A Fine-Grained Peer Sharing Technique for Delivering Large Media Files Over the Internet

Mengkun Yang Zongming Fei
Department of Computer Science, Universtiy of Kentucky
Lexington, KY 40506-0046
Email: {myang0,fei}@cs.uky.edu

***Abstract*— In this paper we propose a fine-grained peer sharing technique for delivering large media files in the context of content distribution networks. The replica servers are divided into groups, and those in the same group cooperate with each other to handle client requests. The key difference of the technique from conventional peer to peer systems is that the unit of peer sharing is not a complete media file, but at a finer granularity. By doing so, we increase the flexibility of replica servers for handling client requests. We design a protocol for peers to exchange the information about available resources with each other, and a scheduling algorithm to coordinate the delivery process from multiple replica servers to a client. Our simulations show that the fine-grained peer sharing approach can reduce the initial latency of clients and the rejection rate of the system significantly over a simple peer sharing method.**

I. Introduction

Content distribution networks (CDNs) have been successful in delivering conventional web documents to clients by distributing content from an origin server to replica servers located at the edges of the network and directing clients to a nearby replica. They usually distribute the whole file of a document to replica servers. The upcoming streaming media files tend to be much larger and the number of files a replica can store is limited, even with the increasing disk space. The result is that either we have to store only a small portion of contents at the replica servers and the rest will be delivered directly from the origin server, or we have to frequently remove previously stored files from the storage to accommodate new files. This consumes quite a lot of network bandwidth and can also overload the origin server.

To improve the service capabilities of resource-constrained replica servers, we employ the peer sharing technique of peer-to-peer systems at the replica servers. Peer sharing among clients is an attractive approach, as proposed by most peer-to-peer systems. However, the difficulty lies in that clients are more willing to get the files than serving other clients. Studies showed that almost 70% of users share no files at all and nearly 50% of all responses are returned by the top 1% of sharing hosts [1]. In this paper we explore the possibility of peer sharing among replica servers, which content service providers have direct control and can equip the new technique if proved useful. In a typical CDN, every replica server achieves the best performance when it stores the most popular files in the vicinity. The result is that files stored at different replica servers have a very large overlap. This is hard for them to help each other to serve client requests. Instead, we can let cooperating replica servers store different files [2]. By careful planning, the overlap between different replica servers can be reduced and the total number of media files stored in a group of replica servers can be increased. Thus their capacities of serving clients are improved.

We go one step further in this paper by proposing a fine-grained peer sharing technique. The unit of peer sharing is not a complete media file. Rather, a media file is divided into segments, which will be distributed to different replica servers. A client makes a request to its nearest replica server, which will act as a *coordinator* to schedule which segment will be streamed from which cooperating replica server to the client at what time. We design a protocol for peers to exchange the information about available resources with each other, and a scheduling algorithm to coordinate the delivery process from multiple replica servers to a client. A nice feature of the scheme is that the coordinator only needs to contact with those peers having the segments involved.

The remainder of the paper is organized as follows. Section II proposes the fine-grained peer sharing technique. Section III describes the scheduling algorithm for planning streaming from cooperating replicas to clients. Performance results are presented in Section IV. We discuss related work in Section V and conclude the paper in Section VI.

II. Fine-Grained Peer Sharing in CDNs

A. Peer Sharing in CDNs

In a CDN, a client is normally directed to the nearest replica server. Under the limited storage constraint, we assume that a replica server can store n media files. Usually it will store the most popular or most recently accessed objects. If a media file is available at the replica server, the client can get it directly. For those files that are not stored in the replica server, the client has to get them from the origin server, or the replica server can fetch them from the origin server on its behalf. This may impose a tremendous load on the origin server, because of the large number of potential clients and the long streaming duration.

One approach to dealing with this problem is *peer sharing*, i.e., cooperating replica servers store different objects and they will serve the requests made to their peer servers. If we have

This work was supported in part by the National Science Foundation under Grants CCR-0204304 and EIA-0101242.

