EURASIP Journal on Applied Signal Processing 2004:2, 317–329


# Medusa: A Novel Stream-Scheduling Scheme for Parallel Video Servers

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

**Dafu Deng**
*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
*Email: dfdeng@hust.edu.cn*

**Liping Pang**
*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
*Email: lppang@hust.edu.cn*

*Received 6 December 2002; Revised 15 July 2003*

Parallel video servers provide highly scalable video-on-demand service for a huge number of clients. The conventional stream-scheduling scheme does not use I/O and network bandwidth efficiently. Some other schemes, such as batching and stream merging, can effectively improve server I/O and network bandwidth efficiency. However, the batching scheme results in long startup latency and high reneging probability. The traditional stream-merging scheme does not work well at high client-request rates due to mass retransmission of the same video data. In this paper, a novel stream-scheduling scheme, called *Medusa*, is developed for minimizing server bandwidth requirements over a wide range of client-request rates. Furthermore, the startup latency raised by Medusa scheme is far less than that of the batching scheme.

**Keywords and phrases:** video-on-demand, stream batching, stream merging, multicast, unicast.

## 1. INTRODUCTION

In recent years, many cities around the world already have, or are deploying, the *fibre to the building* (FTTB) network on which users access the optical fibre *metropolitan area network* (MAN) via the fast LAN in the building. This kind of large-scale network improves the end bandwidth up to 100 Mb per second and has enabled the increasing use of larger-scale *video-on-demand* (VOD) systems. Due to the high scalability, the parallel video servers are often used as the service providers in those VOD systems.

Figure 1 shows a diagram of the large-scale VOD system. On the client side, users request video objects via their PCs or dedicated set-top boxes connected with the fast LAN in the building. Considering that the 100 Mb/s Ethernet LAN is widely used as the in-building network due to its excellent cost/effective rate, we only focus on the clients with such bandwidth capacity and consider the VOD systems with homogenous client network architecture in this paper.

On the server side, the parallel video servers [1, 2, 3] have two logical layers. Layer 1 is an RTSP server, which is responsible for exchanging the RTSP message with clients and scheduling different RTP servers to transport video data to clients. Layer 2 consists of several RTP servers that are responsible for concurrently transmitting video data according to the RTP/RTCP. In addition, video objects are often striped into lots of small segments that are uniformly distributed among RTP server nodes so that the high scalability of the parallel video servers can be guaranteed [2, 3].

Obviously, the key bottleneck of those large-scale VOD systems is the bandwidth of parallel video servers, either the disk I/O bandwidth of parallel video servers, or the network bandwidth connecting the parallel video servers to the MAN. For using the server bandwidth efficiently, a stream-scheduling scheme plays an important role because it determines how much video data should be retrieved from disks and transported to clients. The conventional scheduling scheme sequentially schedules RTP server nodes to transfer segments of a video object via unicast propagation method. Previous works [4, 5, 6, 7, 8] have shown that most clients often request several hot videos in a short time interval. This makes the conventional scheduling scheme send lots of same

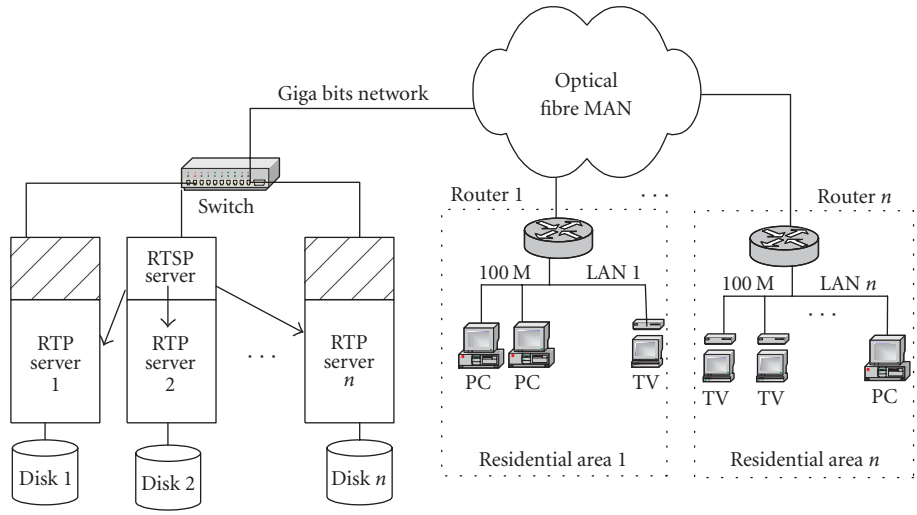


Figure 1: A larger-scale VOD system supported by parallel video servers.

video-data streams during a short time interval. It wastes the server bandwidth and better solutions are necessary.

The multicast or broadcast propagation method presents an attractive solution for the server bandwidth problem because a single multicast or broadcast stream can serve lots of clients that request the same video object during a short time interval. In this paper, we focus on the above VOD system, and then, based on the multicast method, develop a novel stream-scheduling scheme for the parallel video servers, called Medusa, which minimizes the server bandwidth consumption over a wide range of client-request rates.

The following sections are organized as follows. In Section 2, we describe the related works on the above bandwidth efficiency issue and analyze the existing problem of these schemes. Section 3 describes the scheduling rules for the Medusa scheme and Section 4 discusses how to determine the time interval $T$ used in the Medusa scheme. Section 5 presents information of the performance evaluation. Section 6 proposes some discussions for the Medusa scheme. Finally, Section 7 ends with conclusions and future works.

## 2. RELATED WORKS

In order to use the server bandwidth efficiently, two kinds of schemes based on the multicast or broadcast propagation method have been proposed: the batching scheme and the stream-merging scheme.

The basic idea of the batching scheme is using a single multicast stream of data to serve clients requesting the same video object in the same time interval. Two kinds of batching schemes were proposed: first come first serve (FCFS) and maximum queue length (MQL) [4, 6, 9, 10, 11, 12]. In FCFS, whenever a server schedules a multicast stream, the client with the earliest request arrival is served. In MQL, the incoming requests are put into separate queues based on the requested video object. Whenever a server schedules a multicast stream, the longest queue is served first. In any case, a time threshold must be set first in the batching scheme. Video servers just schedule the multicast stream at the end of each time threshold. In order to obtain efficient bandwidth, the value of this time threshold must be at least 7 minutes [7]. The expected startup latency is approximately 3.5 minutes. The long delay increases the client reneging rate and decreases the popularization of VOD systems.

The stream-merging scheme presents an efficient way to solve the long startup latency problem. There are two kinds of scheduled streams: the complete multicast stream and the patching unicast stream. When the first client request has arrived, the server immediately schedules a complete multicast stream with a normal propagation rate to transmit all of the requested video segments. A later request to the same video object must join the earlier multicast group to receive the remainder of the video, and simultaneously, the video server schedules a new patching unicast stream to transmit the lost video data to each of them. The patching video data is propagated at double video play rate so that two kinds of streams can be merged into an integrated stream.

According to the difference in scheduling the complete multicast stream, stream-merging schemes can be divided into two classes: *client-initiated with prefetching* (CIWP) and *server-initiated with prefetching* (SIWP).

For CIWP [5, 13, 14, 15, 16, 17], the complete multicast stream is scheduled when a client request arrives. The latest complete multicast stream for the same video object cannot be received by that client.

For SIWP [8, 18, 19], a video object is divided into segments, each of which is multicast periodically via a dedicated multicast group. The client prefetches data from one or several multicast groups for playback.

