Distributed Cache to Support Video-on-Demand Service over P2P Network

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

Abstract

P2P Video-on-Demand (VoD) is becoming a popular service in the Internet. The first deployments are already in place, and millions of customers worldwide are already using this new medium. But its success is still quite limited and technically, these researches have been deployed in a rush, mostly based on practicality and instability. This paper studies the advantages and potential problems of VoD in peer-to-peer (P2P) network, we propose a system which uses a distributed VoD streaming scheme over P2P network to support media streaming.

1. Introduction

Peer-to-peer (P2P) computing research has provided lots of solutions to many network applications. P2P systems have been used in scalable distributed applications.

In P2P systems, cooperative peers self-organize themselves into overlay networks. Each peer in an overlay network acts as a caching and relaying data for other peers. P2P system is an elegant alternative in which each peer-host may act as a potential server for other clients.

Although there have been extraordinary research in P2P systems during the past few years, one category of P2P systems has so far received less attention: the P2P media streaming system. The major difference between a general P2P system and a P2P media streaming system lies in the data sharing mode among peers: the former uses the 'open-after-downloading' mode, while the latter uses the 'play-while-downloading' mode.

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 interactive video service system. 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 local 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.

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 some 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 are kept by server.

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

Many existing work on P2P research has been conducted

in assessing the viability of P2P live streaming, and some encouraging results have been reported. With the widespread deployment of broadband access, P2P VoD service has become a very popular item in the Internet.

Fig.1 depicts the general framework of a typical P2P-based VoD system. In such a system, cooperative peers are organized into an overlay network via unicast tunnels. There are two overlay networks in the VoD system, i.e., the index overlay for locating the users with expected data and the data overlay for cooperative content delivery. The streaming content is split into a sequence of segments, each of which is a smallest playable unit, and the server distributes these segments among clients with asynchronous demands. Each client caches a limited number of segments around its "play offset", which is the sequence number of the playing segment. The client exchanges the available segments with its partners, which have close play offset and thus can help the client to get the expected data with high probability. For the users in the VoD system, when joining or implementing VCR operations, it needs to search for the partners through the index overlay. After locating the partners, the user will collaborate with them to fetch and deliver the content.

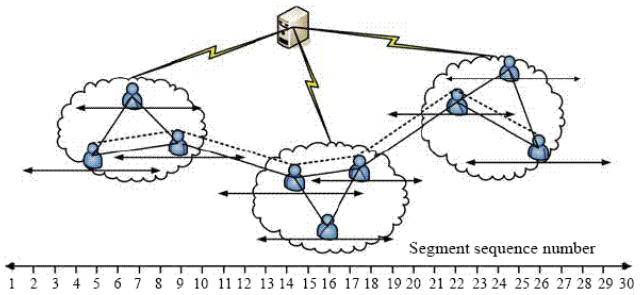

Fig.1 Typical P2P-based VoD system

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.

Considering the condition that there are actions of leaving, jumping, and invalidating, and these actions will lead to the child nodes' service intermittence, we propose a failure recovery that it keeps candidate parent nodes' state information in child nodes. When media streaming service intermits, child node can search new parent node quickly and exactly according to the candidate parent nodes' state information. There is no need to search the whole multicast tree and no necessary to login server S when the service intermittence happens, so it efficiently reduces the stress of server.

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

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

Otherwise, node p_i can not connect the node p_j of the open nodes (Ω_i).The node p_i resends a request joining message to server S. This case is similar to Case1. The different point is $P_i = \Omega_i$.

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

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

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.

In the following analysis, we assume that if a node can receive the content segment from a node, it will do so rather than obtain the video from the server. In addition, according to the parameters in Table 1, it is presented as a theorem, to be used in proving the main theorems on stress of sever.

Theorem1: There are a large number of nodes interested in some video content. This video content is divided into T segments. The nodes cache the fixed-size FIFO segments τ. The nodes joining to the system follow the Poisson distribution (λ). The average time interval of root nodes is $E(y)$. Then the average number of used server channels $E(x)$ is $E(x) = \lambda T e^{-\lambda\tau}$

Proof: Consider our system with a single content serv er, then

$$E(x) = \frac{T}{E(y)} \qquad (2)$$

The average time interval of root nodes $E(y)$ is

$$E(y) = \sum_{i=0}^{\infty} p^i (1-p)(iE(v|v \le \tau) + E(v|v > \tau)) \qquad (3)$$

The average nodes time interval of $v \le \tau$ is

$$E(v|v \le \tau) = \frac{1}{P\{v \le \tau\}} \int_0^{\tau} vf(v)dv = \frac{1}{\lambda} - \frac{\tau e^{-\lambda\tau}}{1-e^{-\lambda\tau}} \qquad (4)$$

Meanwhile, the average nodes time interval of $v > \tau$ is

$$E(v|v > \tau) = \frac{1}{P\{v > \tau\}} \int_{\tau}^{+\infty} vf(v)dv = \frac{1}{\lambda} + \tau \qquad (5)$$

With the relationship among (2) (3) (4) and (5), we can conclude

$$E(x) = \lambda T e^{-\lambda\tau}$$

We observe that the average server stress is determined by the Poisson distribution (λ), length of the video (T), size of cache (τ), not by the scale of system. According to the theorem1, the system is scalable to large number of users with low server bandwidth requirement.

5. Conclusions

Efficient and scalable VoD service over P2P network has become a hot topic recently. In this paper, we propose distributed scheme for P2P VoD service. It has combined the best features of cluster and message dissemination: low delay with a regular cluster, and robust delivery with control protocol. Furthermore, a methodology is formulated to examine our system. Performance evaluation results show that it can achieve scalable application. As future work, we are planning to validate our results by performing experiments.

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.