Computer Networks 56 (2012) 2080–2095

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A bandwidth and effective hit optimal cache scheme for wireless data access networks with client injected updates

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

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

A R T I C L E   I N F O

Article history:
Received 28 October 2010
Received in revised form 18 October 2011
Accepted 17 February 2012
Available online 2 March 2012

Keywords:
Wireless data access
Cache
Replacement policy
Access scheme
Data update
Proactive and reactive data access

A B S T R A C T

In this paper, we propose an optimal cache replacement policy for data access applications in wireless networks where data updates are injected from all the clients. The goal of the policy is to increase effective hits in the client caches and in turn, make efficient use of the network bandwidth in wireless environment. To serve the applications with the most updated data, we also propose two enhanced cache access policies making copies of data objects strongly consistent. We analytically prove that a cache system, with a combination of our cache access and replacement policy, guarantees the optimal number of effective cache hits and optimal cost (in terms of network bandwidth) per data object access. Results from both analysis and extensive simulations demonstrate that the proposed policies outperform the popular Least Frequently Used (LFU) scheme in terms of both effective hits and bandwidth consumption. Our flexible system model makes the proposed policies equally applicable to applications for the existing 3G, as well as upcoming LTE, LTE Advanced and WiMAX wireless data access networks.

1. Introduction

In recent years, we have witnessed extraordinary improvements in computing electronics and wireless communications. These inventions are promising flexibility to our daily lives. Traditionally, cellular devices were used for voice communication. In contrast, modern mobile communication devices, such as smart phones, personal digital assistants (PDAs) and other hand-held computers are powerful general purpose computing devices with communication capabilities. These devices create the platform for computing and data access from any where and at any time by using existing high bandwidth 3G, under deployment LTE, and expected LTE Advanced and IEEE 802.16 m data access networks. Due to these emerging technologies, many necessary and entertaining mobile Internet applications such as Mobile IP telephony, mobile TV, video-on-demand (VOD), video conference, tele-medicine, mobile online banking, stock market tracking, instant messaging, on the road adaptive navigation, multi-player games have become a reality. Live Internet applications, such as online social networking (i.e., Facebook [1], Qzone [2], MySpace [3], Twitter [4]), document storage and sharing (i.e., Skydrive [5], Flickr [6], Picasa [7], Photobucket [8], Dropbox [9]), video sharing (i.e., Youtube [10], Youku [11]) have changed the way people communicate with each other, and store and share their multimedia contents and documents. The benefits and features of these Internet applications are made available at the expense of huge bandwidth consumptions, burdening the communication infrastructure. Mobile devices with faster processor, variety of sensors and powerful operating systems are becoming more affordable. As more users subscribe for wireless data access services, the problem of bandwidth is simply going to get worse. Modernizing wireless communication infrastructure towards higher capacity is extraordinary expensive. Thus, the service provides have no choice but to look for

* Corresponding author.
E-mail addresses: mmakon@bbcr.uwaterloo.ca (M. Akon), mtislam@bbcr.uwaterloo.ca (M.T. Islam), xshen@bbcr.uwaterloo.ca (X. (Sherman) Shen), asingh@uwaterloo.ca (A. Singh).

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2012.02.015

alternate solutions and user incentive mechanisms [12–14].

In a mobile information retrieval system, databases and files are hosted at a remote server. Conventionally these servers are directly connected to the wired networks to make the data access process faster and feasible. When a wireless user accesses data objects from the server, all communications have to pass through the channels of the wireless network. In spite of many improvements, wireless channel bandwidth is the scarcest resources, making data access in wireless networks very expensive. To reduce the data access cost, a client has to be very economic about the amount of data access. Additionally, when data is accessed a client must make the best utilization of the wireless channels. Many efforts have been put towards the best utilization of the available wireless channels. In such efforts, some applications behave adaptively depending on the state of the wireless channel, available bandwidth and other resources. For example, a mobile image retrieval system may retrieve images whose quality is adjusted according to the available bandwidth. A mobile device Internet browser retrieving compressed hypertext documents from the server and decompressing before displaying is another example of environment aware applications. However, developing such network aware applications is not trivial, particularly because the application logic as well as the development process become exceptionally complex [15]. In wireless data access applications, caching recently accessed objects is a very practical approach to reduce the amount of data access, and hence, cost of data access. Notice that cache oriented solutions do not contradict with the idea of developing network aware applications, rather, in many cases, cache can be deployed irrespective of network awareness of the applications.

Incorporating cache in a network system may increase the performance of the system in three ways – (1) the average access time or latency is shortened. As the local cache hosts a number of objects (depending on some criteria) many data objects are delivered from the local cache without retrieving the objects from the remote server. Therefore, caching is heavily used in hardware systems, such as processor cache and disk cache; (2) if caching is not deployed, at an access, the requested object would have to be fetched from the data server. With cache, many data objects are served locally, reducing the amount the data transferred over the networks; and (3) without cache, all data accesses have to be handled by the server. With cache, many of the accesses are handled by the client, cutting down the server load. The latter two benefits also make the data access system more scalable – allowing a server to handle more clients without adding any additional computing or network resources. Use of cache in wireless environment results in another very interesting and crucial benefit to its users. Data communications from wireless devices consume a significant amount of power. Utilization of a cache reduces the amount of communication transmissions. Consequently, data access cost is reduced, power is saved, and battery life is prolonged.

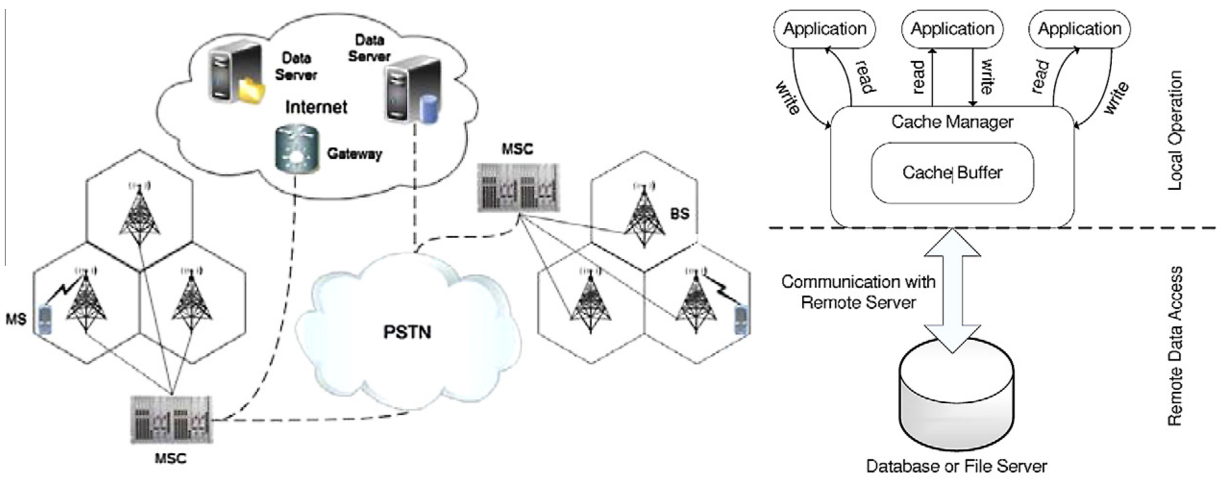
The goal of deployment cache may vary. In wireless data access applications, a cache mechanism needs to address two crucial aspects – cache access and cache replacement policies. A cache access policy determines how a cache is accessed and how the client–server system utilizes the cache. Maintaining consistency between copies scattered over clients and the server is also a task of the cache access policy. Notice that ensuring consistency in a distributed environment is complicated, particularly, if updates are allowed to be injected from any of the clients.

Despite the complexity of the problem, many applications demands availability of the most recently updated information/objects, because updates render all copies of older versions of the updated objects obsolete for further computation. This kind of consistency is called the *latest value* consistency [16], and a cache, satisfying the latest value consistency, is said to be *strongly consistent* [16,17]. In this case, a client has to retrieve the data item from the server. Several strong cache consistency algorithms for mobile/wireless data access have been proposed, such as, *Invalidation Report (IR)* schemes [18,19,16,20–29], *Poll-Each-Read* [17] and *Call-Back* [30,17,31,28].

When a cache is full and a new object is introduced, a decision has to be made – whether caching the new object is beneficial, and if it is, which existing object should be removed to make space. A replacement policy makes this important decision. Most researchers consider *Least Recently Used (LRU)* [18,19,16,20–25,17,26–29] or *Least Frequently Used (LFU)* [32] replacement policies for wireless data access. In general, an access policy can be deployed with any of a set of replacement policies and vice versa. However, performance of different combinations of access and replacement policies varies depending on system characteristics. Hence, a system developer has to be prudent in choosing the appropriate replacement and access policies.

