# VOVO: VCR-Oriented Video-on-Demand in Large-Scale Peer-to-Peer Networks

Yuan He, *Student Member*, *IEEE*, and Yunhao Liu, *Senior Member*, *IEEE*

**Abstract**—Most P2P Video-on-Demand (VOD) schemes focus more on mending service architectures and optimizing overlays but do not carefully consider the user behavior and the benefit of prefetching strategies. As a result, they cannot better support VCR-oriented services in terms of substantive asynchronous clients or free VCR controls for P2P VODs. To address this issue, we propose VOVO, a VCR-oriented VOD for large-scale P2P networks. By mining associations inside each video, the segments requested in VCR interactivities are predicted based on the information collected through gossips. Together with a hybrid caching strategy, a collaborative prefetching scheme is designed to optimize resource distribution among neighboring peers. We evaluate VOVO through extensive experiments. Results show that VOVO is scalable and effective, providing short start-up latencies and good performance in VCR interactivities.

**Index Terms**—Peer-to-peer, Video-on-Demand, VCR-oriented.

## 1 Introduction

Multimedia communications and entertainments have been essential parts of people's daily life. Video streaming, attracting extensive attention during the past years, becomes the most popular activity over the Internet. Compared to other Internet applications, video streaming usually supports a large number of concurrent clients and consumes more network bandwidth. On the other hand, the dramatic development in Peer-to-Peer (P2P) technologies presents great scalability and support for millions of users worldwide.

Video streaming is also boosted by P2P technologies [1], [2], [3], especially after the application level multicast (ALM) is proposed. Video streaming can be classified into two categories: live streaming and video on demand (VOD). In live streaming systems, the source servers broadcast live shows or TV programs, and all the clients play the content at a same progress. As the clients of each program/show have a common downloading target, they can efficiently share their buffers, alleviating the server stress and accelerates the downloading processes. Successful examples include Narada [1], CoolStreaming [4], and PPLive [5]. VOD is an interactive multimedia service, which delivers video content to the users [6]. Differing live streaming, a VOD user expects to enjoy the video with completely free choices. Due to the frequent VCR controls from the users such as play, pause, fast forward, fast search, reverse search, and rewind, existing approaches either present long latencies on the user side or incur excessive stress on the server side. In this work, we focus on the single-video VOD process, where a peer only transfers the video it is currently playing. For ease of expression, in the rest of the paper, we use the terms "client," "peer," and "node" interchangeably.

In order to provide "play-as-download" VOD services, stream reuse techniques such as batching, patching, and chaining are proposed [7], [8], [9], [10]. Generally, the overlay construction with those techniques is tightly coupled with the peers' playback progresses ("playback" refers to the usual state when the video is continuously played by the media player). The stream reusability will be underutilized unless partnering peers keep persistent connections with each other. Consequently, user experiences are seriously degraded when they take frequent VCR controls. To address this problem, prefetching is also employed. Different strategies are adopted such as sequential, random, and global rarest strategies [11], but none of them addresses the content-based associations among different segments of videos.

Fig. 1 illustrates the sequence of segments in a movie. Without loss of generality, we assume the VOD clients prefetch one segment from the next five segments. Hit ratio is defined as the probability with which the VCR requests are satisfied by the prefetched content. Suppose most users are interested in the dotted segments. They probably skip the blank segments which are the period of prelude. After the third segment is played, a user takes a VCR control (i.e., fast search) to request the seventh segment. For the sequential prefetching strategy, the fourth segment is prefetched, $\text{hit ratio} = 0$. For the random prefetching strategy, a random segment from the fourth to the eighth is prefetched, $\text{hit ratio} = 1/5$. For the global rarest prefetching strategy, regardless of the global distribution of the cached segments, the chance to be prefetched for each segment is uniform. Thus, $\text{hit ratio} = 1/5$. With such low hit ratios, the user likely suffers from a long interruption because the requested segment is not prefetched.

For the prefetching strategy that is aware of associations inside the video, however, the first segment to be

• *The authors are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: {heyuan, liu}@cse.ust.hk.*

*Manuscript received 15 Nov. 2007; revised 13 Feb. 2008; accepted 27 May 2008; published online 4 June 2008.*
*Recommended for acceptance by C.-Z. Xu.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-11-0428.*
*Digital Object Identifier no. 10.1109/TPDS.2008.102.*

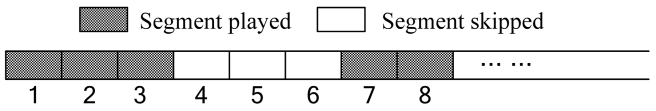

Fig. 1. An example to compare prefetching strategies.

prefetched after the third segment should be the seventh segment. Thus, $\text{hit ratio} = 1$. The user seldom experiences any interruption because the requested segment has the highest priority to be prefetched.

The above example indicates that the experience of VOD users can be greatly improved with a better prefetching strategy. Bearing this in mind, we propose VOVO, a VCR-oriented VOD scheme in large-scale P2P networks, which can be characterized as follows:

1. **VCR oriented.** VCR interactivities are efficiently predicted using the technique of association rule mining. Requests of VCR controls are efficiently resolved locally.
2. **Scalability.** We adopt the combination of batching and patching. VOVO is able to serve many more concurrent clients than the original capacity of the source server.
3. **Flexibility.** Based on the scheme of patching and the hybrid caching strategy, VOVO provides abundant backup resource for asynchronous clients and frequent VCR requests.
4. **Short latency.** VOVO adopts an efficient gossip protocol and a collaborative prefetching strategy. Both the requests of joining in and the dynamic VCR controls can be responded with very short latencies.

The rest of this paper is organized as follows: Section 2 discusses the related work. In Section 3, we show some observations and discuss the basic concepts of this scheme. Section 4 elaborates the system design of VOVO. In Section 5, we theoretically analyze the validity, efficiency, and overhead. Performance evaluation is presented in Section 6. Section 7 concludes the work.

## 2 Related Work

Previous schemes of VOD mainly focus on improving the client-server model. For example, in [12], a video is divided into multiple segments and periodically broadcast to the clients through dedicated server channels. Users joining in asynchronously may receive the streams from different channels. In order to provide uninterrupted streaming for all the users, the server keeps track of all the channels and ensures no interruptions exist between playing segments, resulting in heavy overhead. Further, as the requests randomly come to the server, the numbers of users in certain channels are limited, and some channels are underutilized. Dan et al. [7] propose a pull-based scheme called *batching*, in which requests coming to the server are placed in a waiting queue. When the length of the queue reaches a threshold, or the earliest client has been kept waiting for a certain period of time, a multicast video stream is transmitted to the waiting queue. Nevertheless, early clients might wait even longer especially when the queue length is increased to reduce the server stress. The service provided by the schemes above is actually Near Video on Demand (NVOD), where the playback on clients is constrained by the granularity of the server channels. There are also some schemes proposed with IP multicast for True Video on Demand (TVOD), in which users have full freedom in VCR interactivities [10], [13], [14], [15]. Due to the difficulty of deploying IP multicast, however, many schemes are extended to adopt ALM [8], [9], [16], [17].

