A Two-Level Patching Scheme for Video Multicast

Dongliang Guan, Songyu Yu
Institute of Image Communication & Information Processing
Shanghai JiaoTong University
1954 HuaShan Rd
Shanghai 200030, P. R. China

Abstract—Although patching has shown to be a simple and efficient technique for immediate media on-demand delivery, there is still much scope for further improvement compared with the lower bound of the server network-I/O bandwidth requirement. In this paper, a new concept of two-level patching channel is proposed for the first time, based on which patching channels are rearranged through merging and further patching and a novel channel schedule scheme is developed. Simulation results show that the proposed Two-level patching scheme outperforms the conventional patching technique by a significant margin. It even performs better than the Dynamic Skyscraper Algorithm over a wide range of client request rates. Furthermore, it is reasonably competitive with hierarchical multicast stream merging (HMSM) at low to modest client request rates. Most importantly, the implementation complexity of our algorithm is much lower than the Skyscraper and HMSM.

I. INTRODUCTION

As the Internet grows so does the desire for video-on-demand delivery. The popularity of a specific video may be very high. These pose a big challenge - a huge consumption of server bandwidth. Thus, multicasting may be the only way to satisfy the demand.

The first and most natural idea to exploit the advantage of multicasting is to batch clients together. This solution implies a trade-off between the overall server bandwidth and the start-up delay guaranteed. See [1-3] for various solutions. The main advantage of such solutions lies in their simplicity. The main disadvantage is that the guaranteed start-up delay can be very large for a given amount of bandwidth.

The revolutionary Pyramid solution ([4, 5]) is the first to explore the trade-off with two other resources: the receiving bandwidth of clients and the buffer size of clients. Many solutions follow this line: Fast broadcasting [6], Harmonic broadcasting [7], and, most notably, Skyscraper Broadcasting [8]. The innovative idea in these schemes is that the data for each object is divided into fragments and these fragments are broadcast during predefined Periods on separate channels. Clients must be able to receive on two or more channels simultaneously and must be able to buffer a fragment that is received earlier than needed for playback. All of these solutions assume a static allocation of bandwidth per transmission. Thus, efficient bandwidth saving is achieved only when the client request rates are high. Meanwhile, they cannot provide immediate services. Since object is segmented and transmitted in many channels, a client has to keep track of the starting and ending time of each segment and buffer it accordingly. Consequently, the client has to “hop” from one channel to another at exact times frequently. This leads to design and implementation complexities.

The need for a dynamic allocation motivates the solutions - Dynamic Skyscraper [9, 10] that still use the skyscraper broadcasting model. The Patching solution [11-16], the Piggybacking solution [17-20], and the stream merging solution - HMSM [21, 22] assume the attractive dynamic allocation of bandwidth to transmissions.

Piggybacking is to dynamically speed up and slowdown client processing rates (e.g., display rates for video files) so as to bring different streams to the same file position, at which time the streams can be merged. This technique requires the server to store (or compute in real time) extra encoding of the media data that is delivered most often. Furthermore, the maximum rate at which clients can be merged is bounded by the variation in viewing rate (typically 5%) that can be tolerated by a client.

Patching is a dynamic multicast scheme enabling a new request to join an ongoing multicast. In this scheme, each request is served over one or two channels – either a regular channel alone or the combination of a regular channel and a patching channel. A regular channel delivers the full stream from start to finish while a patching channel delivers only the missing part of the stream. Patching performs much better than Piggybacking.

Dynamic Skyscraper allows each cluster to broadcast a different object in response to client request rather than devoting the channels to a single object. All sequences in the same cluster will broadcast the same object, allowing client requests to batch together during playback, as in the static skyscraper. Dynamic Skyscraper outperforms Patching at modest to high client request rates. But it has the same disadvantage as the static skyscraper scheme because of the segment transmission character.

HMSM gets inspiration from patching and Piggybacking approaches. Clients that request the same file are repeatedly merged into larger and larger groups, leading to a hierarchical merging structure. This technique is reasonably close to the minimum achievable required server bandwidth over a wide range of client request rates. But its repeating merging process leads to implementation complexity. Meanwhile, the wasting of some clients receiving bandwidths usually happens during the merging process.