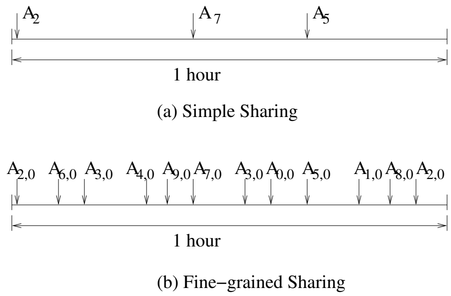

Fig. 1. Comparison between simple sharing and fine-grained sharing

P servers peering with each other, we can store $P*n$ different objects, instead of n in the non-sharing case. We can decrease the load on the origin server substantially and better serve client requests. An observation is that a larger P will result in more objects being served from replica servers. However P cannot be too large. One reason is that grouping more servers together means a longer distance from one of them to the client making the request. This may affect the quality (delay, delay jitter, bandwidth) of the streaming path. The other reason is that the management cost will be higher because of the increased complexity of communication between replica servers.

We envision a two-tier model for organizing the replica servers in the CDN. The replica servers are no longer in a flat structure but divided into groups. The basic principle is that the replica servers in a group should be relatively close in the network distance, and they will cooperate to serve the content to clients attached to any replica server in the group. The focus of this paper is on how replica servers in the same group can cooperate with each other to serve requests from clients.

B. Fine-Grained Peer Sharing

The problem with the simple peer-sharing technique described above is that it is possible that the load on one of the replica servers may be extremely high because it stores one or several hot media files. In the non-sharing case, the requests to these hot media files are distributed to all replica servers. In the sharing case, all these requests are now concentrated on a particular replica server that stores the hot files. Therefore, clients may experience excessive delay in obtaining the media file requested. We propose a fine-grained peer sharing technique to solve the problem.

Instead of storing the whole media file in one server, we divide each media file into segments. Generally one replica server will only store one segment of a media file. A client still makes the request to the nearest replica server, which will communicate with those peering servers having segments of the requested file and schedule streaming of each segment to the client.

We use a simplified example to illustrate the idea of the fine-grained peer sharing technique. Assume four replica servers A, B, C and D peer with each other, and each of them can stream 3 media files simultaneously at the playout rate. Each of them can hold 10 media files that last 1 hour. First consider the *simple peer sharing* case. As a result of careful placement, each of them holds different files. Specifically we assume that A stores files $A_0, A_1, \cdots, A_9$, B stores files $B_0, B_1, \cdots, B_9$, etc. Because each file lasts 1 hour and a server can stream 3 media files at the same time, the total number of requests for files $A_0, A_1, \cdots, A_9$ that we can start to serve within one hour is 3. The same is true of files stored at servers B, C and D.

Now consider the *fine-grained peer sharing* case. We divide each file into 4 segments and store them into different servers. For example, file A_0 is divided into $A_{0,0}, A_{0,1}, A_{0,2}$ and $A_{0,3}$, which will be stored to servers A, B, C and D, respectively. All other files are divided and distributed in a similar way, except that $B_{k,0}$ will be kept in server B, $C_{k,0}$ in C, $D_{k,0}$ in D ($k = 0, 1, 2, \cdots, 9$) and the rest of segments are distributed in a round robin fashion. Now consider the total number of requests for files $A_0, A_1, \cdots, A_9$ that we can start to serve within one hour. Since each segment only lasts 1/4 hour, in each 1/4 hour we can start 3 streams. Over 1 hour period of time, we can start 3*4 = 12 streams for those requests for $A_0, A_1, \cdots, A_9$. As an example, Figure 1(a) shows that the simple sharing method can only start three video files (A_2, A_7, A_5) within one hour, while Figure 1(b) shows that the fine-grained peer sharing method can start 12 video files within one hour.

It is true that the total number of requests that can be served by a given set of cooperating replica servers within a certain period of time is not necessarily increased. However, because the load for serving a given file is evenly distributed, the number of requests for *a specific file* or *a specific subset of files* that can be served within a certain period of time by the fine-grained peer sharing scheme is increased. Therefore the capacity of these replica servers to serve a given media file or a given set of media files is improved. In general, if media files are divided into N segments, the capability of servicing a given set of files will be increased up to N times.

