Design and Evaluation of a Scalable and Reliable P2P Assisted Proxy for On-Demand Streaming Media Delivery

Lei Guo, *Student Member*, *IEEE Computer Society*, Songqing Chen, *Member*, *IEEE Computer Society*, and Xiaodong Zhang, *Senior Member*, *IEEE*

Abstract—To efficiently deliver streaming media, researchers have developed technical solutions that fall into three categories, each of which has its merits and limitations. *Infrastructure-based CDNs* with dedicated network bandwidths and hardware supports can provide high-quality streaming services, but at a high cost. *Server-based proxies* are cost-effective but not scalable due to the limited proxy capacity in storage and bandwidth, and its centralized control also brings a single point of failure. *Client-based P2P networks* are scalable, but do not guarantee high-quality streaming service due to the transient nature of peers. To address these limitations, we present a novel and efficient design of a scalable and reliable media proxy system assisted by P2P networks, called PROP. In the PROP system, the clients' machines in an intranet are self-organized into a structured P2P system to provide a large media storage and to actively participate in the streaming media delivery, where the proxy is also embedded as an important member to ensure the quality of streaming service. The coordination and collaboration in the system are efficiently conducted by our P2P management structure and replacement policies. Our system has the following merits: 1) It addresses both the scalability problem in centralized proxy systems and the unreliable service concern by only relying on the P2P sharing of clients. 2) The proposed content locating scheme can timely serve the demanded media data and fairly dispatch media streaming tasks in appropriate granularity across the system. 3) Based on the modeling and analysis, we propose global replacement policies for proxy and clients, which well balance the demand and supply of streaming data in the system, achieving a high utilization of peers' cache. We have comparatively evaluated our system through trace-driven simulations with synthetic workloads and with a real-life workload extracted from the media server logs in an enterprise network, which shows our design significantly improves the quality of media streaming and the system scalability.

Index Terms—Internet media streaming, peer-to-peer systems, proxy caching, distributed hash table.

1 Introduction

Delivering multimedia contents with high quality and low cost over the Internet is challenging due to the typical large sizes of media objects and the continuous streaming demand of clients. Currently, there are three representative solutions for media streaming on the Internet. First, special content delivery networks (CDNs) have been built to replicate media servers across the Internet to move the contents close to the clients, such as Akamai.[1] This approach is performance-effective but not cost-effective. The second approach is to utilize existing proxies to cache media data, which is cost-effective but not scalable due to limited storages and bandwidths of centralized servers. The third approach is to build client-based P2P overlay networks for media content delivery, which is highly cost-effective but does not guarantee the quality of service because the capacities (CPU, storage, and bandwidth) of peers can be heterogeneous and their availabilities can be transient.

1. http://www.akamai.com/.

- *L. Guo and X. Zhang are with the Department of Computer Science and Engineering, Ohio State University, Columbus, OH 43210. E-mail: {lguo, zhang}@cse.ohio-state.edu.*
- *S. Chen is with the Department of Computer Science, George Mason University, Fairfax, VA 22030. E-mail: sqchen@cs.gmu.edu.*

Manuscript received 23 Mar. 2005; revised 21 July 2005; accepted 27 Oct. 2005; published online 17 Mar. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0104-0305.

Many researchers have delved in the proxy caching approach, a successful approach used for delivering text-based Web content on the Internet, to improve its performance for streaming media delivery. However, a full caching approach of media objects can quickly exhaust the limited proxy cache space. To handle the large sizes of media objects, researchers have developed a number of segment-based proxy caching strategies (e.g., [5] and [30]) to cache partial segments of media objects instead of their entireties.

Although the segment-based proxy caching technique has shown its effectiveness for media streaming, the quality of service it can provide is still not satisfactory to clients for the following reasons: First, the limited storage capacity of a proxy restricts the amount of media data it can cache for clients. In addition to the large size of media objects, research in [8] shows that the reference locality of multimedia objects is much less than that of Web pages. Thus, the cache space problem is much more significant in media proxy systems than in Web proxy systems. Second, the delivery of streaming media normally requires a dedicated reservation of continuous bandwidths for the clients. However, the highly demanded proxy bandwidths limit the number of clients to be served simultaneously. Furthermore, a proxy not only easily becomes a system bottleneck, but also forms a single point of failure, being vulnerable to attacks. On the other hand, the resources of bandwidth, storage, and CPU cycles are richly available and underutilized among the clients. Thus, the P2P file sharing model is very attractive. However, directly borrowing this

1041-4347/06/$20.00 © 2006 IEEE Published by the IEEE Computer Society

model for media streaming cannot guarantee the quality of streaming service for the following three reasons: First, the availability of demanded media data in each peer can be unpredictable because each peer caches and replaces media content independently without any coordination with other peers. Second, the availability of services can also be dynamic due to the transient nature of peers even though the data are always available. Third, the quality of services provided by multiple collaborative peers may not be sufficient for highly dynamic and bursty media streaming requests.

In order to address the scalability problem of proxy-based techniques, and to deliver the high-quality media content to clients, we present a novel and scalable segment-based P2P media delivering system by organizing the proxy and its clients in the same intranet into a P2P system. This system is called **PROP** abbreviated from our technical theme of "collaborating and coordinating **PRO**xy and its **P**2P clients." Our proposed system attempts to address both the scalability and the reliability issues of streaming media delivery in a cost-effective way. In such a system, the clients effectively coordinate and collaborate with the proxy to provide a scalable media storage and to actively participate in the streaming media delivery. Media objects are cached in segment units both in peers and in the proxy for the purposes of self-viewing and global sharing. Based on modeling and analysis of cache redundancy in our proposed system, we have designed replacement algorithms, aiming to approximately achieve the optimal distribution of media segments across the whole system. We have comparatively evaluated our proposed system by trace-driven simulations with synthetic workloads and with a real-life workload trace extracted from the media server logs in an enterprise network. Our experimental results show that PROP significantly improves the quality of media streaming and the system scalability. Specifically, the byte hit ratio can be improved up to 2.4 times over client-based P2P systems and proxy-based P2P systems. Our contributions in this work are threefold:

- The collaboration and coordination between the proxy and its P2P clients in our system address the scalability problem of the proxy-based technique, and also eliminate the concern of unstable quality of services by only relying on self-organized clients. These two system components are complementary to each other: The proxy provides a dedicated storage and reliable streaming services when peers are not available or not capable of doing so, while peers provide a scalable storage for data caching and significantly reduce the service load of the proxy.
- To improve the reliability and maximize the utilization of cached data in each peer, we have proposed a model to analyze the cache redundancy in our peer-to-peer caching system where peers are free to come and go. Our modeling results give the optimal replica distribution in such a system, and provide the guidance to cache replacement policy design.
- According to the modeling and analysis, we have proposed segment-based replacement policies for proxy and peers, which exploit the data locality and balance the load in the PROP system. Our objective is to keep popular media segments in the proxy for global sharing, and leave a certain space in each peer to cache those relatively unpopular segments. With this arrangement, both popular and unpopular segments are fairly treated, improving the overall hit ratios in the PROP system.

The remainder of the paper is organized as follows: Section 2 discusses other related work. Section 3 presents the design and rationale of our proposed media streaming system. We evaluate the performance of our system and its alternatives in Section 4, and make concluding remarks in Section 5.

2 Other Related Work

Efforts have been made on P2P media streaming recently. Acharya and Smith [1] studied the collaboration among multiple proxy servers in an intranet to provide media streaming service. Chae et al. [3] studied the data replacement techniques for cooperative caching among a number of media proxies. Different from these studies, our PROP system exploits the potential storages and processing capacities among media clients, and proposes corresponding data placement and replacement policies. Padmanabhan et al. [22] and Tran et al. [28] propose P2P multicast trees for live media streaming. Although multicast can be used for on-demand media streaming as well, the startup latency is nontrivial. Cui et al. [10] and Rejaie et al. [25] propose P2P streaming schemes for layer-encoded media. The quality of layer-encoded media streaming may not be guaranteed if some layers are not delivered in time, which cannot occur in segment-based systems. In weakly coupled P2P systems such as the ones presented in [22] and [25], users collaborate loosely with each other in a peer-to-peer fashion for on-demand media streaming, instead of being organized into a P2P overlay. Ip et al. [16] propose a cooperative proxy-client caching system for on-demand media streaming, which does not consider cache redundancy in the system. Hefeeda et al. [15] propose a structured P2P media streaming system, targeting media sharing on the global Internet, where each peer is the owner and distributer of its cached media, and there are no extra media providers in this system. In contrast, our system targets on-demand media streaming from professional and commercial media providers for clients in a large intranet, where each peer only serves its cached media as a member of the distributed caching system, instead of owning and distributing these media.

Aside from P2P-based media streaming, the research on the proxy-based and assisted media streaming and delivery techniques has been conducted. Particularly, proxy caching of streaming media has been explored in [1], [2], [13], [19], [26], [27], [30], [32]. Prefix caching and its protocol consideration as well as partial sequence caching are studied in [11], [26]. In video staging [32], a portion of bits from the video frames whose sizes are larger than a predetermined threshold is cut off and prefetched to the proxy to reduce the bandwidth on the server proxy channel. In [20], the proposed approach attempts to select groups of consecutive frames by the selective caching algorithm, while in [21], the algorithm may select groups of nonconsecutive frames for caching in the proxy. The caching problem for layered encoded video is studied in [19]. The cache replacement of streaming media is studied in [24], [27].