In this paper, we provide a strongly consistent and update-aware cache mechanism for wireless clients scattered over a network spanning multiple wireless cells, where data updates may originate from any client. We make three major contributions – firstly, we propose two strongly consistent cache access policies – *Proactive Access Policy* (PAP), and *Reactive Access Policy* (RAP). Secondly, we introduce an Update-oriented Replacement Policy (URP). Our access policies are designed keeping the replacement policy in mind. The access policies collect different access and update related information to facilitate working capability of the cache replacement policy. In turn, the replacement policy aims towards higher effective hits. Thirdly, we analytically prove that our replacement policy ensures the optimal performance. As a result, this research provides the upper boundary for the worst case performance of any caching scheme and a foundation for average case performance comparison. The design goals of the proposed cache mechanism are – (1) to increase the effective hit ratio and (2) to reduce transmission cost (i.e., bandwidth consumption) by the applications. 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 PAP and RAP cache access, and URP cache replacement policies are introduced in Section 3. Quantitative analysis is provided in Section 4. Performance



(a) A Wireless Data Access Network (b) The Software Architecture

Fig. 1. Pictorial view of the system model.

evaluations and comparisons are presented in Section 5. We conclude the paper in Section 6.

2. Background

We start the section with a description of our system model followed by a brief introduction to important related works. We present the design goals and performance metrics at the end of the section.

2.1. System model

Our system model is based on the wireless data networks, already available in the consumer market [33]. In these networks, service areas are divided into a number of *location areas* (LA). 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 connect 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* (PSTN).

An example wireless data access network is shown in Fig. 1a. An MSC is connected to the Internet through either proprietary networks of the wireless career or through PSTN. For practical reasons, data servers are integrated to the Internet or to the service provider's network through wired infrastructure. As a result, any form of communication between a mobile device and a data server has to pass through the wireless section of the network, located between the mobile device and the corresponding BS. Notice that, existing 2G (such as EDGE, CDMA2000 1xRTT), 3G (such UMTS, WCDMA, CDMA2000 1xEV-DO), 3.5G and 3.7G (such as HSDPA, IEEE 802.16e), under deployment 3.9G (such as Long Term Evolution (LTE)) and expected 4G (such as LTE Advanced, IEEE 802.16m) networks are practical examples of our system model.

Users run different client applications on their mobile devices. A group of interrelated data access applications may want to use a cache for the reasons described in the previous section. A cache consists of a buffer and a cache manager.[1] In our model, a data server also plays an important role and collaborates with the cache managers in improving the performance of the client caches scattered over the network, as shown in Fig. 1b. We made the design choice of leaving the wireless network infrastructure unchanged. We are motivated by the fact that changes to wireless infrastructures are expensive and often a very slow process to be universally adopted. Moreover, if caches or cache helpers are implemented at any of the intermediate processing units of the infrastructure (i.e., BSs, MSCs or gateways), cache management becomes complex. As mobile wireless device roams from one cell to another, maintained cache and/or related status information also has to be moved from one processing unit to another, resulting in significant overhead and delay. On the other hand, a locally maintained cache at a mobile device is not influenced by the mobility of the user. In this paper, we consider that communication scheduling, channel condition tracking, packet scheduling, error and flow control are performed by the lower layers of the communication stack. Rather, we focus on reducing bandwidth consumed by the application or *soft bandwidth* – the amount of data the client requests to and from the lower layers to transmit and receive, respectively.

2.2. Related works

The technique of caching has been employed in different problem domains. Caching in wired networks has received significant attention from research community. Applications in wired network enjoy plethora network bandwidth. Thus, most of the researches concentrate on faster data retrieval (i.e., latency) rather than focusing on bandwidth consumption. In [34,35], authors propose a pre-fetch scheme to cache streaming multimedia segments to be

[1] In the rest of the paper, we use these terms – mobile host, client (application), and user, interchangeably. Unless specified precisely, a cache and cache manager is considered to be an integral part of a client.

played in the near future. The characteristics of streaming multimedia application is mostly deterministic. Here, sequential data segments are accessed in sequence. Caching is used in other areas of computer applications to store temporary entries. In [36], alternates to Distributed Hash Table (DHT) are proposed, where caching is employed to store indices among a group of peers in a distributed manner. These techniques enable fast and efficient query processing in structured P2P networks. However, in those applications, where the caches are considered as temporary storages, a significantly large buffer is allocated for storage, and cache eviction is not considered. In [37,38], authors present a collaborative scheme to share file/disk caches among tightly coupled network workstations. The proposed scheme assumes that direct and efficient unicast and broadcast communications among all the clients are provided by the network infrastructure. Our system model does not support such communication features and hence, the presented collaborations are not directly applicable.

Unreliable communication links and user's mobility impose challenges on data consistency for collaborative caching in wireless data access. There are many cache consistency strategies available in recent days. However, each strategy focuses on particular application domain. Cao et al. provide a 3D design framework for evaluating the cache consistency strategies with respect to consistency control, consistency level and consistency status maintenance [39]. Hara et al. further classify the consistency level in four different primitives: Global Consistency, Local Consistency, Time-Based Consistency, and Peer-Based Consistency. Quorum is a basic technique of achieving geographical consistency. However, the authors do not mention bandwidth consumption for quorum formation to achieve the required consistency level [40].

In wireless data access, update process plays an important role because an update renders all the older copies of the updated data object invalid. For this reason, a replacement policy has to be aware of not only the access events but also the update events. The researches in [17] study LRU as the replacement policy without considering the effects of the update process. The method in [41] made efforts to utilize the update information. However, the work is not supported by an analytical model and does not guarantee better performance. Rather, the performance heavily depends on the access and update patterns and frequencies. Other researches [42,43] have also proposed to utilize update information in replacement policies, but the main concentration was to reduce the *stretch*[2] [44] of the invalidation report (IR).

Notice that IR schemes perform well, if the following three conditions are satisfied by the underlying system. Firstly, the channel is a broadcast channel. Secondly, all the clients are in one wireless cell, and finally, the server is available locally, i.e., according to our model, at the base station. If clients are located in different wireless cells, IR reports include too much information which is irrelevant to clients in different wireless location areas. Thus, IR schemes become very expensive [17]. In a real-life wireless network, not all of the three conditions are satisfied due to practical and architectural reasons. For example, the server is more likely to be at a remote site, subscribed clients are scattered all over the wireless networks or even in remote networks, and the entire network is not covered by a single broadcast channel. Moreover, a suitable implementation of an IR scheme requires cross layer support for efficiency, but not much attention from software development community has been attained towards such support. Hence, IR schemes are not realistic in practical data access networks. Beside the access policy, LRU [18,19,16,20], LFU [32], MRU [45] cache replacement policies 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 among the three [46].

[2] 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.

2.3. Performance metrics

We have two major design goals – (1) increase the effective hit ratio and (2) reduce the communication cost. To better understand these goals, we first define the following concepts. When an access takes place, two situation may arise – a *cache hit* or a *cache miss*. A cache miss happens when the accessed data object is not in the cache. Otherwise, a cache hit is considered to take place. However, not all cache hits contribute towards serving a data object from the cache. Thus, cache hits are classified into two groups – *valid cache hit* and *invalid cache hit*. A cached object becomes invalid when an updated version of the object is available. Invalid cache hits are those hits due to invalid cached objects. Given that the cached object is the most recent version, a hit on the object results in a valid cache hit. *Effective hit ratio* is the ratio of a valid cache hit over all accesses.

According to our system model, the wireless channel bandwidth between the mobile devices and corresponding base stations is the most expensive resource. Hence, measuring *cost* involves the amount of average soft bandwidth, consumed by the cache aware applications, to serve a data object request.

3. Proposed scheme

In this section, we first present the PAP and RAP cache access policies. Then, we describe the proposed Update-oriented Replacement Policy or URP.

3.1. Notations

Before describing the proposed policies, we introduce several notations from previous related research publications to make the concepts clear. Let the number of distinct and equal size objects hosted by the server be N. The hosted objects are identified as O_i, where $i = 1, \ldots, N$. Let the maximum number of objects a client can locally cache be K. Let $\mu_i^j(t)$ and $\lambda_i^j(t)$ be the access and update frequencies, respectively, of O_i at client j up to time t. We denote μ_i^j and λ_i^j as the expected access and update rates at the client j for object O_i, respectively. With sufficiently larger value of t, $\mu_i^j(t)$ and $\lambda_i^j(t)$ approach to access and update rate μ_i^j and λ_i^j, respectively. Formally, $\mu_i^j = \lim_{t\to\infty}\mu_i^j(t)$ and $\lambda_i^j = \lim_{t\to\infty}\lambda_i^j(t)$. Let $\mu_i(t)$ and $\lambda_i(t)$ be the total access and update rate for object O_i up to time t, respectively. Expected access and update rate for object O_i over all clients is defined as $\mu_i = \lim_{t\to\infty}\mu_i(t)$ and $\lambda_i = \lim_{t\to\infty}\lambda_i(t)$, respectively. Obviously, $\mu_i(t) = \sum_j \mu_i^j(t)$ and $\lambda_i(t) = \sum_j \lambda_i^j(t)$. Similarly, $\mu_j(t)$, μ^j, $\lambda^j(t)$ and λ^j are defined.

3.2. Access policies preliminaries

We introduce RAP, a reactive, and PAP, a proactive access policy. Through those access policies, we ensure that the objects, served to the applications, are the copies of the most recent version of the objects. Beside serving requested objects, these policies handle updates initiated by the clients, and collect access and update related information to facilitate working requirements of the replacement policy. Our replacement policies use following abstract[3] primitives:

add(**object**): Add *object* to the local cache. The prerequisite of this primitive is the availability of at least one free memory block in the cache buffer.
evict(**id**): Evict the object in the cache buffer with identification – *id*. The post condition of this primitive is one more free memory block in the cache buffer.
replace(**obj**): Overwrite an older copy of locally cached *obj* object with the most recent version of *obj*.
modify(**prof**): Depending on policies in place, access and update related profiles per object (and per client) are maintained. Each profile keeps access and/or update frequencies of the hosted objects and may contain information about availability of objects at different clients. This primitive modifies part of the profiles, indicated by *prof*.
find_id(**replacement_policy**): Find the identification of the object to evict, using the given replacement policy, identified by *replacement_policy*.

