Deploying P2P Techniques to Provide VoD Services under Scheduled Video Delivery Paradigm

Chow-Sing Lin and Yi-Chi Cheng
Department of Information Management
Southern Taiwan University of Technology
Email:{mikelin, yasashi}@msrg.mis.stut.edu.tw

Abstract— **In the past, the discussion of VoD services mainly focused on the on-demand paradigm, in which users are assumed and restricted to request a video for instantly viewing. This paradigm causes the inefficient utilization of the system resources and high rejection probability. In this paper, we address the problem of providing VoD services over peer-to-peer networks under Scheduled Video Delivery (SVD) paradigm, in which users are allowed to issue requests for instantly viewing or for later viewing. We propose an administrative organization to manage the relationship among peers, atop which a proposed novel delivery protocol is applied to take full advantage of the resources contributed by peers and the server. Due to the dynamics inherent in P2P networks, the fast failure recovery is critical to an approach of providing VoD services over P2P networks. Our proposed approach can make the users suffering failure recover quickly.**

Index Terms — **VoD, SVD, P2P**

I. Introduction

In the past, VoD had attracted considerable attention from the academic community. However, most of researches on providing VoD services were focused on the on-demand paradigm, which assumes that users do not issue their requests until they want to view a video immediately. Consequently, users are not permitted to issue a planned-ahead request for later viewing and the server tends to be swamped by a large number of requests in peak times and idle in off-peak times. Besides, people are used to planning ahead to do something, surely including when to watch the desired videos. Owing to the inherent inappropriate assumption of the preceding paradigm, Wu et al [10] proposed a new video delivery paradigm, named Scheduled Video Delivery (SVD). In SVD, users are allowed to issue a request either for instantly viewing or for later viewing. Thus SVD can be considered as a generalization of the traditional VoD paradigm.

In addition to SVD, Wu et al [10] also proposed two approaches to address the problem of providing VoD services under SVD, namely Modified Earliest Deadline First (MEDF) and Least Popularity First (LPF). In MEDF, the requests for the same video are grouped and served as a unit. All the groups are served in ascending order of deadline. The smaller the deadline of a group is, the earlier a group is served. The output of MEDF is a schedule, which specifies which video segments will be delivered for which groups on which channels at what time. MEDF can be regarded as a primitive approach on SVD since it, as its name suggests, concentrates only on how to meet the deadlines of requests and makes no attempt to incorporate more requests into a group to make multicast more efficient. Thus LPF was proposed as an improvement on MEDF. LPF first applies MEDF to obtain an initial schedule. Afterwards, according to the initial schedule, LPF postpones the groups requesting more popular videos as late as possible and advances the groups requesting less popular videos. After this appropriate adjustment, a final schedule is produced, in which the groups requesting popular videos are postponed and expected to incorporate subsequent requests because of the high popularity of their required videos.

Although LPF outperforms MEDF by postponing the groups requesting popular videos to boost the efficiency of multicast, it remains sharing some drawbacks with MEDF. First, unlike Patching [6], both MEDF and LPF specify that a late request cannot join a group that has started their download. As a result, the server may have to retransmit a whole duplicate video specially for a late request, which causes the inefficient utilization of channels. Secondly, neither MEDF nor LPF takes advantage of clients' capability of downloading simultaneously from multiple channels, which has been widely deployed in broadcast and multicast such as Skyscraper [7], Client-Centric Approach [8], and Patching [6]. Accordingly, even though the server has more than one free channel, it still dedicates only one channel to serve a group and has other available channels sitting idle. Finally, both

This work was partially supported by National Science Council of Taiwan, R.O.C. under contract NSC 93-2213-E-218-028.

MEDF and LPF pose an unreasonable prerequisite for their efficient work. The prerequisite requires that all the videos supplied by the server have to be of equal length. If it is not fulfilled, requests from users may be rejected mistakenly even though the server has sufficient available channels to serve them.

