Peer-to-Peer Video-on-Demand with Distributed Cache

Jian-Ji Ren*, Jae-kee Lee*
*Dept. of Computer Engineering, Dong-A University
jimey@donga.ac.kr

Abstract

It is difficult to provide a video on demand (VoD) service to a large number of users via the Internet. This is due to the characteristics of VoD, which require a large bandwidth for a long playing time. The Peer-to-Peer (P2P) concept is proposed to increase the scalability of the VoD service. Many studies have been undertaken to solve this problem. But its success is still quite limited technically, these research have been deployed in a rush, mostly based on practicality and instability. A system which uses a distributed VoD streaming scheme over P2P network is proposed to support media streaming.

1. Introduction

Media streaming over P2P network has become a prevailing research topic in the past few years, because of the excellent match between the requirement of content delivery and the abundant resource in P2P systems. Recent research [1] shows that it is feasible to support large-scale media streaming in the Internet using P2P approach.

In this paper, we propose a P2P scheme to support Video-on-Demand service. It utilizes the large storage capacity of clients' cache to improve the supply of video segments so as to support the large interactive demand in a scalable manner. In our system, video content is divided into smaller segments (identified by segment IDs) and stored in storage server and nodes distributed over the network. An overlay mesh or tree is built upon distributed storage server and nodes to support play and interactive functionalities during forwarding, backwarding the videos. A node store invariable video segments in its cache. The content manage server keeps a list of the nodes which have the previous and the next video segments. The requesting node can quickly find the nodes which have the next requested segments. Furthermore, a node also keeps a list of nodes which store the same segment for load balancing purpose. If a node is added, it redirects one of children to other nodes on its list. In order to provide failure tolerant streaming service, a node has multiple parent nodes which have stored the segment of interest. The parents could be searched for in the P2P network using control protocol with the message comprising segment IDs.

Our system has the following desirable properties:

• Scalability — the system is scalable to large number of users with low server bandwidth requirement. It is completely decentralized, without the need of a server to organize overlay nodes. Every node has a limited list of the nodes

• Efficiency — users could start playing the media with low delay (without downloading the whole movie).

• Failure recovery — the system is robust to node and link failures to offer continuous streaming.

This paper is organized as follows: In section 2, we provide related work. In section 3, we give a detail description of our system. In section 4, we present our experimental results. Finally, we conclude the paper in section 5.

2. Related Work

There have been attempts in the research community to provide VoD service over P2P network [2][3], the closest work to our system is P2VoD [4]. P2VoD organizes nodes into multi-level generations according to their having the same oldest segment cached. Our system is different from P2VoD in three ways. First, nodes in our system always cache the fixed-size FIFO segments of the media streaming, there are not the generation which like P2VoD, while nodes in P2VoD cache the variable-size FIFO segments. Second, P2VoD didn't detect integrality of segments in failure recovery. Third, the nodes in our system need not to keep information of all nodes in their lists, which may be very costly. In P2VoD, the information of all nodes is kept by server.

3. Scheme Description

3.1 System parameter description

We assume a large number of nodes interested in some video content. This video content is divided into T segments. The nodes joining to the system follow the Poisson distribution (λ). The resources of the server are limited and hence, users contribute their own resources to the system, in exchange for better playback experience. To this end, the participating users form an overlay mesh which resembles a random graph. The parameters used throughout the paper are shown in Table 1. We define some definitions for scheme description:

Definition 1: An aggregate of nodes which have only one initially parent is a *cluster*.

Definition 2: In a cluster, a node which is directly received from data server is *root node*. A node which has no child nodes is *leaf node*.

Definition 3: A node which could be a parent node is called *open node*. A cluster which includes one or more open nodes is called *open cluster*.

Definition 4: If a node p_i cached *next*(p_j), but node p_i is not the parent node of node p_j. Then node p_i is *candidate parent node* of node p_j.

Table 1.The parameters used throughout the paper

Parameters	explanations
T	Whole video is divided into T segments
p_i	Node p_i
a_i	IP address of node p_i
τ	Size of cache
t_i	Joining time of node p_i
λ	Exponential times for Poisson distribution
ν	Time interval of nodes
p	Probability of $\nu \le \tau$
min(p_i)	Minimum segment IDs of node p_i
max(p_i)	Maximum segment IDs of node p_i
next(p_i)	Next requesting segment ID of node p_i
r_i	Root node of node p_i's cluster
c_i	Number node p_i's cluster
Ω_i	Aggregate open nodes of node p_i
Q_i	Aggregate candidate parent nodes of node p_i
C_i	Aggregate which $\{c_k \mid p_k \in Q_i\}$

3.2 Control Protocol

In order to maintain the multicast tree topology and transmit exactly media streaming, we requires nodes to exchange control messages with their partners. The requesting node can quickly find the nodes which have the requested segments. In order to reduce the stress of server and recover the service quickly, it keeps a list of its candidate parent nodes. Besides, to provide the nodes' efficient joining and failure recovery, every root node need to keep its cluster nodes' information.