3.3. Working principle of the access policies

3.3.1. The update process

In both PAP and RAP, whenever a client initiates an update, the updated object is forwarded to the server. At the same time, both policies may store the updated object in the local cache in according to the replacement policy (described later). PAP and RAP policy differ in the process of notifying other clients about the update. PAP follows a proactive approach for notification, where as, RAP is reactive. Whenever the server receives an update, in PAP, the server notifies all other clients hosting a copy of the same object about availability of a newer version. This way, a client is able to remove locally cached invalid objects. Notice that in order to notify the clients about the update events, the server has to maintain a profile indicating cache contents of all its clients. In contrast, in RAP, a client checks for consistency of a requested object in reactive manner and the server simply waits for such queries.

[3] Efficient implementation details are left for the application designers and developers.

3.3.2. Access process in RAP

As stated above, in RAP, the clients are reactive in verifying the consistency of the associated data object at each access. In this policy, the server is the most resourceful and knowledgable entity in the system. It maintains per client per data object access and update profiles. It also keeps track of number of objects hosted by each clients. The working steps of RAP is shown in Fig. 2.

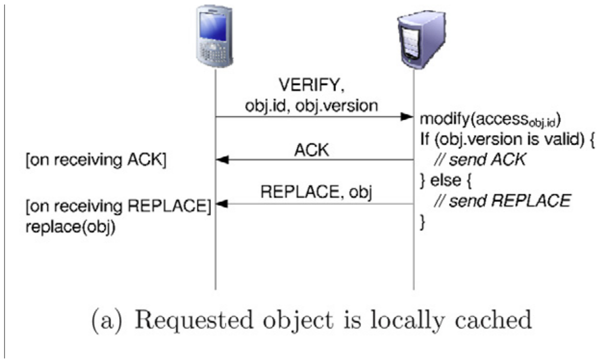
When a client makes a request for an object, in some cases, a copy of the object is available in the local cache. An event of access to such an object begins with a verification step, where the cache manager sends a query to the server (in a VERIFY message) requesting a check on validity of the cached copy. When the cached copy is consistent with the most recent version of the object, the server confirms the validity with an acknowledgement (as an ACK message). Otherwise, the server forwards a copy of the most recent version of the object (with a REPLACE message) and the cache manager replace the older copy with the most recent version, received from the server. Fig. 2a illustrates the entire process.

In the other cases, no copy of the requested object is available in the local cache (as shown in Fig. 2b). Obviously, the client has to make a request to the server to fetch the object (in a REQUEST message). The server considers two different scenarios. If the client cache has more buffer to accommodate at least one more object, the requested object is simply forwarded (in an ADD message) to the client to be added to its cache buffer. Otherwise, a replacement decision has to be made. The server, the most knowledgeable entity in the system, makes the replacement decision and forwards the identification of the object to replace with a copy of the requested object (in an EVICT message).

3.3.3. Access process in PAP

In PAP, as the server informs its clients about all the updates in a proactive manner, each client precisely knows which locally cached objects are consistent. Thus, unlike RAP, at a request from a client, the cache manager does not need to consult the server to verify consistency. Besides, in this policy, the cache manager is responsible to make replacement decision. To facilitate this decision making process, the manager maintains access and update profile for all the cached objects.

When a consistent copy of a requested object resides in local cache, the object is served without any further processing. Otherwise, a sequence of steps are taken to satisfy the request (as shown in Fig. 3). A request to fetch the object is sent to the server (in a REQUEST message). The server fetches the object along with the relevant access and update profiles (in an ADD message). If the cache manager finds that the local cache is full, it makes a replacement decision before accommodating the newly received object. It must be noted that whenever the policy decide to replace an object (either due to replacement or update), the associated profiles are forwarded to the server.



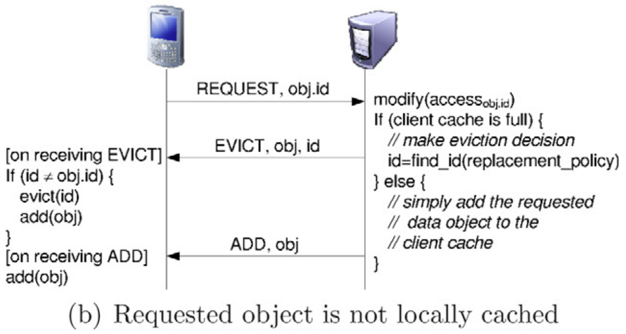

REQUEST, obj.id
modify(access$_{obj.id}$)
If (client cache is full) {
// make eviction decision
id=find_id(replacement_policy)
} else {
// simply add the requested
// data object to the
// client cache
}
EVICT, obj, id
[on receiving EVICT]
If (id ≠ obj.id) {
evict(id)
add(obj)
}
ADD, obj
[on receiving ADD]
add(obj)
(b) Requested object is not locally cached

Fig. 2. Step sequences for RAP access policy.

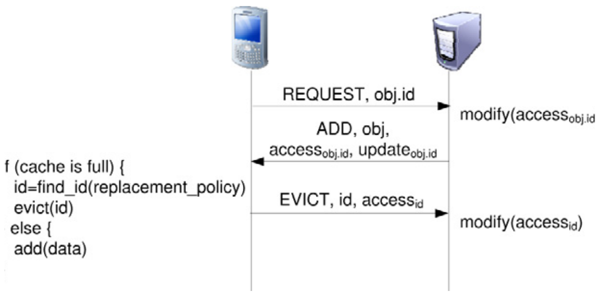


Fig. 3. Step sequences for PAP access policy.

3.4. Update-oriented Replacement Policy (URP)

In this subsection, we present Update-oriented Replacement Policy (URP), which uses both update and access frequencies gathered through the access policies to achieve superior guaranteed performance. We use *gain factor (GF)* of an object to determine which object should be given preference in preserving in the cache. We define GF for the object O_i at client j up to time t according to (1). The logic behind this definition of GF is elaborated in the next section.

$$GF_i^j(t) = \frac{\left(\mu_i^j(t) + \lambda_i^j(t)\right)\mu_i^j(t)}{\mu_i^j(t) + \lambda_i(t)} \tag{1}$$

Similarly, long term GF is defined as,

$$GF_i^j = \lim_{t\to\infty} \frac{\left(\mu_i^j(t) + \lambda_i^j(t)\right)\mu_i^j(t)}{\mu_i^j(t) + \lambda_i(t)} = \frac{\left(\mu_i^j + \lambda_i^j\right)\mu_i^j}{\mu_i^j + \lambda_i} \tag{2}$$

The working steps for replacement is shown in Algorithm 1. The algorithm takes two inputs – the set of (attributes of) the currently cached object and the accessed object (O_a). It finds the object with the minimum GF (O_r) over all the cached and accessed object (line 1–6). If O_r and O_a are not the same (line 7), O_r is evicted from and O_a is saved to the cache. Otherwise, the cache is kept intact and O_a is not saved. We consider replacement and eviction to be different – replacement takes place whenever the cache is full and a new object is introduced to the cache. A replacement results in eviction when an in cache object is replaced with the newly introduced object.

Algorithm 1. Replacement algorithm at client i

```
  input: $C$ is the set of currently cached objects
  input: $O_a$ is the accessed object
  // Cached object with the lowest GF
1 $MinGF \leftarrow +\infty$
2 foreach $O_j \in (C \cup \{O_a\})$ do
3     $GF_i^j \leftarrow \frac{(\mu_i^j+\lambda_i^j)\mu_i^j}{\mu_i^j+\lambda_i}$
4     if $MinGF > GF_i^j$ then
5         $MinGF \leftarrow GF_i^j$
6         $r \leftarrow j$
  // The new object has better GF
7 if $r \neq a$ then
      //So, keep the fetched object
8     $C \leftarrow (C \setminus \{O_r\}) \cup \{O_a\}$
  // Otherwise, leave the cache unchanged
```

4. Quantitative analysis

In this section, we analyze PAP and RAP. We also consider the cases when URP is combined with RAP and PAP. To facilitate the analysis, we number all replacements in the sequence they take place in time. The earliest replacement be identified as the first replacement, the next one as the second replacement and so on. The set consisting of all cached objects before and after the *t*th replacement are denoted with $C(t)$ and $C(t)'$, respectively.