In [2], we proposed an approach to efficiently deal with the problem of providing VoD services under SVD, namely Multi-Channel Multicast Delivery (MCMD). MCMD, like Patching [6], adopts dynamic multicast, which enables a late request to join the early requests to get its needed video segments. Besides, MCMD also takes advantage of clients' capability of multi-channel downloading. The experimental results show that MCMD achieves better performance than MEDF and LPF.

Recently, peer-to-peer (P2P) networks [12][11] have emerged as a promising solution to providing VoD services [15][3][4][9]. In a P2P network, users not only consume the resources from the server and network but also contribute their own resources such as storage capacity and out-bound bandwidth to the system. Therefore, the scalability of the system can be greatly improved. However, a P2P solution also brings some challenges to tackle, one of which is high dynamics [13][14]. In a P2P network, users can join and leave at any time without any restriction. Consequently users may be suspended from downloading or viewing their required videos because of an abrupt departure of their parents. The high dynamics of P2P networks reveal the importance of efficient failure recovery.

In this paper, we intend to augment MCMD in the context of peer-to-peer networks, and thus propose a novel P2P scheme, P2Deliver, as an extension to MCMD. The combination of MCMD and P2Deliver is named P2MCMD. The notion behind P2MCMD is that clients are served by the server as the last resort if and only if they can not acquire their required video segments from other clients. Specifically, on the arrival of a newcomer, P2Deliver searches for and assigns an appropriate parent to the newcomer. The parent then provides the newcomer with the needed video segments as many as possible. Once a client can not get any more from its parent or if no appropriate parent is found, it then resorts to the server via MCMD to make up the video it requests. P2Deliver not only can efficiently take advantage of the resources contributed by peers but also make the peers quickly recover from failure.

The rest of the paper is organized as follows. In section 2, we give a preliminary introduce to SVD and outline MCMD to make the paper self-contained. The proposed method is described in section 3. In section 4, we present our simulation study. Finally, our concluding remarks are given in section 5.

II. Preliminary

In this section, we present some basic notions about SVD and outline our previous work, MCMD, as the groundwork to facilitate the subsequent discussion.

A. SVD

In SVD, each client is assumed to have sufficient storage space to cache the whole video it requests, which is necessary in that a client may complete the download of its required video before viewing. This assumption is not absurd since a typical hard disk nowadays can afford to accommodate a few videos, each of several hundreds MBytes. Furthermore, the adequate storage space enables some VCR functions such as fast rewind and pause. In addition, SVD assumes that a request, R_i, from a client comprises three arguments:

1) $o(R_i)$ indicates the video object R_i requires.
2) $a(R_i)$ represents the time when R_i arrives.
3) $s(R_i)$ specifies the time when R_i starts to view.

The time interval, $s(R_i)$ - $a(R_i)$, is referred to as plan-ahead time. A request with a zero plan-ahead time, like a request in the traditional on-demand paradigm, is for instantly viewing while one with non-zero plan-ahead time is for later viewing.

Suppose that the server provides m video objects for clients to choose from. Then each video object, O_i, is assigned a number in [1, m] to indicate its relative popularity, denoted as $p(O_i)$. The smaller the number is, the more popular the video is. Most requests tend to require a video with a small $p(O_i)$. The video with $p(O_i)$ = 1 has the highest probability to be picked by a new request. Besides, every video is fragmented into a number of segments, each of equal size. For example, video i may be fragmented into k segments, denoted as $v_{i,1}$, $v_{i,2}$, ..., $v_{i,k}$. Sequential transmission of video segments is not necessary but it is required that each segment must be delivered by the time the client starts to consume it.

In this paper, we follow the convention on VoD. Most of the out-bound bandwidth of the server and clients is organized into a set of logical channels, each capable of transmitting a video at the playback rate. The remaining bandwidth is used for control messages. To distinguish the channels of clients from those of the server, we coin a word, $links$, for channels of clients, and the total number of available links of a client is denoted as N_{link}.