S. Sheu et al. [10] propose to find overlaps among client buffers and create chains of clients, called *chaining*. All the clients hold the latest periods of video content. The early clients rather than the video server can thus serve the subsequent clients, so as to alleviate the server stress. Practically, each single chain occupies one dedicated full stream of the video, so the chaining scalability actually depends on the aggregate bandwidth provided by the distribution network. Besides, the chaining structure presents unsatisfactory resiliency against churns and link dynamics. P2Cast [9] is one of the earliest models of P2P VOD. It adopts cooperative streaming using the *patching* technique. In P2Cast, clients are clustered according to their arrival times and form sessions. Each session, together with the server, constructs an application multicast tree. Later, clients can retrieve the missing parts from the server or other clients. P2VoD [8] divides the clients into generations according to their requests. Clients in P2VoD always cache the most recent content of a video, while clients in P2Cast only cache the initial part. Consequently, only one stream from an early client is needed to serve a late client in P2VoD, while two streams, a patching, and a base stream are necessary for serving a late client in P2Cast. Instead of deploying a patch server as P2Cast, failures and dynamics are handled locally in P2VoD, so that the server stress is further reduced.

Rodriguez et al. [11] focus on improving the VOD user experiences, such as providing a small start-up latency and a sustainable playback rate. With observations on prefetching strategies, they find similar result as revealed by the example shown in Fig. 1. Some heuristics are proposed to optimize the global resource distribution in the manner of centralized scheduling and management. A central tracker is used to track all the peers, and the peers are forced to periodically report their progresses to the central tracker. Such a mechanism incurs considerable control overhead on the central server. Also, it is vulnerable to targeted attacks.

Different from the above schemes, VOVO serves asynchronous peers with the combination of batching and patching. The server does not need to record any buffer information of peers. As we will elaborate in Section 4, the VCR requests are resolved in a distributed manner. Moreover, VOVO employs a behavior-content-based scheme of prefetching. We leverage the technique of association rule mining to find the frequent patterns of VCR interactivities, based on which neighboring peers conduct collaborative prefetching. Using this scheme, peers intelligently prefetch the requested segments before the occurrences of VCR interactivities. As a result, the response latencies of VCR interactivities are minimized.

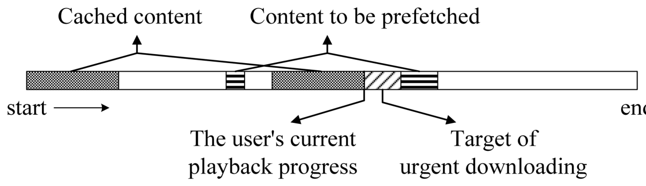
Cached content
Content to be prefetched
start
end
The user's current playback progress
Target of urgent downloading

Fig. 2. Urgent downloading, prefetching, and hybrid caching.

## 3 Design Overview

### 3.1 Observations on VOD User Behavior

Our design is based on the following observations [18], [19]. First, VOD users rarely view the movie continuously from the beginning to the end. As peers with transient life span account for a large proportion, streaming connections among peers are unstable and altered often. Second, accesses to different periods of a video are not uniformly distributed. Some parts always attract more accesses than the others. Hot scenes are always appreciated by most audience and lead to a consensus on the popularity of a video. Third, VCR interactivities occur frequently, but many VCR controls are forward-looking aimlessly. This is due to the user's skips of the video, directly ignoring a period uninteresting. Such discoveries reveal that prefetching the requested content accurately may benefit the performance of VCR interactivities a lot.

### 3.2 Policy of Streaming and Caching

A peer in VOVO always conducts two types of downloading: urgent downloading and prefetching, as shown in Fig. 2. Urgent downloading keeps streaming for the content in the immediate next 2 minutes, which meets the demand of continuous playback. Besides urgent downloading, a peer can potentially prefetch more content with its surplus downloading bandwidth. Considering that the surplus bandwidth on a peer for prefetching is typically 0-3 times of the bandwidth used for urgent downloading [6], we set the size of buffer for prefetching to store 3 minutes of video at most.

As for the management of buffer, VOVO performs a hybrid caching strategy, as illustrated in Fig. 2. Every peer caches the latest 5 minutes of the video played and uses the Least-Recently-Used (LRU) policy for cache replacement. Moreover, all the peers hold the initial 5 minutes of video and never replace this part until they depart from the system. We will discuss cache settings and costs in Section 5.

### 3.3 Associations Inside a Video

The technique of association rule mining is used to discover elements that cooccur frequently within a data set consisting of multiple selections of elements [20]. Association rules have the general form $t_1 \rightarrow t_2$ (where $t_1 \cap t_2 = \Phi$), where $t_1$ and $t_2$ are sets of items. The rule $t_1 \rightarrow t_2$ holds in the transaction set $T$ with certain *support* and *confidence*. *Support* indicates the occurring frequency of $t_1 \cup t_2$ in the transaction set $T$, while *confidence* indicates the strength of the implication from $t_1$ to $t_2$.

Now, we consider the case of VOD. A video program can be regarded as a sequence of continuous media segments. Supported by the VCR functionalities, a user may choose to watch only a subset of the sequence, i.e., several segments. Due to the difference in individual interests, the segments watched by different users vary a lot. However, segments of a video are not completely independent to each other. In other words, if the $x\text{th}$ and the $y\text{th}$ segment of a video are close associated, users who watch the $x\text{th}$ segment will probably watch the $y\text{th}$ segment. Such associations can be inferred from the history information of VOD users' behavior.

The aforementioned association rule mining helps to prefetch corresponding contents. Nevertheless, to collect the history information is nontrivial in a large-scale VOD system. Existing schemes include the central server blessed mechanisms or using an offline data set [19], [21]. Superior to these schemes, VOVO takes full advantage of the P2P networks and enables the peers to exchange their history information in a distributed manner.

Peers in VOVO execute periodical workflows to leverage the design essentials above, as depicted in Fig. 3: while a peer is streaming video, it maintains a record (named playback record) of its playback history. Peers exchange their records through gossips, so that the states of peers are efficiently propagated and updated throughout the networks. Based on the information collected from other peers (the list of playback records), a peer dynamically predicts its future behavior using the technique of association rule mining. Based on such predictions, neighboring peers conduct collaborative prefetching to maximize the resource availability in a local area. As a result, the response latency for any VCR control is restricted to a fairly low level.

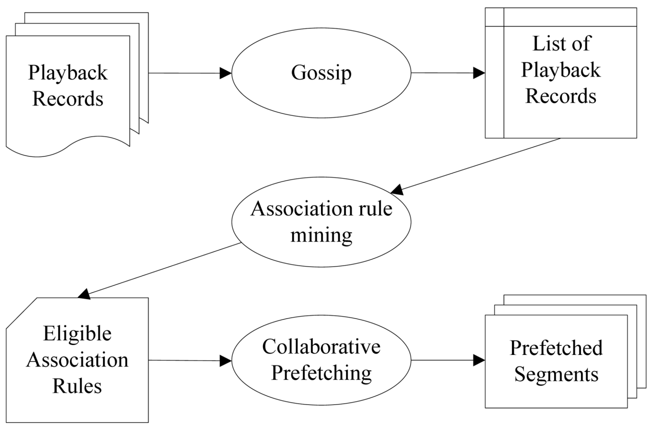

Fig. 3. Work flow from gossip to prefetching.

## 4 System Design

VOVO mainly has four components: overlay manager, state manager, streaming scheduler, and player, as illustrated in Fig. 4. Section 4.1 introduces the overlay management. The working mechanism of the state manager is described in Sections 4.2 and 4.3, including the gossip-based state propagation and the association rule mining. Section 4.4 focuses on the collaborative prefetching strategy executed on the streaming scheduler.