Stream-merging schemes can effectively decrease the required server bandwidth. However, with the increase of client-request rates, the amount of the same retransmitted video data is expanded dramatically and the server

Table 1: Notations and Definitions.

| Notations | Definitions |
|---|---|
| $T$ | The length of time interval and also the length of a video segment (in min) |
| $M$ | The amount of video objects stored on the parallel video server |
| $N$ | The amount of RTP server nodes in the parallel video servers |
| $t_i$ | The $i$th time interval; the interval in which the first client request arrives is denoted by $t_0$; the following time intervals are denoted by $t_1, \ldots, t_i, \ldots$, respectively, $(i = 0, \ldots, +\infty)$ |
| $L$ | The length of the requested video object (in min) |
| $S(i, j)$ | The $i$th segment of the requested object; $j$ represents the serial number of the RTP server node on which this segment is stored |
| $R_i$ | The client requests arriving in the $i$th time interval $(i = 0, \ldots, +\infty)$ |
| $\mathrm{PS}_i$ | The patching multicast stream initialized at the end of the $i$th time interval $(i = 0, \ldots, +\infty)$ |
| $\mathrm{CS}_i$ | The complete multicast stream initialized at the end of the $i$th time interval $(i = 0, \ldots, +\infty)$ |
| $\tau(m, n)$ | The start transporting time for the $m$th segment transmitted on the stream $\mathrm{PS}_n$ or $\mathrm{CS}_n$ |
| $G_i$ | The client-requests group in which all clients are listening to the complete multicast stream $\mathrm{CS}_i$ |
| $b_c$ | The client bandwidth capacity, in unit of stream (assuming the homogenous client network) |
| $\mathrm{PB}_{\max}$ | The maximum number of patching multicast streams that can be concurrently received by a client |
| $\lambda_i$ | The client-request arrival rate for the $i$th video object |

bandwidth efficiency is seriously damaged. Furthermore, a mass of double-rated patching streams may increase the network traffic burst.

## 3. MEDUSA SCHEME

Because video data cannot be shared among clients requesting for different video objects, the parallel video server handles those clients independently. Hence, we only consider clients requesting for the same hot video object in this section (more general cases will be studied in Section 5).

### 3.1. The basic idea of the Medusa scheme

Consider that the requested video object is divided into lots of small segments with a constant playback time length $T$. Based on the value of $T$, the time line is slotted into fixed-size time intervals and the length of each time interval is $T$. Usually, the value of $T$ is very small. Therefore, it would not result in long startup latency. The client requests arriving in the same time interval are batched together and served as one request via the multicast propagation method. For convenient description, we regard client requests arriving in the same time interval as one client request in the following sections.

Similar to stream-merging schemes, two kinds of multicast streams, the *complete multicast streams* and the *patching multicast streams*, can be used to reduce the amount of retransmitted video data. A complete multicast stream responses to transporting all segments of the requested video object while a patching multicast stream just transmits partial segments of that video object. The first arrival request is served immediately by a complete multicast stream. Later starters must join the complete multicast group to receive the remainder of the requested video object. At the same time, they must join as more earlier patching multicast groups as possible to receive valid video data. For those really missed video data, the parallel video servers schedule a new patching multicast stream for transporting them to clients.

Note that the IP multicast, the broadcast, and the application-level multicast are often used in VOD systems. In those multicast technologies, a user is allowed to join lots of multicast groups simultaneously. In addition, because all routers in the network would exchange their information periodically, each multicast packet can be accurately transmitted to all clients of the corresponding multicast group. Hence, it is reasonable for a user to join into several interesting multicast streams to receive video data.

Furthermore, in order to eliminate the additional network traffic arisen by the scheduling scheme, each stream is propagated at the video play rate. Clients use disks to store later played segments so that the received streams can be merged into an integrated stream.

### 3.2. Scheduling rules of the Medusa scheme

The objective of the Medusa scheme is to determine the frequency for scheduling the complete multicast streams so that the transmitting video data can be maximally shared among clients, and determine which segment will be transmitted on a patching multicast stream so that the amount of transmitted video data can be minimized. Notations used in this paper are showed in Table 1. Scheduling rules for the Medusa scheme are described as follows.

(1) The parallel video server dynamically schedules complete multicast streams. When the first request $R_0$ arrives, it schedules $\mathrm{CS}_0$ at the end of time slot $t_0$ and notifies the corresponding clients of $R_0$ to receive and play back all segments transmitted on $\mathrm{CS}_0$. Suppose the last complete multicast stream is $\mathrm{CS}_j$ $(0 \le j < +\infty)$. For an arbitrary client request $R_i$ that arrives in the time

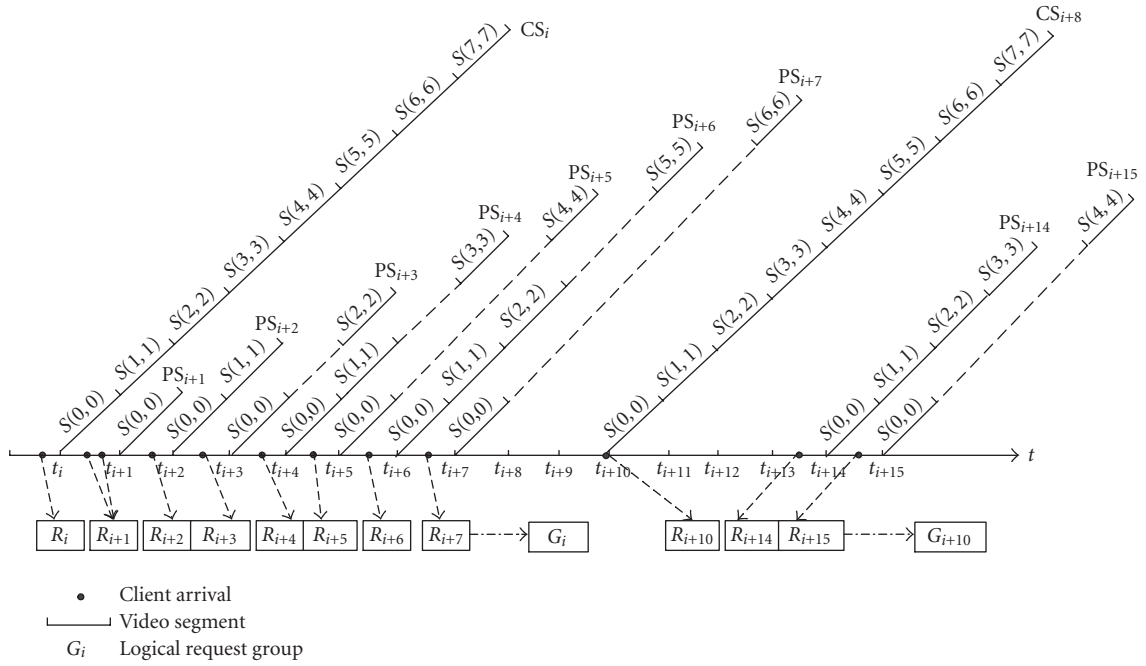


Figure 2: A scheduling example scene for the Medusa scheme.

$t_i$, if $t_j < t_i \leq t_j + \lceil L/T \rceil - 1$, no complete multicast stream need to be scheduled and just a patching multicast stream is scheduled according to rules (2) and (3). Otherwise, a new complete multicast stream CS$_i$ is initialized at the end of the time interval $t_i$.

(2) During the transmission of a complete multicast stream CS$_i$ ($0 \leq i < +\infty$), if a request $R_j$ ($i < j \leq i + \lceil L/T \rceil - 1$) arrives, the server puts it into the logical requests group $G_i$. For each logical request group, a parallel video server maintains a stream information list. Each element of the stream information list records the necessary information of a patching multicast stream, described as a triple $E(t, I, A)$, where $t$ is the scheduled time, $I$ is the multicast group address of the corresponding patching multicast stream, and $A$ is an array to record the serial number of video segments that will be transmitted on the patching multicast stream.

(3) For a client $R_j$ whose request has been grouped into the logical group $G_i$ ($0 \leq i < +\infty$, $i < j \leq i + \lceil L/T \rceil - 1$), the server notifies it to receive and buffer the later $\lceil L/T \rceil - (j - i)$ video segments from the complete multicast stream CS$_i$. Because the begining $j - i$ segments have been transmitted on the complete multicast stream CS$_i$, the client $R_j$ loses them from CS$_i$. Thus, for each begining $j - i$ segments, the server searches the stream information list of $G_i$ to find out which segment will be transmitted on an existing patching multicast stream and can be received by the client. If the $l$th segment ($0 \leq l < j - i$) will be transmitted on an existing patching multicast stream PS$_n$ ($i < n < j$) and the transmission start time is later than the service start time $t_j$, the server notifies the corresponding client $R_j$ to join the multicast group of PS$_n$ to receive this segment. Otherwise, the server transmits this segment in a new initialized patching multicast stream PS$_j$ and notifies the client to join the multicast group of PS$_j$ to receive it. At last, the server creates the stream information element $E_j(t, I, A)$ of PS$_j$, and inserts it to the corresponding stream information list.

