Cooperative caching for multimedia streaming in overlay networks

Won J. Jeon and Klara Nahrstedt

Department of Computer Science,
University of Illinois at Urbana-Champaign,
Urbana, IL 61801, USA

ABSTRACT

Traditional data caching, such as web caching, only focuses on how to boost the hit rate of requested objects in caches, and therefore, how to reduce the initial delay for object retrieval. However, for multimedia objects, not only reducing the delay of object retrieval, but also provisioning reasonably stable network bandwidth to clients, while the fetching of the cached objects goes on, is important as well. In this paper, we propose our cooperative caching scheme for a multimedia delivery scenario, supporting a large number of peers over peer-to-peer overlay networks. In order to facilitate multimedia streaming and downloading service from servers, our caching scheme (1) determines the appropriate availability of cached stream segments in a cache community, (2) determines the appropriate peer for cache replacement, and (3) performs bandwidth-aware and availability-aware cache replacement. By doing so, it achieves (1) small delay of stream retrieval, (2) stable bandwidth provisioning during retrieval session, and (3) load balancing of clients' requests among peers.

Keywords: multimedia, peer-to-peer, overlay, caching, streaming, cooperative

1. INTRODUCTION

In order to address the bottleneck at the multimedia server (i.e., server distance may cause long start-up time and server load may cause low frame rate), in general, a cache or proxy is located between the server and the client and stores parts of multimedia streams originally located at the server. By exploiting temporal and geographical proximity, the cache reduces the server's overload and delay for retrieving streams from the server. Since multimedia streams require lots of memory/disk space for caching, naturally distributed caching schemes were introduced to address the scalability and size problems.

Recently, peer-to-peer (P2P) overlay networks have become popular and many P2P applications such as file sharing and content distribution have been introduced. One of the advantages of P2P overlay networks is their high feasibility of collaboration among peers without any modifications of the server side and the underlying network infrastructure. Naturally, P2P caching schemes for streaming were introduced. However, these approaches manifest (1) inefficiency of cache utilization due to inappropriate cache replacement algorithms, (2) lack of load balancing for peers with popular streams, and (3) no consideration of dynamic membership changes (e.g., due to high peer mobility).

In this paper, we present a novel overlay cooperative caching scheme for multimedia streaming service. Our scheme has the following advances over existing solutions. First, it efficiently utilizes cache space contributed by peers by maximizing cache availability of streams in the cache peer community, and therefore decreases the delay for stream retrieval. Second, it is tolerant with respect to dynamic membership changes such as peer up/down events by considering node availability.

The rest of the paper is organized as follows. In Section 2, we introduce the general cooperative overlay caching and its components. Then, in Section 3, we discuss our caching algorithm. Simulation results show the performance of our algorithm and validate the advances of our algorithm in Section 4. Section 5 describes the related work. Finally, Section 6 concludes the paper.

Further author information: (Send correspondence to Won J. Jeon)
Won J. Jeon: E-mail: wonjeon@cs.uiuc.edu, Telephone: 1 217 333 1515
Klara Nahrstedt: E-mail: klara@cs.uiuc.edu

2. COOPERATIVE OVERLAY CACHING

The cooperative P2P community consists of peer clients that have computing power, storage space, and network connections to shared networks, such as the Internet. Each peer joins and leaves the community with an appropriate authentication procedure. Once accepted in the community, the peer exploits other peers' storage and network resources, and, at the same time, contributes its storage space and network bandwidth to the community. This architecture (as shown in Figure 1) comprises *the cooperative overlay caching space*. When a specific peer tries to retrieve a new multimedia stream such as audio and video from a server, it first looks up the storage space of its peers in its community as shown in Figure 1.* If the stream is found in the community, the client retrieves the stream from its peer(s), not from the server. After retrieving the stream, the client determines whether it caches the stream in its own storage space or not. If the client always stores (caches) the stream, then there might be excessive duplicates of the same stream in the community, which might cause inefficient utilization of shared cache space in the community. If the client does not store the stream in the case of cache hit in the community, then all requests from different peers in the community might be forwarded to the same peer which has the requested stream. This might cause longer delays for stream retrieval and high service unavailability, due to limited service capability affected by computing power and network bandwidth. In addition, any peer in the community could leave without any notification and a peer might sometimes be down for maintenance. Therefore, the following criteria should be considered for designing and maintaining cooperative overlay caching.

