Coping with Unreliable Peers for Hybrid Peer-to-Peer Media Streaming

Sung-Hoon Sohn, Yun-Cheol Baek, and Juno Chang

Division of Computer Software
Sangmyung University
Seoul 110-743, Korea
{shson,ybaek,jchang}@smu.ac.kr

Abstract. In this paper, we propose a hybrid peer-to-peer streaming architecture for large scale media streaming service. The proposed architecture combines peer-to-peer system and centralized server to exploit advantages of the two models. Then, we deal with streaming load allocation problem under the proposed scheme. Given a set of centralized servers and a set of unreliable supplying peers with heterogeneous bandwidth offers, we show how to assign streaming load to a subset of supplying peers, while considering unreliable performance of each peers. Our experimental results show that the proposed scheme, compared with legacy server with similar capacity, increases the number of clients about 67% on the average.

1 Introduction

Traditional client-server architecture along with voluminous nature of multimedia files makes it difficult to provide large scalable multimedia treaming service. A powerful centralized server with a high-bandwidth Internet connection is somewhat easy to deploy, it has a lot of shortcomings in scalability, reliability, high cost, and load on backbone networks. To deal with these problems, architectures such as proxy caching [7,10] or content distribution network (CDN) [1, 8] are introduced. These architectures employ intermediate nodes called proxy server (in caching) or edge server (in CDN) to duplicate some of multimedia contents to geographically closer nodes to clients. However, they still suffer from the same problems as the client-server case, since they are fundamentally based on client-server paradigm from the viewpoint of proxy server or edge server.

Recently peer-to-peer streaming system has gained a lot of attention as an alternative to existing streaming service architectures. Compared with client-server based models, peer-to-peer streaming system is more scalable in terms of the number of concurrent users and provides much larger streaming capacity in a cost-effective manner. However, pure peer-to-peer media streaming service in real world is almost infeasible due to the following non-technical reasons. (1) Peers are very autonomous entities; they join and leave the network whenever they want to (even in the middle of streaming), since they do not care about overall system's serviceability, availability, etc. (2) Peers are extremely unreliable; they may suffer from performance degradation due to network congestion, etc. (3) It is very difficult to service unpopular contents by pure peer-to-peer system only, since peers have inclination to access and store popular contents only. Therefore, in order to deploy peer-to-peer paradigm in media streaming service in real world, there must be a scheme to make up for such weak points in the pure peer-to-peer system.

There have been many works dealing with pure peer-to-peer media streaming systems [2,4,5,6,9,12]. Especially, several multi-source on-demand media streaming systems have been proposed. [14] addresses the media data assignment problem in non-parallel fashion and fast system capacity amplification method for multi-source media streaming. Similar to our work, a hybrid system which integrates peer-to-peer into CDN is proposed in [13]. Different from our work, peers are regarded as reliable entities. The authors assume that peers are always up, have no bandwidth degradation, and never stop streaming anyway. Moreover, once a media file is dispersed throughout the system, subsequent streaming requests for that media file are served by peers without intervention of centralized server. They call it handoff and try to optimal handoff time for a given media file. After handoff time, the hybrid system is regressed to pure peer-to-peer system. Even in the middle of hybrid period, a media streaming session is serviced either by CDN server only or by peers only.

Modeling of peers in previous works is too ideal. As mentioned above, peers are never reliable in real world, and it does not make sense to provide deterministic service guarantee with unreliable peers. In this paper, we first propose a hybrid peer-to-peer streaming architecture for large scale media streaming service. The proposed architecture combines peer-to-peer system and centralized server to exploit advan-tages of the both models. Then, we tackle a problem of streaming load allocation under the proposed architecture. Given a set of centralized servers and a set of peers with heterogeneous bandwidth offers, we suggest a policy to select subset of available centralized servers and supplying peers for streaming a media file. Especially we take account of the unreliable property of peers.

