01, 0.60, and 400, respectively. From the figure, we deduce the following arguments:

- In oppose to the previous case, as cache size increases, the number of effective hits also increases for all combinations of schemes, and OUR performs better in all cases.
- Because of increasing number of hits, the cost per access also declines with larger cache for both the schemes. However, the cost of UCB is reduced at a higher rate than that of UPER.
- As cache size increases, each additional extra cache buffer adds fewer additional effective hits. Hence, a designer must compromise between the cost of adding extra cache buffer and cache performance.

5.4. Impact of Zipf ratio

Figure 9 shows the effect of update Zipf ratio on the performance of both OUR and LFU schemes with the simulation parameters α_a, K, and N to be 0.01, 50, and 500, respectively. It can be seen that

- with different Zipf ratios, the performance of OUR is consistently better than that of LFU in terms of both number of hits and cost per access; and
- the overall effective hits fall and the cost increases as update Zipf ratios increases.

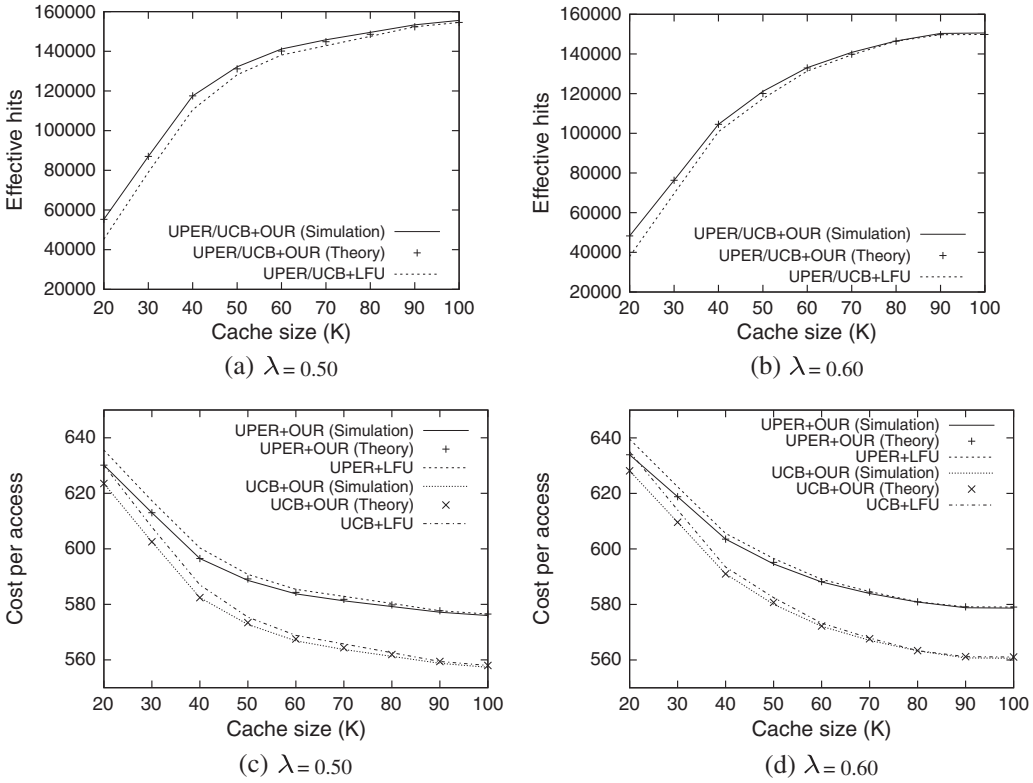


Figure 8. Performance of Optimal Update-based Replacement (OUR) and Least Frequently Used (LFU) for objects with different cache sizes: (a) $\lambda = 0.50$, (b) $\lambda = 0.60$, (c) $\lambda = 0.50$, and (d) $\lambda = 0.60$. UPER, Update-based Poll-Each-Read; UCB, Update-based Call-Back.