Let the probability of guaranteed effective hits after the *t*th replacement for RAP and PAP be $P_{RAP}(t)$ and $P_{PAP}(t)$, respectively, under an arbitrary replacement policy. Similarly, the expected cost of accessing an object resulting from the *t*th replacement are denoted as $C_{RAP}(t)$ and $C_{PAP}(t)$, accordingly, under any replacement policy. We use $P_{RAP+URP}(t)$, $P_{PAP+URP}(t)$, $C_{RAP+URP}(t)$ and $C_{PAP+URP}(t)$ to denote the relevant metrics due to the *t*th replacement when URP is exercised. Let $O_a(t)$ and $O_r(t)$ denote the accessed and replaced objects at the *t*th access, respectively. Let $O_{URP}(t)$ denote the replaced object when URP is used. The cost of transmission of an object, a request (or verification) message, and an acknowledge message are denoted with the notations C_{obj}, C_{req}, and C_{ack}, respectively. In this paper, we leave out the costs of transmitting replacement decisions or cost of piggy backing profile information with a large message, such a message with an object. These information incurs very negligible message overheads. Based on the behavior of URP policy, we conclude the following theorem.

Theorem 1. *URP maximizes the probability of guaranteed effective hits at each replacement for both RAP and PAP access mechanisms.*

Proof. At a cache access, if the accessed object is not available at the local cache, the object is fetched from the server. When the local cache is full and a new object is introduced (due to access), a replacement decision has to be made. The employed replacement policy makes the final decision about preserving the new object. In case the new object is preserved, another object from the local cache is evicted to make space for the new object. Without loss of generality, let *t*th access results in a replacement. To prove the theorem, we must show that the probability of guaranteed effective hits after *t*th replacement obtained by replacing $O_{URP}(t)$ with $O_a(t)$ is larger than or equal to the same probability obtained by replacing $O_r(t)$ with $O_a(t)$. The following conditions holds at the *t*th access event,

$$O_a(t) \notin C(t) \tag{3}$$

$$O_r(t) \in C(t) \cup \{O_a(t)\} \tag{4}$$

$$C(t)' = (C(t) \cup \{O_a(t)\}) \setminus \{O_r(t)\} \tag{5}$$

Similarly, for URP policy,

$$O_{URP}(t) \in C(t) \cup \{O_a(t)\} \tag{6}$$

$$C(t)' = (C(t) \cup \{O_a(t)\}) \setminus \{O_{URP}(t)\} \tag{7}$$

Since access event is a Poison process, at any access, the probability of the accessed object being a given object O_i, $1 \leqslant i \leqslant N$, at client *j* is

$$p_{a,i} = \frac{\mu_i^j}{\sum_{i=1}^{N} \mu_{i,j}} = \frac{\mu_i^j}{\mu^j} \tag{8}$$

Since update event is also a Poison process, the probability of O_i being accessed or locally updated before being updated from any other client is,

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

Note that $\overline{p_{u,i}}$ also represents the probability that O_i is locally accesses or updated before the object become invalid. Accesses to different data items are independent, i.e., they are disjoint events. Therefore, from (5)–(9), no matter what replacement policy is used, we have,

$$\begin{aligned}
P_{RAP}(t) = P_{PAP}(t) &= \sum_{\forall i | O_i \in C(t)'} p_{a,i}\overline{p_{u,i}} \\
&= \sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\} \setminus \{O_r(t)\}} p_{a,i}\overline{p_{u,i}} \\
&= \sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\} \setminus \{O_r(t)\}} \frac{\mu_i^j}{\mu^j} \frac{\mu_i^j + \lambda_i^j}{\mu_i^j + \lambda_i} \\
&= \sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\} \setminus \{O_r(t)\}} \frac{(\mu_i^j + \lambda_i^j)\mu_i^j}{\left(\mu_i^j + \lambda_i\right)\mu^j} \\
&= \sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} \frac{\left(\mu_i^j + \lambda_i^j\right)\mu_i^j}{\left(\mu_i^j + \lambda_i\right)\mu^j} - \frac{\left(\mu_r^j + \lambda_r^j\right)\mu_r^j}{\left(\mu_r^j + \lambda_r\right)\mu^j} \\
&= \frac{1}{\mu^j}\left[\sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} \frac{\left(\mu_i^j + \lambda_i^j\right)\mu_i^j}{\mu_i^j + \lambda_i} - \frac{\left(\mu_r^j + \lambda_r^j\right)\mu_r^j}{\mu_r^j + \lambda_r}\right] \\
&= \frac{1}{\mu^j}\left[\sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} GF_i^j - GF_r^j\right]
\end{aligned} \tag{10}$$

So, the equality when URP is exercised can be driven as follows:

$$P_{RAP+URP}(t) = P_{PAP+URP}(t) = \frac{1}{\mu^j}\left[\sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} GF_i^j - GF_{URP}^j\right] \tag{11}$$

According to the definition of the URP policy, we have,

$$GF_{URP}^j \equiv \min_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} \left(GF_i^j\right) \tag{12}$$

Therefore, we conclude,

$$\begin{aligned}
P_{RAP+URP}(t) &= P_{PAP+URP}(t) \\
&= \frac{1}{\mu^j}\left[\sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} GF_i^j - \min_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} (GF_i^j)\right] \\
&\geqslant \frac{1}{\mu^j}\left[\sum_{\forall i | O_i \in C(t) \cup \{O_a(t)\}} GF_i^j - GF_r^j\right] = P_{RAP}(t) = P_{PAP}(t)
\end{aligned} \tag{13}$$

From the above derivation, it is conspicuous that at each replacement, the probability of effective hits while using URP is greater than or equal to the same metric while using any other replacement policy. Thus, we conclude that URP maximizes the probability of guaranteed effective hits at each replacement for both RAP and PAP access mechanism. □

Corollary 1. *URP minimizes the expected cost of data access at each replacement for both RAP and PAP access mechanism.*

Proof. With the RAP cache access scheme, when an access results in an effective cache hit, the client and the server exchange a request and acknowledgement message only (Fig. 2a). Otherwise, e.g., if the access causes a cache miss or an invalid cache hit, the client sends the server a request message, and the server responds to the client by fetching the requested data object (Fig. 2a). If the cache is full, the server also forwards a replacement decision (Fig. 2b). Therefore, for RAP,

$$\begin{aligned} C_{RAP}(t) &= P_{RAP}(t)(C_{req} + C_{ack}) + (1 - P_{RAP}(t))(C_{req} + C_{obj}) \\ &= C_{req} + C_{obj} + P_{RAP}(t)(C_{ack} - C_{obj}) \end{aligned} \tag{14}$$

In all practical applications, the cost of transmitting a simple message, such as an acknowledgment, is much smaller than the cost of transmitting an entire object, i.e., $C_{ack} \ll C_{obj}$. If simple messages are more or as expensive as fetching data objects, deployment of caches for strongly consistent applications would be more expensive in terms of both computing and communication resources. Thus, $C_{ack} - C_{obj} < 0$, but $P_{RAP} \geqslant 0$ and $P_{RAP+URP} \geqslant 0$. From Theorem 1, we know that $P_{RAP}(t)$ is maximized when URP is used, i.e., $P_{RAP+URP} \geqslant P_{RAP}$. Therefore, when URP is applied, following properties hold:

$$\begin{aligned} C_{RAP+URP}(t) &= C_{req} + C_{obj} + P_{RAP+URP}(t)(C_{ack} - C_{obj}) \\ &\leqslant C_{req} + C_{obj} + P_{RAP}(t)(C_{ack} - C_{obj}) = C_{RAP}(t) \end{aligned} \tag{15}$$

In PAP, in case of an effective cache hit there is no need for communication in between the client and the server. Therefore, the cost of access resulting from the replacement at the *t*th access is:

$$C_{PAP}(t) = (1 - P_{PAP}(t))(C_{req} + C_{obj}) \tag{16}$$

Again, from Theorem 1, $C_{PAP}(t)$ is minimized when URP is employed to make replacement decisions, i.e., $P_{PAP+URP}(t) \geqslant P_{PAP}(t)$. Thus,

$$\begin{aligned} C_{PAP+URP}(t) &= (1 - P_{PAP+URP}(t))(C_{req} + C_{obj}) \\ &\leqslant (1 - P_{PAP}(t))(C_{req} + C_{obj}) = C_{PAP} \quad \square \end{aligned} \tag{17}$$

Based on the characteristics of URP proven above, we propose the following theorem:

Theorem 2. *In the long run URP gives optimal guaranteed effective cache hits.*

Proof. Let the content of a cache be C up to time $t1$. Let object O_{a1} be accessed at $t1$, and there is another replacement policy called *OPT* making a different decision than URP. The objects replaced by *URP* and *OPT* are O_{URP1} and O_{OPT1}, respectively. By definition, $GF_{URP1} \leqslant GF_{OPT1}$. It has been proven in Theorem 1 that the guaranteed hits for URP are higher than or equal to any other replacement policy, including OPT. Hence, $P_{URP}(t1) > P_{OPT}(t1)$. To makeup the loss and outperform URP, let at a later time $t2 > t1$, when O_{a2} is accessed, OPT makes another replacement decision, where URP continues with the existing cache content. Thus, $P_{URP}(t1) = P_{URP}(t2) < P_{OPT}(t2)$. At time $t2$, object O_{OPT2} is decided to be replaced. The events are shown in Fig. 4. Next, we prove the theorem by contradiction and show that OPT replacement policy cannot exist.

$$\begin{aligned} &P_{OPT}(t2) > P_{URP}(t1) \\ &\quad \Rightarrow \sum_{\forall i|O_i \in C} GF_i + GF_{a1} + GF_{a2} - GF_{OPT1} - GF_{OPT2} \\ &\quad > \sum_{\forall i|O_i \in C} GF_i + GF_{a1} - GF_{URP1} \\ &\quad \Rightarrow GF_{a2} > GF_{OPT2} + GF_{OPT1} - GF_{URP1} \end{aligned} \tag{18}$$

Here, $GF_{OPT1} > GF_{URP1}$ and thus, $a2 \neq OPT2$ and the replacement at $t2$ involves in an eviction. It is clear that the following equation must hold:

$$O_{OPT2} \in C \cup \{O_{a1}\} \setminus \{O_{OPT1}\} \tag{19}$$

We consider three possible cases for choosing O_{OPT2}.