Our contribution can be divided into two parts. First, we propose a hybrid peer-to-peer media streaming architecture integrated with centralized client-server one. Sec-ond, we solve a streaming load allocation problem under the proposed architecture to cope with unreliable peers while maximizing the number of concurrent users. We conduct comprehensive performance evaluation of the proposed scheme. Our results show that that the proposed scheme, compared with legacy server with similar capacity, increases the number of clients about 67% on the average.

The rest of this paper is organized as follows. In Section 2, we propose a hybrid peer-to-peer streaming scheme and explain the system operations under the proposed architecture. In Section 3, we identify the streaming load allocation problem based on the model and propose a load allocation scheme and its admission control algorithm which continues streaming service against some peer failure or degradation. We describe the simulation setup and discuss the results of performance evaluation in Section 4. Finally, we present our conclusions in Section 5.

2 System Operations

In this section, we present a hybrid peer-to-peer media streaming architecture and explain overall system operations based on the architecture.

Fig. 1 shows the proposed hybrid peer-to-peer streaming architecture. The hybrid system consists of a few streaming servers, an index server, and a set of ordinary peers. Major roles of centralized servers in our architecture are: (1) legacy streaming servers that participate in hybrid streaming session as a server, (2) source of all media files in the system, i.e. a seed peer in the peer-to-peer network. Hereafter, we call them *source peers* in the sense of "source of all media files." Compared with ordinary peer, source peer is a true server in the meaning that they are always up and in operation. The number of source peers in the system is dependent on the scale of the network and client population.

An *index server* of peer-to-peer network knows all the information about which peer is dead or alive, which peer owns which media files, which peers are joining which streaming sessions, and so on. Each *peer* is either a supplying peer for a requesting peer depending on the role of the peer in the streaming session. A peer may be both supplying peer and requesting peer at the same time. Before receiving any streaming service, the client is a requesting peer. After finishing streaming service, the requesting peer caches the media file in disk and it becomes a supplying peer of the media file. A supplying peer may participate in several streaming session as a server at the same time. The heterogeneity of supplying peers is modeled by its maximum number of out-bound sessions, upper limit on the aggregate out-bound bandwidth, and the degree of reliability. We assume that each peer has enough disk storage to contain several video files.

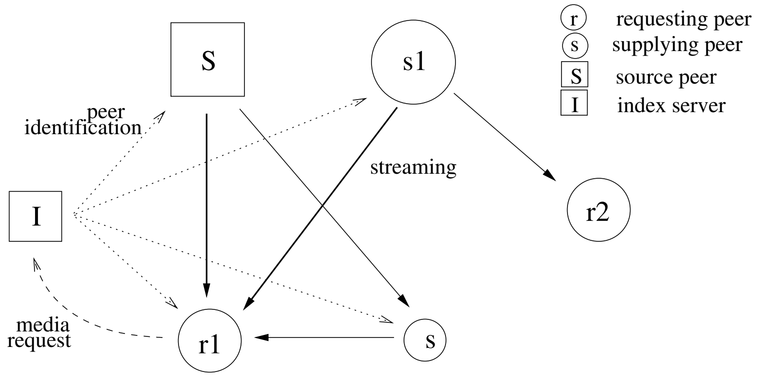

Fig. 1. A hybrid peer-to-peer streaming

The peer-to-peer media streaming in the proposed architecture operates as follows. When a media file is first introduced to the system, it is copied to each source peer[1]. At the beginning, there are no supplying peers. Only source peers stream the media data to receiving peers. After a streaming session, the requesting peer reports to the index servers that it has become a sending peer with *contribution parameters* such as a limited out-bound bandwidth contributed to the system's streaming capacity and a limited number of streaming sessions it would support. The values of the two parameters are dependent on the capacity of the peer such as processor power, network bandwidth, and storage.