Taking node p_i as an example, it need to record parent node's information consisting of IP address and node state. Meanwhile, it needs to record child node's information consisting of IP address, node state and requested segment ID to know which of the next segment will be sent. It need to periodically send *heart-beat* messages containing the IP addresses to its candidate parent nodes in order to confirm its candidate parent nodes' presence. Due to media streaming in on-demand streaming system, a parent-child relationship or candidate parent-child relationship can not be directly set up between any two nodes. So when node p_i sends heart-beat messages to node p_j, it checks cache segments ID of node p_j to know whether their parent-child relationship can be directly set up.

The necessary condition for node p_j being the parent of node p_i should be

$$min(p_j) \le next(p_i) \le max(p_j) \qquad (1)$$

If node p_j is not the parent of node p_i, the node p_i should delete node p_j from the list of node p_i and search upstream nodes of node p_j when *next*(p_i)<*min*(p_j) or search downstream nodes of node p_j when *next*(p_i)<*min*(p_j) until match condition (1).

If node p_i is the root node, it needs to keep a list to record the open nodes' IP address, leaf nodes' IP address node state. The server S needs to keep a list to record every root nodes' IP address, every cluster's bound of segment IDs.

3.3 Join Algorithm

Before a new node p_i joins the VoD system, it needs to send a request joining message to server S. Due to the message exchange in the control protocol; there are two cases to deal.

Case 1: If there is not open cluster, node p_i will be admitted if server S still has enough outbound-bandwidth; otherwise, node p_i will be rejected. In the former case, a new cluster is created, and node p_i is the root node of that cluster.

Case 2: If there are open clusters, server S will transmit the request joining message to the root nodes of open clusters. The root nodes will transmit open nodes' information to the new node p_i. According to the condition (1), node p_i selects the node p_j in the open nodes (Ω_i). We set that

$$P_i = \Omega_i - \{p_j\}, \quad c_i = c_j, \quad r_i = r_j$$

3.4 Failure Recovery

In P2P VoD system, failures are expected to happen often due to the leaving of nodes or the congested traffic in the overlay network. The following recovery operations will then be performed. Table 2 show the messages used throughout the paper.

For node p_i, it performs the following procedures:

Step 1: Check Q_i, if Q_i is empty, then goes to step 3. Otherwise sends REJOIN_SEARCH_BIDIRECTION to nodes in Q_i, then set Ω_i to be empty, starts the BEGIN CONVERSATION TIMER, and goes to step 2.

Step 2: REION_BE_CANDIDATE is received, adds node p_j to the Ω_i. If Ω_i is empty when the BEGIN CONVERSATION TIMER goes to the due time, then goes to step 3, otherwise, selects a parent node. If the connection is done, then goes to step 6, otherwise, goes to step 3.

Step 3: Sends REJOIN_QUERY to server S, if REJOIN_QUERY_WAIT is received, then starts the BEGIN CONVERSATION TIMER, and goes to step 4. If REJOIN_QUERY_NONE is received, then goes to step 5.

Step 4: REION_BE_CANDIDATE is received; adds node p_j to the Ω_i, if Ω_i is empty when the BEGIN CONVERSATION TIMER goes to the due time, then goes to step 5. Otherwise, selects a parent node. If the connection is done, go to step 6. Otherwise goes to step 5.

Step 5: Node p_i sends REJOIN to server S. If REJION_SUCCESS is received, then goes to step 6. If REJION_FAIL is received, then goes to step 7.

Step 6: Node p_i gets a parent node. If the parent node is server S, then updates c_i and sends CLUSTER_HEAD_CHANGE to its child nodes. When the

parent node is p_j, $c_{i=}\ c_j$, then ends the recovery operation. Otherwise, sets $c_{i=}\ c_j$, $r_{i=}\ r_j$ and sends CLUSTER_HEAD_CHANGE to its child nodes, and ends the recovery operation.

Step 7: Node p_i sends QUIT to child nodes.

For server *S*, it performs the following procedures:

Step 1: If REJOIN_QUERY is received, then goes to step 2. If REJOIN is received, then goes to step 3.

Step 2: If there is a node p_j, which cached $next(p_i)$, cj isn't included in C_i, sends REJOIN_SEARCH_DOWN to r_j sends REJOIN_QUERY_WAIT to node p_i. Otherwise sends REJOIN_QUERY_NONE to node p_i.