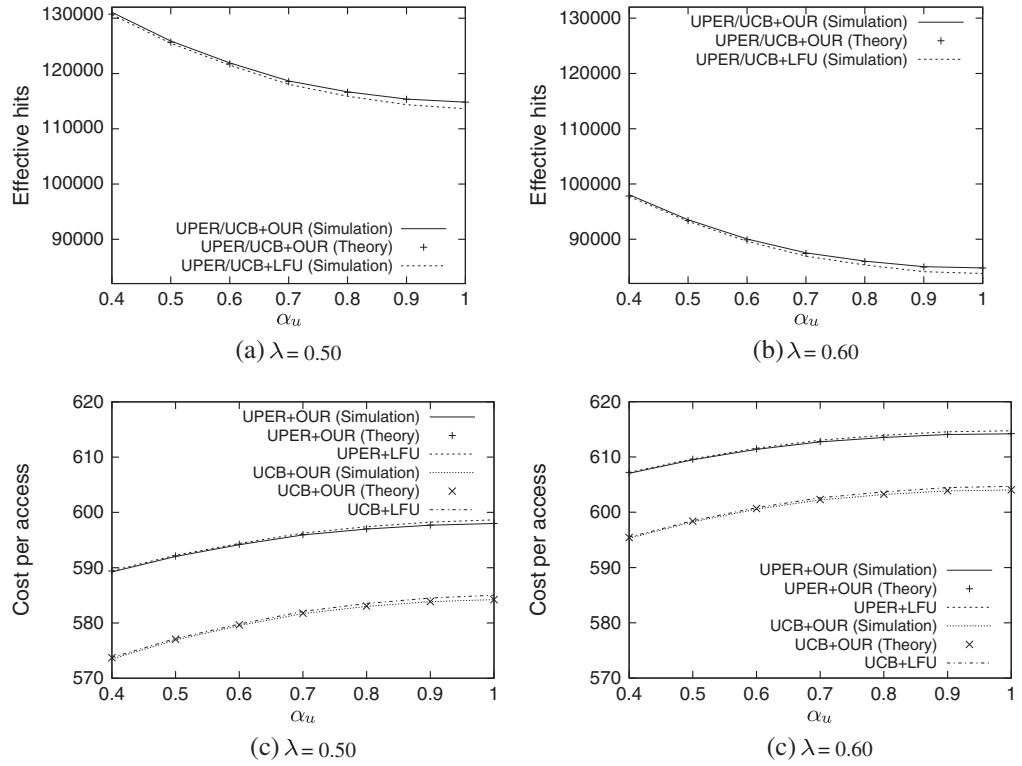
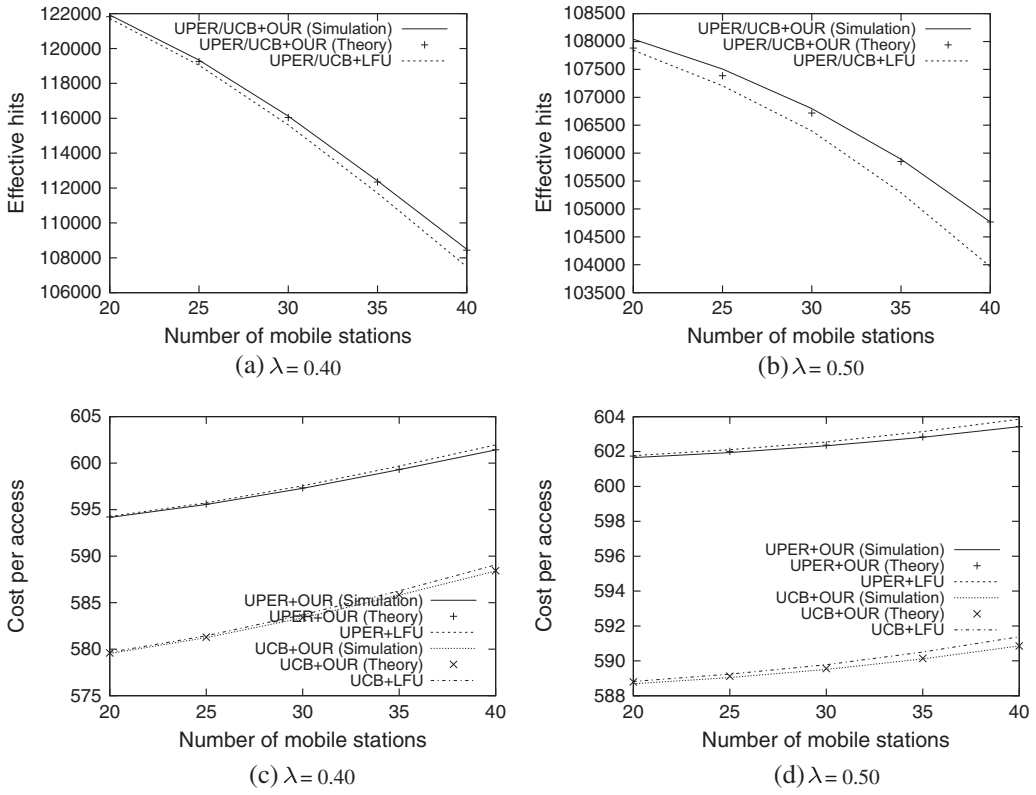
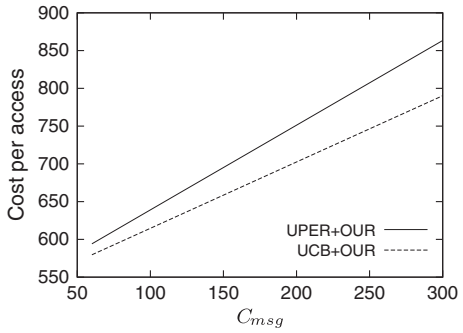
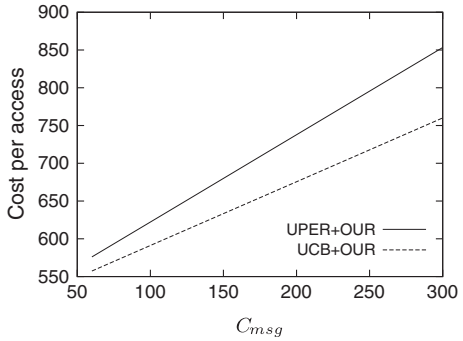