A peer-to-peer streaming session involves at least one source server and several supplying peers. In order to guarantee continuous playback of media file, the sum of their out-bound bandwidth contribution is at least the same as the media playback rate. On receipt of media playback request, the index server first determines source peers in the neighborhood of the requesting peer. It also checks if there are active supplying peers who own the media file such that (1) they have an available out-bound slot and (2) the sum of their out-bound bandwidth, including source peers, is greater than or equal to the playback rate of the media. If so, the request will be served by the selected supplying peers and source peers servers; otherwise, the request will be served by the source peers only.

For a given media file and possible candidate supplying peers (including centralized servers), there may exist many possible ways to select a subset of candidate supplying peers who will participate in streaming session of the media file. There are certain criteria in designing peer selection policy. These include (1) guaranteeing continuous playback, (2) to minimize initial buffering delay, (3) to be resilient from (sending) peer degradation or failure, (4) to maximize the remaining streaming capacity of the system, (5) to minimize load on network, (5) to disseminate the media file as fast as possible. The principal objective of the streaming service is to guarantee continuous playback. In legacy client-server paradigm, continuous playback is guaranteed by reliable centralized server. However, in peer-to-peer paradigm, peers acting as a server are not reliable at all. They may go down, suffer from degradation or failure. Moreover, one cannot predict theses failures in advance. Therefore, the unreliable property of peers makes it difficult to guarantee continuous playback. In what follows, we propose a peer selection policy considering unreliable property of peers.

3 Streaming Load Allocation

In this section, we try to answer the following question; *how to distribute a streaming load among source peers and ordinary supplying peers?* We first define clearly the streaming load allocation problem, and then we suggest a streaming load allocation policy in consideration of peer's reliability. We also propose

[1] A media file can be published by an ordinary peer. In this case the media file should be duplicated to source peers before being serviced to peers

a failure resilience scheme dealing with degradation of peers in the middle of streaming session.

Before discussing the streaming load allocation, we first model the heterogeneity of peers as follows. Each peer has two attributes concerning heterogeneity; the degree of reliability and out-bound bandwidth limit. Let the *degree of reliability* of each supplying peer i be specified as a percentage p_i of the total amount of media data that is supposed to arrive on time. That is, in worst case, a requesting peer receives only p_i of the data the supplying peer i tries to send. If a peer is a source peer, p_i is always 1; otherwise, for ordinary supplying peers, it is less than 1 (p_i is even zero when p_i is down). Let the *out-bound bandwidth limit* of peer i be specified as f_i^{max}. Namely, peer i limits its out-bound bandwidth contribution by f_i^{max}.

Now consider a media streaming session of m possible candidate peers (including source peers). We assume that supplying peers can service the requesting peer by proceeding in periodic rounds, retrieving and sending a fixed amount of media data for each round. Let $f_1, f_2, \ldots, f_n$ denote the amount of media data sent in each round. The problem of streaming load allocation is to find $n(n \leq m)$ and $f_i(i = 1, \ldots, n)$, which satisfy the following inequality:

$$p_1 * f_1 + p_2 * f_2 + \ldots + p_n * f_n \geq q * F \tag{1}$$

subject to $0 \leq p_i \leq 1$, $0 < f_i \leq f_i^{max}$, and $\sum_{i=1}^{n} f_i^{max} \geq F$, where F is the total amount of data needed by requesting peer during a round for best-quality playback and q is the streaming quality requirement provided by requesting peer. The left hand side of the equation represents the lower bound on the expected amount of media data received during a round in worst case. The right hand side means the amount of media data that is needed by client while satisfying the client-supplied QoS parameter[2].

This problem can be solved using the following algorithm. First of all, we should include one or more source peers, if available, in order to guarantee minimum quality of streaming. Then, given m candidate peers, we sort the supplying peers according to their values of $p_i * f_i^{max}$ in decreasing order. We start with the largest value peers and assign its f_i to be $\alpha * p_i * f_i^{max}$, where α is an appropriate constant less than 1. We then continue to assign to each supplying peer this value, beginning with the ones with larger values and moving to the ones with smaller values, until the above equation is satisfied.

