PeerMsm: a Scalable P2P Model for Streaming Media

Ying YANG¹,a, Lei YANG²,b

¹College of Computer & Electronic Information, Guangxi University, P.R. China
²Institute of Application Physics, Guangxi Science Academy, P.R. China

ayingy2004@126.com, bleiy2004@126.com,

Keywords: P2P, Streaming Media, Access Control.

Abstract. Streaming media over Internet is becoming one of the main Internet services, but the limitation of network and server resource makes existing systems the bottleneck of servicing. In this paper, we propose the scalable servicing system based on P2P networks. Through constructing the model of PeerMsm and the FSM of public host, meanwhile, utilizing a new caching and cluster scheduling algorithm, our system makes the media server high QoS and low cost so as to satisfy demands of more users.

Introduction

Streaming Media over Internet is becoming one of the main Internet services, and how to improve the servicing capacity of system is a hot topic. However, the limitations of network and server resource make existing servicing systems low capacity, low QoS and high cost.

P2P has been used wildly in file transferring systems and instant message communication, and gradual application in media on-demand (MoD) system[1][2][3]. C.S.Lin and M.J.Yan presented a P2P MoD system[4] which a dynamic peer buffer adjustment method to improve service availability. But root server chooses application server for host, which increase root server’s overhead significantly. A.Ghaffari Sheshiavani and B.Akbari proposed an adaptive buffer-map exchange mechanism for pull-based in p2p VoD system[5]. Hosts receive and forward data at the same time, but don’t cache data for forwarding later. An Efficient Peer-to-Peer Scheme for Media Streaming[6] let hosts access same clip and construct a multicast tree. Because hosts don’t cache data, the host join the tree later have to get the data that have transmit on the tree from root server, so root server has to manage many additional patching streaming [7].

This paper proposes PeerMsm, a scalable MoD servicing system based on P2P networks. In PeerMsm, the media server divides each clip into several segments; the peerhosts caching same segment consists of a cluster, and several clusters work together to transmit a whole clip to users. Sharing the peer hosts’ resource makes the media server not the bottleneck of servicing system, and system can satisfy more concurrent users.

The Model of PeerMsm

Basic architecture of PeerMsm consists of a WEB server, a streaming media server and hosts that accessing clips storing on media server. This paper presents the model of PeerMsm, and utilizes Finite State Machine (FSM) to describe behaviors of a host in the servicing system. Figure 1 presents the model of PeerMsm. In the model, media server divides clips into n segments. The host that can receive, cache, and forward data at the same time is called public host (Pub-host). Peer host caches one segment at most. The host caching the first segment consists of the cluster named cluster-1 (clu-1), and so on. The host that only receives data is called private host (Pri-host). Host sends requests to WEB server, which redirects the request to media server. Media server maintains the host clusters’ member information, and sends information to host. Pub-host tries to get media data from a member in host cluster firstly. If it can’t get service from a member, Pub-host gets media data from media
server. Peer host will join a host cluster after caching a segment. Web server is the rendezvous of host tree. Besides redirecting hosts to streaming media server, web server maintains and issues the basic information of the clips, such as clip’s introduction, price, etc. Logically, hosts accessing same clip construct a host tree. We call the host sending data parent node, and the host receiving data descendant.

Figure 1. Model of PeerMsm.

The model in figure 1 is two-level (server level and host level), and is designed for a large autonomous network, such as a campus network, etc. It can be scaled a three-level structure, the highest level is a central media library that stores all clips, the second level is composed of streaming media servers and cache servers that are deployed in autonomous networks and the third level includes the hosts that access clips.