B. MCMD

In MCMD, we introduce the concept of $detain$, with which the server can precisely evaluate the degree of urgency of a request. The detain of a request, R_i, is defined as:

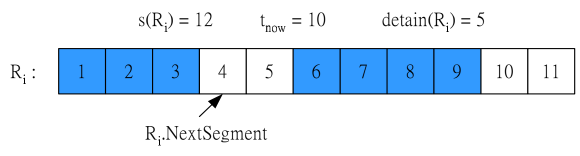


Fig. 1. An example for detain value

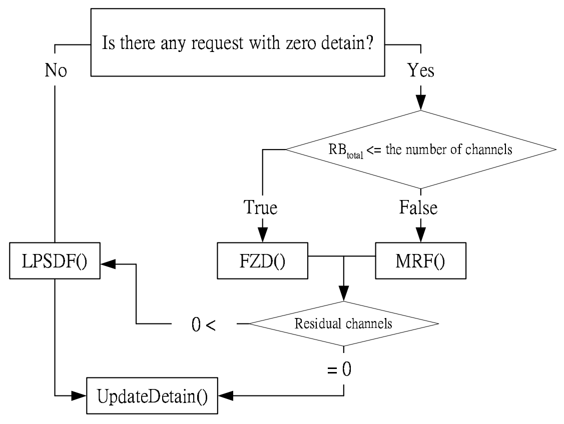

Is there any request with zero detain?
No
Yes
RB_total <= the number of channels
True
False
LPSDF()
FZD()
MRF()
0 <
Residual channels
= 0
UpdateDetain()

Fig. 2. The decision procedure of MCMD

$$detain(R_i) = s(R_i) - t_{now} + R_i.NextSegment - 1 \quad (1)$$

In (1), t_{now} is the current time, and $R_i.NextSegment$ indicates the first segment, from the beginning of $o(R_i)$, that R_i has not downloaded. Figure 1 shows an example, where R_i specifies that the user will start to view the required video at time slot 12, and the current time is time slot 10. The array of squares indicates that the video required by R_i is fragmented into 11 segments and R_i has acquired the shaded segments, $v_{o(R_i),1}$, $v_{o(R_i),2}$, $v_{o(R_i),3}$, $v_{o(R_i),6}$, $v_{o(R_i),7}$, $v_{o(R_i),8}$, $v_{o(R_i),9}$. Therefore, the first segment, from the beginning of $o(R_i)$, that R_i hasn't downloaded is $v_{o(R_i),4}$. Using (1), we can calculate that $detain(R_i)$ = 12 - 10 + 4 - 1 = 5 at time slot 10. The smaller $detain(R_i)$ is, the more urgent R_i is. If the download progress remains unchanged, $detain(R_i)$ will decrease to zero at time slot 15 and it means that R_i is starting to consume $R_i.NextSgment$ but $R_i.NextSgment$ has not be delivered. Thus the server has to serve R_i with $R_i.NextSgment$ immediately.

MCMD primarily consists of four modules: (1) *Least Popularity Shortest Detain First* (LPSDF) (2) *Finishing Zero Detain* (FZD) (3) *Most Requests First* (MRF) (4) *UpdateDetain*. At the beginning of every time slot, the server determines which module to invoke according to the details of the requests currently in the systems. The whole decision procedure is depicted in Figure 2.

If there does not exist any request with zero detain, then LPSDF module is invoked. LPSDF classifies all the requests into a number of groups according to their required videos and sorts the groups in ascending order of popularity of their required videos. Afterwards, beginning from the group requesting the least popular video, LPSDF identifies the request with the smallest detain in each group and dedicates a channel to deliver the segment indicated by $NextSegment$ of the request. The preceding procedure repeats until there is no free channel. Finally, UpdateDetain module will be invoked to update the details of all the requests according to (1). The details of the requests which are not scheduled to get any segments at this time slot will be decremented by one, and those of other requests will be unchanged or incremented by a number, depending on the new $NextSegment$.