Now we discuss the failure resiliency feature of the proposed scheme. By failure resiliency, we mean a recovery from the situation where peer's degree of reliability is changing due to some reasons such as network congestion on links between supplying peer and receiving peer, abrupt overload on supplying peers, etc. Consider a simple case involving two supplying peers. For supplying peer s_1, p_1 is 1.0 and f_1 is 10 and, for supplying peer s_2, p_2 is 0.6 and f_2 is 10. Then, the amount of media data received by requesting peer per each round is $1 * 10 + 0.6 * 10 = 16$. After a while, the degree of reliability of s_1 changes to 0.9,

[2] The equation can be used as an admission control criteria by index server when admitting a new requesting peer

which may results in the decrease of streaming quality in the near future. We can avoid the situation by simply changing the values fi of other supplying peers who shows stable degree of reliability. In our example, we increase the value of f_2 by 2, we can easily maintain the streaming quality such as $0.9*10+0.6*12 \geq 16$.

4 Experiments

We evaluate the performance of the proposed architecture through extensive simulation experiments. We present the simulation setup and the results in this section.

We use the Network Simulator ns-2 [11] in the experiments. We use large Internet-like network topology generated by GT-ITM topology generator [3], add peers via DSL or LAN to the routers in the topology. We use 112 media files of 30-minute durations each recorded at a rate of 192Kb/s. Each peer has 128Kb/s out-bound bandwidth of 2-6 concurrent streaming sessions.

We simulate the following scenario. First source peers introduce media files into the network. According to the uniform arrival pattern, a peer joins the network and requests a media file. Media files are selected according to the zipfian distribution with skew factor of 0.7. Then, the streaming steps described in Section 2 and 3 are put into operation. If the request can be satisfied, i.e., there is a sufficient capacity in the system, connections are established between the supplying peers (and source peers) and the requesting peer and the streaming session begins. The send and receive over UDP and carries CBR traffic. When the streaming session is over, the requesting peer caches the whole media file.

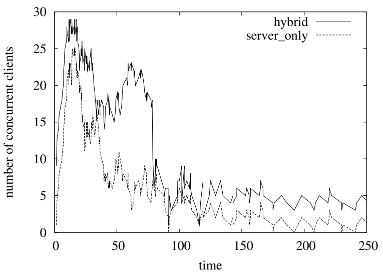

Fig. 2. Number of concurrent users

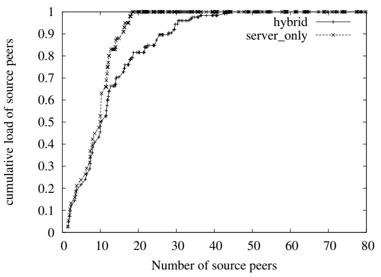

Fig. 3. Load of source peers

Fig. 2 and 3 shows the performance of the proposed hybrid streaming scheme compared with the legacy CDN service. In this experiment, 600 clients make 4,100 requests during 270 minute simulated interval. For legacy streaming service, we use 10 streaming servers and, for hybrid peer-to-peer service, we use 3 source peers. As shown in Fig. 2, the proposed scheme accepts much lager number of client's requests. More specifically, the number of concurrent clients increases by 67.2% on average. Fig. 3 shows the cumulative load on the source peers in hybrid system and on the streaming servers in legacy system. Source peers in the hybrid system are much less loaded due to the help of ordinary supplying peers.

In order to find the optimal number of source peers in the hybrid system streaming service, we measure the effects of the number of source peers on the reject ratio of client's request with various source peer capacities. As shown in Fig. 4, when the number of source peers is relatively small, the reject ratio decrease drastically as the number of source peers increases. However, more source peers does not affect greatly when the number of source peers are large.