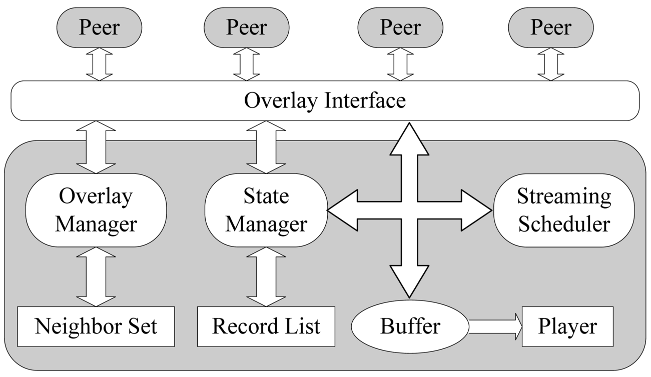

Fig. 4. The diagram of a VOVO node.

## 4.1 Batching Plus Patching: Scaling the Service

In the overlay management, VOVO adopts the combination of batching and patching, as shown in Fig. 5. Such a design is flexible with substantive concurrent and asynchronous clients.

The VOD server uses *batching* to serve asynchronous peers. Every $m$ minutes the server starts a new *batching session* to broadcast the video from the beginning, where $m$ is a predefined *session width*. For example, when $m = 3$, peers joining in the system from the very beginning to the end of the third minutes form *Session* 1. Similarly, later, joining peers successively become participants of *Session* 2, *Session* 3, etc. Sessions may have unequal numbers of participating peers, depending on the varying joining rates of peers.

According to the observed data in [6], a client in a P2P VOD system is usually able to contribute its bandwidth as much as it receives from other peers. Therefore, we assume for every peer in VOVO that the available downloading and uploading bandwidths are both at least equal to the full source rate of the video. The server allocates a certain amount of dedicated outgoing bandwidth for each batching session. In each session, early joining peers directly become the children of the server. After the allocated bandwidth is fully occupied, late peers are redirected by the server and become the descendants of the early ones. Since the peers in a session transfer same video content currently broadcast by the server, the streaming mechanism inside a batching session is similar with P2P live streaming [2], [4], [22]. Peers connected with batching connections are called neighboring peers or neighbors.

Moreover, VOVO reinforces the batching scheme with *patching*. The server sends a list of randomly selected peers to each joining peer. When a peer joins in a session late and misses the initial part of the video, it picks up a few peers from the random list as *patching sources* and immediately starts to download the missing part from them. Fig. 5 shows three typical patching connections in VOVO (for clear display, we do not draw all the patching connections). Note that patching sources are not necessarily peers in the same session. They can be peers from the random list or just the VOD server as long as there is surplus outgoing bandwidth allocated for the current session.

Consequently, a late peer in VOVO simultaneously keeps two kinds of connections in the beginning periods of its life span: batching connections and patching connections. After it finishes downloading the missing part, patching connections are ended. Note that a peer without any VCR interactivity only needs batching and patching connections. Therefore, streaming through batching and patching belongs to urgent downloading.

## 4.2 State Maintenance and Propagation

- *Playback record*. Every minute of video is called a segment. A peer in VOVO maintains its playback record while streaming and playing the video. The playback record is a string of segment indices, which is initially empty. When a segment is played, its index is inserted into the tail of the string. For

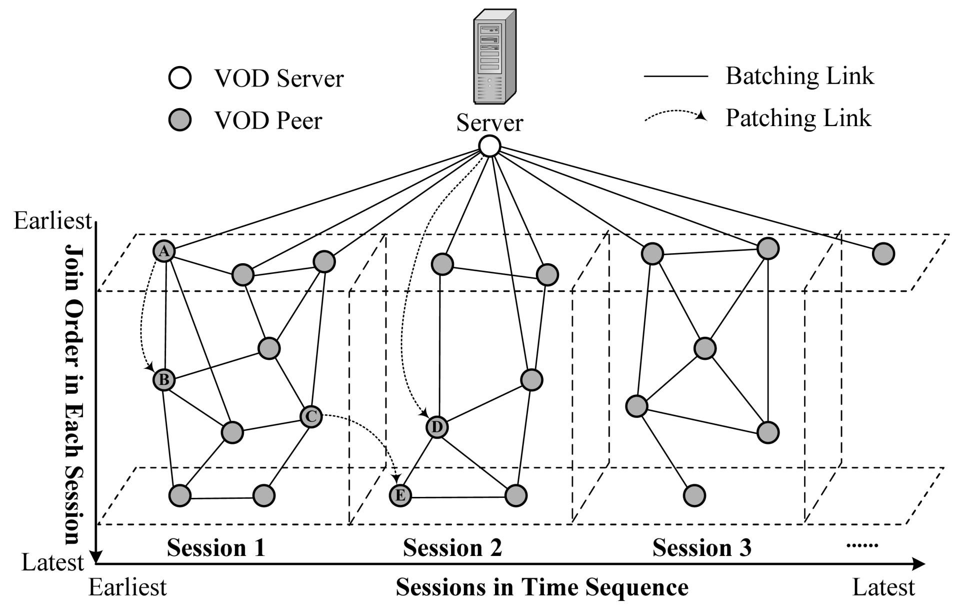

Fig. 5. Batching and patching in VOVO.

example, suppose the current playback record of a peer is (1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14, 15, 7, 8), it depicts a playback history as follows: the peer first plays the video from the first to the eighth minute, fast searches to the 11th minute, plays until the 15th minute, then reverse searches to the seventh minute, and plays the eighth minute before the playback record is last updated.

- *Gossip*. Peers in VOVO conduct periodical gossips to exchange their state information. During each period, a peer generates a state message, including its latest state information. The format of the state message is {*IP, Incremental playback record, Time stamp*}, where *IP* is the peer's IP address, *Incremental playback record* refers to the string of segments the peer plays after it generates last state-message last time, *Time stamp* is the time since the peer joins in. On the other hand, each peer maintains a list of records. Each entry in the list corresponds to a peer and records its latest state.

  On receiving a *state message*, a peer performs relevant operations before it forwards the message to its neighbors: If peer $A$ receives the *state message* of peer $B$ for the first time, it replies to $B$ with a request for the full playback record. On receiving the request, $B$ will send its full playback record to $A$. If peer $A$ has received the *state message* before, it compares two time stamps. If the time stamp of the *state message* is greater than that in the entry, the incremental playback record is inserted into the tail of the playback record in the entry, and the time stamp in the entry is updated.

  Using the gossip-based state propagation, a peer is able to accumulate the information of playback history of all the peers. Furthermore, since the hybrid caching strategy is well known to all, through periodical gossips every peer can keep aware of the global distribution of video data on all the peers.

### 4.3 Mining the Association Rules

Peers in VOVO take the state information collected through gossips as the input of mining. The goal of mining is to find the segments most associated to the segment currently played.

First, it is observed that the user behavior in the next few minutes is closely related with his/her experience during the last few minutes [19], [21]. The preconfigured sizes of item sets in the association rules have apparent impact on the efficiency and accuracy of mining. Thus, we propose to mine all the rules $t_1 \rightarrow t_2$, where $t_1 \cap t_2 = \Phi$, $|t_1| = |t_2| = 3$.

Second, the playback history of a VOD user actually forms a sequence of segments. Input and output of predictions based on the history information ought to be order sensitive. For example, a user who plays the segments (4, 5, 1) will probably continue to watch the second and the third segments, while a user who plays the segments (1, 4, 5) will probably go on with the sixth segment. We have different predictions of their future behavior because they play the same segments in different orders.