(4) Each multicast stream propagates the video data at the video playback rate. Thus, a video segment is completely transmitted during a time interval. For the $m$th segment that should be transmitted on the $n$th multicast stream, the start-transmission time is fixed and the value of this time can be calculated by the following equation:

$$\tau(m, n) = t_n + m^* T, \tag{1}$$

where $t_n$ is the initial time of the $n$th multicast stream.

Figure 2 shows a scheduling example for the Medusa scheme. The requested video is divided into eight segments. Those segments are uniformly distributed on eight nodes in a round-robin fashion. The time unit on the $t$-axis corresponds to a time interval, as well as the total time it takes to deliver a video segment. The solid lines in the figure represent video segments transmitted on streams. The dotted lines show the amount of skipped video segments by the Medusa scheme.

In this figure, when the request $R_i$ arrives at the time slot $t_i$, the server schedules a complete multicast stream CS$_i$.

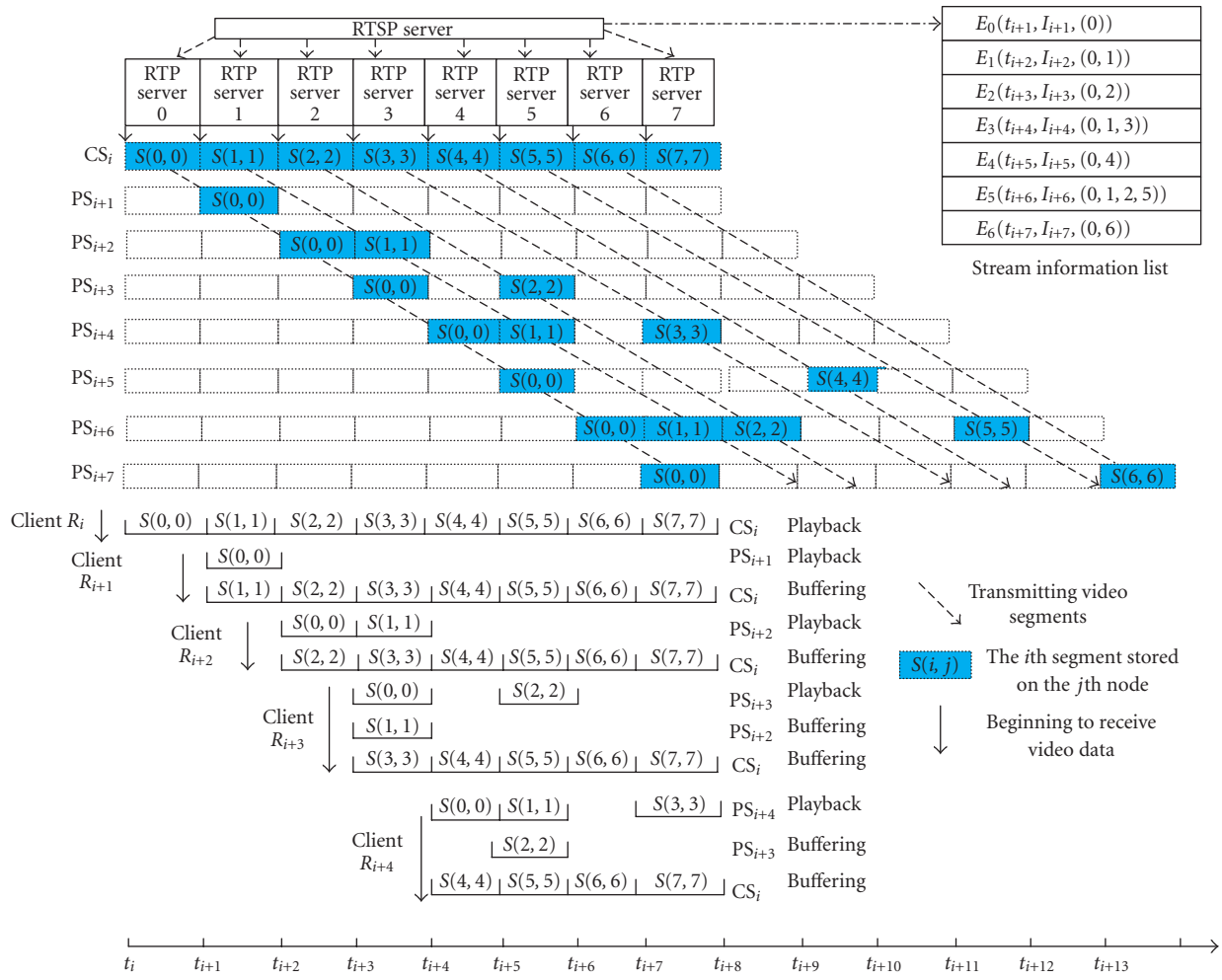


FIGURE 3: The scheduling course of parallel video server for requests of $G_i$ and the corresponding receiving course for clients of $R_i$, $R_{i+1}$, $R_{i+2}$, $R_{i+3}$, and $R_{i+4}$.

Because the complete multicast stream is transmitted completely at time $t_{i+10}$, the video server schedules a new complete multicast stream $CS_{i+10}$ to serve clients corresponding to the request $R_{i+10}$. According to rule (2), requests $R_{i+1} \cdots R_{i+7}$ must be grouped into the logical request group $G_i$, and requests $R_{i+14}$, $R_{i+15}$ must be grouped into logical requests group $G_{i+10}$.

The top half portion of Figure 3 shows the scheduling of parallel video servers for those requests in the group $G_i$ presented in Figure 2. The bottom half portion of Figure 3 shows the video data receiving and the stream-merging for clients $R_i$, $R_{i+1}$, $R_{i+2}$, $R_{i+3}$, and $R_{i+4}$. We just explain the scheduling for the request $R_{i+4}$, others can be deduced by rule (3). When request $R_{i+4}$ arrives, the parallel video server firstly notifies the corresponding clients of $R_{i+4}$ to receive the video segments $S(4,4)$, $S(5,5)$, $S(6,6)$, and $S(7,7)$ from the complete multicast stream $CS_i$. It searches the stream information list, and finds out that segment $S(2,2)$ will be transmitted on patching multicast stream $PS_{i+3}$ and the transmission start time of $S(2,2)$ is later than $t_{i+4}$. Then, it notifies the client $R_{i+4}$ to receive the segment $S(2,2)$ from patching multicast stream $PS_{i+3}$. At last, the parallel video server schedules a new patching multicast stream $PS_{i+4}$ to transmit the missing segments $S(0,0)$, $S(1,1)$, and $S(3,3)$. The client $R_{i+4}$ is notified to receive and play back those missing segments and the stream information element of $PS_{i+4}$ is inserted into the stream information list.

## 4. DETERMINISTIC TIME INTERVAL

The value of time interval $T$ is the key issue affecting the performance of the parallel video servers. In the Medusa scheme, a client may receive several multicast streams concurrently and the number of concurrently received multicast streams is related with the value of $T$. If $T$ is too small, the number of concurrently received streams may be increased dramatically and exceed the client bandwidth capacity $b_c$. Some valid video data may be discarded at the client side. Furthermore, since a small $T$ would increase the number of streams sent by the parallel video server, the server bandwidth efficiency may

be decreased. If $T$ is too large, the startup latency may be too long to be endured and the client reneging probability may be increased.

In this section, we derive the deterministic time interval $T$ which guarantee the startup latency minimized under the condition that the number of streams concurrently received by a client would not exceed the client bandwidth capacity $b_c$. The server bandwidth consumption affected by the time interval will be studied in Section 6.