The idea of fine-grained peer sharing is similar to the disk striping technique [3]–[6] in balancing the load, but there are two key differences. First, the scheduler in disk striping schemes has the knowledge of all the requests and states of all the disks, while the scheduler in the fine-grained peer sharing scheme needs to communicate with other replica servers to know their status to make a decision for individual clients. Second, one goal in disk striping is to improve throughput from the disk to CPU by accessing different parts of a file from different disks at the same time, while the goal of the fine-grained peer sharing technique is to improve the availability of a given set of files.

In the next section, we will describe a protocol for peers to share information and redirect requests to others, and design a scheduling algorithm for a replica server to decide when to get each segment from which sharing peer.

III. Communication Protocol and Scheduling Algorithm

Accessing different segments from different servers needs coordination between these servers for several reasons. First, the load and the streaming schedule at replica servers will change dynamically. We do not want to have a central server to maintain all the information. When a client requests a media file, those replica servers holding segments of the file need to be coordinated to schedule when to send the segments to the client based on the current load. Second, the client can only receive from a limited number of servers at the same time. At any point of time, the number of servers streaming to the client cannot exceed this limit. Third, it is possible that several initial segments are received by the client at an earlier time and the client starts to play back, but later segments cannot be streamed fast enough to catch up with their supposed playout time. This can lead to a gap in the playout by the client. One way to eliminate the gap is to let the client postpone playout with an initial delay. We want to minimize this delay and guarantee that there is no gap in the playout. These factors motivate us to design the communication protocol and the scheduling algorithm described in this section.

When a media file is delivered to a group of P peering replica servers, it is divided into N (usually $N \leq P$) segments, which can be of different size [1]. The first segment will be stored at the replica server that has the smallest number of first segments of other files already there. Assume the server number is i. The second segment will be distributed to server $(i+1) \bmod P$. We continue the distribution of the rest of segments in the round-robin fashion until we finish distributing all the segments. The information about locations of each segment of each file will be distributed to all replica servers in the group, so that they know where to ask for help when needed.

The server and client capacities are abstracted as channels. By a *channel*, we mean the bandwidth used to stream data at the playout rate of media files. If the server bandwidth is B and the playout rate is b, then the number of channels this server has is $B_s = \lfloor B/b \rfloor$. Similarly, we has a parameter B_c, which represents the number of streams (at the playout rate) that a client can receive from replica servers at the same time.

A. Communication Protocol

Streaming a media file from replica servers to a client involves several steps of communications among the client, the nearest replica server and other peer replica servers. The communication protocol can be described as the following steps.

First, the client sends its request for a file to its nearest replica server, which acts as the *coordinator* for this request.

Second, the coordinator looks up in the table about the locations of segments of this file, and forwards *segment requests* to those peering servers having segments of this file.

Third, after receiving segment requests, the peering replica servers will try to find available slots that can be used to stream the segment. The available slots are in the form of a list of pairs (a, b), which means there is a channel that is free between time a and time b. Note b can be ∞. These slots will be reserved for a period of time τ for the coordinator. It is a policy of the replica server that determines how many slots and how long these slots will be reserved for the coordinator.

Fourth, for segment j, the coordinator will receive a list of time slots $(a_{j0}, b_{j0}), (a_{j1}, b_{j1}), \ldots, (a_{jk}, b_{jk})$, where $0 \leq j \leq N-1$. Upon receiving replies from these peers, the coordinator uses the *scheduling algorithm* described in the next subsection to choose an appropriate time slot for streaming each segment, under the assumption that a client can receive from B_c channels at the same time. The goal is to minimize the initial latency and guarantee there is no jitter between playing out different segments.

Fifth, confirmation messages are sent back to peer servers from the coordinator. They will inform each peer which slot (or part of it) has been scheduled to stream the requested segment. At the same time, the coordinator sends the client a message about when to start playing out and where to get each segment. It is up to the client to make the actual requests. We understand that the client may need a certain size of buffer to hold those segments that will be streamed before their playout time.