Third, according to the theory of association rule mining [20], we find all the association rules that have a support and

| Entries in ***L*** | |
|---|---|
| Peer ID | Playback Records |
| B | 1,2,3,4,5,6,7,8, |
| C | 1,2,3,4,5,8,9,10,11 |
| D | 1,2,3,4,8,9,10,11,12,13,14 |
| E | 1,2,3,4,5,8,9,10 |
| F | 1,3,4,7,8,9,10,11,12 |
| G | 1,2,3,2,3,4,5,6,7,8 |
| H | 3,4,7,8,9,10,13,14,15 |
| I | 1,2,3,4,9,10,11 |

(a)

$\Longrightarrow$

| Extracted Sub-strings |
|---|
| 6,7,8 |
| 8,9,10 |
| 8,9,10 |
| 8,9,10 |
| 6,7,8 |
| 9,10,11 |

(b)

Fig. 6. Extract the substrings from the list of records.

a confidence greater than the specified thresholds. For example, we require the eligible association rules to have a support greater than 10 percent and a confidence greater than 25 percent. Since VOVO serves large-scale P2P networks, a single peer may keep a large list of records and extract a good number of substrings during the course of association rule mining. Setting the thresholds of support and confidence helps to avoid the impact of random coincidence and improves the precision and accuracy of mining.

For a particular peer $A$, its playback history in the last 3 minutes is denoted as an ordinal string $(a_1, a_2, a_3)$. Let $L$ be the local list of records kept by peer $A$. Based on the guidelines above, the process of association rule mining in VOVO runs as follows: First, the Knuth-Morris-Pratt algorithm [23] for string searching is used to locate the substring $(a_1, a_2, a_3)$ in each entry of $L$. Then, peer $A$ extracts all those segments which simultaneously satisfy the following requirements: 1) From each record in the list, three segments (if exist) at most are extracted. 2) They do not contain the immediate next two segments after the current progress of playback, because these segments are downloaded through urgent downloading. 3) They are the closest segments following the $a_3\mathrm{th}$ segment in the record.

We take Fig. 6 as an example. Suppose peer $A$ has its list $L$, as shown in Fig. 6a, and $(a_1, a_2, a_3) = (1, 2, 3)$. When peer $A$ searches through the playback record of peer $B$, the fourth and fifth segments will not be extracted. Instead, the substring (6, 7, 8) is extracted, and we get an association rule (1, 2, 3) $\rightarrow$ (6, 7, 8). When peer $A$ searches through the playback record of peer $D$, the fourth segment will not be extracted. The substring (8, 9, 10) is extracted and we get an association rule (1, 2, 3) $\rightarrow$ (8, 9, 10). When peer $A$ searches through the playback record of peer $F$, it extracts nothing because the record of $F$ does not contain the substring (1, 2, 3).

The extracted substrings are listed in Fig. 6b. They become the targets that peer $A$ is likely to play in the next few minutes, so some of them should be prefetched if possible. After filtering all the association rules with the specified thresholds of support and confidence, we finally get two eligible association rules:

$$\begin{aligned}
&(1,2,3) \rightarrow (6,7,8), \mathit{support} = 2/8 = 25 \text{ percent}, \\
&\quad \mathit{confidence} = 2/6 = 33.3 \text{ percent}; \\
&(1,2,3) \rightarrow (8,9,10), \mathit{support} = 3/8 = 37.5 \text{ percent}, \\
&\quad \mathit{confidence} = 3/6 = 50 \text{ percent}.
\end{aligned}$$

The segments to be prefetched are selected from the ones occurring most frequently in the eligible association rules (i.e., the segments 6, 7, 8, 9, and 10).

We may notice that the above scheme of association rule mining does not directly differentiate video segments according to their playing frequencies. Instead, the predictions of the VCR interactivities are closely related to the peer's recent playback behavior. Though different persons might have different preferences and the individual playback behavior might not be predictable, the collection of all the playback records is still able to reveal the nature associations among the segments inside the videos. Basing on such association rules, the collaborative prefetching introduced in Section 4.4 greatly benefits in the performance in VCR interactivities.

## 4.4 On VCR Interactivities

In this section, we first present the taxonomy of response latencies of VCR interactivities. The expected response latency is deduced, based on which we propose the collaborative prefetching strategy.

**Response latencies of VCR interactivities.** To minimize the response latencies of VCR interactivities is a crucial point of a VCR-oriented design. For a particular VCR control, the response latency is defined as the duration of interruption before the playback is resumed. Thus, the response latencies of VCR interactivities $T_i$ $(i = 1, 2, 3, 4)$ have different forms in the following four cases.

When the current peer has not cached the requested segment of the VCR control and the location of the segment is unknown, we have

$$T_1 = T_s + T_r + T_d, \tag{1}$$

where $T_s$ is the time to find the location of the requested segment, $T_r$ is the time to build the streaming connection, and $T_d$ is the time needed to download the requested segment.

When the requested segment is not cached on the current peer but located on an unconnected peer, we have

$$T_2 = T_r + T_d. \tag{2}$$

When the requested segment is not cached on the current peer but cached on a connected neighbor, we have

$$T_3 = T_d. \tag{3}$$

When the requested segment is already cached on the current peer, we have

$$T_4 = 0. \tag{4}$$

Suppose the occurring probability of the four cases above is respectively $p_1$, $p_2$, $p_3$, and $p_4$. Then, we have

$$\begin{aligned} E(T) &= E(p_1 \times T_1 + p_2 \times T_2 + p_3 \times T_3 + p_4 \times T_4) \\ &= p_1 \times E(T_1) + p_2 \times E(T_2) + p_3 \times E(T_3) + p_4 \times E(T_4) \\ &= p_1 \times E(T_1) + p_2 \times E(T_2) + p_3 \times E(T_3) \\ &= p_1 \times E(T_s) + (1 - p_3 - p_4) \times E(T_r) + (1 - p_4) \times E(T_d). \end{aligned} \tag{5}$$

```
segment Select()          // Find the next segment to prefetch
{
  Find the segments $a_i$ with lowest count[$a_i$] in PrefetchingSet;
  if (multiple segments are found)
      return (segment $a_i$ with the highest occurring frequency);
  else return $a_i$;
}

void Prefetch()           // The function to do prefeching
{
    while (PrefetchingSet is not empty)
  {
      segment k = Select();
      Broadcast(k);     // Broadcast the prefetching of segment k;
      if (segment k is cached by a neighbor)
      {
          Download segment k;
          Remove segment k from the PrefetchingSet;
      }
      else if (segment k is located on a remote peer P)
      {
          Connect with the peer P;
          Download segment k;
          Remove segment k from the PrefetchingSet;
      }
      else
      {
          Search the peer P which caches segment k;
          Connect with the peer P;
          Download segment k;
          Remove segment k from the PrefetchingSet;
      }
  }
}
```

Fig. 7. The pseudocode of prefetching.

Regardless of the variance of $E(T_s)$, $E(T_r)$, and $E(T_d)$, we must try to maximize $p_4$ so that the expected response latency $E(T)$ can be minimized. Even if the fourth case cannot be satisfied, the third and second cases must be either satisfied.

In other words, we should adopt a proper prefetching strategy to maximize the hit ratio of requested segments, which in fact is equal $p_4$. In case the requested segments are not downloaded, the chance to find it on a neighbor or a known peer should be maximized, i.e., to maximize $p_3$ or $p_2$. And, the first case, in which a peer has to search for the requested segment after the VCR control, must be avoided.

**Collaborative prefetching strategy.** VOVO adopts a collaborative prefetching strategy to fulfill the requirements above. The pseudocode is shown in Fig. 7, which mainly consists of three stages as follows:

First, every peer sets up its *prefetching set*. It sorts up the segments deduced by the eligible association rules according to their occurring frequencies. Every peer sorts out three most requested segments to prefetch, neglecting those segments already in the cache. If a peer cannot find enough segments after such sorting, it directly chooses the successive unprefetched segments closest to its current progress. This is because a user is likely to watch the successive content. A *prefetching buffer* is allocated to store the prefetched segments. After the prefetched segments are played in the near future, they will be merged into the cached content and replaced according to the LRU policy as well.