Of the proposals in the dynamic solutions, Patching is easy to implement. But it performs poorer than Dynamic Skyscraper and HMSM at modest to high client request rates. In this paper, we introduce a new concept of two-level patching channel, based on which we mainly address how to patch the channels to minimize the server bandwidth requirement. Simulation results show that our algorithm achieves a significant margin over the conventional patching technique. Furthermore, over a wide range of client request rates, it is also better than the Dynamic Skyscraper and close to the HMSM.

Now, we can see that it is difficult to buy a standard size disk drive that cannot store at least an entire video. Hence, the efforts of reducing the storage requirements of the client will become moot. So the buffer size is not considered in this paper.

The remainder of this paper is organized as follows. In
Section II, our Two-level patching scheme is derived and the implementation algorithm is described. Section III gives the simulation results. Finally, we conclude in Section IV.

II. ALGORITHM DERIVATION AND DESCRIPTION

For simplicity of exposition, our analysis assumes that there is infinite number of server channels, i.e., each request is immediately satisfied and no request is batched with previous requests. Note that a video server typically stores many videos and videos of various popularity can achieve good multiplexing effect over the server channel usage. Effectively, a video can use more server bandwidth than its average share at a particular time. Over the long run, a video uses only its average share of the server bandwidth. Therefore, for one video, it is reasonable to assume that there is infinite number of server channels and uses the average server bandwidth requirement as the performance parameter.

In the conventional patching scheme, if we take the regular channel as zero-level channel and the patching channel as one-level patching channel, their key idea is that multi-requests share the zero-level channel and recover their initial missing portion through their one-level patching channels.

In this paper, we introduce a two-level patching channel for the first time to reduce the channel redundancy of the conventional patching scheme. The system is modeled as a renew process, that is the whole duration of the requests are divided into sets of sub-duration (termed as Period in this paper) and each new Period is independent of the past behavior. Each Period is further divided into sets of sub-durations (termed as Time-Window in this paper). The first request in a Period initiates a zero-level channel. In a Time-Window, the first request initiates a one-level patching channel whose length is the value of the Time-Window plus the difference of the arriving time between the first request in this Time-Window and the first request in this Period. Each of other requests in this Time-Window respectively initiates a two-level patching channel whose length is the difference of arriving time between itself and the first request in this Time-Window plus the difference of the arriving time between the first request in this Time-Window and the first request in this Period. Each of other requests in this Time-Window respectively initiates a two-level patching channel whose length is the difference of arriving time between itself and the first request in this Time-Window plus the difference of the arriving time between the first request in this Time-Window and the first request in this Period.

Fig. 1. Global view of our scheme

\[ p[K_n = k] = \left(\frac{\lambda w_n}{2}\right)^k e^{-\lambda w_n} / k! \]

The expectation of the consumed video length during this Time-Window when \( K_n = k \) is

\[ E[L_n | K_n = k] = t_n + w_n + k \lambda w_n / 2. \]  

The expectation of the consumed video length during this Time-Window is

\[ E[L_n] = \sum_{k=0}^{\infty} \left(\frac{\lambda w_n}{2}\right)^k e^{-\lambda w_n} E[L_n | K_n = k] \]

\[ = t_n + w_n + \frac{1}{2} \lambda w_n^2 / k! \]

The expectation of the number the arrivals during this Time-Window is

\[ E[N_n] = 1 + \lambda w_n. \]

The required average server bandwidth during this Time-Window is

\[ \overline{C}_n = b \lambda E[L_n] / E[N_n] \]

\[ = b \lambda \left(t_n + w_n + \lambda w_n^2 / 2\right) / (1 + \lambda w_n). \]

Differentiating both sides of (4) w.r.t \( w_n \),

\[ \overline{C}_n' = b \lambda \left(1 + \lambda w_n^2 / 2 - \lambda t_n + 1 / 2\right) / (1 + \lambda w_n)^2. \]

Setting (5) equal to 0, we obtain the value of \( w_n \) that minimizes the \( \overline{C}_n \).

\[ w_n = \left(\sqrt{2 \lambda t_n - 1} - 1\right) / \lambda. \]
The arriving time of the earliest request in n+1-th Time-Window is

\[ E[N_{n+1}] = T_n + w_n + 1/\lambda . \]  
(7)