Finally, after receiving the reservation messages, each peer server confirms the reservation of the selected time slot for transmitting the segment of the file and releases all other channel slots previously reserved for this request. If a certain server does not receive the reservation message from the coordinator before the lifetime τ expires, it will cancel all channel slots reserved for the segment. It also informs the coordinator, which may cancel channel reservations and transmissions for all other segments of this file accordingly.

One issue worth further discussion is how to deal with concurrent requests from multiple peers. In the third step, a replica server may receive segment requests from multiple coordinators, which may ask for the same segment or different segments. Obviously reserving all available slots for one coordinator will block requests from all other coordinators. Our strategy is to count the number of requests every τ time period. Assume the number in the most recent period is Γ_c and the variable used to represent the load on this server is Γ. We calculate the current load as weighted sum, i.e., $\Gamma = w * \Gamma_c + (1-w) * \Gamma$, where $w = 0.3$. The proportion of slots that will be reserved for each coordinator will be $1/\Gamma$ of all available slots.

B. Scheduling Algorithm

Upon receiving information about available time slots for streaming each segment, the scheduling algorithm of the coordinator will determine the time slot that will be actually used. For the time being, we assume the clocks of these replica servers are synchronized. We will present a method to solve the clock problem in the next subsection. Assume that the client makes the request at time t_{rqst}. The length of the media file is L, which is divided into N segments. Assume the length of segment j is s_j. (If all segments are of equal size, we have $s_j = s = L/N$.) Ideally, there will be no delay if the client

[1] In the simulations of this paper, we use segments of equal size. However, the protocol and the algorithm described in this paper can deal with segments of unequal size as well. The effects of segment size on the performance were discussed in another paper.

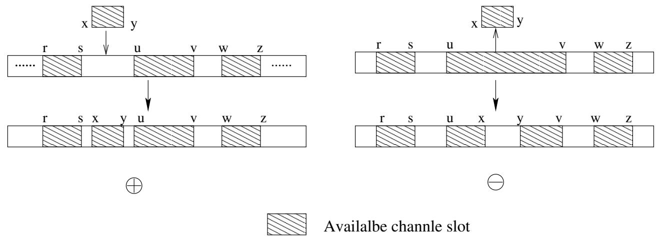

Fig. 2. Operations on Ψ_l: $\Psi_l \oplus (x, y)$ and $\Psi_l \ominus (x, y)$

can start downloading segment j $(0 \le j \le N-1)$ at time $t_j = t_{rqst} + \sum_{k=0}^{j-1} s_k$. However we may have to download segment j before or after t_j because the time slots starting at time t_j on all channels of the replica server may have already been reserved for other requests. While an actual downloading time before t_j will not cause any problem, a downloading time after t_j will cause the client to start playing out the whole media file after some initial latency. Actually the final initial latency is the maximum of the latencies caused by all segments. Our algorithm will try to reduce this latency. We start with some notations used in the algorithm.

- For any request being processed, the available server channel slots returned from the peer replica server holding segment j $(j = 0, \cdots, N-1)$ is $\Omega_j = \{(a_{j0}, b_{j0}), (a_{j1}, b_{j1}), (a_{j2}, b_{j2}), \cdots\}$, where $a_{j0} \le a_{j1} \le a_{j2} \le \cdots$.
- We assume that the client making the request can receive from B_c streams at the same time. The capability for downloading each stream is modeled as a channel. At one point of time, the available slots on the l-th channel of the client is $\Psi_l = \{(x_{l0}, y_{l0}), (x_{l1}, y_{l1}), (x_{l2}, y_{l2}), \ldots (x_{lh_l}, \infty)\}$, where $l = 0, \cdots, B_c - 1$ and $x_{l0} < x_{l1} < x_{l2} < \cdots < x_{lh_l}$. Initially, all channels are (t_{rqst}, ∞).