Second, in the scope of neighborhood, every peer respectively keeps counting the copies of each prefetched segment. For example, when peer $A$ starts to prefetch segment $a_1$, it broadcasts a message to all its neighbors. On receiving the message, every neighbor adds 1 to the count of segment $a_1$. Note that neighboring nodes possibly have different playback histories in the last few minutes. Their *prefetching set* may be similar but different from each other. Due to the link diversity and dynamic user behavior, peers usually execute prefeching asynchronously. To keep counting the copies of prefetched segments helps to synchronize the state of resource distribution in the neighborhood.

Third, based on the resource distribution in its neighborhood, every peer performs prefeching according to the *local-rarest-first* strategy. Due to the constraints in time and bandwidth, a peer is probably unable to download all the segments in the prefetching set before they are requested. Using the *local-rarest-first* strategy, however, the probability is maximized for a peer to find the requested segments locally, i.e., either in its *prefetching buffer* or on a neighbor.

## 5 Discussion and Analysis

### 5.1 Batching, Patching, and Hybrid Caching

Here, we discuss the features of VOVO in overlay management and the hybrid caching strategy.

VOVO combines batching and patching to organize the overlay. Combinatorial approaches have been adopted in some existing works. Annapureddy et al. adopt a mesh-based approach for P2P VOD in [11]. Magharei and Rejaie [22] also adopt a mesh-based approach called PRIME, however, mainly for the applications of P2P live streaming. PRIME combines the schemes of diffusion and swarming to mitigate the "bandwidth bottleneck" and "content bottleneck." Peers are divided into multiple diffusion subtrees, which are similar with the structure of batching in VOVO, only with regard to the overlay topology.

Nevertheless, the existing approaches cannot be directly migrated and applied in VOVO. We propose the scheme of batching and patching mainly for the following purposes:

1. To provide scalable service. Using batching, the bandwidth consumption on the server is limited to no more than the allocated quota, as elaborated in Section 4.1.
2. To support asynchronous accesses to the video content. When the server starts a new batching session, it need not wait any late peers. The early peers in a batching session obtain the video content and become the substitute video sources. Late peers can make up the missing content using patching from the early ones. Meanwhile, patching improves the system flexibility. As observed [18], 37 percent of all the VOD users watch a video for less than 5 minutes. VOVO provides abundant backup stream sources for patching by adopting the hybrid caching strategy, where all the peers keep both the initial 5 minutes and the latest 5 minutes of the video played. Any late peer can instantly find patching sources and make up the missing part immediately after join, no matter if it starts from the beginning or any other offset of the video. Because the patching sources are selected randomly, load balance is kept among the peers.
3. To work in concert with collaborative prefetching. Neighboring peers have very similar playback progresses such that their prefetching sets are similar, too. This feature results in a mutual-complementary effect for the neighboring peers to execute collaborative prefetching. Specifically, even if a peer has not prefetched a requested segment, with high probability, this segment can be located in the prefetched content on its neighbors, as we analyze in Section 5.2.

### 5.2 Efficiency of Prefetching

We define two types of hit ratios for VCR interactivities. *HR1* is the percentage of requests satisfied by the segments prefetched on the current peer. *HR2* is the percentage of requests satisfied by the segments prefetched on the neighboring peers. In Section 4.4, it is illuminated that higher hit ratios lead to shorter response latencies of VCR interactivities. Here, we theoretically compare the hit ratios produced by random prefetching and the scheme proposed in VOVO. Similar results can be obtained from the comparisons with other prefetching strategies. Due to the limit of pages, we just use random prefetching as a compared target.

Let us consider the scenario when a particular VCR control occurs. Without loss of generality, we assume a peer can prefetch $d$ unique segments, while a group of neighboring peers can prefetch $D$ unique segments before the VCR control occurs. Actually, the value of $D$ might be slightly variational with different prefetching strategies. But, the minor difference has less effect in the comparison, so we just treat it as a constant value. $V = \{\text{all the segments possibly to be requested by the VCR control}\}$. $V_i$ is the prefetching set of peer $i$. Let $p_h$ be the probability that the requested segment belongs to the set $V_i$.

For the collaborative prefetching proposed in VOVO, we have

$$HR1 = \frac{p_h \times d}{|V_i|}, \quad HR1 + HR2 = \frac{p_h \times D}{|V_i|}. \tag{6}$$

For random prefetching, we have

$$HR1' = \frac{d}{|V|}, \quad HR1' + HR2' = \frac{D}{|V|}. \tag{7}$$

According to the aforementioned observation results and the associations inside the video, $p_h$ is close to 1. Moreover, $|V_i| \in |V|$, while $|V_i| \ll |V|$. Thus, we have

$$\frac{HR1}{HR1'} = \frac{HR2}{HR2'} = \frac{p_h \times |V|}{|V_i|} \approx \frac{|V|}{|V_i|}. \tag{8}$$

Because $|V_i| \ll |V|$, this result clearly shows the advantage of prefetching scheme in VOVO, which is aware of the associations inside a video. Moreover, based on the collaborative prefetching (also proved by the trace-driven experiments in Section 6), a group of neighboring peers are able to prefetch most of the segments in their *prefetching set*. In other words, $D \approx |V_i|$. From (6), we have $HR1 + HR2 \approx 1$, which means VOVO is able to provide near-optimal hit ratios in the scope of neighboring peers.

It is worth noticing that the scheme of association rule mining in VOVO is applicable for many other P2P VOD frameworks. The mining process is based on the collective behavior of VOD clients. As long as a peer has an effective way to record and share the playback history, it is obviously realizable for the peer to mine the association rules inside a video.

## 5.3 Miscellaneous

In this section, we analyze miscellaneous costs and efficiency of VOVO.

***Memory cost***. The memory cost on a peer in VOVO mainly consists of three parts. The first part is the memory cost to cache the initial 5 minutes and the latest 5 minutes of video played. The second part is the memory cost of prefetching buffer, usually not more than three segments, i.e., 3 minutes. The third part is the memory cost to store the list of records, which are collected through gossips with other peers. Suppose the bit rate of the video is 700 kilobytes per second (Kbps), which is generally playback of high quality. The memory cost of all cached video content will be at most $(700\,\text{Kbps}/8) \times 60\,\text{seconds} \times 13\,\text{minutes} \approx 68\,\text{Mbytes}$. The size of an entry in the list is determined by the size of a playback record. As we know, the playback record is a string of segment indices. Each index in the string costs 1 byte. As we observed in the real VOD data set [24], no video is longer than 120 minutes. Since a VOD user seldom stays in the system longer than the length of the video, we assume the playback record always contains not more than 120 characters. Therefore, the size of a single record is less than $1\,\text{byte} \times 120 \approx 0.12\,\text{Kbytes}$. The size of the list is at most 1.2 Mbytes even if a peer maintains at most 10,000 records. In total, the memory cost of a peer in VOVO is less than 70 Mbytes. It is obviously acceptable for normal PCs.

***Propagation delay***. The propagation delay is defined as the time needed to propagate the state information of a peer throughout the system. Let $m$ be the number of neighbors selected for gossip in each hop. Let $t$ be the average transmission delay of gossip in one hop. Let $N$ be the system size. Then, the expected propagation delay is $T = t \times log_m N$. When $m = 2$, $t = 3\,\text{seconds}$, and $N = 10^6$, we have $T = 60\,\text{seconds}$. In other words, setting the period of gossip as one minute sufficiently satisfies the timing requirement of state propagation.