Figure 9. Performance of Optimal Update-based Replacement (OUR) and Least Frequently Used (LFU) for different Zipf ratios: (a) $\lambda = 0.50$, (b) $\lambda = 0.60$, (c) $\lambda = 0.50$, and (d) $\lambda = 0.60$. UPER, Update-based Poll-Each-Read; UCB, Update-based Call-Back.





(a) Case 1



(b) Case 2

Figure 11. Cost of Optimal Update-based Replacement (OUR) and Least Frequently Used for different message sizes: (a) Case 1 and (b) Case 2. UPER, Update-based Poll-Each-Read; UCB, Update-based Call-Back.

5.5. Impact of number of mobile stations

The number of MSs has significant effect on what PF values different objects get. In the way PF is computed (as shown in Equation (2)), λ_j is a global metric and is not affected by the number of MSs. On the other hand, $\mu_{i,j}$ for all i are affected by the number of MSs, provided that μ for the entire system is given. As the number of MSs increases, the difference between PFs of different items become less distinct, and PFs are mainly determined by λ. Figure 10 shows results with parameters α_a, α_u, K, and N valued at 0.01, 0.06, 50, and 500, respectively. As shown in Figure 9, with more MSs, the hit rate decreases at a faster rate, and hence, the cost per access increases.

5.6. Impact of message size

Finally, we investigate the impact of different message sizes, that is, C_{msg}. We consult two cases: (1) $K = 50$ and $N = 500$ and (2) $K = 100$ and $N = 400$. For both the cases, values of λ, α_a, and α_u are set to 0.40, 0.01, and 0.60. The results are presented in Figure 11. From both the results, it is evident that cost per access for UPER increases at a faster rate than UCB as the ratio of C_{msg} to C_{obj} increases.