We first derive the relationship between the value of $\mathrm{PB}_{\max}$ (defined in Table 1) and the value of $T$. For a request group $G_i$ $(0 \le i < +\infty)$, $\mathrm{CS}_i$ is the complete multicast stream scheduled for serving the requests of $G_i$. For a request $R_k$ $(i < k \le \lceil L/T \rceil - 1 + i)$ belonging to $G_i$, the clients corresponding to the $R_k$ may concurrently receive several patching multicast streams. Assume that $\mathrm{PS}_j$ $(i < j < k)$ is the first patching stream from which clients of $R_k$ can receive some video segments. According to the Medusa scheme, video segments from the $(j-i)$th segment to the $(\lceil L/T \rceil - 1)$th segment would not be transmitted on $\mathrm{PS}_j$, and the $(j-i-1)$th segment would not be transmitted on the patching multicast streams initialized before the initial time of $\mathrm{PS}_j$. Hence, the $(j-i-1)$th segment is the last transmitted segment for $\mathrm{PS}_j$. According to (1), the start time for transporting the $(j-i-1)$th segment on $\mathrm{PS}_j$ can be expressed by

$$\tau(j-i-1, j) = t_j + (j-i-1)^* T. \tag{2}$$

Since the clients of $R_k$ receive some video segments from $\mathrm{PS}_j$, the start transporting time of the last segment transmitted on $\mathrm{PS}_j$ must be later than or equal to the request arrival time $t_k$. Therefore, we can obtain that

$$\tau(j-i-1, j) \ge t_k. \tag{3}$$

Consider that $t_k = t_j + (k-j) \times T$. Combining (2) and (3), we derive that

$$j \ge \frac{k+i+1}{2}. \tag{4}$$

If the clients of the request $R_k$ receive some segments from the patching multicast streams $\mathrm{PS}_j, \mathrm{PS}_{j+1}, \ldots, \mathrm{PS}_{k-1}$, the number of concurrently received streams access to its maximum value. Thus, $\mathrm{PB}_{\max} = k - j$. Combing (4), we can obtain that $\mathrm{PB}_{\max} \le (k-i-1)/2$. In addition, because the request $R_k$ belongs the request group $G_i$, the value of $k$ must be less than or equal to $i + \lceil L/T \rceil - 1$, where $L$ is the total playback time of the requested video object. Thus, $\mathrm{PB}_{\max}$ can be expressed by

$$\mathrm{PB}_{\max} = \left\lceil \frac{L}{2T} \right\rceil - 1. \tag{5}$$

For guaranteeing that the video data would not be discarded at the client end, the client bandwidth capacity must be larger than or equal to the maximum number of streams concurrently received by a client. It means that $b_c \ge \mathrm{PB}_{\max} + 1$, where 1 is the number of complete multicast streams received by a client. Combing (5), we obtain that

$$b_c \ge \left\lceil \frac{L}{2T} \right\rceil \Longrightarrow T \ge \left\lceil \frac{L}{2b_c} \right\rceil. \tag{6}$$

Obviously, the smaller the time interval $T$, the shorter the startup latency. Thus, the deterministic time interval will be the minimum value of $T$, that is,

$$T = \left\lceil \frac{L}{2b_c} \right\rceil. \tag{7}$$

## 5. PERFORMANCE EVALUATION

We evaluate the performance of the Medusa scheme via two methods: the mathematical analysis on the required server bandwidth, and the experiment. Firstly, we analyze the server bandwidth requirement for one video object in the Medusa scheme and compare it with the FCFS batching scheme and the stream-merging schemes. Then, the experiment for evaluating and comparing the performance of the Medusa scheme, the batching scheme, and the stream-merging schemes will be presented in detail.

### 5.1. Analysis for the required server bandwidth

Assume that requests for the $i$th video object are generated by a Poisson process with mean request rate $\lambda_i$. For serving requests that are grouped into the group $G_j$, the patching multicast streams $\mathrm{PS}_{j+1}, \mathrm{PS}_{j+2}, \ldots, \mathrm{PS}_{j+\lceil L/T \rceil - 1}$ may be scheduled from time $t_{j+1}$ to time $t_{j+\lceil L/T \rceil - 1}$, where $L$ is the length of the $i$th video object and $T$ is the selected time interval. We use the matrix $M_{\mathrm{pro}}$ to describe the probabilities of different segments transmitted on different patching multicast streams. It can be expressed as

$$M_{\mathrm{pro}} = \begin{bmatrix} P_{11} & \cdots & P_{1(\lceil L/T \rceil - 1)} \\ P_{21} & \cdots & P_{2(\lceil L/T \rceil - 1)} \\ \vdots & \vdots & \vdots \\ P_{(\lceil L/T \rceil - 1)1} & \cdots & P_{(\lceil L/T \rceil - 1)(\lceil L/T \rceil - 1)} \end{bmatrix}, \tag{8}$$

where the $n$th column represents the $n$th video segment, the $m$th row expresses the patching multicast stream $\mathrm{PS}_{j+m}$, and $P_{mn}$ describes the probability for transmitting the $n$th segment on the patching multicast stream $\mathrm{PS}_{j+m}$ $(1 \le m \le \lceil L/T \rceil - 1,\ 1 \le n \le \lceil L/T \rceil - 1)$. Hence, the expected amount (in bits) of video data transmitted for serving requests grouped in $G_j$ can be expressed as

$$\overline{\Omega} = b \times T \times \sum_{m=1}^{\lceil L/T \rceil - 1} \sum_{n=1}^{\lceil L/T \rceil - 1} P_{mn} + b \times L, \tag{9}$$

where $b$ is the video transporting rate (i.e., the video playback rate) and $b \times L$ represents the number of video segments transmitted on the completely multicast stream $\mathrm{CS}_j$.

According to the scheduling rules of the Medusa scheme, the $n$th $(1 < n \le \lceil L/T \rceil - 1)$ video segment should not be transmitted on patching multicast streams $\mathrm{PS}_{j+1}, \ldots, \mathrm{PS}_{j+n-1}$. Thus,

$$P_{mn} = 0 \quad \text{if } n > m. \tag{10}$$

On one hand, for the $m$th patching multicast stream, the first video segment and the $m$th video segment must be transmitted on it. This is because the first video segment has been transmitted completely on the patching multicast streams $\mathrm{PS}_{j+1}, \ldots, \mathrm{PS}_{j+m-1}$, and the $m$th video segment is not transmitted on such streams. We can obtain that $P_{m1}$ and $P_{mm}$ are equal to the probability for scheduling $\mathrm{PS}_{j+m}$ (i.e., the probability for some requests arriving in the time slot $t_{j+m}$). Since the requests for the $i$th video object are generated by Poisson process, the probability for some requests arriving in the time slot $t_{j+m}$ can be calculated by $P[K \neq 0] = 1 - P[K = 0] = 1 - e^{-\lambda_i T}$. Considering that probabilities for request arriving in different time slots are independent from each other, we can derive that

$$\begin{aligned} P_{11} &= P_{21} = \cdots = P_{(\lceil L/T \rceil - 1)1} = P_{22} \\ &= P_{33} = \cdots = P_{(\lceil L/T \rceil - 1)(\lceil L/T \rceil - 1)} = 1 - e^{-\lambda_i T}. \end{aligned} \tag{11}$$

On the other hand, if the $n$th video segment is not transmitted on patching multicast streams from $\mathrm{PS}_{j+m-n+1}$ to $\mathrm{PS}_{j+m-1}$, it should be transmitted on the patching multicast stream $\mathrm{PS}_{j+m}$. Therefore, the probability for transmitting the $n$th segment on the $m$th patching multicast stream can be expressed as

$$\begin{aligned} P_{mn} = P_{m1} \times \prod_{k=m-n+1}^{m-1} (1 - P_{kn}) \\ (1 < n < m \leq \lceil L/T \rceil - 1), \end{aligned} \tag{12}$$

where $P_{m1}$ represent the probability for scheduling the patching multicast stream $\mathrm{PS}_{j+m}$, and $\prod_{k=m-n+1}^{m-1}(1 - P_{kn})$ indicates the probability for which the $n$th video segments would not be transmitted on patching multicast streams from $\mathrm{PS}_{j+m-n+1}$ to $\mathrm{PS}_{j+m-1}$. Combining (9), (10), (11), and (12), we derive that

$$\begin{aligned} \overline{\Omega} = b \times T \times (1 - e^{-\lambda_i T}) \\ \times \sum_{m=1}^{\lceil L/T \rceil - 1} \sum_{n=1}^{m} \prod_{k=m-n+1}^{m-1} (1 - P_{kn}) + b \times L, \end{aligned} \tag{13}$$

where $P_{kn}$ can be calculated by the following equations:

$$P_{kn} = \begin{cases} 0 & \text{if } k < n, \\ 1 - e^{-\lambda_i T} & \text{if } k = n, \\ \prod_{\ell=k-n+1}^{k-1} (1 - P_{\ell n}) & \text{if } k > n. \end{cases} \tag{14}$$

