HHMSM: A HIERARCHICAL HYBRID MULTICAST STREAM MERGING SCHEME FOR LARGE-SCALE VIDEO-ON-DEMAND SYSTEMS

Hai Jin and Dafu Deng
Huazhong University of Science and Technology, Wuhan, 430074, China
Email: hjin@ hust.edu.cn

ABSTRACT

The key performance bottleneck for large-scale video-on-demand (VoD) systems is the server bandwidth, which controls the number of clients a video server can support. Two existing stream scheduling schemes can save server bandwidth significantly by using multicast method to transmit video data: the batching scheme and the patching scheme. However, the batching scheme results in long start-up latency and high reneging probability. The patching scheme does not work well at high client request rates due to mass retransmission for same video data. In this paper, we propose a hierarchical hybrid multicast stream merging scheme, called HHMSM, which can save server bandwidth significantly over a wide range of client request rates. Furthermore, the start-up latency raised by the HHMSM scheme is far less than that of the batching scheme.

1. INTRODUCTION

In video-on-demand systems, most clients usually request several hot video objects at the same time interval [1, 3, 5]. A conventional stream scheduling scheme simply schedules a unique stream for each client request. This makes that video servers send lots of video data streams during a short time interval. It wastes lots of server bandwidth, and better solutions are necessary.

The batching stream scheduling scheme [1, 3, 4, 5] uses a single multicast stream for clients that request the same video object in a time slot. In order to obtain good bandwidth efficiency, the batching time interval must be at least 7 minutes [2]. Therefore, the expected start-up latency is approximately 35 minutes. The long start-up delay increases the client reneging rate.

The patching scheme [2, 6, 8, 9] presents a good idea to solve the long start-up latency problem. In the patching scheme, a video server schedules a complete multicast stream immediately when the first client request arrived. For those clients who request the same video object during the multicast stream being transmitted, the server notifies them to receive the remaining of the complete multicast stream. Simultaneously, it schedules a unicast stream for every one to patch the lost video data. Later starters must concurrently receive both the complete multicast stream and the patching unicast stream, and then merging them into a unicast video. The patching scheme guarantees the zero start-up latency and saves the server bandwidth effectively at low or moderate client request rates. However, the client request rate is higher, and the amount of retransmitted same video data is larger. For a large-scale VoD system, video server often works at very high client request rates. Thus, it is necessary to develop some approaches for reducing retransmitted video data.

This paper contributes in developing a novel hierarchical hybrid multicast stream merging scheme, called HHMSM, which significantly save the server bandwidth over a wide range of client request rates. The following sections are organized as follows. In section 2 we describe HHMSM scheme. Section 3 presents the experiment results for the performance evaluation. Finally, Section 4 ends with conclusions.

2. HHMSM SCHEME

Because video data can not be shared among clients requesting different video objects, a video server handles those clients’ requests independently. Thus, we just consider requests for the same video object in this section (The general case will be studied in section 3).

The novel HHMSM scheme combines the advantages of the batching scheme and the patching scheme. In particular, clients that request the same video are repeatedly merged into larger groups, leading to a hierarchical hybrid merging structure. Furthermore, clients merge different stream video data into a complete video by buffering the received video data on client disk rather than altering server sending speed or altering client play rates.

2.1. HHMSM delivery technique

Key elements of the hierarchical hybrid multicast stream merging technique include:

1. A time interval T must be selected firstly. Based on selected T, time is divided into lots of small time slots, denoted by \( t_0, \ldots, t_\infty \). The requested video object with \( L \) minutes length is divided into \( \lfloor L/T \rfloor \) fix sized video segments, denoted by \( S_0, \ldots, S_{\lfloor L/T \rfloor - 1} \). Note that server sending speed is equal to client play rates so that each video segment is exactly transmitted completely in a time slot.

2. A video server only schedules streams at the end of time slot. Each data transmission stream is a multicast so that any client can listen to the stream. Thus, clients arriving at the same time slot can be batched together and served as one client.
The key issue affecting the performance of video servers is the selected time interval \( T \). This value determines the number of streams that can be concurrently received by a client. If \( T \) is too small, the number of concurrently received streams may be increased dramatically and the bandwidth of clients may be exhausted. Also, when the value of \( T \) is decreasing, the required server bandwidth will be increased due to less clients being batched. However, if \( T \) is too large, the start-up latency of clients may be too long to be endured. Here, we only consider the effect of client bandwidth consumption and the start-up latency affected by the time interval. The server bandwidth consumption affected by time interval will be studied in section 3. Firstly, we give out some definitions as following:

- \( b_{\text{max}} \): In different network environment, clients have different network bandwidth capacity. We use the notation \( b_{\text{max}} \) to represent the maximum client bandwidth capacity. \( b_{\text{max}} \) is in unit of stream. For example, supposing that the maximum bandwidth of clients is 100Mbits/s and the transmission stream is formatted in MPEG-I (near 1.2 ~ 1.5Mbits/s per stream), \( b_{\text{max}} \) shall be near 100/1.5 \( \approx \) 65 streams.