If there is any request with zero detain, then the server first calculate the total number of segments the requests with zero detain are deficient in, denoted as RB_{total}. If RB_{total} is less than or equal to the number of channels on the server, then FZD module will be invoked. FZD dedicates as many channels as needed to delivering all the segments those requests with zero detain are deficient in, by which all the requests with zero detain will be satisfied in this time slot. If RB_{total} is greater than the number of channels on the server, then MRF module will be invoked. MRF first identifies all the segments indicated by $NextSegment$ of those requests with zero detain. For each identified segment, MRF calculates the total number of requests which lack this segment. Finally, each segment is dedicated a channel to be delivered in descending order of the number of requests. In other words, a segment needed by more requests is granted a higher priority to be delivered. The purpose of MRF is to give the requests with zero detain just the instantly needed segments in order to make those requests able to survive this time slot. After determining which channels will deliver which video segments, both FZD and MRF will invoke LPSDF to utilize the remaining channels, if any. Otherwise, UpdateDetain module will be invoked to update the details of all the requests.

III. **P2Deliver**

In this section, we describe our proposed P2P scheme, P2Deliver, as an extension to MCMD. We start with distinguishing between the roles MCMD and P2Deliver play. As mentioned before, P2MCMD, composed of P2Deliver and MCMD, is an approach to providing video-on-demand services over a peer-to-peer network. For a peer in a peer-to-peer network, there are two kinds of sources of its required video, the server and the early peers. When a new arrived peer requests a video to view, the requested video may have been transmitted by the server to a certain extent for the early peers requesting the same one. With dynamic multicast, the new arrived peer can join the early peers and download the residual part, called $base$, from the server. As to the missing initial part, called $patch$, the new arrived peer may get it from the early peers if the early peers have sufficient out-bound bandwidth. The two components of P2MCMD, namely MCMD and P2Deliver, are responsible for the foregoing two tasks respectively. At the beginning of every time slot, MCMD determines which video segments should be delivered on which channels according to the states of the requests from peers currently in the system. On the other hand, P2Deliver is charged to search for appropriate patch providers for a new arrived peer.

Briefly, it is the duty of MCMD to make peers get the bases of their required videos while it's the duty of P2Deliver to ensure peers timely patches. From another point of view, MCMD is responsible for the planning of the server's resources while P2Deliver is for the planning of peers' contributed resources.

For cooperation of MCMD and P2Deliver, the server has to maintain two separate queues, $TotalQ$ and $MCMDQ$. TotalQ accommodates all the nodes currently in the system, while MCMDQ accommodates only the nodes which have completed their download of patches or have no parent to supply them with their needed patches. When a new node arrives, the server keeps a copy of it in TotalQ. Once either of the following conditions occurs, the server will duplicate a copy of the node in MCMDQ.

1) The node completes its download of patch.
2) The node is abandoned by its parent and no other parent candidate has enough out-bound bandwidth to serve as its new parent.

MCMD takes into account only the nodes in MCMDQ when scheduling, while it considers the nodes in TotalQ when delivering.

Hereafter, the term "request" is used interchangeably with "peer" and "node" since a request comes from and thus represents a peer or node. The terms "parent" and "children" are used with the same meanings as applied for conventional trees.

A. Administrative Organization

For efficient work of P2Deliver, an administrative organization is used to represent the logical relationships among nodes and manage nodes currently in the system. The administrative organization is described as follows.

In P2Deliver, the peers requesting the identical video and arriving during the same time slot forms a $batch$ and are processed together at the beginning of the next time slot. In each batch, a peer is selected as the $batch\ leader$, and the others are called $batch\ nodes$. A batch leader is charged to maintain a list of records about the batch nodes, called a $node\ list$. As shown in Figure 3, the record about a batch node,

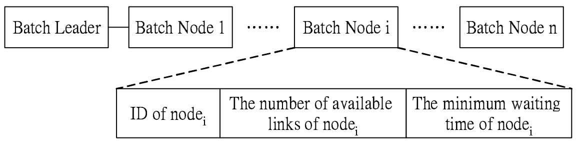

Batch Leader
Batch Node 1
······
Batch Node i
······
Batch Node n
ID of node_i
The number of available links of node_i
The minimum waiting time of node_i