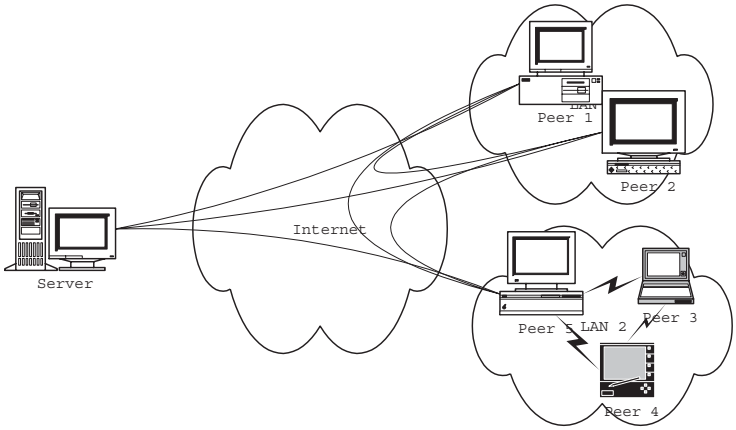

Figure 1. Cooperative P2P Community

- *Creation and maintenance of community:* Peers in a cooperative P2P community could be either predefined in a specific geographical region, such as community network by residential homes, or widely spread in wide-area-networks, such as the Internet. In order to have a tolerable delay for stream retrieval, any two peers in a community should be located within the distance of a tolerable delay d_t. We assume that there is an underlying P2P substrate to measure the distance of peers in terms of network delay, so that any stream retrieval in a community does not have an excessive delay longer than d_t.[7]

- *Handling of peer up/down events:* Since peers are possibly maintained by different maintenance authorities, stable membership in a community is no longer valid. Especially in a heterogeneous computing environment, peers could be mobile devices, which might frequently join or leave the community. The cooperative community should handle this peer's behavior without causing a malfunction or breakdown of the whole system.

*Note that the server's distance from any of the P2P community peers is an order of magnitude higher than the peers' distance from each other. For example, a situation of a server located in Japan and all peer nodes being located at UIUC would satisfy our assumption.

- *Search and lookup capability:* When any cache entry in a peer is changed, the other peers in the community should know this change for lookup in the future upon its stream request. There might be either centralized or distributed ways of cache lookup schemes. In a centralized scheme, there is a specific peer that keeps a record of cache entries of peers in a community. In a distributed scheme, a peer finds cached streams in its peers by flooding or other scalable query/discovery methods.

- *Cache allocation and replacement:* When a stream request is issued by a peer in a community, the community then determines whether this new stream should be cached or not, and if cached, where it should be cached. If there is not a great enough number of peers having popular streams, then these peers would be congested with other peers' cache retrieval, therefore suddenly becoming a performance bottleneck. On the other hand, if there is an excessive number of duplicates of popular streams in a community, it is possible that there is not enough space for unpopular streams, so that the retrieval of such streams might struggle with a longer delay.

- *Fair utilization of shared resources:* Each peer might contribute its resources, such as cache space and serving network bandwidth, to its community. In this scenario, differently contributed resources should be allocated fairly to the group of peers, so that the community minimizes inefficient utilization of cache space and network bandwidth.

3. COOPERATIVE CACHING ALGORITHM

3.1. Assumptions, Models, and Definitions

Suppose a multimedia stream consists of a series of segments (e.g., a segment could be group of pictures (GOP) in MPEG, or a frame in MJPEG video), and a segment is the base unit of cache allocation. Since the size of a multimedia stream spans from a few seconds to a few hours, this segment-based caching has been recommended in many previous works.[4,5,6] In addition, segment-based caching is more compatible with VCR operations such as *pause, fast-forward, backward* for the video-on-demand service, and is regarded more tolerant in a mobile and ubiquitous computing environment. Here without loss of generosity, a segment of each stream is denoted as s_i $(i = 0, 1, ..., I)$ (with the same size), where I represents the total number of segments in the whole stream. In a P2P community group, each peer is denoted as p_j $(j = 0, ..., J)$, where J is the total number of peers in the P2P community.

If a peer p_j wants to retrieve a stream segment s_i from the community and play it locally, then the delay d_j for the retrieval is calculated as:

$$d_j = A(s_i)d_{j*} + (1 - A(s_i))d_{j\infty} \tag{1}$$

where $A(s_i)$ is the availability of the segment s_i in the community ($A(s_i) \in [0, 1]$), d_{j*} is the average delay from the peer p_j to any peer in the community ($d_{j*} \leq d_t$, where d_i is the maximum delay between any pair of peers in a P2P community), and $d_{j\infty}$ is the delay from p_j to the server which originally stores the segment s_i. Since $d_{j*} < d_{j\infty}$, maximizing $A(s_i)$ is to minimize d_j.