Step 3: If server *S* still has enough outbound-bandwidth, sends REJION_SUCCESS to node p_i. Otherwise sends REJION_FAIL to node p_i.

For other node p_k, it performs as the following procedures:

Step 1: If REJOIN_SEARCH_BIDIRECTION is received, then goes to step 2. If REJOIN_SEARCH_UP is received, then goes to step 3. If REJOIN_SEARCH_DOWN is received, then goes to step 4. If CLUSTER_HEAD_CHANGE is received, then goes to step 5.

Step 2: Sends REJOIN_SEARCH_DOWN to child nodes of p_k if $next(p_i) < min(p_k)$. Sends REJOIN_SEARCH_UP to parent nodes of p_j if $next(p_i) > max(p_k)$. Sends REION_BE_CANDIDATE to node p_i, sends REJOIN_SEARCH_UP to the parent nodes of p_k, and sends REJOIN_SEARCH_DOWN to the child nodes of p_k if $\min(p_k) \le next(p_i) \le \max(p_k)$.

Step 3: Sends REION_BE_CANDIDATE to node p_i, and sends REJOIN_SEARCH_UP to parent nodes of p_k if $\min(p_k) \le next(p_i) \le \max(p_k)$. Sends REJOIN_SEARCH_UP to parent nodes of p_k if $next(p_i) > max(p_k)$. Aborts message if $next(p_i) < min(p_k)$.

Step 4: Sends REION_BE_CANDIDATE to node p_i, and sends REJOIN_SEARCH_ DOWN to child nodes of p_k if $min(p_j) \le next(p_i) \le max(p_j)$ .sends REJOIN_SEARCH_ DOWN to child nodes of p_k. Aborts message if $next(p_i) > max(p_k)$.

Step 5: sets $c_{k=}\ c_i$, $r_{k=}\ r_i$, sends CLUSTER_HEAD_CHANGE to child nodes of p_k.

4. Performance Evaluation

In this section, we examine the performance of our system analytically.

Two performances metrics are used: (1) Server stress- the amount of bandwidth used at the server, or the number of streams established at the server. It indicates how congested the video server is; (2) Rejection probability- the probability that a client tries to join the system but can not get the service. This metric illustrates which protocol uses the network resources (bandwidth in this case) more effectively. This metric illustrates which protocol uses the network resources.

4.1 Simulation Setup

In the simulation, the underlying network topology is created using GT-ITM [5]. The whole network consists of one transit network (with 4 nodes), and 12 stub domains (with 96 nodes in total). Assume that each node in this network represents a local area network with the ability to host unlimited number of clients, and to have enough bandwidth to support media streaming. Routing between two nodes is determined by using the shortest path algorithm. Assign bandwidth capacity to links in the network as follows: a backbone link (at least one end point of the link is a transit node) can support 25 concurrent media streams, and an edge link (any link in the network, which is not a backbone link) can support 7 concurrent media streams.

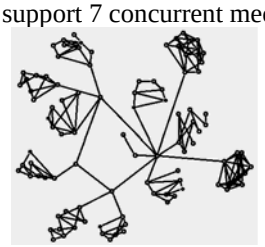

Fig.1 Network topology

Clients arriving to the system follow the Poisson distribution, and are randomly placed into one of the network nodes. Each host receives data from a parent in its upper group and forwards it to its children in its lower group. A

Table 2. The messages used throughout rejoining algorithm

Message name	Parameters	Explanations
REJOIN_SEARCH_BIDIRECTION	$a_i, next(p_i)$	Send from p_i to p_k , its meaning is to search new parent node on multicast tree bidirectional which starts from the destination node
REJOIN_SEARCH_DOWN	$a_i, next(p_i)$	Sent by p_i , its meaning is to search new parent node on multicast tree in the data retransmitting direction
REJOIN_SEARCH_UP	$a_i, next(p_i)$	Sent by p_i , its meaning is to search new parent node on multicast tree in opposite direction of the data retransmitting
REJOIN_QUERY	$a_i, next(p_i), C_i$	Send from p_i to server *S*, its meaning is to query *S* that if any node locating in other clusters (excluding the cluster C_i) has cached the data segment $next(p_i)$
REJOIN_QUERY_WAIT		Wait for the server *S* message
REJOIN_QUERY_NONE		The query is failing
REJION	a_i	Send from p_i to server *S*, its meaning is that wish to join the server *S* directly
REJION_SUCCESS		Send from server *S* to p_i , its meaning is that server is parent node of p_i
REJION_FAIL		The connection is not created
REION_BE_CANDIDATE	a_j	Send from p_j to p_i , its meaning is that p_j can be a candidate parent node of p_i
CLUSTER_HEAD_CHANGE	r_i, c_i	Sent by p_i , its meaning is to notify the destination node to change its cluster number and its cluster header
Quit	a_i	Node p_i quit system