- \( B_{\text{max}} \): Clients may concurrently receive lots of streams. We use the notation \( B_{\text{max}} \) to represent the maximum total number of streams that may be concurrently received by a client. \( B_{\text{max}} \) is in unit of stream.

- \( T^* \): the optimal time interval which not only guarantees that clients have enough bandwidth to concurrently receive all streams, but also minimize the client average start-up latency.

Assuming that the complete multicast stream is initiated at time \( t_0 \). Consider a client request, denoted by \( R_k \), which arrives at the time slot \( t_k(0 < k \leq \lceil L/T \rceil) \). Use \( PS_j \) to represent the first patching multicast stream from which client \( R_k \) can receive one or more segments, where \( PS_j \) is initiated at time \( t_j(0 < j \leq k) \). We can find out that the last segment transmitted on \( PS_j \) is \( S_j \). The start transmission time for the segment \( S_j \) must latter than or equal to the time \( t_k \). Therefore, we can derive that \( 2j \geq k + 1 \). In addition, if client \( R_k \) can receive one or more segments from each of following patching multicast streams. The number of concurrently received streams access to its maximum value. Thus, we can derive that

\[
B_{\text{max}} \leq (k + 1)/2 \leq \lceil L/(2T) \rceil
\]  

In order to guarantee that clients have enough bandwidth to concurrently receive all streams that transmit valid video segments to them, \( b_{\text{max}} \) must be larger than or equal to \( B_{\text{max}} \). Thus, we can obtain

\[
T \geq L/(2b_{\text{max}})
\]

The smaller the selected time interval \( T \) is and the shorter the start-up latency is. Thus, the optimal time interval \( T^* \) shall be the minimum value of \( T \). That is:

\[
T^* = \lceil L/(2b_{\text{max}}) \rceil
\]
Table 1. Parameters for experiment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video length (minutes) L</td>
<td>90 ∼ 120</td>
</tr>
<tr>
<td>The number of stored videos</td>
<td>200</td>
</tr>
<tr>
<td>Video format</td>
<td>MPEG-I</td>
</tr>
<tr>
<td>The maximum client bandwidth (Mbits/s)</td>
<td>100</td>
</tr>
<tr>
<td>The maximum server bandwidth (Mbits/s)</td>
<td>1000</td>
</tr>
<tr>
<td>Client request rates (requests/hour)</td>
<td>200 ∼ 1600</td>
</tr>
</tbody>
</table>

3. PERFORMANCE EVALUATION

Typically, multiple videos with varying popularity are stored in a video server. In this section, we demonstrate the performance of the HHMSM scheme via experiment. Firstly, we describe our experimental environment. Then, the performance of HHMSM scheme will be analyzed systematically at two aspects: the start-up latency and reneging probability of clients, the bandwidth consumption for video servers.

3.1. Experimental parameters

We had to decide two factors for each video: length and popularity. For its length, the data from Internet Movie Database (http://www.imdb.com) has shown a normal distribution with a mean of 102 minutes and a standard deviation of 16 minutes. The popularity of each video was modeled using a Zipf-like distribution. Empirical evidence suggests that the parameter θ of the Zipf-like distribution is 0.271 to give a good fit to real video rental [1, 4, 5]. In addition, the number of videos stored on parallel video server is 200 and the video format is MPEG-I.

Client requests were generated using a Poisson arrival process with an interval time of $1/\lambda$, for varying λ values, between 200 to 1600 arrivals per hour. Once generated, clients simply selected a video, and waited for their request to be served. The waiting tolerance of the client is independent of each other, and each is willing to wait for a period time μ ≥ 0 minutes. If its requested movie is not displayed by then, it reneges (Note that even if the start time of a video is known, a client may lose its interest in the video and cancel its request. If it is delayed too long, in this case, the client is defined "reneged"). We consider the following exponential reneging function $R(\mu)$, which is used by most VoD studies [1, 4, 7]. Clients are always willing to wait for a minimum time $U_{\min} ≥ 0$. The additional waiting time beyond $U_{\min}$ is exponentially distributed with mean τ minutes, i.e.

$$R(u) = \begin{cases} 0, & 0 ≤ u ≤ U_{\min} \\ 1-e^{-(u-U_{\min})/\tau}, & Otherwise \end{cases}$$ (4)

Obviously, the larger τ is, the more delay clients can tolerate. We chose $U_{\min} = 0$ and τ = 15 minutes in our experiment. If the client is not reneging, it simply plays back the received streams until those streams is transmitted completely. Other parameters are showed in Table 1.