5.7. General observation

We have the following general observations while preparing and running the simulator and evaluating the results:

- It is difficult to synthetically generate access and update traces that satisfy all the parameters and capture the worst-case requirements. Hence, most of the traces are generated with high update rates and update Zipf ratios to make them behave more closer to the worst-case scenarios. In all the simulations, the theoretical probability of hits are computed at each replacement only. Thus, in most cases, simulation results are slightly better than the theoretical results.
- For cases where cache suffers from higher number of misses, or cases where there are fewer to no

Figure 10. Performance of Optimal Update-based Replacement (OUR) and Least Frequently Used (LFU) with different number of mobile stations: (a) $\lambda = 0.40$, (b) $\lambda = 0.50$, (c) $\lambda = 0.40$, and (d) $\lambda = 0.50$. UPER, Update-based Poll-Each-Read; UCB, Update-based Call-Back.

misses, the temporal objects introduced in between two consecutive cache replacements result in fewer extra cache hits. Thus, in those cases, the difference between theory and simulation subsides.
- For cases where cache suffers from higher number of misses, the benefit of OUR scheme becomes more visible. As the system approaches to a system with read-only objects (i.e., objects with no updates), both LFU and OUR can achieve the optimal performance.

6. CONCLUSION

In this paper, we have proposed an optimal cache replacement scheme, named *Optimal Update-based Replacement* (OUR) scheme, to make efficient use of the network bandwidth in wireless data networks by increasing effective cache hits. To accommodate access and update information in the caching system, we have also proposed two enhanced cache access schemes—UPER and UCB access scheme. We have analyzed the cache access schemes and proved that if combined with OUR scheme, the schemes guarantee optimal number of effective cache hits and optimal cost of data access in terms of network bandwidth. Extensive simulations also demonstrate that the proposed scheme outperforms LFU in terms of both effective hits and communication costs.

Currently, we are working towards finding optimal cache system policies for data access applications where updates are initiated by the clients, rather than the server. We are also investigating cache systems for other form of wireless networks, particularly, networks with limited broadcasting capabilities, such as wireless LAN. In many applications, objects sizes are different. We plan to carry on researches where the same objects size constraint is relaxed.

REFERENCES

1. Facebook, Inc. facebook, 2010. http://www.facebook.com/.
2. Qzone QQ. Qzone, 2010. http://qzone.qq.com/.
3. News Corp. Digital Media. Myspace, 2010. http://www.myspace.com/.
4. Twitter, Inc. twitter, 2010. http://www.twitter.com/.
5. YouTube, LLC. Youtube, 2010. http://www.youtube.com/.
6. Youku. Youku, 2010. http://Youku.com/.
7. Windows Live SkyDrive. Skydrive, 2010. http://skydrive.live.com/.
8. Yahoo! Inc. flickr, 2010. http://www.flickr.com/.
9. Google Inc. Picasa, 2010. http://picasa.google.com/.
10. Photobucket. Photobucket, 2010. http://photobucket.com/.
11. Dropbox, Inc. Dropbox, 2010. http://dropbox.com/.
12. AT&T - News Room. AT&T launches pilot Wi-Fi project in Times Square, 2010. http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=30838.
13. AT&T - News Room. AT&T Wi-Fi handles more than 85 million total connections in 2009, more than four times 2008, 2010. http://www.att.com/gen/press-room?pid=4800&cdvn=news&-news articleid=30433&mapcode=consumer.
14. AT&T - News Room. AT&T Wi-Fi network usage soars to more than 53 million connections in the first quarter, 2010. http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=30766.
15. Creus GBI, Niska P. System-level power management for mobile devices. In *International Conference on Computer and Information Technology*, 2007; 799–804.
16. Perrucci GP, Fitzek F, Sasso G, Kellerer W, Widmer J. On the impact of 2G and 3G network usage for mobile phones' battery life. In *European Wireless*, 2009; 255–259.
17. Cao G. Proactive power-aware cache management for mobile computing systems. *IEEE Transaction on Computers* 2002; **51**(6): 608–621.
18. Wang X, Fan P. A strongly consistent cached data access algorithm for wireless data networks. *Wireless Networks* 2009; **15**(8): 1013–1028.
19. Lin Y-B, Lai W-R, Chen J-J. Effects of cache mechanism on wireless data access. *IEEE Transactions on Wireless Communications* 2003; **2**(6): 1247–1258.
20. Barbará D, Imieliński T. Sleepers and workaholics: caching strategies in mobile environments. *ACM SIGMOD Record* 1994; **23**(2): 1–12.
21. Cai J, Tan K-L. Energy-efficient selective cache invalidation. *Wireless Networks* 1999; **5**(6): 489–502.
22. Cao G. A scalable low-latency cache invalidation strategy for mobile environments. *IEEE Transactions on Knowledge and Data Engineering* 2003; **15**(5): 1251–1265.
23. Kumar A, Sarje AK, Misra M. Prioritised predicted region based cache replacement policy for location dependent data in mobile environment. *International Journal of Ad Hoc and Ubiquitous Computing* 2010; **5**(1): 56–67.
24. Madhukar A, Özyer T, Alhajj R. Dynamic cache invalidation scheme for wireless mobile environments. *Wireless Networks* 2009; **15**(6): 727–740.
25. Tian K-L, Cai J, Ooi BC. An evaluation of cache invalidation strategies in wireless environments. *IEEE Transactions on Parallel and Distributed Systems* 2001; **12**(8): 789–807.
26. Yin J, Alvisi L, Dahlin M, Lin C. Volume leases for consistency in large-scale systems. *IEEE Transaction on Knowledge and Data Engineering* 1999; **11**(4): 563–576.