***Traffic cost***. The traffic cost is defined as the cost for a peer to update all the entries in its local list. Note that the size of the *state message* generated in each period of gossip is the sum of the following items: the IP address, the incremental playback record, and the time stamp. That is, $4\,\text{bytes} + 1\,\text{byte} + 1\,\text{byte} = 6\,\text{bytes}$. To update at most 10,000 entries in the list, the average traffic cost per peer in each period of gossip is $6\,\text{bytes} \times 10^4/10^3 = 60\,\text{Kbytes}$. Since a period of gossip lasts for 1 minute, the traffic cost on a peer is 1 Kbps, which is almost negligible compared to the traffic of video streaming.

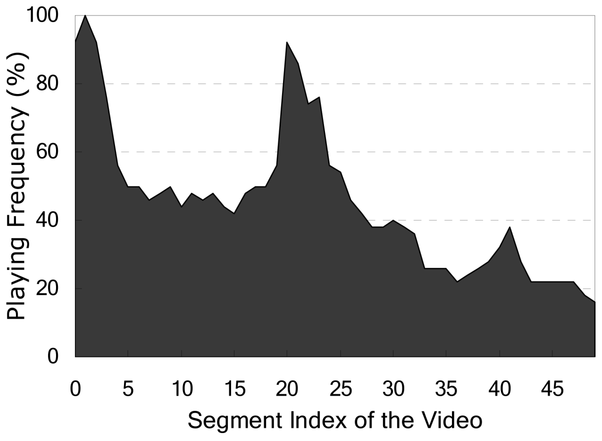

Fig. 8. Playing frequency of the video.

***Overhead of mining***. We have analyzed the memory cost on a peer to store the video content and the list of playback record. As shown in Section 4.3, there is hardly any additional memory cost in mining the association rules, except a few variables used to count and filter the association rules. And, it is well known that the temporal complexity of the Knuth-Morris-Pratt algorithm is $O(n + m)$, where $n$ is the length of main string, and $m$ is the length of the substring to search for. In the case of association rule mining in VOVO, $m = 3$ while $n$ is the length of a playback record, which usually varies in [1, 120]. Moreover, we observe a real VOD data set [24] and find that three-quarters of all clients play not more than 50 percent of a video, as means that the values of $n$ must be even smaller in practical scenarios. Therefore, we may regard association rule mining in VOVO as a procedure with constant temporal complexity. Actually data mining through several thousands of records is trivial to the powerful computing capability of PCs. It takes less than 1 second in our experiments to execute the corresponding procedures.

## 6 Performance Evaluation

### 6.1 Experimental Methodology

For comparison, we simulate a system called SBR which adopts simple batching and random prefetching strategy. We compare VOVO with SBR in groups of experiments. We take a movie of 50 minutes as a test video and simulate 1,000 peers. The accumulated playing frequency of segments in the video conforms to the distribution in Fig. 8, which is collected from a real deployed VOD system [24]. Then, we generate a playback record for each of the 1,000 peers. According to the record, the playback procedure of a peer can be determined. For example, a peer with playback record (1, 2, 3, 6, 7, 8, 15, 16, 17, 18, 19, 20, 45, 30, 31, 32, 33, 34, 35, 45, 46, 47, 48, 49, 50) has a lifetime of 25 minutes and takes five VCR controls in total. Fig. 9 shows the numbers of VCR controls taken by

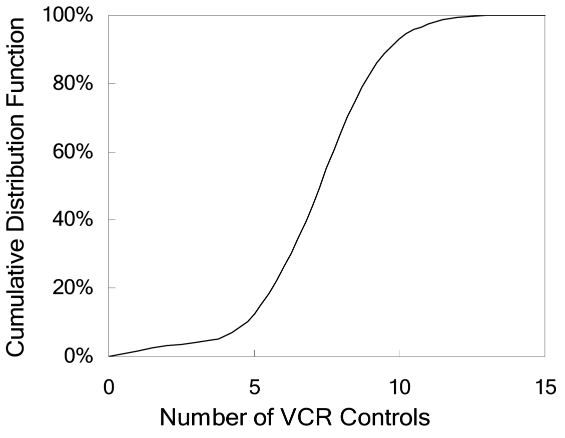

Fig. 9. VCR behavior of 1,000 peers.

1,000 peers with a curve of cumulative distribution. We assume that the aggregated join rates of peers conform to a Poisson process with parameter $\lambda$. In the order of joining, peers are assigned with 0-based IDs. The session width of batching is set at 1 minute. For the source server, the allocated bandwidth per session is denoted by the parameter $\theta$. We set the full source rate of the video as $S = 800\ \mathrm{Kbps}$ to represent high quality videos. The outgoing bandwidth capacity of the source server is 200 times the full source rate, i.e., $200S = 160\ \mathrm{Mbps}$. The total available downloading bandwidth of each peer is randomly distributed in $[1.5S, 5S]$. The end-to-end bandwidth for transmission is uniformly equal to $S$. The playout buffer size is $30\ \mathrm{seconds} \times S$. In other words, a peer has to fill the buffer with 30 seconds video before it starts playback. Without loss of generality, the time needed to locate a segment and build a connection varies from 5 to 15 seconds.

## 6.2 Experimental Results

**Hit ratio.** Let $\lambda = 20$, $\theta = 6S/\mathrm{minutes}$. Fig. 10 plots the sums of *HR1* and *HR2*, the two hit ratios defined in Section 5.2. We can see that the accumulated hit ratios (sum of *HR1* and *HR2*) of most peers are over 70 percent. This figure implies the conclusion: a peer cannot prefetch all the requested

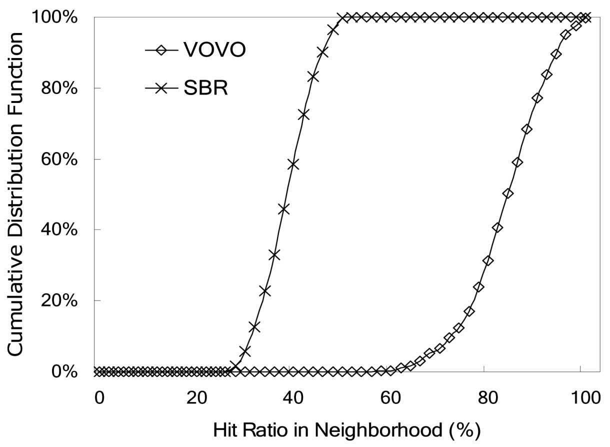

Fig. 11. Comparison of hit ratio.

segments ahead of the occurrence of VCR controls; the majority of the requests of VCR controls, however, can be resolved within its neighborhood using the collaborative prefetching strategy. Fig. 11 plots the *HR2* of VOVO and SBR. We can see VOVO improves *HR2* by 90 percent-120 percent, indicating the high accuracy and efficiency of the collaborative prefetching strategy.

**Impact of join rate.** Let $\theta = 6S/\mathrm{minutes}$, we tune the value of $\lambda$ to measure the impact of join rates. Fig. 12 shows the server stress over time with different join rates. The server stress is denoted as percentage to the total bandwidth capacity of the source server.

As we see from the curves, there is not apparent correlation between the server stress and the join rates. During the first 25 minutes, the server stress presents nearly linear increase as peers join in. After the 30th minute, the server stress keeps steady at around 70 percent. This fact should be explained as follows: during the initial stage of the experiment, the peers join the system in succession and consume the allocated bandwidth from the server. Early peers obtain and cache the video content in their buffers. Afterwards, they become the substitute video sources and alleviate the server stress, despite that the number of peers keeps increasing.