Figure 2 describes finite state machine (FSM) of a host. Host in init isn’t a member of PeerMsm, and host becomes a member after entering recv. Each host has playback cache, which is used to keep playback smoothly when host enter temp_con or temp_ser. Besides playback cache, peer host has cache one of some clip’s segments. Compared with playback cache, the duration of a segment is much longer. So, peer host caches the segment on hard disk as a temporary file. Pri-host will change from recv to init after finishing playback. Peer host download a segment from parent node in recv. After download, peer host joins in the segment’s host cluster and enter recv_ser. In recv_ser, peer host can playback and forwarding at same time. $M$ is number of current descendants. The interval that peer host stay in recv_ser is called peer host’s life span. At the end of life span, peer host enter serv_end, and doesn’t accept any requests, but continue providing service to current descendants. When $m=0$, peer host has no descendant, and return to init. Besides this, the owner of peer host can execute exit to make peer host return to init before the end of life span. Host failure and link failure
can make host enter error state. Peer host’s failure changes itself from \textit{recv}, \textit{recv\_ser}, \textit{serv}, or \textit{serv\_end} to \textit{host\_err}. When in \textit{serv} or \textit{serv\_end}, peer host doesn’t need the support from parent nodes any more, so it doesn’t care about the failure of parent nodes and uplink. When in \textit{recv} or \textit{recv\_ser}, peer host can’t distinguish the difference between failure of parent node and failure of uplink. So, if peer host can’t communicate with parent node, it enters \textit{temp\_con} or \textit{temp\_ser}, and use playback cache to keep playback smoothly in a short time. Peer host must connect with original parent node or create a link with new parent node before depleting playback cache. Otherwise, peer host will enter \textit{link\_err} after depleting playback cache.

**Host Caching Strategy**

Caching several seconds or one minute of data makes host be able to tolerate the variation of QoS on network and server, and playback the clip smoothly, but can’t satisfy the requirement of PeerMsm. Due to the unpredictability of the request’ arrival time, the duration time of media data cached on host must long enough to increase the utilization of cached data. Obviously, the longer the data stay on host, the higher the probability of the data used by other hosts. The duration time of cached data is related to the host’s cache space. The larger the cache space is, the longer the data’s duration time is, but large cache space will occupy host’s resource too much. Besides this, PeerMsm system hopes the hosts to run as a parent node as long as possible to keep the cached data available to other hosts. But, host has the trend of leaving host tree as soon as possible at the end of playback. So, we design a cache strategy to take advantage of hosts’ shared resource efficiently.

Caching a segment is the foundation of providing service to other hosts. Media server has stored many clips. Each clip is divided into several segments, and each segment maps a host cluster. Dividing a clip with longer segments can reduce the number of parent nodes host connect to get a whole clip, but the connect time with each parent node is extended. Due to the unpredictability of parent host, the longer connect time will increase the unpredictability of the QoS, and increase the overhead of parent node. So, dividing a clip should pursue the balance between host’s overhead, QoS and parent node’s overload. Besides this, a smaller segment can shorten the download time, and make peer host join host cluster earlier. Media server specifies which segment peer host should cache. The clips’ access probabilities are Zipf distribution. In order to improve the utilization of peer host’s shared resource, peer host only caches the segment that belongs to the first \(k\) clips with highest accessing probability. Media server divides \([0,1]\) into \(k\) intervals, and the width of interval \(i\) is mapped to the accessing probability of clip \(i\). When peer host enter \textit{recv}, media server creates a random number \(p\), \(0 \leq p \leq 1\), and specified the clip ID of segment that peer host will cache according to \(p\).

Normally, the clip peer host playback isn’t the clip peer host cache, so peer host creates two uplinks with two parent nodes to playback and download at same time. Media server specify segment ID for peer host, too. The accessing probability of clip’s head is higher than the accessing probability of clip’s tail, so the number of peer hosts that cache clip’s beginning segment should be more than other segments. Media server uses a three-dimension list to maintain host cluster. \(x\) dimension is clip ID, and \(y\) dimension is segment ID, and \(z\) dimension is host ID. Let \(i \in x, j \in y, k \in z\). The peer host caching the \(j^{th}\) segment of the \(i^{th}\) clip is a member of \(C_{ij}\), and \(h_{ij}\) is register information of the \(k^{th}\) member of \(C_{ij}\). The register information of peer host includes clip ID, segment ID, host address, join time, and its life span. If peer host fail silently, media server will delete its registration information at the end of its life span.