Because the mean number of arrived clients in the group $G_j$ is $L \times \lambda_i$, we can derive that, in the time epoch $[t_j, t_{j+\lceil L/T \rceil - 1})$, the average amount of transmitted video data for a client, denoted by $\beta_c$, is

$$\begin{aligned} \beta_c &= \frac{\overline{\Omega}}{L \times \lambda_i} \\ &= \frac{b \times T \times (1 - e^{-\lambda_i T}) \sum_{m=1}^{\lceil L/T \rceil - 1} \sum_{n=1}^{m} \prod_{k=m-n+1}^{m-1} (1 - P_{kn})}{L \times \lambda_i} \\ &\quad + \frac{b}{\lambda_i}, \end{aligned} \tag{15}$$

where $P_{kn}$ can be calculated by (14).

Consider the general case from time 0 to $t$. We derive the required average server bandwidth by modeling the system as a renewal process. We are interested in the process $\{B(t) : t > 0\}$, where $B(t)$ is the total server bandwidth used from time 0 to $t$. In particular, we are interested in the average server bandwidth $B_{\text{average}} = \lim_{t \to \infty} S(t)/t$. Let $\{t_j \mid (0 \leq j < \infty), (t_0 = 0)\}$ denote the time set for a parallel video server to schedule a complete multicast stream for video $i$. These are renewal points, and the behavior of the server for $t \geq t_j$ does not depend on past behavior. We consider the process $\{B_j, N_j\}$, where $B_j$ denotes the total server bandwidth consumed and $N_j$ denotes the total number of clients served during the $j$th renewal epoch $[t_{j-1}, t_j)$. Because this is a renewal process, we drop the subscript $j$ and have the following result:

$$B_{\text{average}} = \lambda_i \frac{E[B]}{E[N]}. \tag{16}$$

Obviously, $E[N] = \lambda_i \times L$. For $E[B]$, let $K$ denote the number of arrivals in an interval of renewal epoch length $L$. It has the distribution $P[K = \kappa] = (\lambda_i \times L)^{\kappa} e^{-\lambda_i L}/\kappa!$. For $E[B \mid K = \kappa]$, we have

$$\begin{aligned} E[B \mid K = \kappa] &= \kappa \beta_c \\ &= \left( \frac{b \times T \times (1 - e^{-\lambda_i T}) \sum_{m=1}^{\lceil L/T \rceil - 1} \sum_{n=1}^{m} \prod_{k=m-n+1}^{m-1} (1 - P_{kn})}{L \times \lambda_i} \right. \\ &\quad \left. + \frac{b}{\lambda_i} \right) \kappa. \end{aligned} \tag{17}$$

This indicates that $\kappa$ Poisson arrivals in an interval of length $L$ are equally likely to occur anywhere within the interval. Removal of the condition yields

$$E[B] = \sum_{\kappa=1}^{\infty} \frac{(\lambda_i \times L)^{\kappa} e^{-\lambda_i L}}{\kappa!} E[B \mid K = \kappa]. \tag{18}$$

Combining (17) and (18), we derive that

$$\begin{aligned} E[B] = b \times T \times (1 - e^{-\lambda_i T}) \\ \times \sum_{m=1}^{\lceil L/T \rceil - 1} \sum_{n=1}^{m} \prod_{k=m-n+1}^{m-1} (1 - P_{kn}) + b \times L. \end{aligned} \tag{19}$$

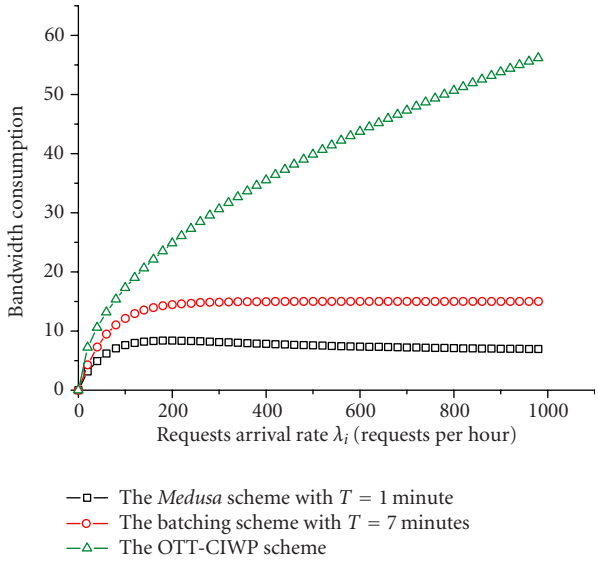


FIGURE 4: Comparison of the expected server bandwidth consumption for one video object among the Medusa scheme, the batching scheme, and the OTT-CIWP scheme.

According to (16) and (19), we derive that

$$B_{\text{average}} = b \times T \times \left(1 - e^{-\lambda_i T}\right) \times \sum_{m=1}^{\lceil L/T \rceil - 1} \sum_{n=1}^{m} \prod_{k=m-n+1}^{m-1} \frac{1 - P_{kn}}{L} + b. \tag{20}$$

For the batching schemes, since all scheduled streams are completely multicast streams, the required server bandwidth for the $i$th video object can be expressed as

$$B_{\text{average}}(\text{batching}) = b \times \left(1 - e^{-\lambda_i T}\right) \times \left\lceil \frac{L}{T} \right\rceil. \tag{21}$$

For the stream merging schemes, we choose the optimal time-threshold CIWP (OTT-CIWP) scheme for comparison. Gao and Towsley [20] have showed that the OTT-CIWP scheme outperforms most other stream-merging schemes and the required server bandwidth for the $i$th video object has been derived as

$$B_{\text{average}}(\text{OTT - CIWP}) = b \times \left(\sqrt{2L\lambda_i + 1} - 1\right). \tag{22}$$

Figure 4 shows the numerical results for comparing the required server bandwidth of one video object among the Medusa scheme, the batching scheme, and the OTT-CIWP scheme. In Figure 4, the chosen time interval $T$ for the Medusa scheme is 1 minute while the batching time threshold for the batching scheme is 7 minutes. In addition, the length of the $i$th video object is 100 minutes. As one can see, the Medusa scheme significantly outperforms the batching scheme and the OTT-CIWP scheme over a wider range of request arrival rate.

### 5.2. Experiment

In order to evaluate the performance of the Medusa scheme in the general case that multiple video objects of varying popularity are stored on the parallel video servers, we use the *Turbogrid* streaming server[1] with 8 RTP server nodes as the experimental platform.

#### 5.2.1. Experiment environment

We need two factors for each video, its length and its popularity. For its length, the data from the 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. For its popularity, Zipf-like distribution [21] is widely used to describe the popularity of different video objects. Empirical evidence suggests that the parameter $\theta$ of the Zipf-like distribution is 0.271 to give a good fit to real video rental [5, 6]. It means that

$$\pi_i = \frac{1}{i^{0.729}} \left[ \sum_{k=1}^{N} \frac{1}{k^{0.729}} \right], \tag{23}$$

where $\pi_i$ represents the popularity of the $i$th video object and $N$ is the number of video objects stored on the parallel video servers.

Client requests are generated using a Poisson arrival process with an interval time of $1/\lambda$ for varying $\lambda$ values between 200 to 1600 arrivals per hour. Once generated, clients simply select a video and wait for their request to be served. The waiting tolerance of each client is independent of the other, and each is willing to wait for a period time $U \geq 0$ 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 exponential reneging function $R(u)$, which is used by most VOD studies [6, 7, 15]. Clients are always willing to wait for a minimum time $U_{\min} \geq 0$. The additional waiting time beyond $U_{\min}$ is exponentially distributed with mean $\tau$ minutes, that is,

$$R(u) = \begin{cases} 0 & \text{if } 0 \leq u \leq U_{\min} \\ 1 - e^{-(u - U_{\min})/\tau}, & \text{otherwise.} \end{cases} \tag{24}$$

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

Considering that the popular set-top boxes have similar components (CPU, disk, memory, NIC, and the dedicated client software for VOD services) to those of PCs, we use PCs to simulate the set-top boxes in our experiment. In addition, because the disk is cheaper, faster, and bigger than ever, we

[1] Turbogrid streaming server is developed by the Internet and Cluster Computing Center of Huazhong University of Science and Technology.