The proxy caching of streaming media has also been integrated with other techniques to further improve cache efficiency. In [29], the batching, patching, and stream merging are combined with proxy caching. A circular buffer is used in [4], while in [6], a set of existing techniques

are evaluated and the running buffer is efficiently utilized together with patching for efficiently delivering the media content.

3 Our System Design and Rationale

3.1 Infrastructure Overview

The two main components of the PROP system are 1) the proxy and 2) all the client peers receiving the media streaming service, which are self-organized into a P2P overlay network. The proxy is the bootstrap site of the P2P system and the interface between the P2P system and media servers. When an object is requested for the first time or when no peer in the system is able to serve a streaming request, the proxy is responsible to fetch the requested media data from the remote server, divide the object into small segments, and cache them locally.

Each peer in the PROP system has three functionalities. First, a peer is a *client* that requests media data. Second, a peer is a *streaming server* that provides media streaming service to clients. Each peer caches the media data in segments while its content accessing is in progress, and shares the cached data with other peers in the system. Third, a peer is also an *index server* that maintains a subset of indices of media segments in the system for content location. Peers in our system are self-organized into a structured P2P overlay supporting a *distributed hash table* (DHT), which maps the identifier of each media segment to the index of the segment (see Section 3.2 for details). The P2P operations in our system are overlay independent though we use CAN [23] in our simulation.

In our system, the media segments and their corresponding indices are decoupled. In other words, they may be maintained by different peers. The index of a segment contains a location list of peers, each of which caches a copy of the media segment, and the access information of this segment, which is used for replacement operations. The segment locating is conducted in two steps: The first step is to route the request to the peer maintaining the index of the demanded segment, and the second step is to select a peer that caches a copy of the segment. Comparing with the central indexing solution like Napster,[2] distributed indexing is not only scalable but also out of the single-point-of-failure problem. Meanwhile, our approach is efficient and cost-effective for two reasons. First, the selection of a serving peer can be optimized according to the capacities and workloads of peers caching the demanded media data because the index server maintains all access information of segments. Second, the cost of content locating is distributed over the P2P network so that the burden of P2P routing on each peer is trivial. Once the demanded segment is successfully located, the media streaming between the serving peer/proxy and the requesting peer becomes point-to-point.

3.2 P2P Routing and Media Streaming

The distributed hash table supported by the P2P overlay stores the $(key, value)$ maps where each key is the identifier of a media segment and the corresponding $value$ is the index of the segment. The identifier of a media segment is a globally unique identifier (GUID) hashed from the URL of the media object and the offset of the segment in the object with SHA1 algorithm. In our system, each peer is assigned a key space zone when joining the system, and maintains the segment indices mapped to this zone. Joining P2P routing entails getting the key space zone and taking over the corresponding indices from a neighbor while leaving P2P routing entails handing over the segment indices and merging the key space zone to a neighbor [23].

The following operations on the distributed hash table are designed in our system for content locating and data management: $publish$, $unpublish$, $request$, $update$, and $notify$. All these operations can be built on top of the common functionalities provided by distributed hash tables: $put(key, value)$, $get(key)$, and $delete(key)$.

3.2.1 Publishing and Unpublishing Media Segments

The $publish(seg\_id, location)$ operation publishes a cached copy of media segment in the P2P system, in which seg_id is the segment identifier, and $location$ is the IP address and port number of the peer that caches the segment copy. Correspondingly, the $unpublish(seg\_id, location)$ operation unpublishes the copy of media segment stored in $location$. To publish or unpublish a segment, a peer routes its $location$ and the seg_id to the target peer that maintains the segment index. Then, the target peer put the $location$ into or remove it out of the location list in the segment index. These operations correspond to the put or $delete$ functions in the DHT interface. Each segment index is created by the proxy when segmentation, and the index server is responsible for maintaining the consistency between the media segments and corresponding indices. A peer publishes a segment as soon as it caches the full segment and unpublishes a segment as soon as it deletes the segment. A peer publishes all segments it caches when joining the P2P system and unpublishes all segments it caches when leaving the P2P system.

3.2.2 Requesting and Serving Media Segments

A peer requests media data segment by segment, and searches in its local cache first. If the local search fails, it calls the $request(seg\_id, URL)$ operation, which requests a segment of the object designated by the URL. When a peer requests a media object that it does not cache, it routes the URL to the target peer that maintains the key space zone that the identifier of the object's first segment (its seg_id) is mapped to. This operation corresponds to the get function in the DHT interface. If the corresponding index does not exist, meaning the object is requested for the first time, the target peer sends a request to the proxy, which fetches the requested object from the media server, and creates the index and publishes the object. Then, the target peer routes the proxy's location back to the requesting peer, redirecting the peer to the proxy to get the media data. If the target peer finds the segment index, but the location list is empty, the target peer sends a request to the proxy, which fetches the segment and publishes it. The first five steps in Fig. 1a show such an example. If the location list is not empty, the target peer checks the validation of each location link, then returns the location of the peer with the maximal available bandwidth to the requesting peer. The first three steps in Fig. 1b show such an example. Then, the serving peer provides the requested data to the requesting peer, as the last step shown in Fig. 1a and Fig. 1b. A client can buffer the

2. http://www.napster.com/.

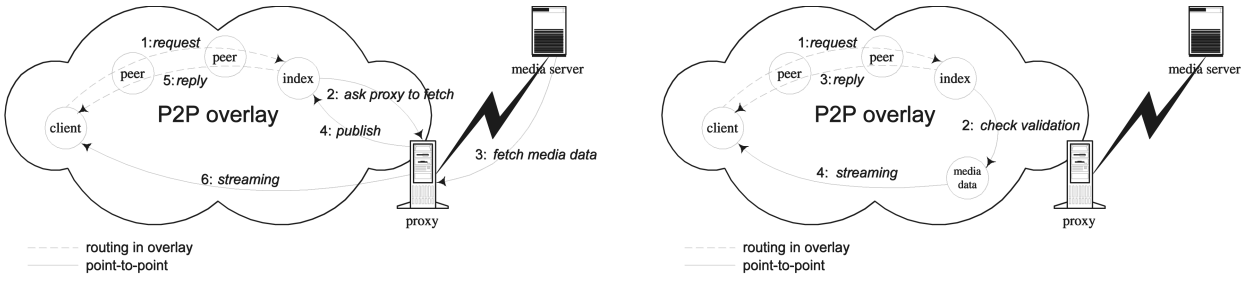

Fig. 1. The requesting and serving of media data. (a) The proxy fetches and serves the requested media data. (b) A peer serves the requested media data.

next segment when the current segment is played back. If a serving peer wants to leave the P2P system before the current streaming terminates, it must push the rest of the segment to the requesting peer before exiting the P2P system.

3.2.3 Updating Segment Popularity and Utility Values

PROP uses the popularity and utility values of segments to manage cached data (see Section 3.4 in details). These values depend on the access information and number of copies of corresponding media segments in the system. When the proxy or a peer finishes serving a segment streaming task, it calls $update(seg\_id, access\_info)$ operation, which routes the access information to the target peer maintaining the segment's index, and then the target peer updates the corresponding information items. When the segment popularity or utility values change, the index server notifies all peers that cache the segment new values by $notify(peerset, seg\_id, value)$ operation, where $peerset$ is the peers in the location list of the segment index, and $value$ is the popularity or utility value of the segment designated by seg_id. These operations correspond to the put function in the DHT interface.

3.2.4 Message Routing Overhead

In PROP, for each segment a client requests, a $request$ and an $update$ message are generated. For each segment replica that is cached or evicted from cache, a $publish$ or $unpublish$ message is generated. Although a $notify$ operation may generate multiple messages, it can be postponed if the popularity or utility value changes little. Furthermore, our replacement algorithm can keep the location list of the segment index in a moderate size (see Section 3.4 for details). Thus, the routing overhead in PROP is trivial compared to the media data transfered. Our simulation shows it is less than 1 percent of the streaming media data (including TCP/IP and Ethernet headers) for 100 KB segment size. To further reduce the routing overhead, we can increase the segment size, or use variable-sized segmentation such as exponential segmentation [30] and adaptive and lazy segmentation [5].

3.3 Redundant Cache Model

Due to the dynamical peer arrivals and departures, an object may not be available if it is only cached by a single peer. A solution is to let a peer that wants to go offline transfer the object it maintains to other peers, such as the *home* mode of Squirrel [17]. However, Squirrel is a peer-to-peer-based Web proxy system, where the objects are usually very small. In contrast, the sizes of multimedia objects are much larger. Frequently transferring cached objects among peers is not desired since it consumes more bandwidth and storage. Furthermore, a peer may fail due to many reasons. In our solution, we use two methods to provide the reliable service: 1) the proxy can maintain the most popular objects as a backup service and 2) an object (or a segment) can be cached in multiple peers to provide redundancy.

Assume the popularity (the reference probability) of media segments follows a Zipf-like distribution. If we rank all media segments in the descending order of their popularities, the popularity of the ith media segment, p_i, can be expressed as

$$p_i = \frac{A}{i^{\alpha}}, \tag{1}$$

where $A = \sum_{i=1}^{m} \frac{1}{i^{\alpha}}$, m is the number of all media segments, and α is a constant.