Problem Formulation: For any request, given server slots $\Omega_0, \Omega_1, \cdots, \Omega_{N-1}$ together with client channels $\Psi_l = \{(t_{rqst}, \infty)\}$ for $l = 0, \cdots, B_c - 1$, we need to find the streaming time u_j for segment j $(0 \le j \le N-1)$, such that $(u_j, u_j + s_j)$ is a time slot that does not collide with time slots for downloading other segments in the same client channel, and there exists a k such that $(u_j, u_j + s_j) \subseteq (a_{jk}, b_{jk})$. The latency caused by segment j is $d_j = \max(u_j - t_j, 0)$, and the final initial latency of the client will be $d = \max\{d_j | j = 0, \cdots, N-1\}$. The client can start playing out the media file at time $t_{rqst} + d$ without jitter for each segment [2]. The goal of the algorithm is to minimize d.

The basic idea of the algorithm is to reserve the *earliest possible slot* on the client channels for each segment first, then try to make adjustment to minimize the initial latency d. Given the current available slots Ω_j from peer servers and client channels $\Psi_0, \cdots, \Psi_{B_c-1}$, the current *earliest possible slot* for segment j is $(u_j, u_j + s_j)$ satisfying: 1) $u_j \ge t_{rqst}$; 2) $(u_j, u_j + s_j) \subseteq (a_{jk}, b_{jk}) \cap (x_{lg}, y_{lg})$ for some k, l, g; and 3) for any e, such that $e \ge t_{rqst} \wedge (e, e + s_j) \subseteq (a_{jk'}, b_{jk'}) \cap (x_{l'g'}, y_{l'g'})$ for some k', l', g', we have $u_j \le e$. We call such a slot satisfying the *earliest condition* in the algorithm.

[2]To accommodate the latency from the replica servers to the client, we may have to add an extra to d. This extra can be the maximum value of one way delay from the replica servers to the client. For the clarity of presentation, we will not mention this extra in the rest of the paper.

Scheduling Algorithm

1: **for** $(j = 0$ to $N-1)$ **do**
2: find $(u_j, u_j + s_j)$ satisfying *earliest condition* for segment j on a client channel and record that the slot is on l_j-th client channel.
3: $\Psi_{l_j} \leftarrow \Psi_{l_j} \ominus (u_j, u_j + s_j)$
4: $d_j \leftarrow \max(u_j - t_j, 0)$
5: Sort d_j and fill in $Q[0 \ldots N-1]$ accordingly.
6: $j \leftarrow N-1$
7: **while** $j > 0$ **do**
8: $i \leftarrow Q[0]$, $i' \leftarrow Q[j]$
9: $\Psi_{l_i} \leftarrow \Psi_{l_i} \oplus (u_i, u_i + s_i)$
10: $\Psi_{l_{i'}} \leftarrow \Psi_{l_{i'}} \oplus (u_{i'}, u_{i'} + s_{i'})$
11: find $(v, v + s_i)$ satisfying *earliest condition* for segment i on a client channel and record that the slot is on l-th client channel.
12: find $(v', v' + s_{i'})$ satisfying *earliest condition* for segment i' on client channel and record that the slot is on l'-th client channel.
13: **if** $v < u_i$ and $\max(v' - t_{i'}, 0) < d_i$ **then**
14: $u_i \leftarrow v$, $l_i \leftarrow l$
15: $u_{i'} \leftarrow v'$, $l_{i'} \leftarrow l'$
16: Update d_i, $d_{i'}$, and $Q[0 \ldots N-1]$
17: $j \leftarrow N-1$
18: **else**
19: $\Psi_{l_i} \leftarrow \Psi_{l_i} \ominus (u_i, u_i + s_i)$
20: $\Psi_{l_{i'}} \leftarrow \Psi_{l_{i'}} \ominus (u_{i'}, u_{i'} + s_{i'})$
21: $j \leftarrow j - 1$

Fig. 3. The scheduling algorithm

The greedy reservation for the earliest possible slots favors early segments and may use up the only slots that can be used to prefetch some later segments that have a larger latency. So the next step in the algorithm is to gradually reduce the highest segment latency by moving a segment with zero or very low latency from its current reserved slot to a later free slot, and let the vacated slot on a client channel to accommodate the segment with the highest latency.