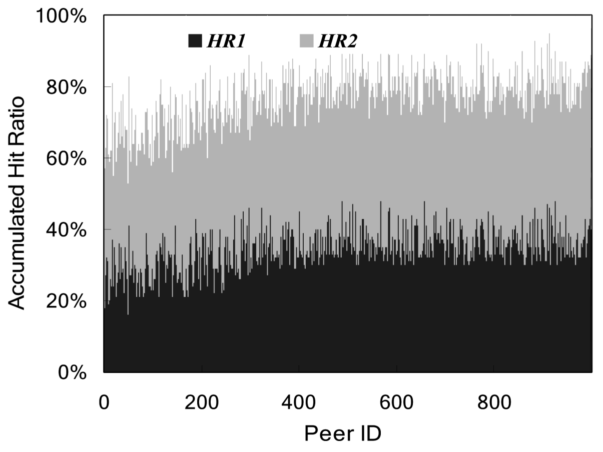

Fig. 10. Accumulated hit ratio.

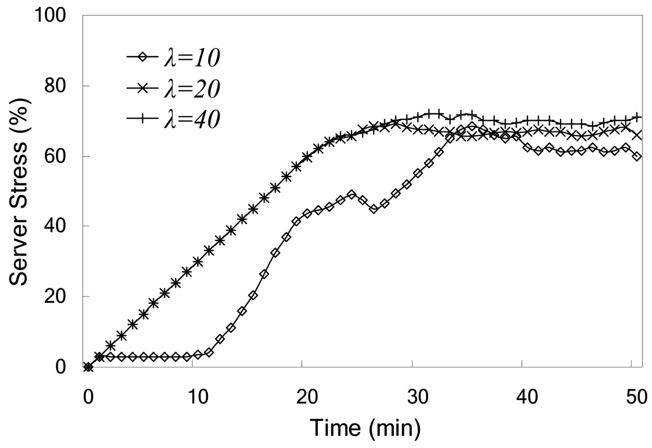

Fig. 12. Impact of join rate.

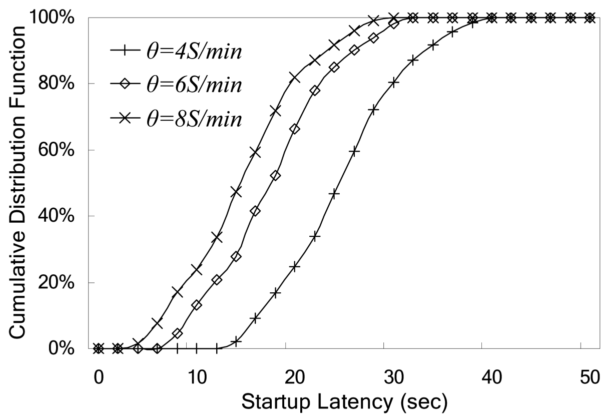

Fig. 13. Impact of allocated bandwidth per session (1).

Note that for any of these experiments, VOVO can serve much more concurrent users than the capacity of the source server. For example, when $\lambda = 20$, the observed peak number of concurrent users is 531; when $\lambda = 40$, the observed peak number of concurrent users is 1,248; but the server can directly serve 200 clients at most.

**Impact of allocated bandwidth per session.** Let $\lambda = 20$, we tune the value of $\theta$ to measure the impact of allocated bandwidth per session.

Fig. 13 plots the cumulative distribution of start-up latencies with different settings of $\theta$. As can be seen from the figure, VOVO offers shorter start-up latencies as the allocated bandwidth per session increases.

Meanwhile, Fig. 14 shows the server stress over time. We find the maximum server stress is determined by $\theta$. First, the server stress increases as the allocated bandwidth per session increases. This is because more peers obtain the video content directly from the server when the allocated bandwidth per session increases. Second, with different settings of $\theta$, the server stress presents similar increasing trends over time. Third, the server cannot be overloaded until $\theta = 9S/\text{minutes}$. According to the results in Fig. 13, it is viable to allocate bandwidth as much as possible for each session so that the start-up latencies are expected to be even shorter.

**Performance comparison.** In this section, we compare VOVO with SBR in start-up latencies and response latencies.

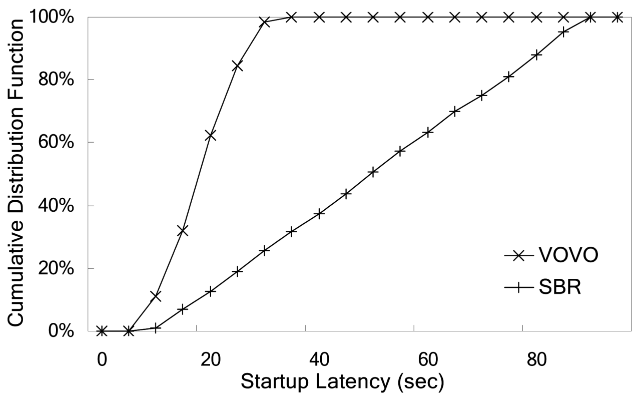

Fig. 15. Comparison of start-up latencies.

Related parameters are set as $\lambda = 20$, $\theta = 6S/\text{minutes}$. The corresponding server stress is shown in Fig. 14. Since VOVO adopts batching to allocate outgoing bandwidth on the server, the server stress is almost same with that of SBR in various scenarios.

Fig. 15 compares the cumulative distribution of start-up latencies for peers in VOVO and SBR. In SBR, the start-up latencies are distributed uniformly from 10 seconds to 90 seconds with an average of 50 seconds. But, the start-up latencies in VOVO are typically ranging from 8 seconds to 30 seconds with an average of 18 seconds. In comparison, VOVO improves the start-up speed by 10 percent-200 percent. Moreover, VOVO offers more equitable start-up speed for all the peers.

With the same parameter settings, Fig. 16 compares VOVO and SBR in terms of response latencies in VCR interactivities. The response latencies in VCR interactivities are improved by 30 percent-140 percent, because VOVO achieves higher accuracy and efficiency by using the collaborative prefetching strategy.

We plot the response latencies of all the peers in VOVO in Fig. 17. Interestingly, we find that the response latencies of early peers are a bit longer than that of the late peers. The reason lies in two aspects: First, early peers cannot find abundant source to download when they take VCR controls. Second, the precision and accuracy of predictions depends on the size of data set collected through gossips.

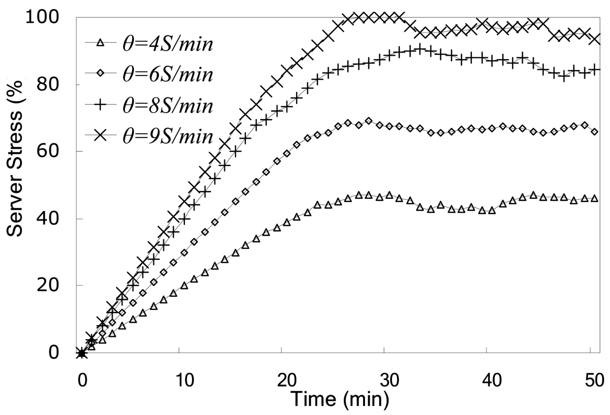

Fig. 14. Impact of allocated bandwidth per session (2).

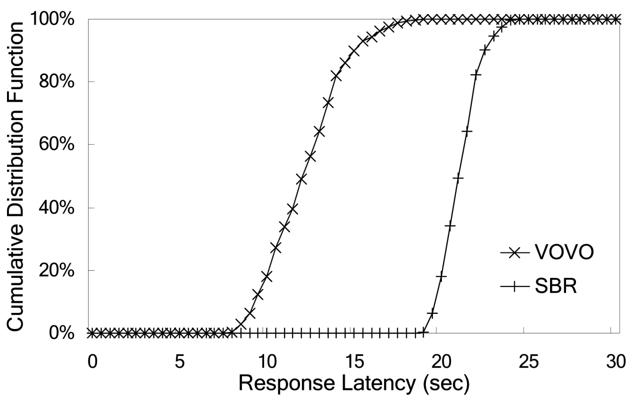

Fig. 16. Comparison of response latencies.