27. Yuen JC-H, Chan E, Lam K-Y, Leung HW. Cache invalidation scheme for mobile computing systems with real-time data. *ACM SIGMOD Record* 2000; **29**(4): 34–39.
28. Nelson MN, Welch BB, Ousterhout JK. Caching in the Sprite network file system. *ACM SIGOPS Operating Systems Review* 1987; **21**(5): 3–4.
29. Hu Q, Lee DL. Cache algorithms based on adaptive invalidation reports for mobile environments. *Cluster Computing* 1998; **1**(1): 39–50.
30. Kahol A, Khurana S, Gupta SKS, Srimani PK. A strategy to manage cache consistency in a distributed mobile wireless environment. *IEEE Transaction on Parallel and Distributed Systems* 2001; **12**(7): 686–700.
31. Robinson JT, Devarakonda MV. Data cache management using frequency-based replacement. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 18, 1990; 134–142.
32. Tanenbaum AS. *Computer Networks*. Pearson Education: NJ, USA, 2002.
33. Chi H-C, Zhang Q. Deadline-aware network coding for video on demand service over P2P networks. *Journal of Zhejiang University—Science A* 2005; **7**(22–23): 755–763.
34. Chi H-C, Zhang Q, Shen X, (Sherman). Efficient search and scheduling in P2P-based media-on-demand streaming service. *IEEE Journal on Selected Areas of Communications* 2007; **25**(1): 119–130.
35. Skobeltsyn G, Aberer K. Distributed cache table: efficient query-driven processing of multiterm queries in P2P networks. *Technical Report LSIRRE-PORT-2006-010*, EPFL, Lausanne, Switzerland, 2006.
36. Akon M, Islam T, Shen X, (Sherman), Singh A. SPACE: a lightweight collaborative caching for clusters. *Peer-to-Peer Networking and Applications* 2010; **3**(2): 80–96.
37. Korupolu MR, Dahlin M. Coordinated placement and replacement for large-scale distributed caches. *IEEE Transactions on Knowledge and Data Engineering* 2002; **14**(6): 1041–4347.
38. Chen H, Xiao Y, Shen X, (Sherman). Update-based cache access and replacement in wireless data access. *IEEE Transactions on Mobile Computing* 2006; **5**(12): 1734–1748.
39. Xu J, Hu Q, Lee DL, Lee W-C. SAIU: an efficient cache replacement policy for wireless on-demand broadcasts. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, 2000; 46–53.
40. Xu J, Hu Q, Lee W-C, Lee DL. Performance evaluation of an optimal cache replacement policy for wireless data dissemination. *IEEE Transaction on Knowledge and Data Engineering* 2004; **6**(1): 125–139.
41. Acharya S, Muthukrishnan S. Scheduling on-demand broadcasts: new metrics and algorithms. In *The 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1998; 43–54.
42. Chou H-T, DeWitt DJ. An evaluation of buffer management strategies for relational database systems. In *VLDB '85 Proceedings of the 11th International Conference on Very Large Data Bases*, 1985; 127–141.
43. Abrams M, Standridge CR, Abdulla G, Fox EA, Williams S. Removal policies in network caches for world-wide web documents. In *SIGCOMM '96 Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 1996; 293–305.
44. Breslau L, Cao P, Fan L, Phillips G, Shenker S. Web caching and Zipf-like distributions: evidence and implications. In *IEEE INFOCOM*, Vol. 1, March 1999; 126–134.