Case 1, where $O_{OPT2} \in C \backslash \{O_{OPT1}, O_{URP1}\}$: URP would make an eviction and accommodate O_{OPT2}, as $C \backslash \{O_{OPT1}, O_{URP1}\} \subseteq C \cup \{O_{a1}\} \backslash \{O_{URP1}\}$.
Case 2, where $O_{OPT2} \equiv O_{URP1}$: From (18), we can derive $GF_{a2} > GF_{OPT1}$. As $O_{OPT1} \in C \cup \{O_{a1}\} \backslash \{O_{URP1}\}$, URP would evict $O_k \equiv \min_{\forall i|O_i \in C \cup \{O_{a1}\} \backslash \{O_{URP1}\}} O_i$, where either $k \equiv OPT1$ or $(GF_{OPT1} \geqslant GF_k \Rightarrow GF_{a2} > GF_k) \wedge (k \neq OPT1)$.
Case 3, where $O_{OPT2} \equiv O_{a1}$: With similar argument of case 2, it can be shown that URP would make a replacement decision to evict some $O_k \in C \cup \{O_{a1}\} \backslash \{O_{URP1}\}$ to accommodate O_{a2}.

Therefore, it is not be possible that OPT makes an eviction decision to accommodate O_{a2} at $t2$ while satisfying (18), and at the same time, URP does not also accommodate O_{a2} by evicting one of the cached objects. Hence, a policy like OPT does not exist. Using the same argument, it can be shown that there exists no sequence of replacements (by another replacement policy) which results in higher guaranteed effective hits than URP. □

5. Performance evaluation

To evaluate the performance of our proposed policies, we have performed a series of extensive simulations using a detail discrete event simulator written in C++. Based on the discussion in Section 2, we have also evaluated the performance of the access policies combined with the popular Least Frequently Used (LFU) [47] and Least Recently Used (LRU) [18] replacement policies. Among LFU and LRU, LFU consistently demonstrates better performance, supporting the findings in [46]. Therefore, comparative discussion is limited in between LFU and our proposed

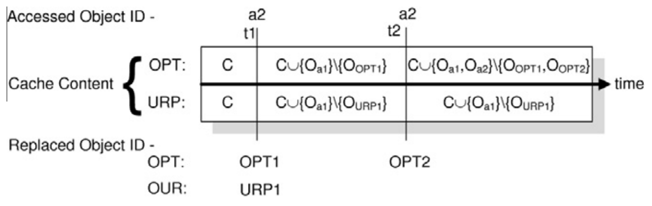


Fig. 4. Replacement in URP and other policies.

policies. In our simulations, we have collected information related to different performance metrics, discussed in Section. Our simulation environment is described in Section 5.2. We summarize our findings from the simulations in Section 5.3–5.9. We also present results from the theoretical analysis, wherever applicable.

5.1. Performance metrics

We consider two performance metrics – effective hits and cost per access. Let $n_{a,j}$ be the total number of accesses to object O_j from all the clients. 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, i.e, $n_a = \sum_{j=1}^{N} n_{a,j}$. We compute the effective hit ratios as follows:

$$P_{RAP} = 1 - \frac{n_{miss}}{n_a} = P_{PAP} \tag{20}$$

Then, we deduce normalized effective hits as,

$$EH_{RAP} = P_{RAP} \times \mu \times T = P_{PAP} \times \mu \times T = EH_{PAP} \tag{21}$$

where T is the simulation time and $\mu = \sum_i \sum_j \mu_i^j$. In our simulations, we do not compute the cost of piggy backing information. For example, when access or update frequencies are sent with other large message load (such as an object), the cost of transmitting access or update frequencies is ignored. We also keep aside the cost of forwarding updates from the clients, because irrespective of the choice

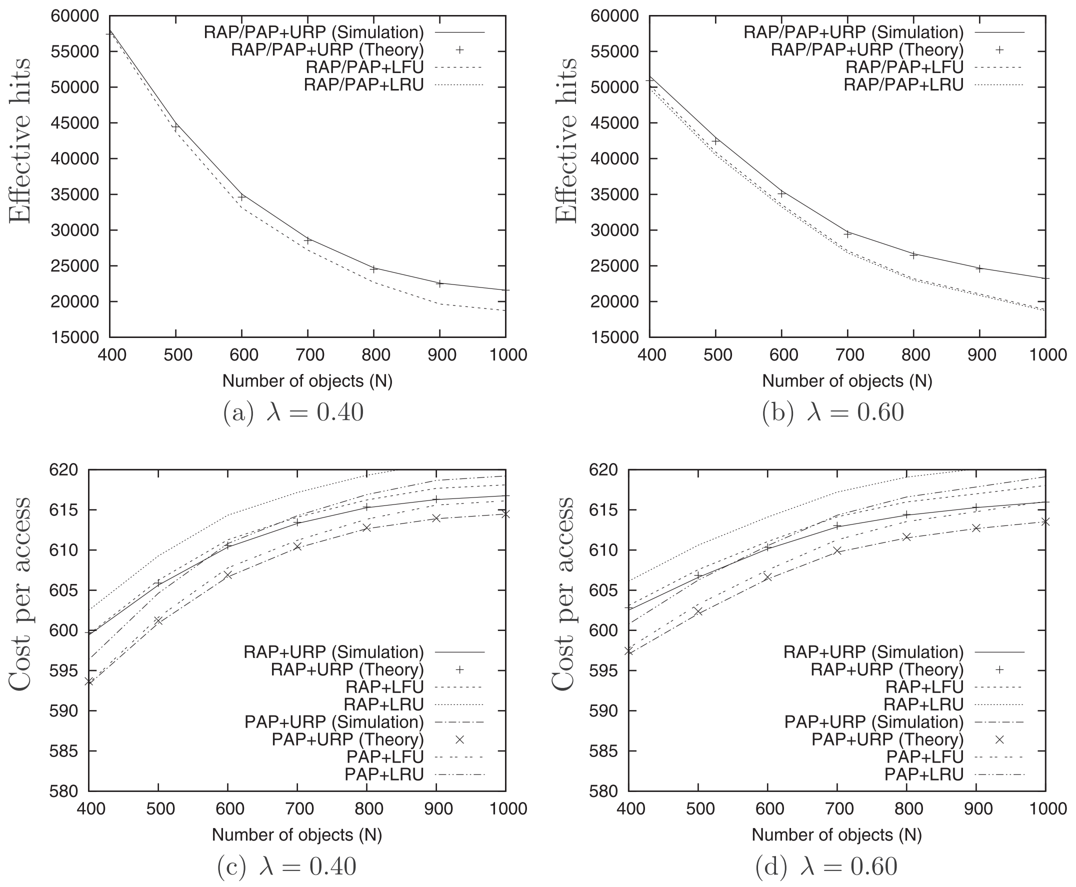


Fig. 5. Performance of URP, LFU and LRU policy with different object population sizes.

of the access policy, this cost is fixed. Finally, the costs per access for RAP and PAP are computed according to (22) and (23), respectively.

$$C_{RAP} = \frac{1}{n_a}[(n_a - n_{miss}) \times (C_{req} + C_{ack}) + n_{miss} \times (C_{req} + C_{obj})] \tag{22}$$

$$C_{PAP} = \frac{1}{n_a}[n_{miss}(C_{req} + C_{obj})] \tag{23}$$

5.2. Simulation setup

In data access applications, popularity of different objects are different. Researches have shown that user interest in different online objects follow *Zipf-like* distributions [47,17]. In our simulations, we assume *Zipf-like* distributions for object access or update pattern, and at an access or update event, an object with rank 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 $\alpha \geqslant 0$ and is called the *Zipf ratio*. Note that, when $\alpha = 0$, $p_i = 1/N$, for all i, and all objects are chosen with the same probability of $1/N$. Let α_a and α_u be the Zipf ratio for access and update events, respectively. We uniquely rank each object within the range from 1 to N to find its access and update probability. Note that rank of the object O_i may not be related to the object identifier i, as well, rank for access and update may be distinct. By default, we consider that C_{req} and C_{ack} have the same value of C_{msg}. Unless mentioned otherwise, the value of C_{msg}, C_{obj} and μ are 60, 600, and 1, respectively. We consider that 20 mobile users are in our network.

5.3. Impact of objects population

In this subsection, we discus our study on the effect of object population size (N). In Fig. 5, the performance of RAP/PAP + URP is compared with RAP/PAP + LFU for different object populations sizes. We consider the values for the parameters α_a, α_u and K in these simulations to be 0.20, 0.60 and 20, respectively. We observe following behaviors from the results:

- Given a fixed size cache, as the object population size increases, the chance of cache hit reduces, irrespective of the replacement policy. However, the gain of using URP becomes prominent with larger population sizes.