Combining (6) and (7), we have

\[
w_{n+1} = \left(\frac{2\lambda T_{n+1} - 1 - 1}{\lambda}\right) = \frac{\left(\sqrt{\left(1 + \lambda w_n + 1\right)^2 - 1} - 1\right)}{\lambda} .
\]  
(8)

The arriving time of the first request in a Period is always

\[ t_0 = 0 . \]  
(9)

The second request starts the first Time-Windows whose value, combining with (6), is set to

\[ w_1 = 0 . \]  
(10)

Assume the number of Time-Windows in a Period is \( T \). The consumed video length in this Period is

\[
L_{\text{Period}} = L + \sum_{n=1}^{T} L_n
= L + \sum_{n=1}^{T} \left(t_n + w_n + 1/\lambda \right) .
\]  
(11)

The first term in (11) is the video length initiated by the first request in this Period, the second term is the video length consumed by the requests in \( T \) Time-Window. Combining with (7), (8), (9) and (10), (11), can be reduced to

\[
L_{\text{Period}} = L + \sum_{n=1}^{T} \left(t_n + w_n + 1/\lambda \right) = L + \sum_{n=1}^{T} \left(t_n + w_n + 1/\lambda \right) + \sum_{n=1}^{T} \left(\lambda w_n^2 / 2 - \sum_{n=1}^{T} \lambda w_n^2 / 2\right) .
\]  
(12)

The expectation of number of requests in this Period is

\[ N_{\text{Period}} = 1 + \sum_{n=1}^{T} E[N_n] = 1 + \sum_{n=1}^{T} \left(1 + \lambda w_n\right) .
\]  
(13)

Then the average server bandwidth is

\[
\bar{C} = bL_{\text{Period}}/N_{\text{Period}} = bL(T^2/3\lambda + T^2/4\lambda - 7T/12\lambda + 1) .
\]  
(14)

Here, we temporarily extend the definition domain of the \( T \) to the real number. Differentiating both sides of (14) w.r.t \( T \),

\[
\bar{C} = \frac{bT^4 + bT^3 + \frac{17}{2} \lambda T^2 + \left(\frac{1}{2} - \lambda \right) T - \left(\frac{L_2}{2} + \frac{7}{12} - \lambda \right)}{T^2 + T + 1} .
\]  
(15)

Let

\[ f(T) = \frac{T^4 + T^3 + \frac{17}{2} \lambda T^2 + \left(\frac{1}{2} - \lambda \right) T - \left(\frac{L_2}{2} + \frac{7}{12} - \lambda \right)}{T^2 + T + 1} . \]
(16)

Then we have

\[ f'(T) = \frac{2T^3 + T^2 + \left(\frac{17}{2} - \lambda \right) T + \left(\frac{1}{2} - \lambda \right)}{T^2 + T + 1} . \]
(17)

If \( 2T^2 + T + \left(\frac{17}{6} - \lambda \right) = 0 \) has a root \( T_2 \) which greater than 0, then \( f'(T) \) is strictly monotone descending during \( [0, T_2] \) and is strictly monotone increasing when \( T \) is greater than \( T_2 \). If \( 2T^2 + T + \left(\frac{17}{6} - \lambda \right) = 0 \) have no root, then \( f'(T) \) is strictly monotone increasing during \( [0, \infty) \). Under the normal case while \( L\lambda - 1/2 > 0 \), we have \( f'(0) < 0 \). There exists one \( T_1 \) that satisfies \( f'(T_1) = 0 \). So \( f'(T) \) crosses the x-axis only once and denotes the intersection point is \( T_1 \). Then \( f(T) \) is strictly monotone decreasing during \( [0, T_1^*] \) and is strictly monotone increasing when \( T \) is greater than \( T_1^* \). Under the normal case while \( L\lambda / 2 + 7/12 - \lambda > 0 \), we have \( f(0) < 0 \). There exists one \( T_0 \) that satisfies \( f(T_0) > 0 \). So \( f(T) \) crosses the x-axis only once and denote the intersection point is \( T^* \). Then we have \( \bar{C}'(T) < 0 \) when \( [0, T^*] \); and \( \bar{C}'(T) > 0 \) when \( (T^*, \infty) \). So \( \bar{C}(T) \) achieves its minimum value on \( T^* \). The optimal \( T^* \) is obtained from the following recursion:

\[ T^* = T - f(T')/f'(T) . \]  
(19)

The recursive equation (19) is terminated when
Here $\epsilon$ is a predefined small value. The obtained $T^*$ from the above recursions is a real number which satisfying: $[T^*] \leq T^* \leq [T^*] + 1$. We now have to recover the definition domain of $T$ to the interger space. Submits $[T^*]$ and $[T^*] + 1$ into (14) and obtains two $C$, then the expected optimal $T^*$ is one of $[T^*]$ and $[T^*]+1$ which make $C$ smaller.