Two operations, $\oplus$ and $\ominus$, are defined on Ψ_l, as illustrated in Figure 2. Basically, $\oplus$ inserts a slot (x, y) into the channel Ψ_l, and $\ominus$ removes (x, y) from the channel. If inserting a slot makes some slots on a channel contiguous to each other, we further combine them to one larger slot. The priority queue $Q[0 \ldots N-1]$ stores segment indices such that $d_{Q[0]} \ge d_{Q[1]} \ge \cdots \ge d_{Q[N-1]}$.

The scheduling algorithm is outlined in Figure 3. Lines 1-5 reserve slots on client channels for segments as early as possible. So at the time we schedule for segment j, $(u_j, u_j + s_j)$ on the channel l_j is the earliest possible slot. The priority for reservation decreases as the segment index

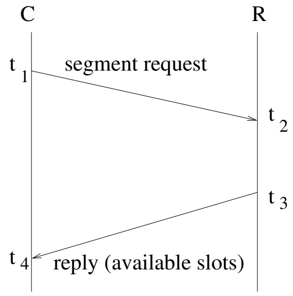

Fig. 4. Clock synchronization

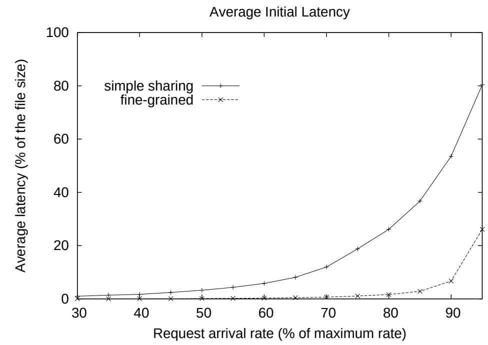

Fig. 5. Average latency using different approaches

increases. Some segments may get high latencies since their residing servers cannot stream them on time, and the only possible slots able to prefetch them have already been taken up by other segments. To solve this problem, lines 6-21 move the segment with the highest latency to an earlier slot previously reserved for another segment with lower latency. In this way, the initial latency of the client can be decreased. The process is continued until no such moving is possible.

C. Clock Synchronization

In the algorithm description we assume that the clocks of the replica servers are synchronized. In reality they may be skewed. We present a mechanism of adding timestamps in the messages exchanged between replica servers to solve the problem. The similar timestamps have been used in RTCP [7].

In the second step of Section III-A, the coordinator (C) adds a send timestamp (t_1) when it sends a segment request to a peering replica server (R), as shown in Figure 4. Upon receiving the request, the replica server (R) records the receive time (t_2). When R is ready to send the reply to the coordinator, it puts a reply timestamp (t_3), together with t_1 and t_2 in the message to C. The coordinator C will record the receive time t_4.

The coordinator can calculate the round trip time to the peering server R as $RTT = (t_4 - t_1) - (t_3 - t_2)$. Under the assumption of symmetric path, the one-way delay will be $RTT/2$. The local time t_2 at R should be translated as local time $t_1 + RTT/2$ at C. Therefore, we should shift each time t reported by the peering server R by applying the function $f(t) = t - (t_2 - (t_1 + RTT/2))$. In particular, when the peering server R reports available slots $\Omega_j = \{(a_{j0}, b_{j0}),\ (a_{j1}, b_{j1}), (a_{j2}, b_{j2}), \cdots\}$, the coordinator will transform them into $\Omega'_j = \{(f(a_{j0}), f(b_{j0})), (f(a_{j1}), f(b_{j1})), (f(a_{j2}), f(b_{j2})), \cdots\}$ and use it as the input to the scheduling algorithm.

The outputs of the scheduling algorithm are time slots $(u_j, u_j + s_j)$, which are in local time at C. Rather than sending a slot directly to the replica server, the coordinator first applies another function to adjust it to the clock of the replica server. The function used is $g(t) = f^{-1}(t) = t + (t_2 - (t_1 + RTT/2))$. That is, it will reserve slot $(g(u_j), g(u_j + s_j))$ at the replica server R holding segment j.