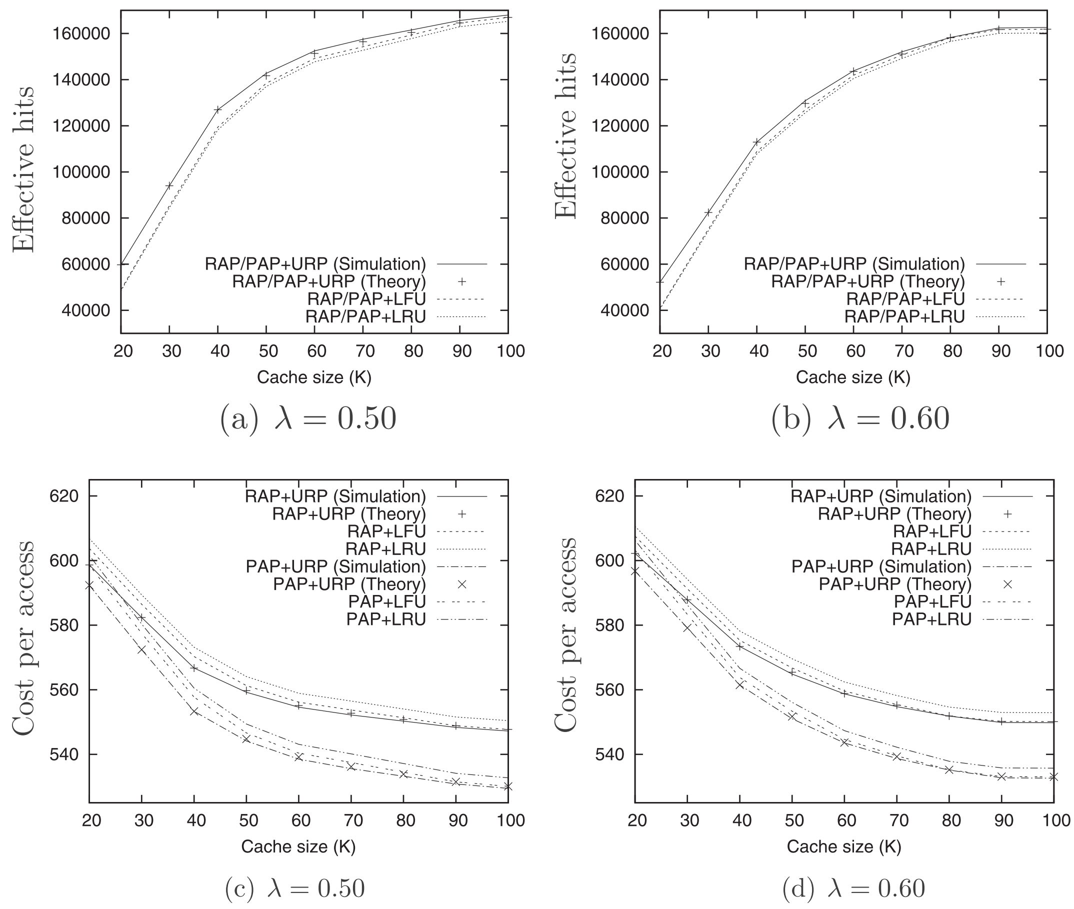


Fig. 6. Performance of URP, LFU and LRU for objects with different cache sizes.

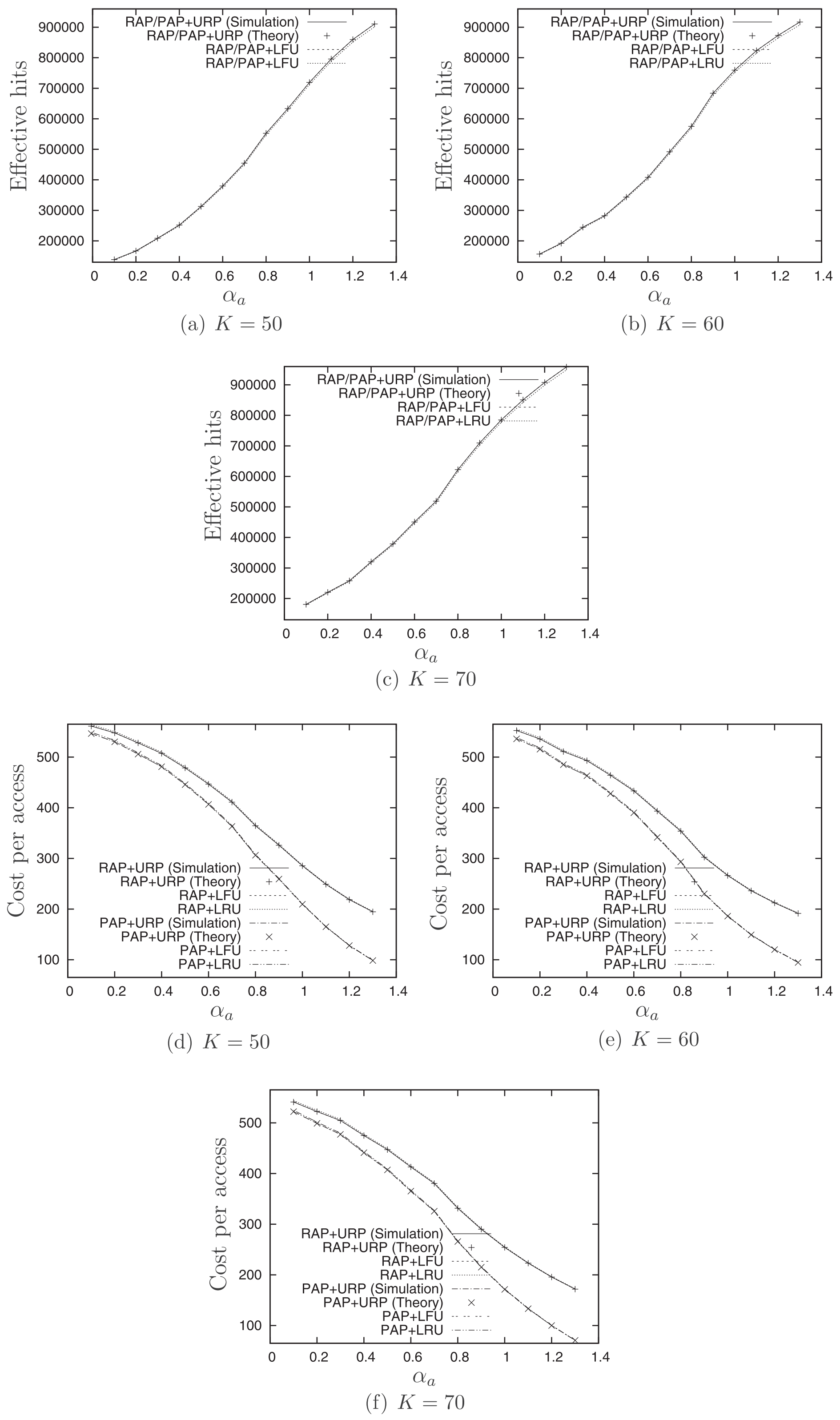


Fig. 7. Performance of URP, LFU and LRU for objects with no update.

- Effective hits for all combinations of access and replacement policies decrease with the increment of object population size. However, in all cases, URP shows better results.
- PAP + URP policy gives the best performance in terms of cost per access and PAP + LFU closely follows. Provided that $C_{obj} \gg C_{msg}$, according to (22), the difference between PAP + URP and PAP + LFU is dominated by n_{miss}. Hence, with smaller miss rate, the gap between URP and LFU is inconspicuous.
- With any of the replacement policies, cost per access for PAP is less than that's of RAP. However, URP reduces cost per access further as compared to LFU policy.

5.4. Impact of cache size

Performance characteristics for different cache sizes are shown in Fig. 6. For these simulations, the values for the parameters α_a, α_u and N are chosen to be 0.01, 0.60 and 400, respectively. From the figures, we deduce following arguments:

- In oppose to the previous discussion on object population, as cache size increases, the number of effective hits also increases for all combinations of policies. At the same time, URP demonstrates superior performance in all cases.
- Due to increasing number of hits, the cost per access, for both policies, also declines with larger cache. However, cost with PAP reduces at a higher rate than that's of RAP.
- With increment of cache size, each additional extra cache buffer results in fewer additional effective hits. Hence, a designer must find a tradeoff between the increment of cache performance and the cost of adding extra cache buffer space.

5.5. Impact of objects with no updates

Fig. 7 shows effective hit ratio and cost per access for URP and LFU policy when no update to any object takes place, i.e., λ_i^j for all i,j is 0. In other word, the objects are for read-only. To gather these results, in all the simulations, an object population size of 500 is considered. With no update events, GF_i^j is found by μ_i^j (as can be found from (2)) and as a result, while making a replacement (as well as, eviction) decision, both URP and LFU choose the same object. Thus both the policies result in exactly the same performance.

Besides, we have the following observations:

- LFU performance is also optimal where no update event takes place in the caching system.