Assume there are N peers and m media segments in the system. Each peer contributes a storage of size C and has a failure probability $q\,(q < 1)$ when serving a segment. Assume the size of each segment is 1 unit of storage size. For segment i cached in r_i peers, the cache failure probability is q^{r_i}. Our objective is to find the distribution of $\{r_i\}$, which can minimize the total number of failed requests of all media segments, with the constraint of the total cache size

$$\{r_i\} = \arg\min_{\{n_i\}} \left\{ \sum_{i=1}^{m} p_i q^{n_i} \right\}, \tag{2}$$

where

$$\sum_{i=1}^{m} n_i = CN. \tag{3}$$

We have

$$\sum_{i=1}^{m} p_i q^{n_i} \geq m \left(\prod_{i=1}^{m} p_i q^{n_i} \right)^{\frac{1}{m}}, \tag{4}$$

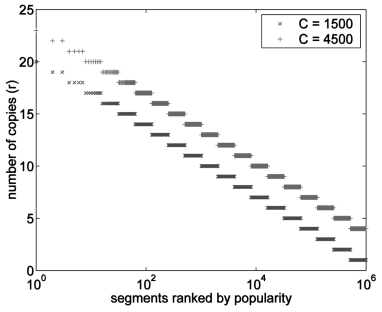

Fig. 2. Segment distribution.

where the minimum value holds when

$$p_1 q^{r_1} = \ldots = p_i q^{r_i} = \ldots = p_m q^{r_m} = B. \tag{5}$$

For segment i, we have

$$p_i q^{r_i} = B, \tag{6}$$

$$r_i = \frac{\log p_i - \log B}{\log \frac{1}{q}} = \frac{\log A - \alpha \log i - \log B}{\log \frac{1}{q}}. \tag{7}$$

Since

$$\sum_{i=1}^{m} r_i = \frac{m \log A - \alpha \log m! - m \log B}{\log \frac{1}{q}} = CN, \tag{8}$$

we have

$$\begin{aligned} \log B &= \log A - \alpha \frac{\log m!}{m} - \frac{CN}{m} \log \frac{1}{q} \\ &\approx \log A - \alpha(\log m - 1) - \frac{CN}{m} \log \frac{1}{q}, \end{aligned} \tag{9}$$

$$\begin{aligned} r_i &\approx \frac{\log p_i}{\log \frac{1}{q}} + \left(\frac{CN}{m} + \alpha \frac{\log m - 1}{\log \frac{1}{q}} - \frac{\log A}{\log \frac{1}{q}} \right) \\ &= -\frac{\alpha}{\log \frac{1}{q}} \log \frac{i}{m} + \left(\frac{CN}{m} - \frac{\alpha}{\log \frac{1}{q}} \right). \end{aligned} \tag{10}$$

When $\frac{CN}{m} \geq \frac{\alpha}{\log \frac{1}{q}} + 1$, we have $r_m = \frac{CN}{m} - \frac{\alpha}{\log \frac{1}{q}} \geq 1$. That is, when the total cache size of all peers is large enough so that all media segments can be cached, increasing cache size leads to evenly increase the number of copies for each segment. Fig. 2 shows the number of copies for different segments in the system, where 1,000 peers cache 10^6 different media segments, with each peer contributing 1,500 storage units ("×" line) and 4,500 storage units ("+" line), respectively.

In reality, it is difficult to get the values of parameters such as α, q, and A. In Section 3.4.2, we will present a heuristic distributed algorithm to achieve such a distribution without knowing the values of these parameters.

3.4 Cache Management

In the PROP system, the proxy serves as a persistent cache site, but the storage size is limited. On the other hand, the total storage space contributed by peers is huge but the available contents change dynamically because peers come and go frequently. To fully utilize the storage and to improve the streaming service quality, we propose efficient replacement policies for both proxy and peers based on the global information of segment accesses using the following parameters:

- T_0, the time when the segment is accessed for the first time,
- T_r, the most recent access time of the segment,
- S_{sum}, the cumulative bytes that the segment has been accessed,
- S_0, the size of the segment in bytes,
- n, the number of requests for this segment, and
- r, the number of replicas of the segment in the system.

3.4.1 Popularity-Based Proxy Replacement Policy

The proxy takes over the streaming service whenever the requested media segment cannot be served by any peer in the system. Thus, the proxy should hold those popular media objects to minimize the performance degradation due to peer failures. We use a popularity-based replacement policy instead of Least Recently Used (LRU) policy that is commonly used in Web proxies because LRU is not efficient for file scan operations, which are typical in media streaming services and can only exploit the locality of reference to the proxy instead of the whole system. We define the *popularity* of a segment as

$$p = \frac{\frac{S_{sum}}{S_0}}{T_r - T_0} \times \min\left(1, \frac{\frac{T_r - T_0}{n}}{t - T_r}\right), \tag{11}$$

where t is the current time instant, $\frac{\frac{S_{sum}}{S_0}}{T_r - T_0}$ represents the average access rate of the segment in the past, normalized by the segment size, and $\min(1, \frac{\frac{T_r - T_0}{n}}{t - T_r})$ represents the probability of future access: $\frac{T_r - T_0}{n}$ is the average time interval of accesses in the past; if $t - T_r > \frac{T_r - T_0}{n}$, the possibility that a new request arrives is small. Otherwise, it is highly possible a request is coming soon. The segment with the smallest popularity is chosen as the victim to be replaced when the proxy cache is full. Considering both the recent access and past access information, the proxy can cache the most useful media data for clients.

The study explained in [5] uses a similar approach to estimating the popularity of objects in proxy caching systems. However, our system architecture and replacement policies are completely different from these systems. In order to quickly collect space for new media objects and to reduce the cache management overheads, the exponential segmentation approach [30] and the adaptive-lazy segmentation approach [5] have been proposed. In contrast, due to the distributed indexing and caching service provided by peers, the load of proxy in the PROP system is much smaller than that in traditional proxy caching systems. PROP uses an evenly sized, fine granulated segmentation with low overheads, and our simulations show that finer granulated data management is more efficient than coarser granulated data management for cache utilization.

3.4.2 Utility-Based Replacement Policy for Client Peers

Independently exploiting reference locality on each client side is neither efficient from the system's perspective nor effective from the user's perspective. First, due to the client access patterns, the popular objects get more accesses from peers, and thus, have more copies cached in the system, which are already cached on the proxy side. Keeping those unnecessary copies of popular objects degrades the cache efficiency since the cache space could have been used to cache other objects. Second, the locality on each client side is limited and the cached data is prone to be flushed in a long streaming session if LRU replacement is used. Third, the segments of a media object may be cached in a single peer; thus, the data availability is very sensitive to the peer failure and leaving. On the other hand, the reference locality of all clients is much more significant than that of a single client and the difference between the access latency in an intranet and the local disk is not important for media streaming. Thus, although the cached data in PROP system are distributed across all client peers, they should be managed collectively to achieve best utilization.

The purpose of our peer replacement policy is to use an adaptive mechanism to dynamically adjust the distribution of media data cached in the system. We have three objectives:

- The distribution of cached segments should be close to the optimal distribution (described by (10)) to maximize the cache utilization.
- Different segments of the same object should be cached in different peers. Thus, the failure of individual peers can only affect part of the media object.
- The segment distribution should not involve extra data transferring.

To achieve these goals, we design a heuristic algorithm as follows. Our peer replacement policy is designed to replace both those media segments with diminishing popularities because they rarely get accessed, and those popular media segments with too many copies being cached. As a result, peers accessing media objects completely will cache the latter segments and evict the beginning segments of the objects because they are more popular and have more replicas in the system than the latter segments. Peers that access only the beginning segments will cache the beginning segments. Thus, naturally, a peer will cache only a few segments of each object it has accessed, while the segments of each object are distributed across many peers in the system according to their popularities, reducing the negative effects caused by peer failures. The above operation needs no extra data transferring except the data requested by users.

To achieve the optimal distribution, for the peer replacement policy, we define the *utility function* of a segment as

$$u = \frac{(f(p) - f(p_{min})) \times (f(p_{max}) - f(p))}{r^{\alpha+\beta}}, \qquad (12)$$

where p represents the popularity of the segment, p_{min} and p_{max} estimate the minimum and maximum of segment popularities in the P2P system, respectively, r is the number of replicas of this segment in the system, and $f(p)$ is a monotonic nondecreasing function, which is called the *adjustment factor* of the utility function. The desired distribution of media data in the PROP system is $r \propto f(p)$, as described in (10). In our system, we choose $f(p) = \log p$ and $\alpha = \beta = 1$, and we will show the performance of different adjustment factors in our evaluation. The values of p_{min} and p_{max} can be maintained by the proxy and propagated across the P2P overlay by a flooding when necessary. The term $\frac{f(p)-f(p_{min})}{r^{\alpha}}$ captures the segments with small popularities and large numbers of replicas while $\frac{f(p_{max})-f(p)}{r^{\beta}}$ captures the segments with large popularities and large numbers of replicas. These two kinds of segment replicas should be replaced by the replicas of segments with moderate popularities but a small number of copies. So, in our system, we choose those segments with the smallest utility value as the victims to be replaced when a peer's cache is full.

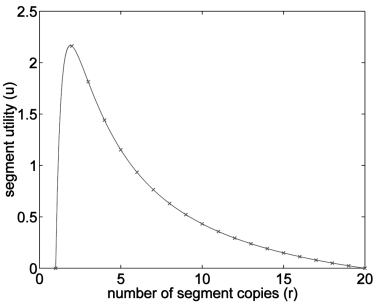

Fig. 3. Utility variations of media segments.