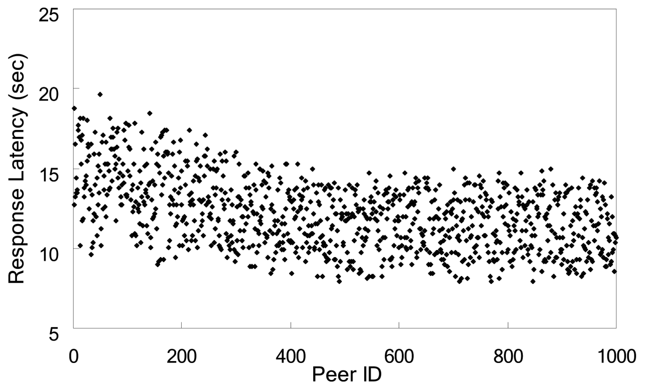

Fig. 17. Response latencies in VOVO.

## 7 Conclusions

Enabling large-scale VOD service in wide area Internet is crucial for many commercial applications. In order to provide a VCR-oriented VOD service in large scale P2P networks, we propose VOVO scheme. VOVO combines batching and patching as the basic service architecture and reinforces it with a hybrid caching strategy. Based on the observations on user behavior and VCR interactivities, we adopt the technique of association rule mining to exploit the associations within videos. The segments requested in VCR interactivities are thus accurately predicted according to the information collected through gossips among peers. Moreover, a collaborative prefetching strategy is designed to optimize the resource distribution on the neighboring peers. We evaluate VOVO through comprehensive simulations and compare it with previous schemes. In the next step, we are going to work on the in-session topology optimization with VOVO. The incentive mechanisms and economic issues in P2P VOD systems are also being studied.

## Acknowledgments

The authors thank Hongliang Yu of Tsinghua University for providing a valuable data set, covering 219 days and more than 150,000 users in a real VOD system [24] deployed by China Telecom. This research was supported in part by the National High Technology Research and Development Program of China (863 Program) under Grant 2007AA01Z180, NSF China Key Project 60736016, NSF China Grant 60573140, Hong Kong RGC Grants HKUST6169/07E and N_HKUST614/07, and HKUST Nansha Research Fund NRC06/07.EG01.

## References

[1] Y. Chu, S.G. Rao, and H. Zhang, "A Case for End System Multicast," *Proc. ACM SIGMETRICS,* 2000.
[2] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," *Proc. ACM SIGCOMM,* 2004.
[3] H. Shen and C.-Z. Xu, "Locality-Aware and Churn-Resilient Load-Balancing Algorithms in Structured Peer-to-Peer Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, 2007.
[4] X. Zhang, J. Liu, B. Li, and T.S.P. Yum, "Donet/Coolstreaming: A Datadriven Overlay Network for Live Media Streaming," *Proc. IEEE INFOCOM,* 2005.
[5] Y. Huang, T.T.J. Fu, D.M. Chiu, J.C.S. Lui, and C. Huang, "Challenges, Design, and Analysis of a Large-Scale P2P VoD System," *Proc. ACM SIGCOMM,* 2008.
[6] C. Huang, J. Li, and K.W. Ross, "Can Internet Video-on-Demand Be Profitable," *Proc. ACM SIGCOMM,* 2007.
[7] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Policies for an On-Demand Video Server with Batching," *Proc. Second ACM Int'l Conf. Multimedia (Multimedia),* 1994.
[8] T. Do, K. Hua, and M. Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," *Proc. IEEE Int'l Conf. Comm. (ICC),* 2004.
[9] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-Peer Patching Scheme for VoD Service," *Proc. 12th Int'l Conf. World Wide Web (WWW),* 2003.
[10] S. Sheu, K.A. Hua, and W. Tavanapong, "Chaining: A Generalized Batching Technique for Video-on-Demand Systems," *Proc. Int'l Conf. Multimedia Computing and Systems (ICMCS),* 1997.
[11] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Is High-Quality VoD Feasible Using P2P Swarming?" *Proc. 16th Int'l Conf. World Wide Web (WWW),* 2007.
[12] A. Hu, "Video-on-Demand Broadcasting Protocols: A Comprehensive Study," *Proc. IEEE INFOCOM,* 2001.
[13] K. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services," *Proc. Sixth ACM Int'l Conf. Multimedia (Multimedia),* 1998.
[14] S. Sen, L. Gao, J. Rexford, and D. Towsley, "Optimal Patching Schemes for Efficient Multimedia Streaming," *Proc. 11th Int'l Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV),* 1999.
[15] A. Mahanti, D. Eager, M. Vernon, and D. Sundaram-Stukel, "Scalable On-Demand Media Streaming with Packet Loss Recovery," *Proc. ACM SIGCOMM,* 2001.
[16] L. Pinho and C. Amorim, "Assessing the Efficiency of Stream Reuse Techniques in P2P Video-on-Demand Systems," *J. Network and Computer Applications,* vol. 29, pp. 25-45, 2006.
[17] S. Rollins and K. Almeroth, "Pixie: A Jukebox Architecture to Support Efficient Peer Content Exchange," *Proc. 10th ACM Int'l Conf. Multimedia (Multimedia '02),* Dec. 2002.
[18] H. Yu, D. Zheng, B. Zhao, and W. Zheng, "Understanding User Behavior in Large-Scale Video-on-Demand Systems," *Proc. EuroSys Conf.,* 2006.
[19] C. Zheng, G. Shen, and S. Li, "Distributed Prefetching Scheme for Random Seek Support in Peer-to-Peer Streaming Applications," *Proc. Workshop Advances in Peer-to-Peer Multimedia Streaming,* 2005.
[20] T. Menzies and Y. Hu, "Data Mining for Very Busy People," *Computer,* vol. 36, pp. 22-29, 2003.
[21] C. Huang and T. Hsu, "A User-Aware Prefetching Mechanism for Video Streaming," *Proc. 12th Int'l Conf. World Wide Web (WWW),* 2003.
[22] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEshed-Based Streaming System," *Proc. IEEE INFOCOM,* 2007.
[23] D. Knuth, J.H. Morris, and V. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Computing,* vol. 6, pp. 323-350, 1977.
[24] PowerInfo Co., Ltd., http://www.sjdd.com.cn/english/englishindex.htm, 2007.

**Yuan He** received the BE degree from the Department of Computer Science and Technology, University of Science and Technology of China, in 2003 and the ME degree from the Institute of Software, Chinese Academy of Sciences, in 2006. He is currently a PhD student in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, supervised by Dr. Yunhao Liu. His research interests include peer-to-peer computing, sensor networks, and pervasive computing. He is a student member of the IEEE and the IEEE Computer Society.

**Yunhao Liu** received the BS degree from the Automation Department, Tsinghua University, China, in 1995, the MA degree from Beijing Foreign Studies University, China, in 1997, and the MS and the PhD degrees in computer science and engineering from Michigan State University in 2003 and 2004, respectively. He is currently with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He is also an adjunct professor of Xi'an Jiaotong University, Jilin University, and the Ocean University of China. His research interests include wireless sensor network, peer-to-peer computing, and pervasive computing. He is a senior member of the IEEE and a member of the ACM. He received the Grand Prize of Hong Kong ICT Best Innovation and Research Award in 2007.