The availability of the segment s_i, $A(s_i)$ in a community is determined by the node availability, link capacity of the peer, and the number of duplicates of the segment:

$$A(s_i) = 1 - \prod_{j=1}^{J}(1 - q_j a_j X_{ij}) \tag{2}$$

where q_j is the probability of availability of p_j $(0 \leq q_j \leq 1)$, a_j is the admission probability of request for s_i at p_j $(0 \leq a_j \leq 1)$, and X_{ij} is the $I \times J$ matrix with the element x_{ij} where:

$$x_{ij} = \begin{cases} 1 & \text{if } s_i \text{ is cached at } p_j, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

To simplify, if the admission probability a_j is determined by the capacity of servicing network bandwidth, then a_j can be expressed as the servicing (outgoing) network bandwidth, normalized by the maximum servicing network bandwidth:

$$a_j = \frac{b_j}{max(b_1, b_2, ..., b_J)} \tag{4}$$

If the node availability q_j with its servicing link bandwidth a_j is regarded as a single parameter, Q_j, then Equation 2 is rewritten as:

$$A(s_i) = 1 - \prod_{j=1}^{J}(1 - Q_j X_{ij}) \tag{5}$$

The purpose of our caching algorithm is to increase $A(s_i)$ for all s_i in the community, therefore minimizing d_j. However, since cache space in the community is limited, it is not possible to increase $A(s_i)$ for all s_i.

3.2. Availability-based caching Replacement Algorithm

When the entire cache space of the community is occupied, then an appropriate *cache replacement scheme* should select a victim stream which is evicted in the cache space and replaced with the stream newly requested by peers. Traditional cache replacement algorithms such as Least Recently Used (LRU), utilizing temporal locality, have been known to be less effective for stream caching than data caching, due to its lack consideration of stream size.[19] Usually the size of streams is large, varying a few kilobytes to a few hundred megabytes. In this case, an unpopular large stream could evict a bunch of small popular streams in the cache, which is not good for the performance of caching. In this section, we will explain our *availability-based caching replacement algorithm* in the distributed and cooperative environment. First, we look at the homogeneous environment where each peer has the same probability of availability (e.g., $Q_j = Q, \forall j \in J$). Then the following theorem is derived:

Theorem 1. If the probability of availability of all peers is the same (e.g., $Q_j = Q, \forall j \in J$), and there is a (cache) capacity constraint at peers, then the average availability for a stream segment ($\sum_{i=1}^{I} A(s_i)$) is constant at the steady state (e.g., cache space is completely occupied with cached streams).

Proof. With $Q_j = Q(\forall j \in J)$, $\frac{1}{I}\sum_{i=1}^{I} A(s_i)$ is:

$$\begin{aligned} \frac{1}{I}\sum_{i=1}^{I} A(s_i) &= \frac{1}{I}\sum_{i=1}^{I}(1 - \prod_{j=1}^{J}(1 - QX_{ij})) \\ &= 1 - \frac{1}{I}\sum_{i=1}^{I}\prod_{j=1}^{J}(1 - QX_{ij}) \end{aligned} \tag{6}$$

Since there is a capacity constraint in the cache space, which means $\sum_{j=1}^{J} C_j < \sum_{i=1}^{I} size(s_i)$, where C_j is the cache size of the peer p_j and $size(s_i)$ is the size of segment s_i, there is at least one X_{ij} which is zero. In equation 6, we select two elements, $X_{i_1 j_1}$ and $X_{i_2 j_2}$, which are 1 and 0, respectively, and then flip them into 0 and 1. The resulting $\sum_{i=1}^{I} A(s_i)$ is not changed by this flipping when $\sum_{i=1}^{I}\sum_{j=1}^{J} X_{ij}$ is constant, which means that, in steady state, any reallocation of any cached stream in the community does not affect the total availability of streams. □

From Theorem 1, the following corollary is derived.

Corollary 2. If the probabilities of availability of all peers are the same, and a caching replacement algorithm replaces the segment s_i with $s_{i'}$, then the availabilities of the segments except s_i and $s_{i'}$ are not changed.

From Equation 1 and Theorem 1, in order to have a minimized delay for retrieval of a segment s_i, increasing the minimum $A(s_i)$ for s_i without decreasing other $A(s_{i'})$ (where $s_{i'}, \forall i' \in \{1, 2, ..., I\}, i' \neq i$) too much, is a key factor. In reality, the availability of each peer (Q_j) might dynamically change over time, so we propose the following algorithmic approach to achieve the minimum delay for the stream retrieval.