When the cache system reaches its optimal distribution of segment replicas, we have

$$u = \left(\log\frac{1}{q}\right)^2 \left(\frac{r_{max}}{r} - 1\right)\left(1 - \frac{r_{min}}{r}\right). \qquad (13)$$

Let $\frac{du}{dr} = 0$, we have

$$r_0 = \frac{2 r_{max} r_{min}}{r_{max} + r_{min}}. \qquad (14)$$

That is, the utility u reaches its maximum when $r = r_0$. Fig. 3 shows the relation between utility and number of copies. The figure indicates that our replacement policy can effectively reduce the number of popular segment copies that can be accumulated quickly (so as to better utilize the cache space), as well as reduce those unpopular segments when they are cached more than needed. Thus, the data distribution is optimized naturally along with the progress of media accessing, and the efficiency of cache utilization is maximized. Our simulation shows that the utility-based replacement policy has more benefits for workload with low reference locality than workload with high reference locality, due to the segment distribution it results in the PROP system. This advantage is very attractive for streaming media delivery systems since the reference locality in streaming media workload is much smaller than that in Web caching workload [8].

TABLE 1
Workload Summary

Trace Name	# of Req.	# of Obj.	# of Peers	Size (GB)	λ	α	Obj. Length	Trace Duration
REAL	11559	400	1663	24	-	-	6-131 m	10 days
PART	15188	400	376	51	4	0.47	2-120 m	1 day
WEB	15188	400	376	51	4	0.47	2-120 m	1 day

3.5 Streaming Task Dispatch

In PROP, a streaming session is divided into a number of streaming tasks in a moderate granularity determined by the segment size (we use 100-500 KB in our simulation). These tasks can be served by different peers and it is the index server's responsibility to dispatch streaming tasks to different streaming servers. Instead of having multiple peers to collaboratively serve media steaming for a client like [15], only one streaming server is needed at a time in PROP. The failure of the streaming server has little impact on the client, and the media player only needs to buffer one segment for a smooth playback. Furthermore, the streaming tasks can be dispatched fairly and efficiently based on the information in segment indices and the quality of the streaming servers. Currently, we only use the available bandwidth of the serving peer as the criterion to dispatch streaming tasks for load balance, and always dispatch streaming tasks to client peers first to decrease the proxy burden. We leave the dispatch optimization and data prefetching as future work.

3.6 Fault Tolerance

When a peer fails, both the media data it caches and the segment indices it maintains are lost. In PROP, each peer periodically checks the validation of the replica location links in the segment indices it maintains and simply removes dead links. The loss of segment indices can be recovered by using the recovery mechanism of distributed hash table, e.g., CAN supports multiple realities to improve fault tolerance of routing [23].

When the proxy fails or is overloaded so that it cannot fetch data for clients, the requesting peer connects to the media server directly, fetches data, and caches them locally until the proxy is recovered. Since the indices of media segments are distributed in the P2P system, the content locating mechanism still works and the system performance degrades gracefully. Compared with the solution of maintaining a central index in the proxy like the browser-aware proxy system [31], our system not only removes the single point of failure, but also significantly reduces the burdens of index maintenance and segment locating on the proxy.

4 Performance Evaluation

By using trace-driven simulation, we have comparatively evaluated the performance of our proposed system with different proxy cache sizes and different peer cache sizes, and showed the different roles of the proxy and peers. When the total cache size of peers is zero, the system is equivalent to a *proxy caching system*. When the cache size of the proxy is zero, the system is equivalent to a *client-based P2P system* without proxy server. In the following evaluations, if not specified, the default replacement policy on the proxy is popularity-based, and the default replacement policy on peers is utility-based for the PROP system, or LRU for client-based the P2P system, respectively.

We use the following metrics in our evaluations: The major metric, *streaming jitter byte ratio*, is defined as the amount of data that is not served to the client in time by the proxy and peers, thus causing the potential playback jitter on the client side, normalized by the total bytes all clients demand. The second metric is the *delayed start request ratio*, which is defined as the number of requests suffering startup delays, normalized by the total number of requests. These two metrics reflect the QoS of the media streaming to the client. The third metric we use is the *byte hit ratio*, which is defined as the total bytes of media data served by the proxy and peers, normalized by the total bytes of media data all peers consume. Byte hit ratio represents the storage utilization and outgoing network traffic reduction of the system.

4.1 Workload Summary

Table 1 outlines some properties of the workloads. REAL denotes the workload extracted from the server logs of HP Corporate Media Solutions, covering the period from 1 May through 10 May 2001. WEB and PART are two synthetic workloads for media viewing in the Web environment, where we assume the popularities of media objects follow Zipf-like distribution

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}, f_i = \frac{1}{i^{\alpha}}$$

and the request arrivals follow Poisson distribution

$$p(x, \lambda) = e^{-\lambda} \frac{\lambda^x}{x!}, x = 0, 1, 2 \ldots,$$

where i represents the file popularity rank and N denotes the total number of different files. λ is the average request arrival rate. These assumptions and the corresponding parameters for two synthetic workloads are based on the existing workload characterization studies [7], [8], which indicate that Internet media accesses exhibit substantially different characteristics from common Web accesses. Recently, we have further confirmed this distribution in [14]. WEB denotes complete viewing scenario while PART denotes partial viewing scenario in the Web environment.

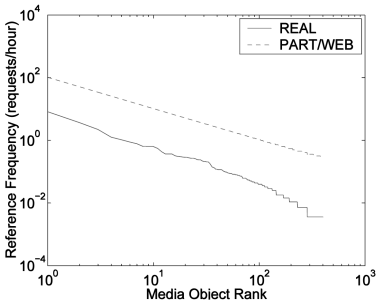

Fig. 4. The distribution of the reference frequency of media objects.

In PART, 80 percent of the requests only view 20 percent of the objects and then terminate.

The reference frequency of media objects in the three different workloads is shown in Fig. 4. REAL has a steeper Zipf distribution because the accesses in this workload are limited in the internal media servers of HP Corporation only (paper [7] presents the measurement and analysis of this workload in details). The reference locality of objects in the PART workload is the same as that in the WEB workload while the reference locality of segments is higher than that in the WEB workload because in the PART workload, the initial parts of media data are accessed more frequently than the latter parts. Thus, these three workloads represent three kinds of access patterns with different reference localities: For media segments, the order of reference localities is WEB $<$ PART $<$ REAL.

For the REAL workload, we set the bandwidth of the link between the proxy and media server as the clients' available bandwidths in the trace, ranging from 0 bps (unreachable) to 6.248 Mbps. The media encoding rate in the REAL workload ranges from 6.9 Kbps to 2.055 Mbps. For synthetic workloads, we choose the bandwidth of the link between the proxy and media server randomly from 0.5 to 2 times the encoding rate of corresponding media objects. In our simulation, the segment size for the REAL workload is 100 KB, and the segment size for the PART and WEB workload is 500 KB. We assume the bandwidth of the intranet is broad enough for media streaming. We also assume each peer is online randomly, and can serve at most five clients at the same time, which is the common configuration in P2P software such as BitTorrent [9]. The caches of the proxy and peers are empty at the beginning of our simulation, and each peer's cache size is proportional to the amount of media content it accesses in the workload. For client-based P2P system model without proxy caching, each peer fetches media data individually while it caches its accessed data and publishes them in the system in the same way as our proposed system.

4.2 Performance Results

To simulate the system performance in real environments, in our experiments, the peer arrival and departure to the system are assumed to be random. We believe this reflects (or at least it is close to) the behavior of real peers. Note that different from the Web accesses, in P2P systems, there is no consensus on the peer arrival and departure patterns. We first present the overall performance of PROP under different proxy cache sizes and peer cache sizes, then evaluate the performance improvement of our proposed proxy replacement and peer replacement policy.

4.2.1 Overall Performance

Fig. 5 shows the performance results of the three different system models (PROP, proxy caching, and client-based P2P system) on the REAL workload. We selected the proxy cache size as 10 percent, 3 percent, and 1 percent of the total accessed media data in the workload and ranged the total peer cache size from 0 to about three times the total accessed media data. In each figure, "no proxy caching" denotes the performance results of the client-based P2P system. The y axis denotes the performance results of the proxy caching system since the total peer cache size is zero. The results in Fig. 5 show that our proposed system can significantly improve the QoS of media streaming and the byte hit ratio with the increase of peer cache size. For example, compared with the proxy caching system, our system can reduce up to 87 percent streaming jitter bytes as shown in Fig. 5a, reduce up to 82 percent delayed start requests as shown in Fig. 5b, and increase the byte hit ratio up to 2.4 times as shown in Fig. 5c. This is because more media data can be cached in the system due to the more storage contributed by peers and, thus, improves both the byte hit ratio and QoS of media streaming correspondingly.