IV. Performance Evaluation

We evaluate the performance of the *fine-grained* peer sharing and its protocol and scheduling algorithm by simulation. Each client is assumed to have B_c (with the default value 4 unless specified otherwise) channels for receiving media files, and every replica server has B_s (default value 10) channels to transmit data. The file size is $L = 4000$ second MPEG video and each server can store 50 files. Each peering group consists of P (default value 20) replica servers. So the total storage size of the group is $P * 50$ files. In the *fine-grained* scheme, a file is divided into N (default value 10) segments, and they are placed onto servers using the *round-robin placement*. In the *simple sharing* case, the whole file is always distributed to the server currently having the smallest sum of popularity rankings of files already on it, so long as the server has enough available storage. Request arrivals are modeled as a Poisson process and the file popularities are approximated by a Zipf-like distribution with $\alpha = 0.733$ [8]. We assume requests are equally likely originated from any client. The highest possible request arrival rate the system can support is $\lambda_{max} = \frac{B_s \times P}{L}$. That is the case in which *all* channels of *all* servers are kept busy *all the time*. The actual arrival rate λ in our simulation is set as a percentage of λ_{max}. Every experiment simulates a time period of L*25 second long.

The performance measures we study include: 1) *Initial latency*. It is the average interval between the time a client makes a request and the starting time of playout. 2) *Rejection rate*. It is the ratio of the number of rejected requests over the total number of requests.

Figure 5 shows the initial latency of the *simple sharing* and the *fine-grained* schemes, expressed as the percentage of the video length. As the volume of traffic increases, the difference between the initial latencies of the *simple sharing* and the *fine-grained* schemes becomes much larger. When the request rate is 80% of the system capacity, the initial latency of the simple sharing is almost 25% of the video length, i.e., more than 15 minutes, while the initial latency of the fine-grained scheme is still very close to 0.

Figure 6 plots the rejection rate when the admission control is applied. Specifically, if a coordinator finds that the initial latency is greater than a threshold $T_{thresh} = L/20$, the request

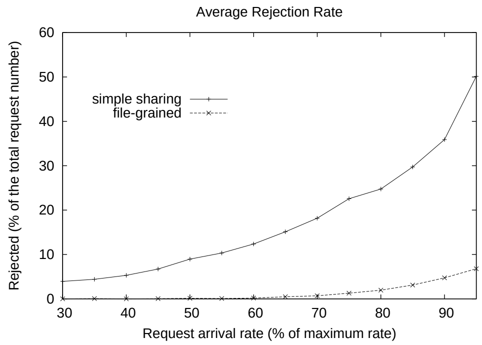

Fig. 6. Rejection rate using different approaches

will be rejected. We note that much lower rejection rate can be achieved in the *fine-grained* scheme. For example, when the request rate is 80% of the system capacity, the rejection rate of the *fine-grained* scheme is less than 2%, while the rejection rate of the *simple sharing* scheme is as high as 27%. Even though the request rate is only 30% of the system capacity, the *simple sharing* scheme has to reject about 4% of the requests.

V. RELATED WORK

Streaming multimedia content over the Internet has been studied extensively [9]–[11]. Proxy caches have been used to reduce the backbone bandwidth requirement for scalable video delivery [12]–[14]. Recently, prefix caching has been proposed to reduce the storage requirement for the proxy servers [15], [16]. Only the prefix part is delivered to replica servers while the suffix part is kept at the origin server and usually depends on multicast for its delivery. The peer sharing technique was originally used in peer-to-peer systems [17]–[19]. However, the unit of sharing is a complete file.

Segmentation of large media files has been used in large video servers. The data striping technique has been used in media servers to improve the disk access speed and reliability [3], [6], [20]. It allows multiple disk transfers to occur in parallel. Data striping [21] has also been used in traffic shaping of multimedia transmission to decrease short-term burstiness and diminish short-term correlations, in order to improve the buffer efficiency of the queueing system. However, the scheduling task is easier in the video server because it has knowledge of all files and load on all disks. Our scheduling algorithm is more complicated because it requires communication between replica servers. It is executed by each replica server and only involves those replicas having segments of the requested file.