Algorithm for Cooperative Cache Replacement

Upon the request for the stream $\{s'_1, s'_2, ..., s'_K\}$,

```
1   for k = 1 to K
2       Replace_segment(s'_k);
```

Function *Replace_segment*(s'_i)

```
1   A_min := ∞, i_min := 0;
2   Δ_max = 0, i_max = 0; j_max = 0;
3   for i = 1 to I
4       if (A_min ≥ A(s_i)) then
5           A_min := A(s_i);
6           i_min := i;
7   for (i = 1 to I) and (j = 1 to J)
8       Calculate A'_j(s_i), ∀s_i with possible replacement and updated Q_j.
9       δ_inc,j = A'_j(s_{i_min}) − A_j(s_{i_min});
10      δ_dec,j = Σ_{i≠i_min} (A'_j(s_i) − A_j(s_i));
11      Δ_ij = δ_inc,j − δ_dec,j;
12      if (Δ_ij > 0 ∧ Δ_max < Δ_ij) then
13          Δ_max := Δ_ij;
14          i_max = i;
15          j_max = j;
16  if (i_max ≠ 0) then
17      Replace s_{i_max} located at p_{j_max} with s'_i.
```

In the algorithm, specifically on line 8, 'possible replacement' represents all the cases when s_i ($i \neq i_{min}$) in any peer p_j, $\forall j \in J$ is replaced with $s_{i'}$ ($i' = i_{min}$). In this case, X_{ij} which was one, changes to zero; and $X_{i'j}$ which was zero, changes to one, after the replacement. Note that this algorithm measures exactly the availabilities of all stream segments from all peers, but it has $O(I^2J)$ time complexity and $O(J^2)$ message exchange for X_{ij} among peers upon every change of cache entries, which might be expensive when there are many segments and peers.

3.2.1. Example of *Replace_segment*

Suppose that there are three peers p_1, p_2 and p_3 in a P2P cache community. Each peer has a two-segment size of cache space. p_1, p_2, p_3 has its availability (Q_j) of 1.0, 0.8, 0.6, respectively. In steady state, let us assume that p_1, p_2, p_3 caches the segment sets $\{s_1, s_2, s_3\}$, $\{s_3, s_4, s_5\}$, $\{s_4, s_6, s_7\}$, respectively. In this configuration $Conf_1$, the availabilities of each segment are calculated from Equation 5:

$Conf_1 : A(s_1) = 1.0, A(s_2) = 1.0, A(s_3) = 1.0, A(s_4) = 0.92,$
$A(s_5) = 0.8, A(s_6) = 0.6, A(s_7) = 0.6, A(s_8) = 0.0$

Later, let us suppose that there is a new segment request for s_8 at p_2. Then, p_2 locally computes the change of availabilities in the 9 possible replacement cases. Among them, the replacement of s_4 at p_2 with s_8 is the case when Δ on the line 11 in the above algorithm will be Δ_{max}.

$Conf_2 : A(s_1) = 1.0, A(s_2) = 1.0, A(s_3) = 1.0, A(s_4) = 0.8,$
$A(s_5) = 0.8, A(s_6) = 0.6, A(s_7) = 0.6, A(s_8) = 0.6$
$Conf_1 \rightarrow Conf_2 : \delta_{inc} = 0.6, \delta_{dec} = 0.92 - 0.8 = 0.12, \Delta = 0.48$

Therefore, s_4 at p_2 is replaced by s_8.

3.3. Approximation of Algorithm

In reality, it is not possible for each peer to have the same probability of availability, and to keep a record of all cache entries located at neighbor peers, and to exchange every updated information of X_{ij} upon every change of cache entries. Instead, it is more feasible in terms of the number of exchanging messages for each peer to ask the number of duplications of a segment s_i in the community from the subset of neighbor peers. This type of message exchange and information gathering is more compatible with popular epidemic protocols[25]in P2P networks.

Suppose that, at a given time, peer p_j has $\tilde{n}$ number of close neighbor peers, and has only the information about the number of duplications of the segment s_i from them. Originally, the availability of the segment s_i, $A(s_i)$ is given in Equation 5. Recall that the probability of availability of a peer p_j, Q_j is a parameter which is measured by its neighbor peers, whereas X_{ij} is the information that the peer p_j sends to its neighbors. To plug the information about the number of duplicates of the segment into this formula, let us sacrifice the accuracy of the Q_j at this point. We will compensate for the error caused by this approximation of Q_j by selecting a victim peer later in this section. The average of Q_j, $\bar{Q}$ is given as:

$$\bar{Q} = \frac{\sum_{j=1}^{J} Q_j}{J} \tag{7}$$

Then, the availability of the segment s_i is derived from Equation 5:

$$\begin{aligned} A(s_i) &= 1 - \prod_{j=1}^{J} (1 - \bar{Q} X_{ij}) \\ &= 1 - (1 - \bar{Q})^{\tilde{N}(X_i)} \end{aligned} \tag{8}$$

Where the approximation of the number of duplicates of the segment s_i in the community is given as ($\tilde{J}$ is the number of subset neighbor peers which exchange messages with the peer p_j):

$$\tilde{N}(X_i) = \frac{J}{\tilde{J}} \sum_{j \in \tilde{J}} X_{ij} \tag{9}$$

Equation 8 means that, from the approximation of Q_j and incomplete information of X_{ij}, the availability of a stream segment can be calculated. Based on Equation 8, the approximation of cooperative caching replacement is as follows. Note that this algorithm only selects the victim segment s_i, not the exact peer location.