Fig. 5 also shows that the proxy plays an important role in P2P media streaming system. The QoS and the byte hit ratio are improved significantly with the increase of the proxy cache size. Compared with the client-based P2P

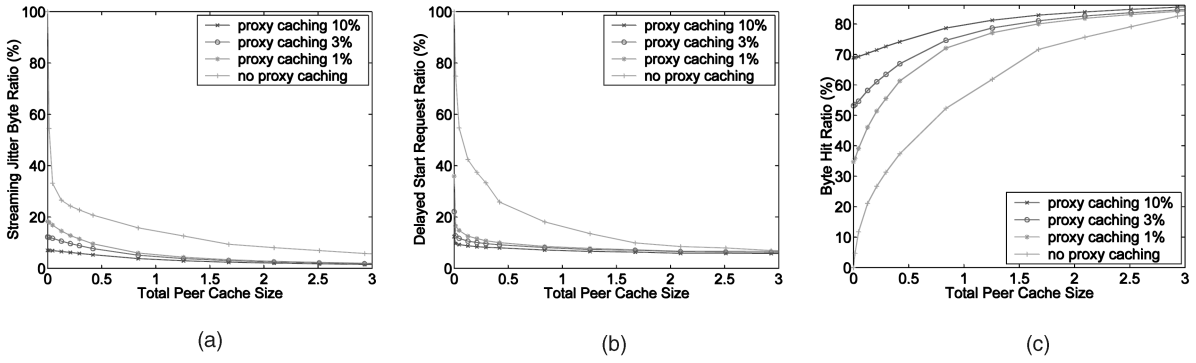

Fig. 5. Performance evaluation on the REAL workload. (a) Streaming jitter byte ratio. (b) Delayed start request ratio. (c) Byte hit ratio.

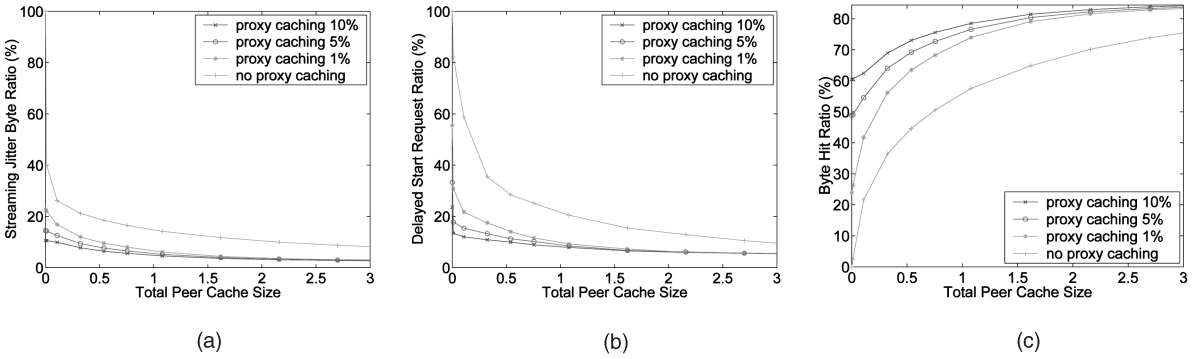

Fig. 6. Performance evaluation on the PART workload. (a) Streaming jitter byte ratio. (b) Delayed start request ratio. (c) Byte hit ratio.

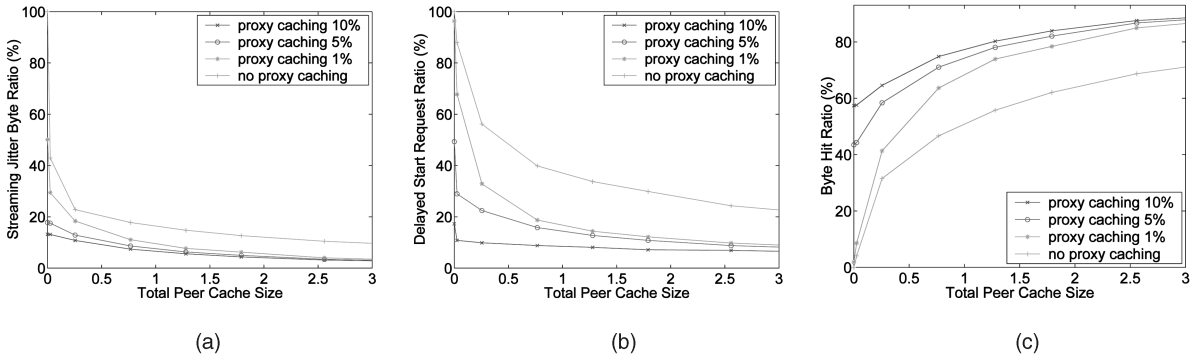

Fig. 7. Performance evaluation on the WEB workload. (a) Streaming jitter byte ratio. (b) Delayed start request ratio. (c) Byte hit ratio.

system, our proposed system with a proxy capable of caching 1 percent of all media data can reduce up to 93 percent of streaming jitter bytes, up to 88 percent of delayed start requests, and the byte hit ratio can be as high as 85 percent, as shown in Fig. 5a, Fig. 5b, and Fig. 5c, respectively. In our system, the streaming performance can be effectively improved as we increase the proxy size. The reason is that a peer is not a dedicated server. It comes and leaves randomly and can only serve a small number of clients simultaneously. Thus, it is possible that a client cannot be served immediately by peers that cache the requested data. On the other hand, the proxy provides a persistent cache and dedicated service to all clients; it takes over the streaming service whenever the peers are not capable of providing the streaming service, ensuring the quality of media streaming and reducing the outgoing bandwidth cost.

Fig. 6 and Fig. 7 show the performance results with the PART workload and the WEB workload, respectively. As shown in these figures, the performance results with these two workloads have similar trends to that with the REAL workload. At the first glimpse, it is surprising to find that the performance of workload WEB is slightly better than that of workload PART, which does not seem to be intuitive since the reference locality of media segments in the PART workload is greater than that in the WEB workload. Dissecting the two workloads, we find that, in the WEB workload, the total amount of bytes consumed by all clients is about 2.8 times greater than that in the PART workload, while the total amount of media objects is nearly the same as that in PART. Thus, the compulsory misses in the WEB workload are relatively smaller than those in the PART workload. Comparing the performance results of the PROP system evaluated by the three workloads, we can see that the proxy plays a more important role for workload with smaller reference locality. For example, under the same condition, when proxy cache size increases, the streaming jitter reduction of the WEB workload is up to four times greater than that of the PART workload, the delayed startup reduction of the WEB workload is up to two times greater than that of the PART workload, and the byte hit increase of the WEB workload is up to 64 percent greater than that of the PART workload. This confirms the effectiveness of our design of the PROP system: The proxy and its P2P clients should collaboratively provide media streaming service.

4.2.2 Proxy Load Changes

To further evaluate the collaboration and coordination between the proxy and its P2P clients, we have measured and observed the proxy load changes of our proposed system with different proxy cache sizes and peer cache sizes. The proxy load includes the total amount of bytes it serves peers and the total amount of bytes it fetches from the media server. Fig. 8a, Fig. 8b, and Fig. 8c show the proxy load for REAL, PART, and WEB, respectively. In these figures, the proxy load is normalized by twice of the total bytes all peers consume, the maximal load that is possible on the proxy. Increasing the cache size of peers can significantly reduce the proxy load due to the streaming service provided by peers in the system. The proxy load reduction is up to 72 percent for the REAL workload compared with the proxy caching system, as shown in Fig. 8a. We also observe that the proxy load decreases with the increase of the proxy cache size. This is because when

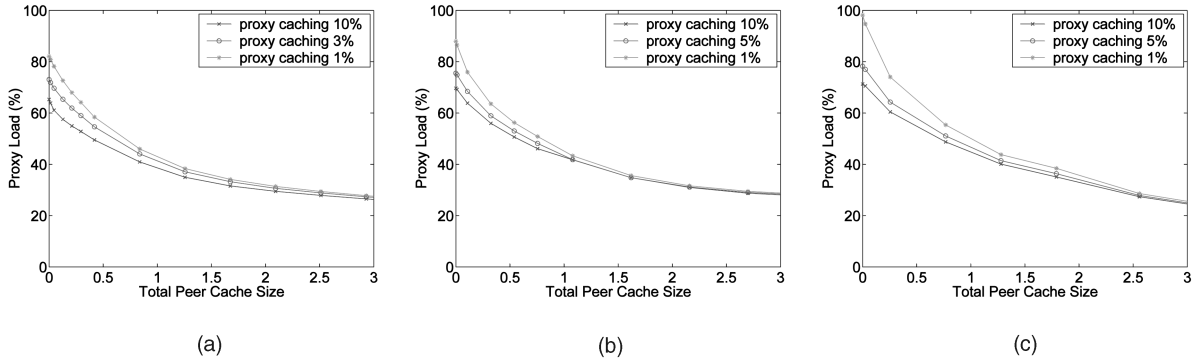

Fig. 8. Proxy load changes. (a) With the REAL workload. (b) With the PART workload. (c) With the WEB workload.

the proxy's cache size increases, more media data can be cached, which reduces the outgoing fetching load on the proxy.

Fig. 8b and Fig. 8c show similar trends when proxy cache size increases or peer cache size increases. Comparing the proxy load on the three different workloads, we also observe that the proxy cache size is more sensitive to workload with smaller reference locality. For example, under same conditions, when proxy cache size increases, the proxy load reduction of the WEB workload is up to 50 percent greater than that of the PART workload.