Table 2: Experiment parameters.

| | |
|---|---|
| Video length (min) $L$ | 90 ~ 120 |
| Number of videos $Nv$ | 200 |
| Video format | MPEG-1 |
| Clients' bandwidth capacity (Mbits/s) | 100 |
| Maximum total bandwidth of parallel video server (Mbits/s) | 1000 |
| Clients arrival rate $\lambda$ (hour) | 200 ~ 1600 |

do not consider the speed limitation and the space limitation of disk. Table 2 shows the main experimental environment parameters.

#### 5.2.2. Results

For parallel video servers, there are two most important performance factors. One is startup latency, which is the amount of time clients must wait to watch the demanded video, the other is the average bandwidth consumption, which indicates the bandwidth efficiency of the parallel video servers. We will discuss our results in these two factors.

(A) *Startup latency and reneging probability*

As discussed in Section 4, in order to guarantee that clients can receive all segments of their requested video objects, the minimum value of time interval (i.e., optimal time interval) $T$ will be $\lceil L/(2b_c) \rceil \cong 120/2^*60 = 1$ minute. We choose time interval $T$ to be 1, 5, 10, and 15 minutes for studying the effect on the average startup latency and the reneging probability, respectively. Figures 5 and 6 display the experimental results at these two factors. By the increase of time interval $T$, the average startup latency and the reneging probability are also increased. When $T$ is equal to the deterministic time interval $T = 1$ minute, the average startup 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 startup latency is increased to near 750 seconds and almost 45% of clients renege. Figures 7 and 8 display a startup latency comparison and a reneging probability comparison among the FCFS batching scheme with time interval $T = 7$ minutes, and the OTT-CIWP scheme [20] and the Medusa scheme with deterministic time interval $T = 1$ minute. We choose 7 minutes because [7] has presented that FCFS batching could obtain a good trade-off between startup latency and bandwidth efficiency at this batching time threshold. As one can see, the Medusa scheme outperforms the FCFS scheme and is just little poorer than the OTT-CIWP scheme at the aspect of the system average startup latency and reneging probability. The reason for this little poor performance compared with OTT-CIWP is that the Medusa scheme batches client requests arriving in the same time slot. This will effectively increase the bandwidth efficiency at high client-request rates.

(B) *Bandwidth consumption*

Figure 9 shows how the time interval $T$ affects the server's average bandwidth consumption. We find out that the server's

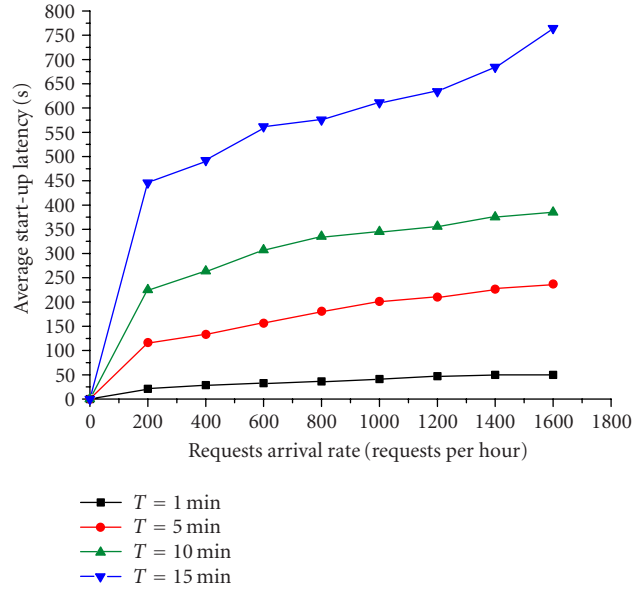


Figure 5: The effect of time interval $T$ on the average startup latency.

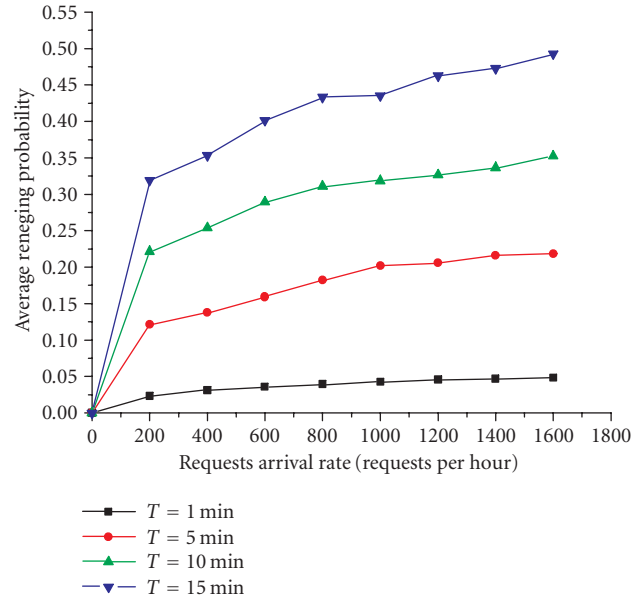


Figure 6: The effect of time interval $T$ on the average reneging probabity.

average bandwidth consumption is decreased by some degree by increasing the time interval. The reason is that more clients are batched together and served as one client. Also, we can find out that the decreasing degree of bandwidth consumption is very small when client-request arrival rate is less than 600 requests per hour. When the arrival rate is more than 600, the decreasing degree tends to be distinct. However, when the request arrival rate is less than 1600 requests

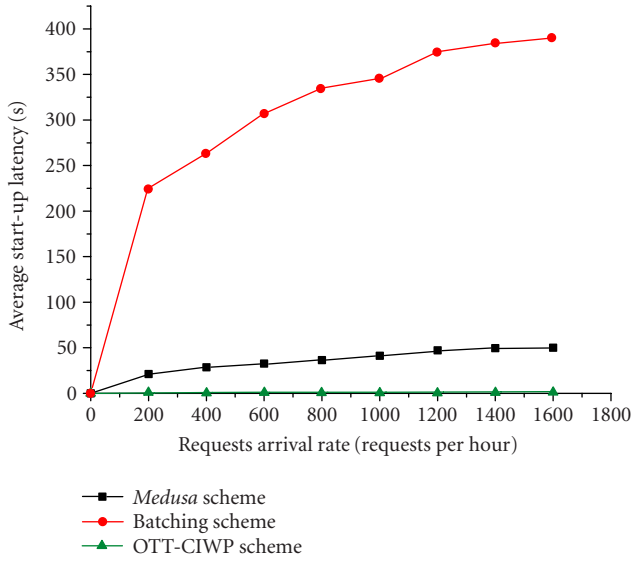


Figure 7: A startup latency comparison among the batching scheme with time interval $T = 7$ minutes, the OTT-CIWP scheme, and Medusa scheme with time interval $T = 1$ minute.

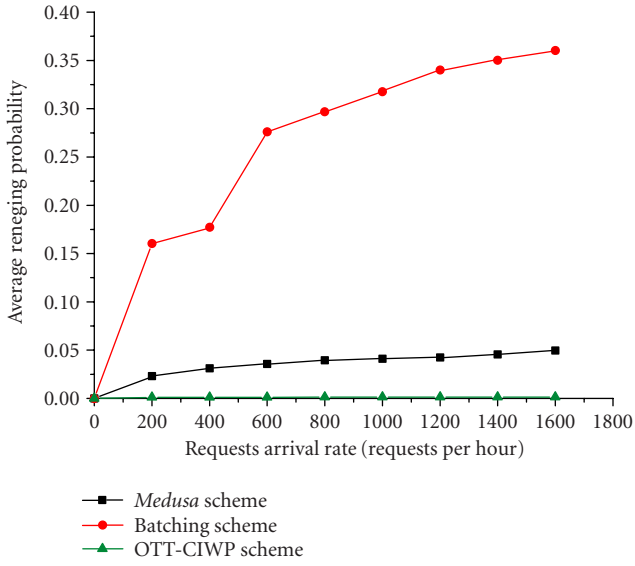


Figure 8: A reneging probability comparison among the batching scheme with time interval $T = 7$ minutes, the OTT-CIWP scheme, and the Medusa scheme with time interval $T = 1$ minute.