Approximation Algorithm for Selecting Victim Segments

Upon the request for the stream $\{s'_1, s'_2, ..., s'_K\}$,

$S_{victim} = \emptyset$; $s_{victim} = 0$;
for $k = 1$ **to** K
    $s_{victim} \leftarrow$ *Find_Victim*(s'_k);
    $S_{victim} \leftarrow S_{victim} \cup s_{victim}$;

Function *Find_Victim*(s'_i)

$A_{min} := \infty, i_{min} := 0$;
$\Delta_{max} = 0, i_{max} = 0$;
for $i = 1$ **to** I
    if $(A_{min} \geq A(s_i))$ **then**
        $A_{min} := A(s_i)$;
        $i_{min} := i$;
for $i = 1$ **to** I
    Calculate $A'(s_i), \forall s_i$ from Equation 8.
    $\delta_{inc} = A'(s_{i_{min}}) - A(s_{i_{min}})$;
    $\delta_{dec} = \sum_{i \neq i_{min}} (A'(s_i) - A(s_i))$;
    $\Delta_i = \delta_{inc} - \delta_{dec}$;
    if $(\Delta_i > 0 \wedge \Delta_{max} < \Delta_i)$ **then**
        $\Delta_{max} := \Delta_i$;
        $i_{max} = i$;
Return s'_i.

Once a victim segment s_{victim} is selected by this approximation algorithm, there might be more than one peer that caches the segment s_{victim}. When there are multiple peers that cache the victim segment s_{victim}, there might be different policies to select a specific peer having s_{victim}. This peer might be either one with high availability, one with high network bandwidth, or one that is randomly selected. The peer which would be selected here will initiate a transmission of s_{victim} from the peer p_j to itself. Therefore, in order to decrease delay for stream retrieval, selecting a peer which has large available network bandwidth and high availability is a key factor. The following is our proposed peer selection algorithm which will perform a cache replacement algorithm for each s_{victim}, for the segment selected by the previous victim segment selection algorithm.

Approximation Algorithm for Cache Replacement

For each s'_i, among the candidate peers $\{p_j\}$ $(j \in J')$ having each s_{victim},

$B'_{max} = 0, j_{max} = 0$;
for $j = 1$ **to** J'
    $B'_j = Q_j B_j$;
    if $(B_{max} < B'_j)$ **then**
        $B'_{max} := B'_j$;
        $j_{max} := j$;
Replace s_{victim} with s'_i at $p_{j_{max}}$.

In this algorithm, J' represents the number of candidate peers with s_{victim} $(J' \subseteq J)$, B_j represents the currently available network bandwidth serving neighbor peers, and B'_j represents the calibrated network bandwidth with availability Q_j.

3.4. Caching Protocols

The approximation of victim segment selection and the cache replacement in previous sections assume that there is a P2P substrate that not only measures the distance between every peer to form a cache community, but also measures the probability of availability Q_j. This information might be the ratio of blackout time of the peer p_j over total execution time of the system, which is measured by its neighbor peers. Q_j is not a specific information different to peers, but a global information in the community. Therefore, in order to reduce the flooding of this information, a particular peer could measure this information for only a subset of peers in the community and could deliver it to the other peers which does not have the information, with any epidemic protocol.

Once a particular peer p_j receives a request for a stream segment s'_i, then p_j selects the victim segment s_i based on Q_j from its neighbor peers. Once the victim segment is selected by p_j, then it selects its neighbor peer among candidate peers p_j having s_{victim}, based on the information of calibrated network bandwidth, B'_j. These two algorithm are locally executed at p_j based on the information of Q_j and $\tilde{N}(X_i)$. The real transmission of s'_i is performed from p_j to the one of candidate peer, $p_{j_{max}}$, after the playback of s'_i is done at p_j.

4. SIMULATION

In this section, we compare our availability-based caching replacement algorithm with the segment-based LRU algorithm. We measure the average delay of segments of streams for our non-approximated cache replacement algorithm, then compare it with the approximated version of the algorithm and the segment-based LRU algorithm.