4.2.3 Proxy Replacement Policy

Several proxy caching techniques for media streaming have been proposed, such as exponential segmentation [30] and adaptive and lazy segmentation [5]. Both techniques use variable segment size to improve byte hit ratio and to reduce the cache management overheads. These techniques can be used in PROP as well. However, due to the service load taken by clients, the service load of the proxy in the PROP system is much smaller than that in traditional proxy caching systems. Therefore, finer granulated segmentation is feasible in the PROP system for better cache utilization. In this section, we compare the performance of popularity-based replacement with that of LRU replacement using uniform segment size to show the advantage of popularity-based replacement. Given the existence of many variants of the LRU-based replacement policies, we select LRU for comparisons because LRU is the most commonly used policy in the deployed proxy cache such as SQUID.[3] Due to this reason, we assume the LRU-based replacement policy for peers too in the next section.

Fig. 9a, Fig. 9b, and Fig. 9c show the comparisons of byte hit ratios between popularity-based replacement and LRU replacement for three workloads REAL, PART, and WEB, respectively. For PART, the byte hit ratio of popularity-based replacement is up to 23 percentage points higher than that of LRU (about three times of the byte hit ratio achieved by LRU). For WEB, the byte hit ratio improvement of popularity-based replacement over LRU is even higher, up to 26.5 percentage points (about 4.8 times byte hit ratio achieved by LRU). The improvement of the REAL workload is not so significant because the reference locality in this workload is relatively high, so that the potential improvement of byte hit ratio is relatively small. LRU replacement is not efficient for media streaming proxy because the data access pattern of media streaming is sequential in most cases, i.e., a user tends to view media from the beginning until the end. This access pattern is similar to file scanning, for which the inefficiency of LRU is well known [18]. With LRU, the latter segments of a media file will replace the initial segments when the cache is full, which is more likely to be accessed in the future. The popularity-based policy, on the other hand, performs replacement operations based on the probability of the future accesses of media segments, maximizing the utilization of proxy cache.

4.2.4 Peer Replacement Policy: Overall Performance

To further evaluate the efficiency of resource management of our proposed system, we compared the performance of our global caching with utility-based replacement on each peer to that of independent caching with LRU replacement on each peer, and then discussed the performance of different adjustment factors for utility function.

Fig. 10, Fig. 11, and Fig. 12 show the comparisons between the performance of the system with our utility-based peer replacement policy and that of the system with the LRU peer replacement policy, for the three workloads REAL, PART, and WEB, respectively. For the REAL workload, we only show the results of the system without proxy caching. For the PART and WEB workloads, we only show the results of the system with proxy caching 5 percent of the total accessed media data, using the popularity-based replacement policy on the proxy. For other proxy cache sizes, similar performance results are achieved.

From these figures, we can see the streaming jitter bytes and delayed start requests of the utility-based replacement policy decrease much faster than those of the LRU replacement policy, while the byte hit ratio of the utility-based policy increases much faster than that of the LRU policy, indicating our utility-based replacement policy is much more effective than the LRU policy. For the REAL workload, compared with the system using LRU, the system using the utility-based policy can reduce up to 36 percent streaming jitter bytes and up to 42 percent delayed start requests, and the improvement of byte hit ratio is as high as 59 percent, as shown in Fig. 10a, Fig. 10b, and Fig. 10c, respectively. For workload PART, the streaming jitter reduction and the delayed start request

3. http://www.squid-cache.org/.

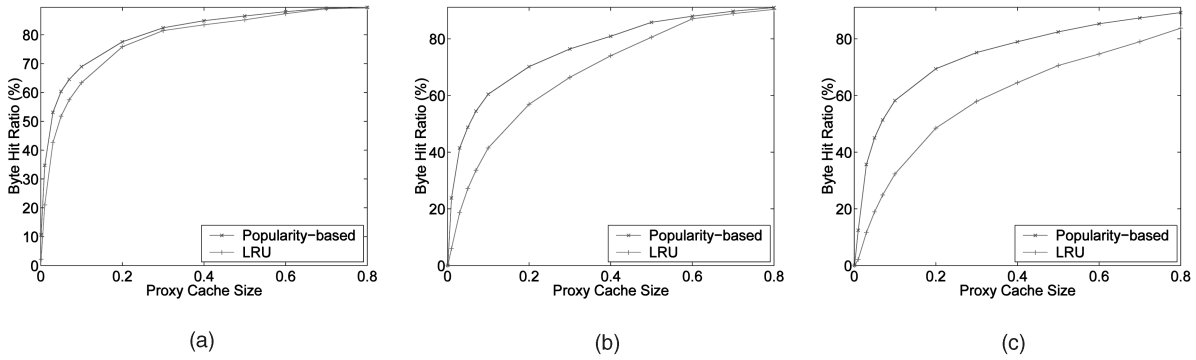

Fig. 9. The comparison between popularity-based replacement and LRU replacement. (a) With the REAL workload. (b) With the PART workload. (c) With the WEB workload.

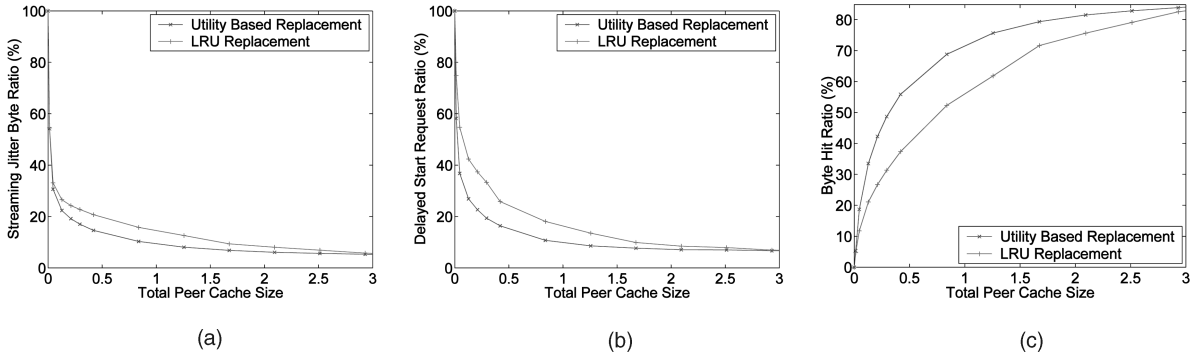

Fig. 10. Replacement policy comparisons on the REAL workload (without proxy caching in the system). (a) Streaming jitter byte ratio. (b) Delayed start request ratio. (c) Byte hit ratio.

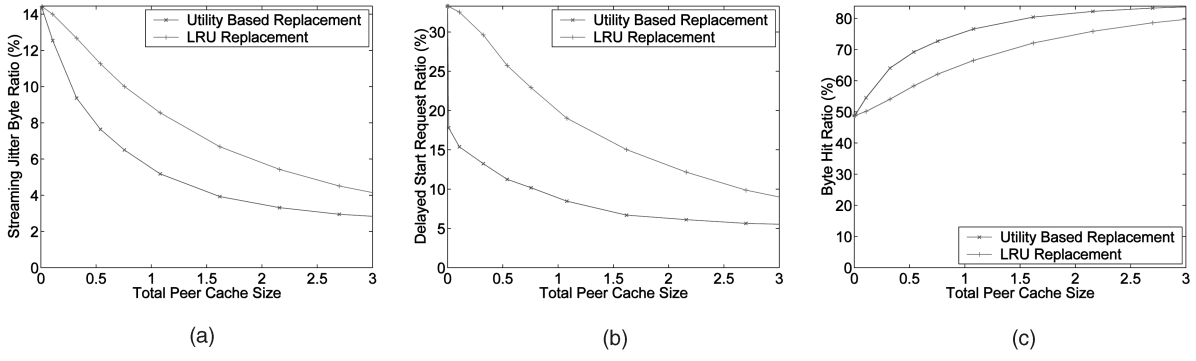

Fig. 11. Replacement policy comparisons on the PART workload (with proxy caching 5 percent of the total accessed media data in the system). (a) Streaming jitter byte ratio. (b) Delayed start request ratio. (c) Byte hit ratio.

reduction is up to 41 percent and 56 percent, respectively, and the improvement of byte hit ratio is up to 19 percent, as shown in Fig. 11a, Fig. 11b, and Fig. 11c, respectively. For workload WEB, the streaming jitter reduction and the delayed start request reduction is up to 57 percent and 61 percent, respectively, and the improvement of byte hit ratio is up to 24 percent, as shown in Fig. 12a, Fig. 12b, and Fig. 12c, respectively.

The ranking order of overall performance improvements for the three workloads is REAL $<$ PART $<$ WEB, in the reverse order of reference localities of the three workloads presented in Section 4.1, showing significant advantage of utility-based replacement. Meanwhile, although REAL has the highest reference locality in the three workloads, the maximal improvement of byte hit ratio of the REAL workload in Fig. 10c is higher than that of PART workload in Fig. 11c and WEB workload in Fig. 12c because there is no proxy caching for the case of REAL workload in Fig. 10c, evidencing the important role of proxy in our PROP system. The utility-based replacement policy can greatly reduce delayed start requests with only a small cache size in each peer because the initial segments of media objects are generally more popular than the latter segments and have a greater possibility to be cached. On the contrary, the initial segments are more likely to be replaced by LRU, with the progress of media viewing. At the same time, utility-based peer replacement policy takes into account both the popularities and the numbers of replicas of cached segments to maximize the storage utilization; therefore, it reduces the outgoing bandwidth consumptions and improves the QoS of media streaming.