AUTHORS' BIOGRAPHIES

Mursalin Akon received his B.Sc. Engg. degree in 2001 from the Bangladesh University of Engineering and Technology (BUET), Bangladesh, and his M.Comp.Sc. degree in 2004 from the Concordia University, Canada and Ph.D. degree in 2011 from the University of Waterloo, Canada. His current research interests include peer-to-peer computing and applications, network computing, and parallel and distributed computing.

Mohammad Towhidul Islam received his B.Sc. Engg. degree in 2001 from the Bangladesh University of Engineering and Technology (BUET), Bangladesh, and his M.Sc. degree in 2004 from the University of Manitoba, Canada. He is currently working towards his Ph.D. degree at the University of Waterloo, Canada. His current research interests include peer-to-peer computing and applications, service oriented architectures, and mobile computing.

Xuemin (Sherman) Shen received the B.Sc. (1982) degree from Dalian Maritime University (China) and the M.Sc. (1987) and Ph.D. degrees (1990) from Rutgers University, New Jersey (USA), all in electrical engineering. He is a Professor and the Associate Chair for Graduate Studies, Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on mobility and resource management in wireless/wired networks, wireless security, ad hoc and sensor networks, and peer-to-peer networking and applications. He is a co-author of three books, and has published more than 300 papers and book chapters in different areas of communications and networks, control and filtering. Dr. Shen serves as the Technical Program Committee Chair for IEEE Globecom'07, General Co-Chair for Chinacom'07 and QShine'06, the Founding Chair for IEEE Communications Society Technical Committee on P2P Communications and Networking. He also serves as the Editor-in-Chief for Peer-to-Peer Networking and Application; founding Area Editor for IEEE Transactions on Wireless Communications; Associate Editor for IEEE Transactions on Vehicular Technology; KICS/IEEE Journal of Communications and Networks, Computer Networks; ACM/Wireless Networks; and Wireless Communications and Mobile Computing (Wiley), etc. He has also served as Guest Editor for IEEE JSAC, IEEE Wireless Communications, and IEEE Communications Magazine. Dr. Shen received the Excellent Graduate Supervision Award in 2006, and the Outstanding Performance Award in 2004 from the University of Waterloo, the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada, and the Distinguished Performance Award in 2002 from the Faculty of Engineering, University of Waterloo. Dr. Shen is a registered Professional Engineer of Ontario, Canada.

Ajit Singh received the B.Sc. degree in electronics and communication engineering from the Bihar Institute of Technology (BIT), Sindri, India, in 1979 and the M.Sc. and Ph.D. degrees from the University of Alberta, Edmonton, AB, Canada, in 1986 and 1991, respectively, both in computing science. From 1980 to 1983, he worked at the R & D Department of Operations Research Group (the representative company for Sperry Univac Computers in India). From 1990 to 1992, he was involved with the design of telecommunication systems at Bell-Northern Research, Ottawa, ON, Canada. He is currently an Associate Professor at Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research interests include network computing, software engineering, database systems, and artificial intelligence.