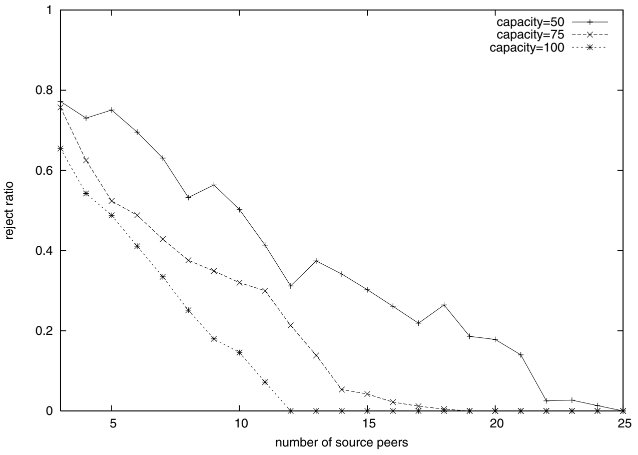

Fig. 4. Effects of number of source peers on reject ratio

5 Conclusions

In this paper, we propose a hybrid peer-to-peer media streaming architecture integrated with legacy client-server architecture. We solve the streaming load allocation problem under the proposed scheme to maximize the number of concurrent users for a given set of peers. The proposed hybrid architecture and the load allocation scheme has the following features: (1) the proposed scheme takes advantage of both client-server streaming and pure peer-to-peer streaming, (2) at least one source peer is involved with each streaming session, which results in more stable streaming quality, (3) by considering the degree of reliability of peers in streaming load allocation, the scheme can be easily applicable to real-world service, (4) the allocation scheme easily adapts to peer's failure or network congestion against unreliable peers. We conduct comprehensive performance evaluation of the proposed scheme. Our results show that the proposed scheme, compared with legacy server with similar capacity, increases the number of clients about 67% on the average.

References

1. Akamai. http://www.akamai.com.
2. C-star. http://www.centerspan.com.
3. Calvert, K., Doar, M., and Zegura, E.: Modeling Internet Topology. IEEE Transactions on Communications, pages 160–163, December (1997)
4. Castro, M., Druschel, P., Kermarrec, A., Nandi, A., Rowstron, A., and Singh, A.: SplitStream: High-bandwidth Content Distribution in a Cooperative Environment. In *Proceedings of International Workshop on Peer-to-Peer Systems* (2003) 000–000
5. Deshpande, D., Bawa, M., and Garcia-Molina, H.: Streaming Live Media over a Peer-to-Peer Network. Stanford Database Group Technical Report **2001-20** (2001)
6. Gummadi, K.P., Dunn, R.J., Saroiu, D., Gribble, D., Levy, H.M., and Zahorjan, J.: Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of ACM Symposium on Operating Systems Principles* (2003) 000-000
7. Jin, S., Bestavros, A., and Iyengar, A.: Accelerating Internet streaming media delivery using network-aware partial caching. In *Proceedings of IEEE ICDCS 02*, Vienna, Austria, July (2002)
8. Nguyen, T. and Zakhor, A.: Distributed Video Streaming over Internet. In *Proceedings of SPIE/ACM MMCN* (2003) 000–000
9. Padmanabhan, V.N., Wang, H.J., Chou, P.A., and Sripanijkulchai, K.: Distributing Streaming Media Content Using Cooperative Networking. In *Proceedings of NOSSDAV* (2002) 000-000
10. Sen, S., Rexford, J., and Towsley, D.: Proxy prefix caching for multimedia streams. In *Proceedings of IEEE INFOCOM 99*, New york, USA, (1999)
11. The Network Simulator – ns2.: http://www.isi.edu/nsnam/ns
12. Tran, D.A., Hua, K.A., and Do, T.T.: A Peer-to-Peer Architecture for Media Streaming. IEEE Journal on Selected Areas in Communications, **22(1)** (2004) 000–000
13. Xu, D., Chai, H., and Kulkarni, S.: Analysis of a Hybrid Architecture for Cost-Effective Media Distribution. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking* , Santa Clara, CA (2003)
14. Xu, D., Hefeeda, M., Hambrusch, S., and Bhargava, B.: On Peer-to-Peer Media Streaming. In *Proceedings of ICDCS* (2002) 000–000