new node tries to join the lowest group or forms a new lowest group. If it fails to find an available parent from the tree and the server has enough bandwidth, it directly connects to the server. In the experiment, use Smallest Delay Selection for parent selection process. And, set the system parameter K = 6 and the cache size MB to be 5 minutes of video in the system (K is the maximum number of clients allowed in the first generation of each video session). The simulation system is built by NS-2 [6]. In the simulations, the length and bit rate of a movie are 120 minutes and 1 Mbps, respectively. Each segment is 5 minutes long and of size about 36 MBytes. Each peer stores one video segment at its local storage. The peers participating in the system follow a Poisson arrival, each being randomly attached to a router node and request the video from the beginning. Group size, i.e. the total number of peers in a measurement session, varies from 200 to 12800. The underlying network topology is created using GT-ITM. In case a new client is unable to find available parent(s), the client is rejected to be served.

4.2 Simulation Results

(1) Server Stress We first compare the system server stress. Fig.2 illustrates the relationship between server stress and parameter K when fixing the value of MB. The first observation is that the server is most stressed when the workload is lightest. This seemingly counter-intuitive phenomenon can be explained as follows. When the workload is light, clients arriving to the system are far away from each other in time. Therefore, a new client hardly finds an open video session to join, which implies that the server has to create a new video session for almost every client. On the other hand, when the workload on the system is heavier, clients arriving to the system are close to each other in time. This makes a new client be more likely to find an open video session to join, which also means a new client usually gets the video stream from another client, not the server. In short, almost every clients get the video stream directly from the server in the case of light workload, while in the case of heavy workload most clients get the video stream from some other client, not the sever.

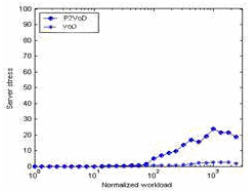

Fig. 2. Server stress

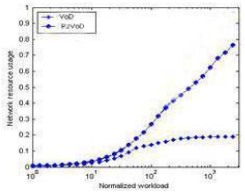

Fig. 3. Rejection probability

(2) Rejection Probability Fig.3 shows the rejection probability as a function of MB and the normalized workload. With a value of MB, the rejection probability increases with the increasing size of the normalized workload. This is due to the limited available bandwidth of the underlying network. When there are already a large number of clients in the system, the probability that a new client can find a path with sufficient bandwidth to get the stream from any existing client is smaller. Since the rejection probability is a good indicator of how scalable a system is, MB can be used to adjust the system to a desirable scalability.

5. Conclusions

In this paper, a scheme system is proposed to support VoD service in P2P networks. This system utilizes the large cache capacity of peers to amplify the supply of videos so as to easily support the large demand in a scalable manner. In this system, videos are divided into smaller segments and cached in peers distributed over the Internet. A video mesh is built upon peers to support playback forwarding and backwarding during playback. A peer, who has a video segment cached in its cache, connects to the peers who have the same, the previous and the next video segments. As a result, its children can be redirected to the peers who have the required segments. Through simulation, it shows that the system outperforms a recent research work, P2VoD. It shows that the system has low segment missing rate under random member join/leave and random injection of background network traffic, which means that the video quality is good in the presence of group dynamics and bandwidth fluctuation. Adding pointers to peers which are caching video segments far away from the current playback position can help a user seek a far away position more efficiently.

How to add other pointers in the mesh and the effect of adding them is the future work. Also, in current implementation, peers randomly choose video segments to download. Since a popular segment is more in demand than the others, it is better to balance the supply and demand in a strategically manner. However, how to obtain the popularity of each segment in a video and how to update it dynamically are very challenging in P2P VoD system.

References

[1] X. Zhang, J.C. Liu, B. Li, and P. Yum. Coolstreaming/Donet: A data-driven overlay network for efficient live media streaming. In Proceedings of IEEE INFOCOM, March 2005.

[2] Guo Y, Suh K, Kurose J, Towsley D. P2Cast: P2P patching scheme for VoD service. In: Proc. of the WWW 2003. New York: ACM Press, 2003. 301−309.

[3] M. Zhou and J. Liu, A Hybrid Overlay Network for Video-on-Demand, in Proceedings of IEEE International Conference on Communications (ICC), Seoul, Korea, May 2005.

[4] Do T, Hua K, Tantaoui M. P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. In: Proc. of the IEEE ICC 2004. Paris: IEEE Communications Society, 2004. 1467−1472.

[5] Calvert K, Doar M, Zegura E, “Modeling internet topology,” IEEE Communication Magazine, 35(6), pp. 160-163 (1997).

[6] NS-2, http://www.isi.edu/nsnam/ns