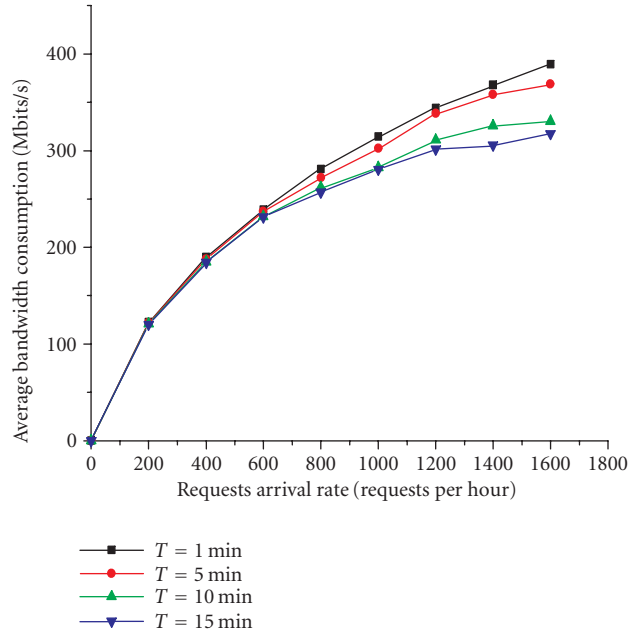


Figure 9: How time interval $T$ affects the average bandwidth consumption.

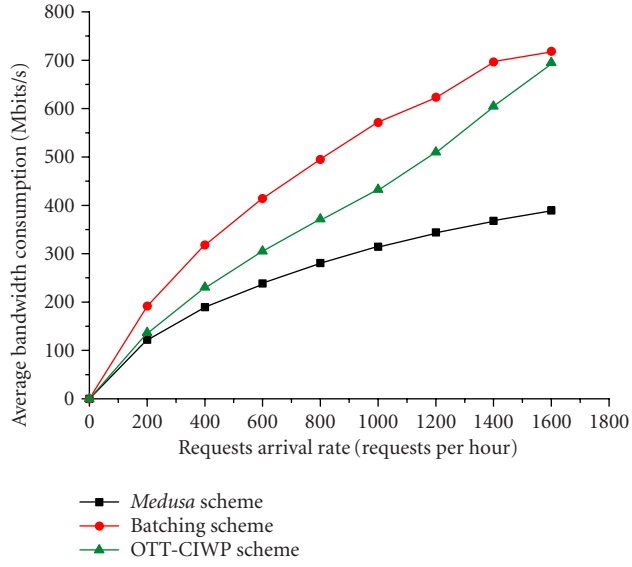


Figure 10: Average bandwidth consumption comparison among the batching scheme with time interval $T = 7$ minutes, the OTT-CIWP scheme, and the Medusa scheme with time interval $T = 1$ minute.

per hour, the total saved bandwidth is less than 75 Mbits/s by comparing deterministic time intervals $T = 1$ minute and $T = 15$ minutes. On the other hand, the clients reneging probability is dramatically increased form 4.5% to 45%. Therefore, a big time interval $T$ is not a good choice and we suggest using $\lceil L/(2b_c) \rceil$ to be the chosen time interval.

As showed on Figure 10, when the request arrival rate is less than 200 requests per hour, the bandwidth consumption of three kinds of scheduling strategies are held in the same level. But by increasing the request-arrival rate, the bandwidth consumption increasing degree of the Medusa scheme is distinctly less than that of the FCFS batching and the OTT-CIWP. When the request-arrival rate is 800, the average bandwidth consumption of the Medusa scheme is approximately 280 Mbits/s. At the same request-arrival rate, the average bandwidth consumption of the FCFS batching is

approximately 495 Mbits per second and that of the OTT-CIWP is approximately 371 Mbits per second. It indicates that, at middle request-arrival rate, the Medusa scheme can save approximately 45% bandwidth consumption compared with FCFS batching, and can save approximately 25% bandwidth consumption compared with OTT-CIWP.

When the request arrival rate is higher than 1500 requests per hour, the bandwidth performance of OTT-CIWP is going to be worse and worse. It is near to the FCFS batching scheme. In any case, the Medusa scheme significantly outperforms the FCFS scheme and the OTT-CIWP scheme. For example, as shown in Figure 10, the Medusa scheme just consumes 389 Mbits/s server bandwidth at the request arrival rate 1600 requests per hour, while the FCFS batching scheme consumes 718 Mbits/s server bandwidth and the OTT-CIWP scheme needs 694 Mbits/s. Therefore, we can conclude that the Medusa scheme is distinctly outperforming the batching scheme and the OTT-CIWP scheme at the aspect of bandwidth performance.

## 6. DISCUSSIONS

For the Medusa scheme, two issues must be considered carefully, the client network architecture and the segments placement policy. In this section, we give out some discussions on the effect of these two issues.

### 6.1. Homogenous client network versus heterogeneous client network

In the above discussions, we discuss the homogenous client network based on the FTTB network architecture. If the parallel video servers are used for serving the VOD systems with heterogeneous client network architecture such as the cable modem access and 10 M LAN access, the basic Medusa scheme is not recommended. This is because the small client bandwidth capacity would result in a large deterministic time interval $T$, as well as the long startup latency and the high reneging probability. However, we can extend the Medusa scheme as following for the heterogeneous client network.

For cable modem users, because the client bandwidth capacity is lower than 2 Mb per second, it just has the capacity to receive one MPEG-I stream (approximately $1.2 \sim 1.5$ Mb per second per stream). In this case, the stream merging schemes and the Medusa scheme are not suitable. We use the batching scheme to schedule streams. Note that the client bandwidth capacity is sent to the parallel video servers during the session being in setup. Thus, the parallel video server can distinguish the category of clients before determining how to schedule streams for serving them.

For 10 M LAN users, the client bandwidth capacity is enough to concurrently receive near 6 MPEG-I streams. In this case, if we use the basic Medusa scheme, the deterministic time interval for a video object with 120 minutes length is 10 minutes and the expected startup latency is near 5 minutes. It is too long for most clients. However, we can extend the basic Medusa scheme to use a time window $W$ to control the scheduling frequency of the complete multicast streams. If requests arrive in the same time window, the parallel video server schedules patching multicast streams according to the basic Medusa scheduling rule (3). Otherwise, a new complete multicast stream will be scheduled. According to the deriving course discussed in Section 4, we can easily obtain that the deterministic time interval $T$ should be $\lceil W/(2b_c) \rceil$. Obviously, if the value of time window $W$ is smaller than the length of the requested video object, the deterministic time interval $T$ and the expected startup latency can be decreased. However, a small time window $W$ would increase the required server bandwidth. The detailed relationship between the time window $W$, the expected startup latency, and the required server bandwidth will be studied in our further works.

### 6.2. Effect of the segment placement policy

For the scheduling of the Medusa scheme, the begining segments of a requested video are transmitted more frequently than the later segments of that video. It is called *intra-movie skewness* [22]. If segments of all stored videos are distributed from the first RTP server node to the last RTP server node in a round-robin fashion, the intra-movie skewness would result in that the load for the first RTP server node is far heavier than the load of other RTP server nodes so that the load balance of parallel video servers is destroyed.

Two kinds of segments placement policies were proposed to solve the intra-movie skewness problem: the symmetric pair policy [22, 23] and the random policy.