Let the duration time of clip is \(t_d\) units, and host can cache \(t_c\) units, \(t_d > t_c\). Host caches and forwards the data received from parent node. Due to \(t_d > t_c\), after cache is filled, the oldest data in cache space is given up to reclaim resource for the new arrival data. Let cache space of host \(h_i\) is \(t_c\), and \(h_i\) begin to receive data from server at \(t_i\). Host \(h_j\) choose \(h_i\) as parent node, and begin to get data from \(h_i\) at \(t_j\). If \(t_j - t_i \leq t_c\), \(h_j\) can get all of the data \(h_i\) caches. Normally, if the interval between the arrival times of two consecutive requests that accessing clip \(c\) is always smaller than the duration time that media data cached on a host, one host tree can satisfy all of the users that access clip. Though it’s impossible to
assure above statement, making the host tree as deep as possible can make use of the resource of host sufficiently. But along with the increase of depth of the tree, the number of hosts influenced by the failure of intermediate host in host tree will increase remarkably. So, we define a constant time $t_c$ for host tree. An host tree created at $t$ when a host become the immediate descendant of media server, and other host can only join this tree before $t+t_c-t_i$, $t_i$ is the interval between the time host send request and the time host join host tree. And $t_c-t_i$ is the duration time that media data cached on a host. Media server records the deadline of each host tree. For every clip, at any time, at most one host tree available for hosts to join. In this way, we can control the weak consistency of the host tree.

**Cluster Scheduling Algorithm**

In a server cluster, a dedicate scheduler chooses the server with lowest workload to provide service to new arrival request. The merits that scheduler using to choose server include throughput, number of links, and response time, etc. Scheduler must keep track of the servers’ state real time to make correct choice. In PeerMsm, keeping track of the state of peer hosts real time and choosing parent node for each host increase media server’s overload heavily, and will balance the benefit of using peer host to reduce server’s overload. Therefore, host itself gets cluster information from media server, and makes its decision by getting state information from each member in the cluster. The other parameter related with peer host’s factor is the round trip time $tr$ between sender and peer host. A smaller $tr$ means the distance between sender and peer host closer. Cluster Scheduling Algorithm is described as flow,

1. Host sends a send_request message to media server, and the return value (address of members in host cluster) is store in array peer_host.
2. Host uses get_status message to get peer hosts’ state.
3. Host calculates each peer host’s factor, and choose peer host with largest factor as parent node.
4. Host connect parent node with connect(), if return is true, host get service from this node, otherwise connect the node with largest factor in the left peer hosts.
5. If no peer host can provide service, host gets service from media server directly.
6. In order to control the overload caused by forwarding segment, peer host specify $num$, the number of descendants it can support concurrently.
7. Peer host uses cur_num to record the number of current descendants, and free_num to record free links. It’s clearly that, $num$ is the sum of cur_num and free_num.
8. When receiving get_status message, peer host responses with its free_num.
9. Let host send request message to $n$ peer host, and get $free_num_i$ and $tr_i$ of peer host $i \in n$, Then the factor of peer host is calculated as below,

$$
\text{factor}_i = p_1 \left( \frac{num_i}{\sum_{i=1}^{n} num_i} \right) - p_2 \left( \frac{tr_i}{\sum_{i=1}^{n} tr_i} \right) \tag{1}
$$

In equation 1, $p_1$ and $p_2$ are parameters, and $p_1 + p_2 = 1, 0 \leq p_1 \leq 1, 0 \leq p_2 \leq 1$. The first item in equation 1 calculates peer host’s num ratio, and the larger the ratio is, the higher the probability of peer host accepting sender is. The second item in equation 1 calculates peer host’s tr ratio, and the larger the ratio is, the farther the distance between peer host and sender is. The second item has side effect on peer host’s factor, so its sign is negative.

The algorithm uses parameters to control the effect of num and tr of peer host’s factor. Host has to connect with several parent nodes to playback clip completely. To keep playback smoothly, host get the members’ register information of next cluster form media server and choose new parent node before the end of playback current segment. If host itself cache next segment, host needn’t require media server provide cluster information. When parent host or uplink is failed, host should choose new parent node. Because the state of members in the host cluster changes during host playback, host should get the update information from media server before choosing new parent node.
Through utilizing innovative caching strategy and cluster scheduling algorithm, PeerMsm makes the media server not the bottleneck of media on-demand servicing and satisfy more users on-line concurrently.

Summary
The paper proposes the model of PeerMsm, the FSM of the public host, meanwhile, gives host cache strategy and cluster scheduling algorithm. PeerMsm needs more peerhosts with longer life span to share their resource with other hosts. So, PeerMsm should develop a rewarding strategy to encourage host work as peer. The other works in the future is optimizing cache strategy and cluster scheduling algorithm.

Acknowledgement
The paper was supported by the Fund of Guangxi Natural Science (2013GXNSFAA019344), Gui Financial Education Number [2013] 19. It was also supported by the Fund for the fourth batch of distinguished experts in Nanning City.

References