VI. CONCLUDING REMARKS

In this paper we present a novel fine-grained peer sharing technique for streaming large media files in the context of content distribution networks. Our scheme does not depend on the availability of multicast deployment. Our design employs peer-to-peer sharing at a finer granularity to improve the flexibility of peering replica servers to handle client requests. We design a communication protocol and a scheduling algorithm to achieve the goal of reducing the initial latency and the rejection rate. The simulations demonstrate a significant performance gain of the fine-grained peer sharing technique.

REFERENCES

[1] E. Adar and B. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, no. 10, http://www.firstmonday.dk/issues/issue5_10/adar/.

[2] J. Kangasharju, J. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," in *Proceedings of Sixth International Workshop on Web Caching and Content Distribution (WCW'01)*, June 2001, pp. 39–53, boston, MA.

[3] J. R. Santos and R. Muntz, "Comparing random data allocation and data striping in multimedia servers," in *Proceedings of ACM Sigmetrics*, June 2000, santa Clara, CA.

[4] P. Shenoy and H. M. Vin, "Efficient striping techniques for variable bit rate continuous media file servers," *Performance Evaluation*, vol. 38, no. 3, pp. 175–199, December 1999.

[5] J. H. Hartman and J. K. Ousterhout, "The zebra striped network file system," *ACM Trans. on Computer Systems*, vol. 13, no. 3, pp. 274–310, 1995.

[6] A. L. Drapeau, P. M. Chen, J. H. Hartman, E. K. Lee, E. L. Miller, K. Shirriff, S. Seshan, R. H. Katz, G. A. Gibson, and D. A. Patterson, "RAID-II: A high-bandwidth network file server," in *Proceedings of 21st International Symposium on Computer Architecture*, April 1994, chicago, IL.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," *RFC 1889*, 1996.

[8] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence, and implications," in *Proceedings of INFOCOM'99*, March 1999, pp. 126 –134.

[9] T. Kim and M. Ammar, "A comparison of layering and stream replication video multicast schemes," in *Proceedings of NOSSDAV'01*, 2001.

[10] Z. Ge, P. Ji, and P. Shenoy, "A demand adaptive and locality aware (dala) streaming media server cluster architecture," in *Proceedings of NOSSDAV'02*, May 2002, pp. 139–146, miami, FL.

[11] J. M. Almeida, D. L. Eager, M. Ferris, and M. K. Vernon, "Provisioning content distribution networks for streaming media," in *Proceedings of IEEE Infocom'02*, June 2002, pp. 1746–1755, new York, NY.

[12] Z. li Zhang, Y. Wang, D. H. Du, and D. Su, "Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks," *IEEE/ACM Transactions on Networking*, vol. 4, no. 8, pp. 429–442, August 2000.

[13] Y. Wang, Z.-L. Zhang, D. H. C. Du, and D. Su, "A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers," in *Proceedings of IEEE Infocom'98*, 1998.

[14] O. Verscheure, C. Venkatramani, P. Frossard, and L. Amini, "Joint server scheduling and proxy caching for video delivery," *Computer Communications, Special Issue*, vol. 25, March 2002.

[15] S. Ramesh, I. Rhee, and K. Guo, "Multicast with cache (mcache): an adaptive zero-delay video-on-demand service," in *Proceedings of IEEE Infocom'01*, 2001.

[16] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proceedings of IEEE Infocom'99*, April 1999.

[17] N. Inc., "http://www.napster.com."

[18] Kaaza, "http://www.kaaza.com."

[19] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM 2001*, August 2001, pp. 149–160, san Diego, CA.

[20] W. Aref, D. Bushmitch, I. Kamel, and S. Mukherjee, "An inexpensive, scalable, and open-architecture media server," in *Proceedings of First IEEE?RPS Joint Conference on Internet Technologies and Services*, Oct. 1999, moscow, Russia.

[21] D. Bushmitch, S. S. Panwar, and A. Pal, "Thinning, striping and shuffling: Traffic shaping and transport techniques for variable bit rate video," in *Proceedings of GLOBECOM 2002*, Nov. 17-21, 2002, taipei.