Fig. 3. The structure of a record about a batch node

$node_i$, consists of three fields, including "identification", "the number of available links", and "the minimum waiting time", denoted as ID_{node_i}, $N_{link}^{node_i}$, and $T_{min}^{node_i}$ respectively. ID_{node_i} represents the address of $node_i$ and can be used to uniquely identify $node_i$; $N_{link}^{node_i}$ indicates how many links of $node_i$ remain available and therefore, indicates how many more children $node_i$ can adopt. $T_{min}^{node_i}$ indicates at least how much more time it will take for an occupied link of $node_i$ to be released, and therefore, how much more time a future child of $node_i$ has to wait until adopted by $node_i$ and beginning its download. The three equation below give formal definitions of the three fields respectively.

$$ID_{node_i} = the\ address\ of\ node_i \tag{2}$$

$$N_{link}^{node_i} = \frac{available\ bandwidth\ of\ node_i}{the\ playback\ rate\ of\ o(node_i)} \tag{3}$$

$$T_{\min}^{node_i} = \begin{cases} 0, & \text{for} N_{link}^{node_i} > 0 \\ \underset{node_j \in children(node_i)}{Min} (node_j.SkewPoint - node_j.NextSegment), & \text{for} N_{link}^{node_i} = 0 \end{cases} \tag{4}$$

In (4), $node_j.SkewPoint$ designates the first segment of the base for $node_j$. For example, $node_j.SkewPoint$ would be four if the first three segments of $o(node_j)$ have been delivered by the server when $node_j$ arrives. Therefore, $node_j.SkewPoint - node_j.NextSegment$ indicates for how much more time $node_j$ will occupy the link of its parent to complete the download of its patch. The minimum difference between $node_j.SkewPoint$ and $node_j.NextSegment$ determines the time for a first-released link of $node_i$ to be released, provided that $node_j$ is $node_i$'s child.

Similar to batch leaders, the server takes the responsibility of keeping a list of records about batches, called the $batch\ list$. As shown in Figure 4, the record about a batch, $batch_j$, comprises three fields, including "identification", "the number of available links", and "the minimum waiting time", denoted as ID_{batch_j}, $N_{link}^{batch_j}$, and $T_{min}^{batch_j}$ respectively. ID_{batch_j} identifies the batch leader of $batch_j$; $N_{link}^{batch_j}$ indicates the total number of available links in $batch_j$; $T_{min}^{batch_j}$

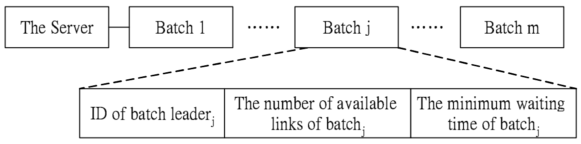

The Server
Batch 1
Batch j
Batch m
ID of batch leader$_j$
The number of available links of batch$_j$
The minimum waiting time of batch$_j$

Fig. 4. The structure of a record about a batch

indicates at least how much more time it will take for an occupied link of $batch_j$ to be released. It is evident that there exists a relation between a batch record and its corresponding node records, which can be deduced from the definitions above and expressed as follows.

$$\begin{aligned} ID_{batch_j} &= \{\ ID_{node_i} \mid node_i \in batch_j \\ &\quad and\ node_i\ is\ the\ leader\ \} \end{aligned} \tag{5}$$

$$N_{link}^{batch_j} = \sum_{node_i \in batch_j} \left(N_{link}^{node_i} \right) \tag{6}$$

$$T_{min}^{batch_j} = \underset{node_i \in batch_j}{Min} \left(T_{min}^{node_i} \right) \tag{7}$$

The rules below are applied when constructing an administrative organization.

1) Nodes requesting the identical video and arriving in the same time slot forms a batch, which implies that all the nodes in a batch achieve the same download progress.
2) Batches requesting the identical video constitute a $video\ session$.
3) In a video session, batches are numbered from one. A smaller batch number signifies an earlier arrival time and thus a higher layer in a video session of an administrative organization.

Figure 5(a) captures a snapshot of a video session of an administrative organization sometime, where the number inside each node indicates which batch the node belongs to. Under the rule (3) above, a batch arriving earlier is assigned a smaller batch number and located at a higher layer in the hierarchy. A forthcoming batch will constitute the forth batch and layer of the video session of the administrative organization. Figure 5(b) presents the same video session in another perspective.