4.1. Simulation Methodology

To evaluate the performance of the caching schemes, we implemented them in our event-driven distributed caching simulator, given a multimedia workload which contains the information of each peer's access to multimedia streams, the simulator performs one of distributed cache replacement algorithms in a predefined network topology. In this simulation, 100 peers were generated in a P2P cache community, and each peer had 100 MB of cache space. Delays between any pair of peers are randomly selected to be less than 50ms (e.g. $d_t = 50ms$). When there is a cache miss for a particular segment in the community, the delay to accessing a multimedia server to retrieve the segment was set to be $[100ms, 300ms]$. The inbound and outbound network bandwidth of each peer ware set to 5 Mbps. When there are multiple streaming sessions among peers, the available network bandwidth is proportionally utilized for each session. If there is a peer overloaded with requests for cache retrieval from other peers, then it is congested and causes a longer delay for cache retrieval.

We generated a multimedia workload using *MediSyn*, the synthesized multimedia workload generator.[22] One of the characteristics of this workload is to allow for variable session duration time, which could have a similar effect as the *stop* VCR operation. This feature simulates the case when a peer stop the request for transmission of the stream at any time during a streaming session. The default configuration parameters were used, except that the trace duration was modified to four days. Table 1 shows a brief summary of the multimedia workload used in this simulation. In addition, we allocated each request in the workload trace to one peer in the community randomly.

4.2. Simulation Result

4.2.1. Effect of Segment Size:

In order to investigate the effect of the segment size of multimedia streams on the cache performance, we had different segment sizes (50kB, 100KB, 500KB, and 1MB) and performed the simulation. In this simulation, the availabilities of peers (Q_j) were set to 1.0. After confirming each peer's steady state (e.g., its cache space is fully occupied), the delay for each retrieval of stream segment was recorded. Figure 2 shows the results of different algorithms in one peer in the community. Note that the other peers in the community showed similar statistics.

ORIGIN, APPROX, and LRU represent the non-approximated algorithm, approximated algorithm, and LRU algorithm, respectively. As the segment size gets bigger, the average delay becomes longer. This is because, as the segment size becomes bigger, the number of cache slots in each peer decreases. In addition, our approximated

Table 1. Summary of Multimedia Workload for Simulation

Parameters	**Values**
Duration of simulation	4 Days
Total number of requests	101464
Average access rate	0.29 (1/sec)
Total number of streams	1000
Average stream size	5.992 MB
Max. stream size	258.83 MB
Min. stream size	7 KB
Stream encoding rates	28.8,56,128,256,350 kbps

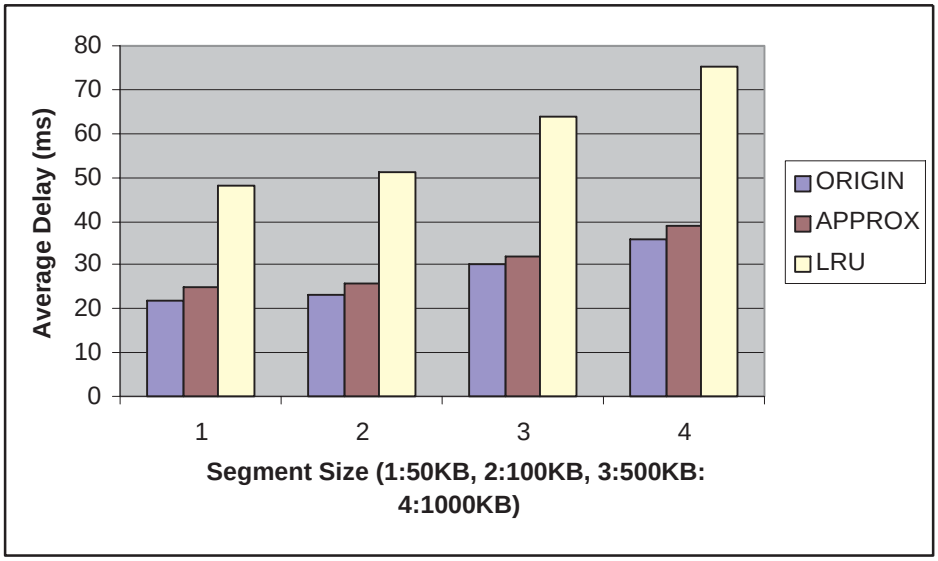

Figure 2. Effect of Segment Size on Cache Performance

algorithm has a very similar result to the non-approximated one, and the LRU algorithm shows the longest delays in all segment sizes. This can be explained by the access pattern in the workload trace. The multimedia workload that we used has a popularity distribution for each stream access (e.g., Zipf-like distribution), so that a small portion of popular streams have dominant access frequencies over unpopular streams. In this configuration, especially with a large cache segment size, an unpopular stream which is requested infrequently are prone to be evicted by popular streams. However, our availability-based cache replacement algorithm shows that it tries to keep the minimum availability of unpopular segments, which decreases the average delay for stream retrieval.