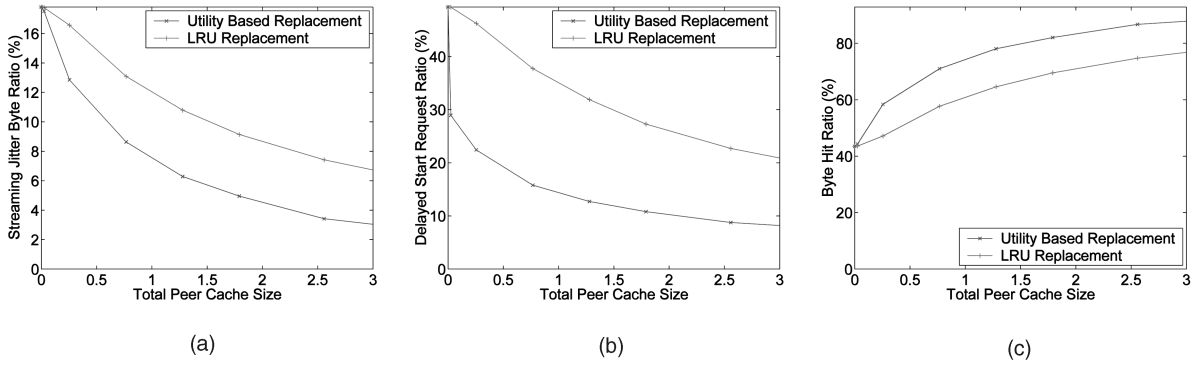

Fig. 12. Replacement policy comparisons on the WEB workload (with proxy caching 5 percent of the total accessed media data in the system). (a) Streaming jitter byte ratio. (b) Delayed start request ratio. (c) Byte hit ratio.

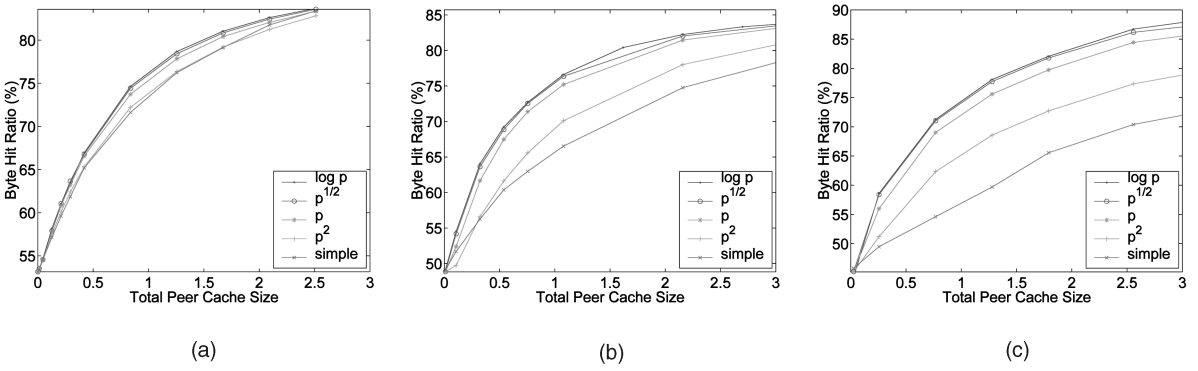

Fig. 13. The comparisons of different adjustment factors for utility function. (a) With the REAL workload. (b) With the PART workload. (c) With the WEB workload.

4.2.5 Peer Replacement Policy: Utility Function

Fig. 13a, Fig. 13b, and Fig. 13c show the byte hit ratios of systems with five different utility functions for workload REAL, PART, and WEB, respectively. For REAL, the proxy caches 3 percent of the total accessed media data. For PART and WEB, the proxy caches 5 percent of the total accessed media data. With the *simple* utility function, the system tries to cache media data as much as possible. Each segment is cached at most in one place, and the more popular segments have higher priorities to be cached. For other utility functions, the adjustment factors are p^2, p, $p^{1/2}$, and $\log p$, respectively, in which p is the segment popularity.

The results of all three workloads show the same ranking order for the performance of the five adjustment factors: $simple < p^2 < p < p^{1/2} < \log p$, in terms of byte hit ratio. For example, for WEB, the maximal byte hit ratio improvement with adjustment factor $\log p$ over that with adjustment factor the $simple$, p^2, p, and $p^{1/2}$ are 18.4, 9.51, 2.61, and 0.56 percentage point, respectively. For REAL and PART, the corresponding byte hit ratio improvements are smaller because the reference localities of these two workloads are greater than that of WEB, but the ranking orders are the same.

The system with $simple$ utility function achieves the worst performance because caching only one copy of each media segment cannot ensure data availability due to the random joining and leaving of clients. Utility functions with adjustment factors p^2, p, $p^{1/2}$, and $\log p$ cache multiple copies of media segments based on their popularities. Maintaining the number of copies of media segments proportional or polynomial to their popularities will quickly exhaust the available cache size, so that the segments with moderate popularities cannot be cached. Although significantly reducing the proxy's service burdens, maintaining too much replicas of popular segments is not efficient for storage utilization since these popular segments are more than enough in the system. On the other hand, the order of logarithmic function is lower than that of any polynomial function. PROP uses $\log p$ as the adjustment factor of the utility function, which reflects the number of segments replicas in the optimal distribution, as analyzed in Sections 3.3 and 3.4.2. Thus, the storage can be utilized more efficiently and the proxy burden can still be relieved with a small number of redundant popular segment replicas in the system. Our simulation results confirm the above analysis. As shown in Fig. 13, lower-level polynomial adjustment factors perform better than higher-level polynomial adjustment factors, but still not so good as logarithmic adjustment factors.

Fig. 14, Fig. 15, and Fig. 16 show the distribution of the number of replicas of media segments with different popularities under utility-based replacement policy on the REAL, PART, and WEB workload, respectively. For REAL, the proxy caches 3 percent of total accessed media data. For PART and WEB, the proxy caches 5 percent of total accessed media data. The x axis denotes segments, sorted by the corresponding popularity values. These figures show that the segment replica distribution resulted from our utility-based replacement is close to the optimal distribution of segment replicas based on (10) (a logarithmic function). The number of replicas for popular segments is a little smaller than the optimal one. However, this is not a

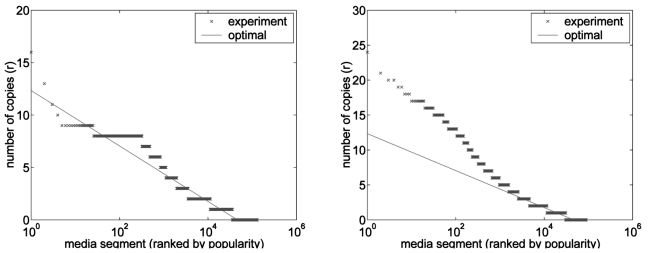

Fig. 14. The adaptiveness of utility-based replacement on the REAL workload. (a) With adjustment factor $\log p$. (b) With adjustment factor p.

problem in PROP since the proxy can take over the streaming requests. Furthermore, the popular segments can be quickly replicated in the system due to the high request rate. For replacement with adjustment factor p, the number of popular segments is much larger than the optimal value, preventing low popular segments from being cached. These trends are shown consistently in all three workloads. In general, although the data placement in PROP is passive (a client only caches the data it has viewed), our utility-based replacement can adaptively maintain the data cached in the system in the desired distribution for media streaming service.

4.3 Implications of Our Performance Evaluation

Our intensive trace-driven performance study provides insights into the PROP system design and impact of P2P sharing on the streaming system. We briefly summarize the implications of our performance evaluation.

- We have shown that our P2P assisted proxy system significantly improves the quality of streaming service mainly because the caching storage in PROP has been effectively and highly enhanced. Thus, media segments can be timely and smoothly delivered to any end user in the system either by other end users or/and by the proxy collaboratively.
- Our study shows that an increment of proxy cache size can be much more effective to the performance improvement of streaming content delivery than that of text-based Web content delivery. Our P2P approach well responds the high demand of large caching space for streaming contents.
- The involvement of P2P segment sharing among users has significantly reduced the traffic to the proxy. This provides us with an opportunity of making data replacement in the proxy at a fine grain level (a segment-based rather than an object-based method). We show that a segment-based replacement method can improve proxy cache utilization.
- One important factor to improve the efficiency of the PROP system is the coordination of segment replacement among peers and the proxy. Without such a mechanism, popular media objects/segments can be unnecessarily duplicated in both proxy and many client caches, wasting cache space. We show that segment replacement in each client based on our utility function effectively coordinates segment allocations in the global PROP caching space.

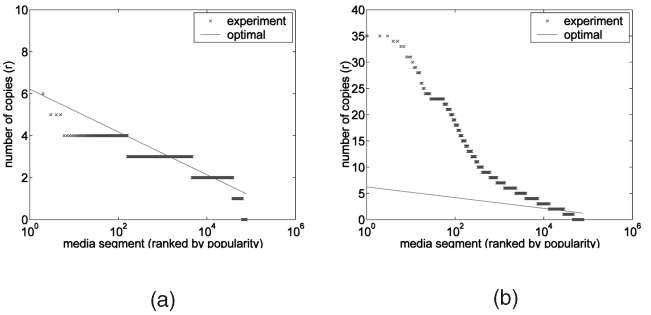

Fig. 15. The adaptiveness of utility-based replacement on the PART workload. (a) With adjustment factor $\log p$. (b) With adjustment factor p.