B. Control Protocol

To keep the node list stored in each batch leader and the batch list stored in the server up-to-date, a control protocol is necessary. Whenever a change in the state of a node occurs, the control protocol is triggered. The control protocol can be considered in the following two cases.

Case 1: Adoption of a new child: Due to the adoption of a new child, the number of available links of the parent will be decreased by one, and the minimum waiting time of the parent may be changed. In such a case, the parent needs to re-evaluate N_{link}^{parent} and T_{min}^{parent} according to (3) and (4), provided that the new child set is the union of the old child set and the new child. Afterwards the parent informs its batch leader of updated N_{link}^{parent} and T_{min}^{parent}, according to which the batch leader will update the corresponding record in the node list and then, if necessary, instructs the server to do a similar update via an update message including the updated N_{link}^{batch} and T_{min}^{batch} evaluated through (6) and (7).

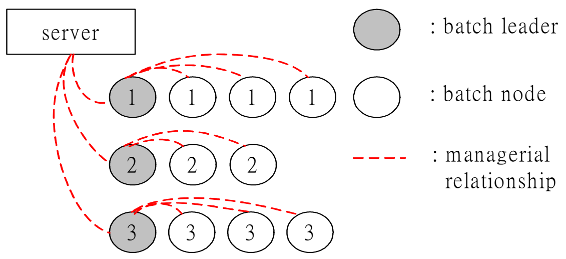
Case 2: Transfer or failure of an existing node: The transfer or failure of an existing node will release an occupied link and thus cause the number of available links of its parent increased by one. In addition, the minimum waiting time of the parent may be changed. As in case 1, the parent needs to re-evaluate N_{link}^{parent} and T_{min}^{parent}, provided that the new child set of the parent is the difference between the old child set and the transferred or departing child. The parent then notifies its batch leader to update the corresponding record in the node list by issuing an update message including updated N_{link}^{parent} and T_{min}^{parent}. If necessary, the batch leader in turn instructs the server to do a similar update on the corresponding record in the batch list by providing updated N_{link}^{batch} and T_{min}^{batch}.

C. Join Algorithm

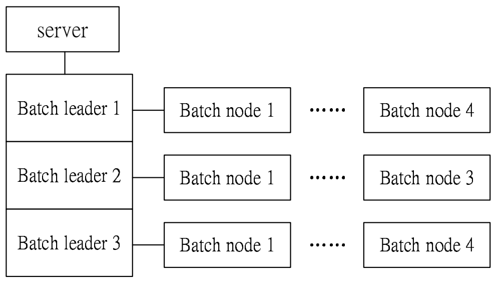
The purpose of the join algorithm is to designate an appropriate node as the patch provider, or the parent, for an orphaned node, like a new arrived node. For an orphaned node, n_{orph}, any node, n_j, possessing the following three properties can serve as the parent.

1) n_j requests the same video as n_{orph}, that is, $o(n_j) = o(n_{orph})$.
2) n_j is ahead of n_{orph} in the download progress of the patch, that is, $n_j.NextSegment > n_{orph}.NextSegment$.
3) n_j has sufficient out-bound bandwidth to accept n_{orph} as its child, that is, $N_{link}^{n_j} > 0$.

All the nodes possessing the preceding three properties are called the candidates for the parent of n_{orph} and denoted as $Parent_{candidate}(n_{orph})$ = $\{\ n_j \mid o(n_j) = o(n_{orph})$ and $n_j.NextSegment > n_{orph}.NextSegment$ and $N_{link}^{n_j} > 0\ \}$. From $Parent_{candidate}(n_{orph})$, we choose such a node as the parent of n_{orph} that the difference between the chosen node and n_{orph} in the download progress of the patch is minimal, and denote it as $Parent_{electee}(n_{orph})$. In other words, $Parent_{electee}(n_{orph}) = \{\ n_j \mid n_j \in Parent_{candidate}(n_{orph})$ and ($n_j.NextSegment$ - $n_{orph}.NextSegment$) is minimal $\}$. From the construction rules associated with an administrative organization, it can be deduced that $Parent_{candidate}(n_{orph})$ must exists in the