3.2. Results

As discussed in section 2.2, in order to guarantee clients receive all segments of their requested video objects, the minimum value of time interval (i.e. optimal time interval $T^*$) shall be $L/(2B_{\max})$ ∼ 0.5. During server bandwidth consumption of video servers. In Fig. 2, we show the average server bandwidth consumption vs. request arrival rates.

Fig. 2. The average start-up latency vs. request arrival rates

Fig. 3. The client reneging probability vs. request arrival rates

Fig. 4. The average server bandwidth consumption vs. request arrival rates
120/2 \times 60 = 1 \text{ minute}. \text{ We choose the time interval } T \text{ to be 1, 5, 10, 15 minutes for studying the effect on the server performance. For schemes that are compared with the HHMSM scheme, we choose the FCFS batching scheme with parameter 7 minutes batch time interval [4] and an optimal time-threshold patching scheme [8].}

Fig. 2 and Fig.3 shows the results at the aspect of the start-up latency and the reneging probability respectively. For the HHMSM scheme, the average start-up latency and the reneging probability are increased dramatically by the increasing of time interval \( T \). When \( T \) is equal to optimal time interval \( T^* = 1 \text{ minute} \), the average start-up latency is less than 45 seconds and the average reneging probability is less than 5%. But when \( T \) is equal to 15 minutes, the average start-up latency is increased to near 750 seconds and almost 45% clients reneging. We can conclude that the HHMSM scheme outperforms the FCFS batching scheme significantly in these two aspects when the selected time interval for the HHMSM scheme is the optimal time interval \( T^* \). In addition, the HHMSM scheme with the optimal time interval \( T^* \) is little poorer than the patching scheme in these two aspects. The reason of little poor performance compared with the patching scheme is that HHMSM scheme batches client requests arrived in the same time slot. This will effectively increase the bandwidth efficiency at high client request rates.

Fig.4 shows results for testing the server bandwidth performance. At the aspect of the server bandwidth efficiency, the HHMSM scheme outperforms the FCFS batching scheme and the optimal time-threshold patching scheme significantly over a wide range of client request rates. When request arrival rate is less than 200 requests/hour, the bandwidth consumption for the HHMSM scheme is just little less than that of other two schemes. By the increasing of request arrival rate, the bandwidth consumption increasing degree for the HHMSM scheme is distinctly less than that of other two schemes. When request arrival rate is 800, the average bandwidth consumption for the HHMSM scheme is approximately 280 Mbits/s. At the same request arrival rate, the average bandwidth consumption for the FCFS batching scheme is approximately 495 Mbits/s and that of the optimal time-threshold patching scheme is approximately 371 Mbits/s. The HHMSM scheme can save approximately 45% bandwidth consumption compared with the FCFS batching scheme, and can save approximate 25% bandwidth consumption compared with the optimal time-threshold patching scheme. When request arrival rate is higher than 1500 requests per hour, the bandwidth performance of optimal time-threshold patching scheme is going to be worse. It is near to the FCFS batching scheme. In any case, the HHMSM scheme significantly outperforms them all. (FCFS: approximate 718 Mbits/s; OTT-Patching: approximate 694 Mbits/s; HHMSM: 389 Mbits/s; at request rate: 1600 request/hour).

Fig.4 also shows how the time interval affects the average server bandwidth consumption for the HHMSM scheme. The average server bandwidth consumption is decreased in some degree by increasing time interval. The reason is that more clients are batched together and served as one client. However, the decreasing degree for average server bandwidth consumption is very small. When the request arrival rate is less than 1600 requests per hour, the total saved server bandwidth is less than 75Mbits/s by comparing optimal time interval \( T = 1 \text{ minute} \) and \( T=15 \text{ minutes} \). On the other hand, the clients reneging probability is dramatically increased from 4.5% to 45%. Therefore, a big time interval \( T \) is not a good choice and we suggest using \( [L/(2b_{max})] \) to be the optimal time interval.

4. CONCLUSIONS

This paper presents a novel stream scheduling scheme that significantly reduces the required server bandwidth over a wide range of client request rates. Unlike the existing batching scheme and the patching scheme, the HHMSM scheme utilizes multicast propagation method for all transmission streams so that the missed video segments can be get both from the complete stream and existed patching streams. Furthermore, the HHMSM scheme with optimal time interval can improve the start-up latency performance significantly compared with the batching scheme.

5. REFERENCES