In the symmetric pair policy, based on the serial number of video objects, all stored video objects are divided into two video sequences, the odd sequence and the even sequence. For the odd video sequence, the $j$th segment of the $i$th video object ($i = 1, 3, 5, \ldots, 2k + 1$) is located on the $((2^*N - 1 - (j + (i/2)) \bmod N) \bmod N)$th RTP server node, where $N$ is the total number of RTP server nodes. For the even video sequence, the $j$th segment of the $i$th video object ($i = 0, 2, 4, \ldots, 2k$) is located on the $((j + (i/2)) \bmod N)$th RTP server node. As discussed in [22, 23], these placement rules can uniformly distribute segments with high transmission frequency to different RTP server nodes so that the load balance of the parallel video server can be guaranteed.

The random placement policy randomly distributes video segments on different RTP server nodes so that the probabilistic guarantee of load balancing can be provided. Santos et al. [24] have shown that the random placement policy has better adaptability to different user access patterns and can support more generic workloads than the symmetric pair policy. For load balancing performance, these two schemes have very similar balancing results [24]. However, the random placement scheme only provides probabilistic guarantee of load balancing and it has the drawback of maintaining a huge video index of the striping data blocks. Hence, we use the symmetric pair policy to solve the load balancing problem in the Medusa scheme.

## 7. CONCLUSIONS AND FUTURE WORKS

In this paper, we focus on the homogenous FTTB client network architecture and propose a novel stream-scheduling

scheme that significantly reduces the demand on the server network I/O bandwidth of parallel video servers. Unlike existing batching scheme and stream-merging scheme, the Medusa scheme dynamically groups the clients' requests according to their request arrival time and schedules two kinds of multicast streams, the completely multicast stream and the patching multicast stream.

For the clients served by patching multicast streams, the Medusa scheme notifies them to receive the segments that will be transmitted by other existing patching multicast streams and only transmit the missed segments on the new scheduled stream. This guarantees that no redundant video data are transmitted at the same time period and that the transmitting video data are shared among grouped clients. The mathematical analysis and the experiment results show that the performance of the Medusa scheme significantly outperforms the batching schemes and the stream-merging schemes.

Our ongoing research includes

(1) designing and analyzing the *extended-Medusa* scheme for clients with heterogeneous receive bandwidths and storage capacities,
(2) evaluating the impact of VCR operations on the required server bandwidth for the Medusa scheme,
(3) developing optimized caching models and strategies for the Medusa scheme,
(4) designing optimal real-time delivery techniques that support recovery from packet loss.

## ACKNOWLEDGMENT

This paper was supported by the National Hi-Tech Project under Grant 2002AA1Z2102.

## REFERENCES

[1] C. Shahabi, R. Zimmermann, K. Fu, and S.-Y. D. Yao, "Yima: a second generation continuous media server," *IEEE Computer magazine*, vol. 35, no. 6, pp. 56–64, 2002.
[2] G. Tan, H. Jin, and L. Pang, "A scalable video server using intelligent network attached storage," in *Management of Multimedia on the Internet: 5th IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, vol. 2496 of *Lecture Notes in Computer Sciences*, pp. 114–126, Santa Barbara, Calif, USA, October 2002.
[3] G. Tan, H. Jin, and S. Wu, "Clustered multimedia servers: architectures and storage systems," in *Annual Review of Scalable Computing*, vol. 5, pp. 92–132, World Scientific, Singapore, 2003.
[4] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optional batching policies for video-on-demand storage servers," in *Proc. 3rd IEEE International Conference on Multimedia Computing and Systems*, pp. 312–316, Hiroshima, Japan, June 1996.
[5] S. W. Carter and D. D. E. Long, "Improving video-on-demand server efficiency through stream tapping," in *Proc. 6th International Conference on Computer Communication and Networks*, pp. 200–207, Las Vegas, Nev, USA, September 1997.
[6] S.-H. G. Chan and F. Tobagi, "Tradeoff between system profit and user delay/loss in providing near video-on-demand service," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 8, pp. 916–927, 2001.
[7] J.-K. Chen and J.-L. C. Wu, "Heuristic batching policies for video-on-demand services," *Computer Communications*, vol. 22, no. 13, pp. 1198–1205, 1999.
[8] D. L. Eager and M. K. Vernon, "Dynamic skyscraper broadcasts for video-on-demand," Tech. Rep. 1375, Department of Computer Science, University of Wisconsin, Madison, Wis, USA, 1998.
[9] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "The maximum factor queue length batching scheme for video-on-demand systems," *IEEE Trans. Comput.*, vol. 50, no. 2, pp. 97–110, 2001.
[10] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. 2nd ACM International Conference on Multimedia*, pp. 15–23, San Francisco, Calif, USA, October 1994.
[11] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Systems*, vol. 4, no. 3, pp. 112–121, 1996.
[12] H. J. Kim and Y. Zhu, "Channel allocation problem in VOD system using both batching and adaptive piggybacking," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 3, pp. 969–976, 1998.
[13] S. W. Carter and D. D. E. Long, "Improving bandwidth efficiency on video-on-demand servers," *Computer Networks*, vol. 30, no. 1-2, pp. 99–111, 1999.
[14] S.-H. G. Chan and E. Chang, "Providing scalable on-demand interactive video services by means of client buffering," in *Proc. IEEE International Conference on Communications*, pp. 1607–1611, Helsinki, Finland, June 2001.
[15] D. Eager, M. Vernon, and J. Zahorjan, "Bandwidth skimming: A technique for cost-effective video-on-demand," in *Proc. Multimedia Computing and Networking 2000*, San Jose, Calif, USA, January 2000.
[16] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. 6th ACM International Multimedia Conference*, pp. 191–200, Bristol, UK, September 1998.
[17] W. Liao and V. O. K. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol. 4, no. 4, pp. 51–62, 1997.
[18] J.-F. Paris, S. W. Carter, and D. D. E. Long, "A hybrid broadcasting protocol for video on demand," in *Proc. 1999 Multimedia Computing and Networking Conference*, pp. 317–326, San Jose, Calif, USA, January 1999.
[19] H. Shachnai and P. Yu, "Exploring wait tolerance in effective batching for video-on-demand scheduling," *Multimedia Systems*, vol. 6, no. 6, pp. 382–394, 1998.
[20] L. Gao and D. Towsley, "Threshold-based multicast for continuous media delivery," *IEEE Trans. Multimedia*, vol. 3, no. 4, pp. 405–414, 2001.
[21] G. Zipf, *Human Behavior and the Principle of Least Effort*, Addison Wesley, Boston, Mass, USA, 1949.
[22] S. Wu and H. Jin, "Symmetrical pair scheme: a load balancing strategy to solve intra-movie skewness for parallel video servers," in *International Parallel and Distributed Processing Symposium*, pp. 15–19, Fort Lauderdale, Fla, USA, April 2002.
[23] S. Wu, H. Jin, and G. Tan, "Analysis of load balancing issues caused by intra-movie skewness for parallel video servers," *Parallel and Distributed Computing Practices*, vol. 4, no. 4, pp. 451–465, 2003.
[24] J. Santos, R. Muntz, and B. Ribeiro-Neto, "Comparing random data allocation and data striping in multimedia servers," in *Proc. International Conference on Measurement and Modeling of Computer Systems*, pp. 44–55, Santa Clara, Calif, USA, June 2000.

**Hai Jin** is a Professor at the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. He received M.S. and Ph.D. degrees at HUST in 1991 and 1994, respectively. He was a Postdoctoral Fellow at the Department of Electrical and Electronics Engineering, University of Hong Kong, and a visiting scholar at Department of Electrical Engineering-System, University of South California, Los Angeles, USA, from 1999 to 2000. His research interests include cluster computing, grid computing, multimedia systems, network storage, and network security. He is the Editor of several journals, such as *International Journal of Computers and Applications, International Journal of Grid and Utility Computing*, and *Journal of Computer Science and Technology*. He is now leading the largest grid project in China, called ChinaGrid, funded by the Ministry of Education, China.

**Dafu Deng** received his Bachelor degree in engineering from the Tongji University, Shanghai, China, in 1997, and is a Ph.D. candidate at the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. His research interests include cluster computing, grid computing, multimedia systems, communication technologies, and P2P systems.

**Liping Pang** is a Professor at the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. In 1995, she was awarded the "Golden medal in education of China." In the recent three years, she has over 40 publications and 3 books in computing science and education. Her research interests include parallel and distribution computing, grid computing, cluster computing, and multimedia technology.