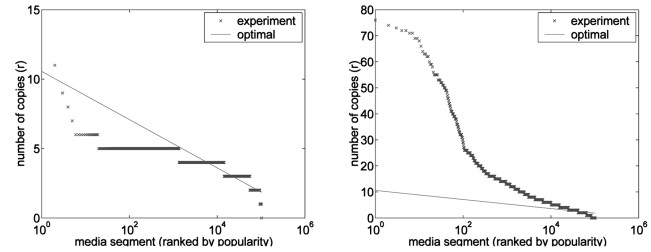

Fig. 16. The adaptiveness of utility-based replacement on the WEB workload. (a) With adjustment factor $\log p$. (b) With adjustment factor p.

5 Conclusion

Existing Internet streaming media delivering techniques are either based on a client-server model, such as, proxy caching and server replications by CDNs, or based on a client-based P2P structure. The disadvantage of the client-server model is its limited scalability and high cost, while the disadvantage of a client-based P2P system is its unreliable quality of streaming media delivery due to the dynamic nature of peers. In this study, we propose P2P assisted proxy systems to address these two limitations. In our system, the proxy is a member of the P2P network managed by the distributed hash table. In addition, the proxy also plays an important and unique role to ensure the quality of media delivery due to its dedicated and stable nature. To improve the cache utilization, we have proposed a model and designed the replacement policies for the collaboration and coordination between the proxy and clients accordingly, making the entire streaming media system both performance-effective and cost-efficient. We have conducted extensive experiments to evaluate various aspects of our design, and the results shows that our system can achieve 2.4 times improvement in byte hit ratio compared to client-based P2P systems and proxy-based P2P systems.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments. This work is supported in part by the US National Science Foundation under grants CCF-0129883, CNS-0405909, CNS-0509054/0509061 and a grant from the HP Labs. They also thank Theresa Long and Robert Richard Painter for reading this paper and their constructive comments. Some preliminary results of this work have been presented in [12].

REFERENCES

[1] S. Acharya and B. Smith, "Middleman: A Video Caching Proxy Server," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video,* June 2000.

[2] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and Implementation of a Caching System for Streaming Media over the Internet," *Proc. IEEE Real Time Technology and Applications Symp.,* May 2000.

[3] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura, "Silo, Rainbow, and Caching Token: Schemes for Scalable Fault Tolerant Stream Caching," *IEEE J. Selected Areas in Comm.,* vol. 20, no. 7, pp. 1328-1344, Sept. 2002.

[4] S.-H.G. Chan and F.A. Tobagi, "Distributed Server Architectures for Networked Video Services," *IEEE/ACM Trans. Networking,* vol. 9, no. 2, pp. 125-136, Apr. 2001.

[5] S. Chen, B. Shen, S. Wee, and X. Zhang, "Adaptive and Lazy Segmentation Based Proxy Caching for Streaming Media Delivery," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video,* pp. 22-31, June 2003.

[6] S. Chen, B. Shen, Y. Yan, S. Basu, and X. Zhang, "SRB: The Shared Running Buffer in Proxy to Exploit Memory Locality for Multiple Streaming Sessions," *Proc. IEEE Int'l Conf. Distributed Computing Systems,* pp. 787-794, Mar. 2004.

[7] L. Cherkasova and M. Gupta, "Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workloads," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video,* pp. 33-42, May 2002.

[8] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and Analysis of a Streaming Media Workload," *Proc. Third USENIX Symp. Internet Technologies and Systems,* Mar. 2001.

[9] B. Cohen, "Incentives Build Robustness in BitTorrent," *Proc. First Workshop the Economics of Peer-to-Peer Systems,* June 2003.

[10] Y. Cui and K. Nahrstedt, "Layered Peer-to-Peer Streaming," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video,* pp. 162-171, June 2003.

[11] S. Gruber, J. Rexford, and A. Basso, "Protocol Considerations for a Prefix-Caching for Multimedia Streams," *Computer Network,* vol. 33, nos. 1-6, pp. 657-668, June 2000.

[12] L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang, "PROP: A Scalable and Reliable P2P Assisted Proxy Streaming System," *Proc. IEEE Int'l Conf. Distributed Computing Systems,* pp. 778-786, Mar. 2004.

[13] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "Analysis of Multimedia Workloads with Implications for Internet Streaming," *Proc. World Wide Web Conf.,* pp. 519-528, May 2005.

[14] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "DISC: Dynamic Interleaved Segment Caching for Interactive Streaming," *Proc. IEEE Int'l Conf. Distributed Computing Systems,* pp. 763-772, June 2005.

[15] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "PROMISE: A Peer-to-Peer Media Streaming Using CollectCast," *Proc. ACM Multimedia Conf.,* pp. 45-54, Nov. 2003.

[16] A. Ip, J. Liu, and J. Lui, "COPACC: A Cooperative Proxy-Client Caching System for On-Demand Media Streaming," *Proc. Conf. Networking,* May 2005.

[17] S. Iyer, A. Rowstron, and P. Druschel, "SQUIRREL: A Decentralized, Peer-to-Peer Web Cache," *Proc. ACM Symp. Principles of Distributed Computing,* pp. 213-222, July 2002.

[18] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter-Reference Recency Set Replacement to Improve Buffer Cache Performance," *Proc. ACM SIGMETRICS Conf.,* pp. 31-42, June 2002.

[19] J. Kangasharju, F. Hartanto, M. Reisslein, and K.W. Ross, "Distributing Layered Encoded Video through Caches," *IEEE Trans. Computers,* vol. 51, June 2002.

[20] W. Ma and H. Du, "Reducing Bandwidth Requirement for Delivering Video over Wide Area Networks with Proxy Server," *Proc. Int'l Conf. Math. Education,* vol. 2, pp. 991-994, July 2000.

[21] Z. Miao and A. Ortega, "Scalable Proxy Caching of Video Under Storage Constraints," *IEEE J. Selected Areas in Comm.,* vol. 7, pp. 1315-1327, Sept. 2002.

[22] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video,* pp. 177-186, May 2002.

[23] S. Ratnasamy, P. Francis, M. Handley, and R. Karp, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM Conf.,* pp. 161-172, Aug. 2001.

[24] M. Reisslein, F. Hartanto, and K.W. Ross, "Interactive Video Streaming with Proxy Servers," *Proc. SPIE/ACM Conf. Multimedia and Computer Networking,* Feb. 2000.

[25] R. Rejaie and A. Ortega, "PALS: Peer-to-Peer Adaptive Layered Streaming," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video,* pp. 153-161, June 2003.

[26] S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," *Proc. IEEE INFOCOM Conf.,* vol. 3, pp. 1310-1319, Mar. 1999.

[27] R. Tewari, A.D.H. Vin, and D. Sitaram, "Resource-Based Caching for Web Servers," *Proc. SPIE/ACM Conf. Multimedia and Computer Networking,* Jan. 1998.

[28] D. Tran, K. Hua, and T. Do, "Zigzag: An Efficient Peer-to-Peer Scheme for Media Streaming," *Proc. IEEE INFOCOM Conf.,* Apr. 2003.

[29] B. Wang, S. Sen, M. Adler, and D. Towsley, "Proxy-Based Distribution of Streaming Video over Unicast/Multicast Connections," *Proc. IEEE INFOCOM Conf.,* June 2002.

[30] K. Wu, P.S. Yu, and J. Wolf, "Segment-Based Proxy Caching of Multimedia Streams," *Proc. WWW Conf.,* pp. 36-44, May 2001.

[31] L. Xiao, X. Zhang, and Z. Xu, "On Reliable and Scalable Peer-to-Peer Web Document Sharing," *Proc. IEEE Int'l Parallel and Distributed Processing Symp.,* Apr. 2002.

[32] Z. Zhang, Y. Wang, D. Du, and D. Su, "Video Staging: A Proxy-Server Based Approach to End-to-End Video Delivery over Wide-Area Networks," *IEEE Trans. Networking,* vol. 8, pp. 429-442, Aug. 2000.

Lei Guo received the BS degree in space physics and the MS degree in computer science from the University of Science and Technology of China in 1996 and 2002, respectively. He received the S. Park Graduate Research Award from the College of William and Mary in 2005. He is currently a PhD candidate in the Department of Computer Science and Engineering at the Ohio State University. His research interests are in the areas of distributed systems, peer-to-peer systems, multimedia systems, and Internet measurement. He is a student member of the IEEE Computer Society.

Songqing Chen received the PhD degree in computer science from the College of William and Mary. He is an assistant professor in the Department of Computer Science at George Mason University, Fairfax, Virginia. His research interests include high-performance computing, operating systems, and distributed systems. He is a member of the IEEE Computer Society.

Xiaodong Zhang received the BS degree in electrical engineering from the Beijing Polytechnic University in 1982, and the PhD degree in computer science from the University of Colorado at Boulder in 1989. He is the Robert M. Critchfield Professor in Engineering, and Chair of Department of Computer Science and Engineering at the Ohio State University. He served as the program director of advanced computational research at the US National Science Foundation, 2001-2004. He is the associate editor-in-chief of the *IEEE Transactions on Parallel and Distributed Systems*, and is serving on the editorial boards of the *IEEE Transactions on Computers* and *IEEE Micro*. He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.