(a) A snapshot of the administrative organization



(b) Another perspective on the same administrative organization

Fig. 5. An example for the administrative organization

batches higher than the batch n_{orph} lies in. To locate $Parent_{electee}(n_{orph})$, we begin our search from the batch next higher than the batch n_{orph} lies in and proceed until $Parent_{electee}(n_{orph})$ is not empty or the first batch is reached. The search procedure is called $SearchForParent$ and described in Figure 6. As Figure 6 shows, the search procedure is divided into two phases. In the first phase from step 1 to step 7, the server searches the batch list for $Parent_{electee}(n_{orph})$, beginning from the batch next higher than the batch n_{orph} lies in, and then the request from n_{orph} will be forwarded to the batch leader of $Parent_{electee}(n_{orph})$ if $Parent_{electee}(n_{orph})$ is not empty. In the second phase from step 8 to step 11, on receipt of the request from n_{orph}, the batch leader searches its node list for a node, at least one link of which remains available, and then forwards the request of n_{orph} to the found node. Finally, n_{orph} can begin end-to-end communication with and download its patch from $Parent_{electee}(n_{orph})$.

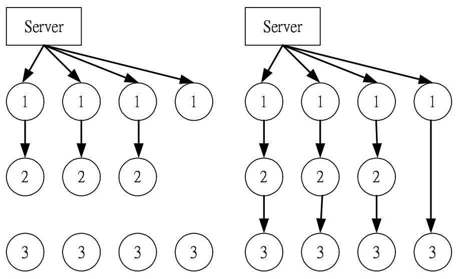
Figure 7 gives an example of how the $SearchForParent$ procedure works. Figure 7(a) shows that two batches constitute a certain video session, and the third batch just arrives. After applying the $SearchForParent$ procedure, each node in batch 3 has a $Parent_{electee}$ as its patch provider, and the delivery layout of the video session is showed in Figure 7(b). Following from the $SearchForParent$ procedure, it can be observed that the nodes in batch 2 are first considered and then the nodes in batch 1 in turn when designating a node as the parent of a node in batch 3.

If $Parent_{candidate}(n_{orph})$ (and thus $Parent_{electee}(n_{orph})$) is empty, it means that either the batch of orphaned nodes are the first batch in the video session, or all the links of the nodes in higher batches are occupied. In such a case, the server will check whether n_{orph} can wait until a link from a node in a higher batch is released. In other words, the server checks whether a link from a node in a higher batch will be released

```
$n_{orph}$: an orphaned node
$k$: the number of the batch $n_{orph}$ lies in

$SearchForParent(\ n_{orph},\ k\ )$

On the server side
1. i ← $k$ - 1
2. While ( i ≥ 1 and $N_{link}^{batch_i}$ = 0 )
3.       i ← i - 1
4. If i = 0
5.       $SearchForParent$ terminates with failure
         to find a parent
6. Else
7.       forwards the request from $n_{orph}$ to the
         batch leader, $ID_{batch_i}$

On the batch leader side
8. j ← 1
9. While ( $N_{link}^{node_j}$ = 0 )
10.      j ← j + 1
11. forwards the request from $n_{orph}$ to the batch
    node, $ID_{node_j}$
```

Fig. 6. The *SearchForParent* procedure

before $n_{orph}.detain$ decreases to zero. If there exists such a node, called $Parent_{future}(n_{orph})$, the server will find it according to the procedure, $SearchForFutureParent$, described in Figure 8. As Figure 8 shows, the server first compares the detain of n_{orph} with ”the minimum waiting time” of each batch in the batch list, beginning from the batch next higher than the batch n_{orph} belongs to, until a batch whose ”the minimum waiting time” is smaller than $n_{orph}.detain$ is found. Then the server forwards the request to the found batch leader and adds n_{orph} into the waiting queue, $WaitQ$. On receipt of the request, the batch leader will make a similar comparison to find $Parent_{future}(n_{orph})$



(a) Before applying the $SearchForParent$ procedure (b) After applying the $SearchForParent$ procedure

Fig. 7. An example for the *SearchForParent* procedure

in its node list and then forward the request to $Parent_{future}(n_{orph})$. As to n_{orph}, it will stay in $WaitQ$ till the corresponding link is released, and be enrolled as a member of the new batch just formed then.