Below, we give our complete algorithm:

Step 1: $\lambda$ is estimated from the reciprocal of the average interval of the latest 10 requests.

Step 2: $T$ is calculated out from the above recursive equations: (19) and (20).

Step 3: The first request starts a Period and initiates a zero-level channel. The second request (its arriving time is $t_1$) starts the first Time-Window (its duration is $w_1$) and there are $T$ Time-Windows whose duration is $w_i$ $(1 \leq i \leq T)$. In each Time-Window, the first request initiates a one-level patching channel; other successive requests initiate two-level patching channels. After this Period is over, goes to step 1 to repeat the above steps.

III. SIMULATION RESULTS

Two request arrival sequences were simulated and the results were shown in the Table I. The required average server bandwidth $C(P)$ was obtained from Poisson request arrival sequence; $C(S)$ was obtained from Self-similar request arrival sequence. $N$ is the average number of requests for the file that arrives during the whole Period of length of the file. The file length is 100 time-units. Comparing $C(P)$ with $C(S)$, we draw a conclusion that Poisson request arrival stream is likely to be reasonably representative for full streaming media file requests.

Below, we compare our algorithm with related prevailing techniques that provide immediate service to client requests under the Poisson request arrival stream. Fig. 2 shows the required average server bandwidth, in units of the file play rate, as a function of $N$ for optimized patching technique, optimized partitioned Dynamic Skyscraper system (segment size progressions: 1, 1, 2, 2, 6, 6, 12, 12, …), HMSM (2,1) and our proposed algorithm. To compare, we also plot the theoretic lower bound, which is derived from the assumption that the file can be divided into arbitrarily small segments and the clients have unlimited client receive bandwidth.

The proposed algorithm has significantly lower bandwidth requirements than Optimized patching technique especially when the client request rates are high. When the client request rates $N$ is less than 300, our algorithm even performs better than optimized partitioned Dynamic Skyscraper delivery system (segment size progressions: 1, 1, 2, 2, 6, 6, 12, 12, …), which has the same receive bandwidth requirement as our algorithm (i.e., three times the file play rate). Furthermore, it is reasonably competitive with HMSM (2,1) at low to modest client request rates.

IV. CONCLUSION

In this paper, a novel Two-level patching scheme is proposed for the first time, based on which further channel is implemented. The system is modeled as a renew process ( termed as Period in this paper). The duration of a Period is divided into a set of sub-duration ( termed as Time-Window in this paper). Then we mainly address two problems: one is how to set the value of each Time-Window (i.e., $w_n$ in this paper); the other is how to set the number of the Time-Window (i.e., $T$ in this paper). The effectiveness of our algorithm was justified through extensive simulations. We showed that the proposed scheme performs very well, while not damaging the performance of algorithm complexity compared with the conventional patching technique.

REFERENCES


### Table I

<table>
<thead>
<tr>
<th>( N )</th>
<th>3</th>
<th>6</th>
<th>10</th>
<th>30</th>
<th>60</th>
<th>100</th>
<th>300</th>
<th>600</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{C} ) (P)</td>
<td>1.90</td>
<td>2.62</td>
<td>3.25</td>
<td>5.11</td>
<td>6.53</td>
<td>7.90</td>
<td>11.54</td>
<td>14.88</td>
<td>17.83</td>
</tr>
<tr>
<td>( \bar{C} ) (S)</td>
<td>2.16</td>
<td>3.04</td>
<td>3.70</td>
<td>5.51</td>
<td>7.09</td>
<td>8.34</td>
<td>12.09</td>
<td>15.28</td>
<td>18.11</td>
</tr>
</tbody>
</table>