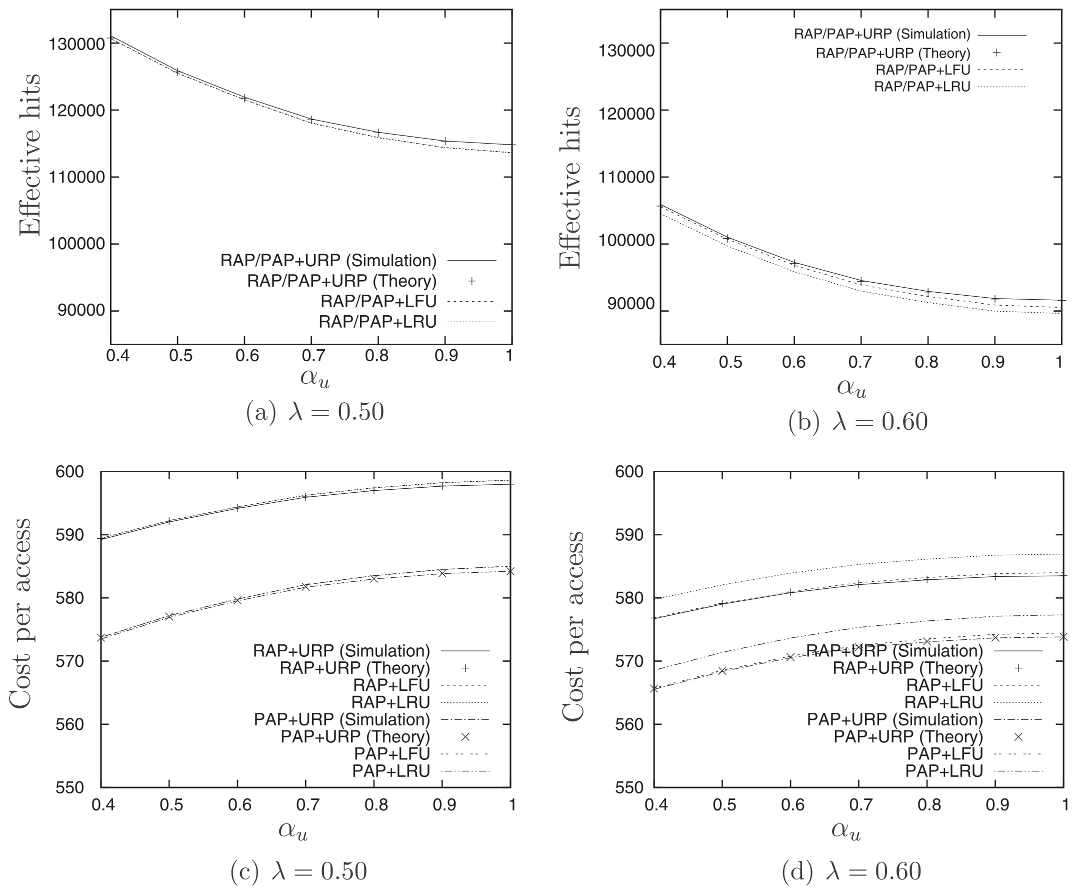


Fig. 8. Performance of URP, LFU and LRU for different Zipf ratios.

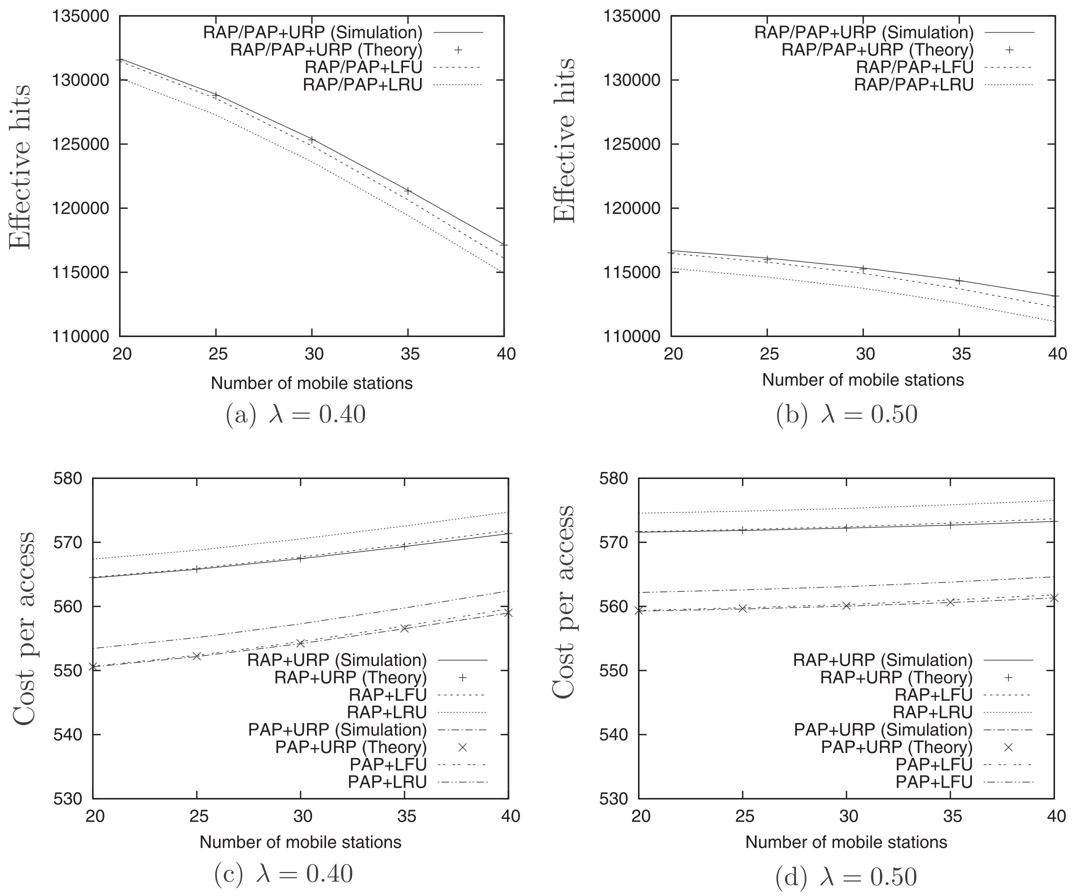


Fig. 9. Performance of URP, LFU and LRU different number of mobile stations.

- With increment of α_a, number of effective hits also increases. The higher α_a is, the smaller set of objects is accessed more frequently, contributing towards fewer misses.
- Larger cache helps alleviating cache misses. However, this behavior is clearly observed when α_a is smaller. With larger α_a, most of the access are due to fewer objects and hence, increasing cache size results in fewer number of additional effective hits.
- All combinations of access and replacement policies enjoy higher effective hits and lower cost per access when the objects are for read only and no update takes

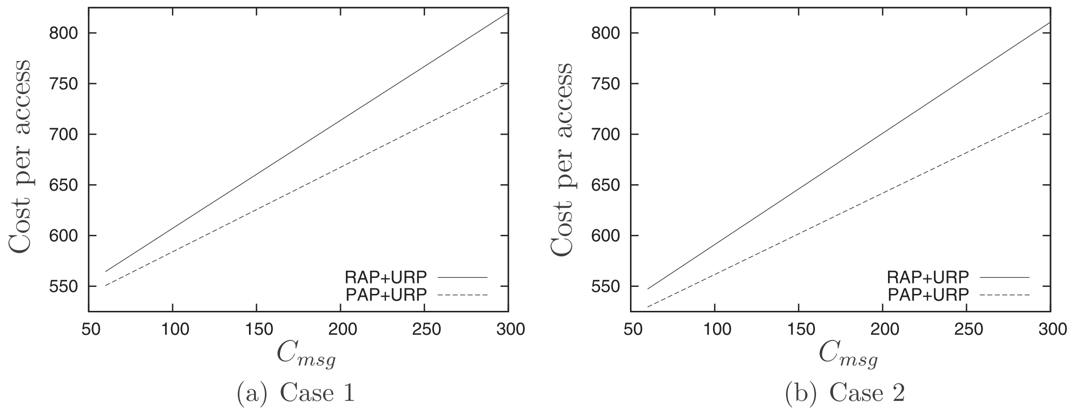


Fig. 10. Cost of URP for different message sizes.

place (i.e., $\lambda = 0$). Static databases, audio and video files sharing in wireless environment are examples of this kind of application.

5.6. Impact of Zipf ratio

Fig. 8 shows the effect of update Zipf ratio on the performance of both URP and LFU policies. In these results, we set the simulation parameters α_a, K and N to 0.01, 50 and 500, respectively. The results demonstrate that:

- With different Zipf ratios, performance of URP is consistently better than that's of LFU in terms of both number of hits and cost per access.
- In these simulations, while computing Zipf ratios, we have considered that both the ranks of object O_i are i. As a result, more frequent updates to more frequently accessed objects results in fewer overall hits. Note that, if rank for access and update Zipf ratio for all objects are arbitrary, the performance of the system may approach to a system with no update at all (see Section 5.5).

5.7. Impact of number of mobile stations

Number of mobile stations has significant impact on GF values of different objects. As shown in (2), λ_i is a combined metric and the number of mobile stations has no effect on it. In contrary, given a fixed μ_i and λ_i, μ_i^j and λ_i^j are affected by the number of mobile stations. As number of mobile stations increases, the difference between GFs of different objects becomes less distinctive and GFs are mainly determined by λ_i. Fig. 9 presents results from simulations with parameters α_a, α_u, K, and N valued at 0.01, 0.06, 50, and 500, respectively. As shown in the figure, with more MSs the number of effective hits decreases at a faster rate and hence, the cost per access increases.