4.2.2. Effect of Peer Availability:

In order to see the effect of peer availability (Q_j), different probabilities of peer availability were set up: 0.6, 0.8, 0.9, and 1.0. The segment size was set to 100 KB. Each peer performs its operation only in its available time period, and does not have any response from its neighbors in its blackout time. The start times of this blackout period were randomly assigned to each peer. Figure 3 shows the average delays for stream retrieval with three different algorithms. Again, these statistics are gathered in one of peers in the community, and the rest of the peers showed a similar performance.

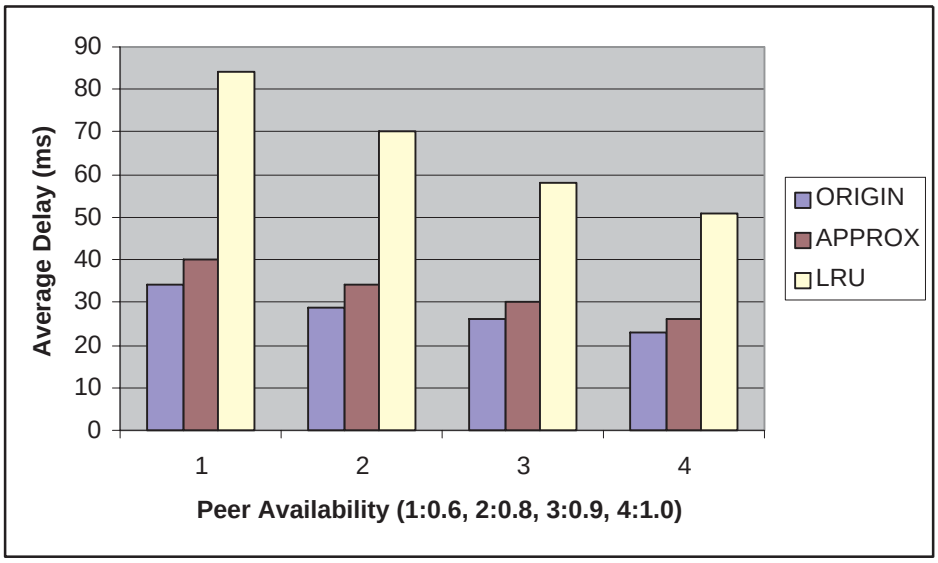

Figure 3. Effect of Peer Availability on Cache Performance

Figure 3 shows how our cache replacement algorithm is tolerant to the peer up/down event. Again, the LRU algorithm shows the worst performance in this scenario, because it has a very diverse range of duplications for each segment of streams; therefore, the segments with less availabilities ($A(s_i)$) are easily unreachable to neighbor peers, which might cause longer delays for stream retrieval.

5. RELATED WORK

Different from traditional data caching, multimedia caching for streaming multimedia at the proxy was proposed by different researchers.[8,9,12] Segment-based caching was investigated and supported, particularly for multimedia caching in different works.[4,5,6,10,20] However, these were discussed only in the single cache scenario, therefore lacking cooperative schemes such as replication and load balancing problems. Various interval-based caching schemes were also introduced.[14,15] However, these schemes are closer to the buffer management scheme at the proxy, which do not have much consideration of requests for multiple streams by users.

The limitation of a single cache was addressed in several works.[18,17] Distributed caching has been mostly discussed in data caching including web caching, and have either hierarchical[16] or flat structures.[23] Recently introduced P2P caching schemes[24] mostly have flat structures. Different P2P caching schemes generally focuse on search algorithms and replication techniques, but have little consideration for network distance and bottleneck problem. Moreover, these work are primarily for web caching, but there is little work for multimedia caching in a P2P overlay environment.[1]

Work by Kangasharju *et al.*[11] and Yu *et al.*[26] discusses the optimal duplication of data objects in P2P communities and the minimum-cost replication problem, respectively, that might have similar configurations for our caching scheme, however, they lack of consideration of multimedia streaming issues.

Streaming media workload has also been discussed by Chesire *et al.*[21] and Tang *et. al.*[22] Even though the traces they gathered from multimedia servers help to understand the access behavior of multimedia clients, the pattern of access/query in P2P overlay networks is not clearly investigated yet.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed our novel distributed cooperative caching algorithm for multimedia streaming, supporting a large number of peers over peer-to-peer overlay networks. In order to facilitate multimedia streaming

and downloading service from servers, our caching scheme determines the appropriate availability of cached stream segments in a cache community, determines the appropriate peers for cache replacement, and performs bandwidth-aware and availability-aware cache replacement. In doing so, it achieves a small delay for stream retrieval, stable bandwidth provisioning during retrieval session, and load balancing of clients' requests among peers.

In our future work, we will consider non-cooperative environments. In P2P overlay networks, where each peer voluntarily joins or leaves freely, the definition of 'fairness' is not well defined. In previous work without consideration of each peer's rational behavior, based on economic incentive, each peer is assumed to be altruistic and willingly contributing its resources such as cache space and network bandwidth, to the community of peers. However, this has been shown not to be true in current operating P2P networks for file sharing applications.[2] In host cases, more ore than half of the peers do not contribute their resources to the P2P community, which might be contradictory to ordinary P2P configurations. Therefore, motivating peers to participate in a community and guaranteeing the fairness based on their resource contribution is very important and is currently being investigated.

REFERENCES

1. M. Tran and W. Tavanapong, "Overlay caching scheme for overlay networks," in *Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN 2003)*, 2003.
2. E. Adar and B. Huberman, "Free riding on GnuTella," *Technical Report*, Xerox PARC, August 10, 2000, First Monday.
3. V. Padmanabhan, H. Wang, and P. Chou, "Distributed streaming media content using cooperative networking," *Microsoft Research Technical Report*, MSR-TR-20002-37, 2002.
4. K. Wu, and P. Yu, and J. Wolf, "Segment-based proxy caching for multimedia streams," in *Proc. of Tenth International World Wide Web Conference (WWW10)*, Hong Kong, May 1-5, 2001.
5. S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. of IEEE INFOCOM 1999*, 1999.
6. W. Jeon and K. Nahrstedt, "Peer-to-peer multimedia caching and streaming service," in *Proc. of IEEE International Confernce of Multimedia and Expo (ICME 2002)*, Laussane, Switzerland, August 2002.
7. T. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. of INFOCOM 2002*, New York, June 2002.
8. M. Hofmann, T. Ng, K. Guo, S. Paul, and H. Zhang, "Caching technique for streaming multimedia over the Internet," in *Bell Labs Technical Memorandum BL011345-990628-05TM*, June 1999.
9. S. Acharya and B. Smith, "Middleman: A video caching proxy server," in *Proc. of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2000)*, 2000.
10. Y. Chae, K. Guo, M. Buddhikot, and S. Suri, "Silo, Rainbow, and Caching Token: Schemes for scalable, fault tolerant streaming caching,' in *IEEE Journal on Selected Areas in Communications*, 2002.
11. J. Kangasharju, K. Ross, and D. Turner, "Adaptive content management in structured P2P communities," *Manuscript*, 2002.
12. Z. Zhang, Y. Wang, D. Du, D. Su, "Video Staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks," in *IEEE/ACM Trans. on Networking*, vol. 8, no. 4, August 2000.
13. E. Bommaiah, K. Guo, M. Hofmann, and S. Paul, "Design and implementation of a caching system for streaming media over the Internet," in *Proc. of RTAS 2000*, 2000.
14. A. Dan and D. Sitram, "Buffer management policy for an on-demand video server," *IBM Research Report RC 19347*, Yorktown Heights, NY, 1993.
15. A. Dan and D. Sitram, "A generalized interval caching policy for mixed interactive and long video workloads," in *Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN 2003)*, 2003.
16. C. Bowman, P. Danzig, D. Hardy, U. Manber, and M. Schwartz, "The Havest information discovery and access system," in *Proc. of the Second World Wide Web Conference (WWW2)*, October 1994.
17. A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrel, "A hierarchical Internet object cache," in *Proc. of USENIX Technical Conference*, January 1996.

18. R. Malpani, J. Lorch, and D. Berger, "Making world wide web caching servers cooperate," in *Proc. of Fourth International World Wide Web Conference (WWW4)*, December 1995.
19. P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proc. of USENIX Symposium on Internet Technologies and Systems (USITS 1997)*, 1997.
20. S. Chen, B. Shen, S. Wee, and X. Zhang, "Investigating performance insights of segment-based proxy caching of streaming media strategies," in *Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN 2004)*, 2004.
21. M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in em Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS 2001), March 2001.
22. W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat, "MediSyn: A synthetic streaming media service workload generator," in *Proc. of the 13th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2003)*, New York, NY, 2003.
23. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. of the 16th ACM International Conference on Supercomputing*, New York, USA, June 2002.
24. S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in em Proc. of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC 2002), New York, NY, 2002.
25. I. Gupta, A. -M. Kermarrec, and A. Ganesh, "Efficient epidemic-style protocols for reliable and scalable multicast," in *Proc. of Symposium on Reliable Distributed Systems (SRDS 2002)*, Osaka, Japan, 2000.
26. H. Yu and A. Vahdat, "Minimal replication cost for availability," in *Proc. of Annual ACM Symposium on Principles of Distributed Computing (PODC 2002)*, New York, NY, 2002.