5.8. Impact of message size (Cmsg)

Finally, we investigate the impact of different message sizes, i.e., C_{msg}. We consult two different cases, where (1) $K = 50$ and $N = 500$ and (2) $K = 100$ and $N = 400$. For both the cases, we consider the values of λ, α_a and α_u to be 0.40, 0.01 and 0.60, respectively. From the results, presented in Fig. 10, we observe that cost per access for RAP increases at a faster rate than PAP as the ratio of C_{msg} to C_{obj} increases.

5.9. Other observations and discussions

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

- Cases where cache suffer from a higher number of misses, or cases where there are fewer to no 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.
- Cases where cache suffers from a higher number of misses, the benefit of URP becomes more visible. As the system approaches to a system with readonly objects (i.e., objects with no updates), both LFU and URP approaches same (and optimal) performance.
- It is difficult to synthetically generate access and update traces which 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, simulation results are in general slightly better than the theoretical ones.

6. Conclusion

In this paper, we have proposed an optimal cache replacement policy, named *Update-oriented Replacement Policy* (URP) for wireless data access applications where updated are injected from all the clients. The goal of the policy is to make efficient use of the network bandwidth in wireless environment, by increasing effective cache hits. To maintain strong consistency among copies of objects and facilitate working environment for URP, we have also proposed two enhanced cache access policies – *Proactive Access Policy* (PAP), and *Reactive Access Policy* (RAP). We have proved that if PAP or RAP is combined with URP, the cache system guarantees optimal number of effective cache hits and optimal cost (in terms of network bandwidth) per data object access. Due to our comprehensive system model, the proposed policies are equally applicable to existing 2G and 3G, as well as upcoming Long Term Evolution (LTE), LTE Advanced and WiMAX wireless data access networks.

We are currently investigating optimal caching schemes for other form of wireless network infrastructures, particularly those networks supporting limited broadcast, such as WLANs. We are also considering a cooperative caching scheme so that clients can be opportunistic in fetching objects that are requested by other clients. At the same time, an idle client can assist other neighboring busy clients to cache objects for future use.

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] Windows Live SkyDrive, Skydrive, 2010. <http://skydrive.live.com/>.
[6] Yahoo! Inc., flickr, 2010. <http://www.flickr.com/>.
[7] Google Inc., Picasa, 2010. <http://picasa.google.com/>.
[8] Photobucket, Photobucket, 2010. <http://photobucket.com/>.
[9] Dropbox, Inc., Dropbox, 2010. <http://dropbox.com/>.
[10] YouTube, LLC, Youtube, 2010. <http://www.youtube.com/>.
[11] Youku, Youku, 2010. <http://Youku.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 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>.

[14] 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&-newsarticleid-=30433&mapcode=consumer>.

[15] G.B.I. Creus, P. Niska, System-level power management for mobile devices, in: International Conference on Computer and Information Technology, 2007, pp. 799–804.

[16] G. Cao, Proactive power-aware cache management for mobile computing systems, IEEE Transaction on Computers 51 (6) (2002) 608–621.

[17] Y.-B. Lin, W.-R. Lai, J.-J. Chen, Effects of cache mechanism on wireless data access, IEEE Transactions on Wireless Communications 2 (6) (2003) 1247–1258.

[18] D. Barbará, T. Imieliński, Sleepers and workaholics: caching strategies in mobile environments, ACM SIGMOD Record 23 (2) (1994) 1–12.

[19] J. Cai, K.-L. Tan, Energy-efficient selective cache invalidation, Wireless Networks 5 (6) (1999) 489–502.

[20] G. Cao, A scalable low-latency cache invalidation strategy for mobile environments, IEEE Transactions on Knowledge and Data Engineering 15 (5) (2003) 1251–1265.

[21] B.Y.L. Chan, A. Si, H.V. Leong, Cache management for mobile databases: design and evaluation, in: Proceedings of the Fourteenth International Conference on Data Engineering, 1998, pp. 54–63.

[22] C.C.F. Fong, J.C.S. Lui, M.H. Wong, Quantifying complexity and performance gains of distributed caching in a wireless mobile computing environment, in: Proceedings of the Thirteenth International Conference on Data Engineering, 1997, pp. 104–113.

[23] Q. Hu, D.L. Lee, Cache algorithms based on adaptive invalidation reports for mobile environments, Cluster Computing 1 (1) (1998) 39–50.

[24] J. Jing, A. Elmagarmid, A.S. Helal, R. Alonso, Bit-sequences: a new cache invalidation method in mobile environments, Mobile Networks and Applications 2 (2) (1997) 115–127.

[25] A. Kahol, S. Khurana, S. Gupta, P. Srimani, A strategy to manage cache consistency in a distributed mobile wireless environment, IEEE Transaction on Parallel and Distributed Systems 12 (7) (2001) 686–700.

[26] K.-L. Tian, J. Cai, B.C. Ooi, An evaluation of cache invalidation strategies in wireless environments, IEEE Transactions on Parallel and Distributed Systems 12 (8) (2001) 789–807.

[27] K.-L. Wu, P.S. Yu, M.-S. Chen, Energy-efficient caching for wireless mobile computing, in: Proceedings of the Twelfth International Conference on Data Engineering, 1996, pp. 336–343.

[28] J. Yin, L. Alvisi, M. Dahlin, C. Lin, Volume leases for consistency in large-scale systems, IEEE Transaction on Knowledge and Data Engineering 11 (4) (1999) 563–576.

[29] J.C.-H. Yuen, E. Chan, K.-Y. Lam, H.W. Leung, Cache invalidation scheme for mobile computing systems with real-time data, ACM SIGMOD Record 29 (4) (2000) 34–39.

[30] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, M.J. West, Scale and performance in a distributed file system, ACM Transactions on Computer Systems 6 (1) (1988) 51–81.

[31] M.N. Nelson, B.B. Welch, J.K. Ousterhout, Caching in the sprite network file system, ACM SIGOPS Operating Systems Review 21 (5) (1987) 3–4.

[32] J.T. Robinson, M.V. Devarakonda, Data cache management using frequency-based replacement, in: ACM SIGMETRICS Performance Evaluation Review, vol. 18, 1990, pp. 134–142.

[33] A.S. Tanenbaum, Computer Networks, Pearson Education, NJ, USA, 2002.

[34] H.-C. Chi, Q. Zhang, Deadline-aware network coding for video on demand service over P2P networks, Journal of Zhejiang University – Science A 7 (22-23) (2005) 755–763.

[35] H.-C. Chi, Q. Zhang, X. Shen, Efficient search and scheduling in P2P-based media-on-demand streaming service, IEEE Journal on Selected Areas of Communications 25 (1) (2007) 119–130.

[36] G. Skobeltsyn, K. Aberer, Distributed Cache Table: Efficient Query-Driven Processing of Multiterm Queries in P2P Networks, Tech. Rep. LSIRRE-PORT-2006-010, EPFL, Lausanne, Switzerland, 2006.

[37] M.R. Korupolu, M. Dahlin, Coordinated placement and replacement for large-scale distributed caches, IEEE Transactions on Knowledge and Data Engineering 14 (6) (2002) 1041–4347.

[38] M. Akon, T. Islam, X. Shen, A. Singh, SPACE: A lightweight collaborative caching for clusters, Peer-to-Peer Networking and Applications 3 (2).

[39] J. Cao, Y. Zhang, G. Cao, L. Xie, Data consistency for cooperative caching in mobile environments, IEEE Computer 40 (4) (2007) 60–66.

[40] T. Hara, S.K. Madria, Consistency management strategies for data replication in mobile ad hoc networks, IEEE Transactions on Mobile Computing 8 (7) (2009) 950–967.

[41] H. Chen, Y. Xiao, X. Shen, Update-based cache access and replacement in wireless data access, IEEE Transactions on Mobile Computing 5 (12) (2006) 1734–1748.

[42] J. Xu, Q. Hu, D.L. Lee, W.-C. Lee, SAIU: an efficient cache replacement policy for wireless on-demand broadcasts, in: Proceedings of the Ninth International Conference on Information and Knowledge Management, 2000, pp. 46–53.

[43] J. Xu, Q. Hu, W.-C. Lee, D.L. Lee, Performance evaluation of an optimal cache replacement policy for wireless data dissemination, IEEE Transaction on Knowledge and Data Engineering 6 (1) (2004) 125–139.

[44] S. Acharya, S. Muthukrishnan, Scheduling on-demand broadcasts: new metrics and algorithms, in: The 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1998, pp. 43–54.

[45] H.-T. Chou, D.J. DeWitt, An evaluation of buffer management strategies for relational database systems, in: VLDB, 1985, pp. 127–141.

[46] M. Abrams, C.R. Standridge, G. Abdulla, E.A. Fox, S. Williams, Removal policies in network caches for world-wide web documents, in: SIGCOMM, 1996, pp. 293–305.

[47] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and zipf-like distributions: evidence and implications, in: IEEE INFOCOM, vol. 1, 1999, pp. 126–134.

M. 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. 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, network computing, and parallel and distributed computing.

T. 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.

X. 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. He 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. He is a registered Professional Engineer of Ontario, Canada.

A. 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.