The comprehensive join algorithm is shown in Figure 9. Any new arrived node is first put into the input queue, $InputQ$, and processed at the beginning of the next time slot. At the beginning of every time slot, if $InputQ$ is not empty, the server tries to find a parent immediately for each node in $InputQ$ according to the $SearchForParent$ procedure. If such a parent exists, then the new arrived node can begin its download of the patch. Otherwise, the server will determine whether a link from any node in a higher batch will be released before the detain of the new arrived node decreased to zero. If there exists such a node, the server will find it according to the $SearchForFutureParent$ procedure. Otherwise, the server will duplicate the new arrived node from $TotalQ$ to $MCMDQ$, which makes the new arrived node resort to the server to get its patch. Throughout the join procedure, it can be observed that a new node only needs to contact two nodes to tell whether there exists an appropriate parent and find out where it is, if any. Accordingly, a new node incurs low overheads in joining. The failure recovery procedure is similar to the join procedure in the sense that the purpose of a failure recovery procedure is to find a new parent for an abandoned node.

IV. Experimental Simulation

In this section, we evaluate the performance of P2MCMD under SVD paradigm through simulation experiments. The experiment environment is described as follows. Requests arriving to the system follow the Poisson distribution with a mean of λ (requests/minute). Interarrival times of requests are exponentially distributed with a mean of $\frac{1}{\lambda}$. The video required by a requests is determined by the Zipf distribution [5]. We set the skew factor α of the Zipf distribution to be 0.7, which is typical for

```
$n_{orph}$: an orphaned node
$k$: the number of the batch $n_{orph}$ lies in
$beginning$: the number of the batch from which
SearchForFutureParent begins. If beginning is
not specified, SearchForFutureParent will begin
from the batch k-1.

$SearchForFutureParent(\ n_{orph},\ k,\ beginning\ )$

On the server side
1. If $beginning \neq$ NULL
2.     i $\leftarrow$ $beginning$
3. Else
4.     i $\leftarrow$ k - 1
5. While ( i $\geq$ 1 and $T_{min}^{batch_i} > n_{orph}.detain$ )
6.     i $\leftarrow$ i - 1
7. If i = 0
8.     $SearchForFutureParent$ terminates with
       failure to find a future parent
9. forwards the request from $n_{orph}$ to the batch
   leader, $ID_{batch_i}$
10. add $n_{orph}$ into the waiting queue, WaitQ

On the batch leader side
11. j $\leftarrow$ 1
12. While ( $T_{min}^{node_j} > n_{orph}.detain$ )
13.     j $\leftarrow$ j + 1
14. forwards the request from $n_{orph}$ to the batch
    node, $ID_{node_j}$
```

Fig. 8. The *SearchForFutureParent* procedure

VoD applications [1]. We assume that the server has 100 channels to support up to 100 media streaming simultaneously and provides N videos, each of equal length L, for users to choose from. We use the normal distribution, N(μ, σ), to model the plan-ahead times of requests, and set a lower bound of $\mu - 3 * \sigma$ (minutes) and an upper bound of $\mu + 3 * \sigma$ (minutes) for the generated plan-ahead times. Any generated plan-ahead time falling outside the bounds would be rounded off to ($\mu - 3 * \sigma$) or ($\mu + 3 * \sigma$). Accordingly, users are restricted to issue requests at least ($\mu - 3 * \sigma$) and at most ($\mu + 3 * \sigma$) minutes early before viewing.

We carry out two experiments to examine the performance of P2MCMD by comparing it with MEDF, LPF and MCMD. In the first experiment, we evaluate the performance of P2MCMD in terms of the rejection rate. The parameters used for the first experiment is summarized in Table 1, and the corresponding results are shown in Figure 10. From Figure 10, it can be observed that MCMD evidently outperforms MEDF and LPF. For example, the rejection rates of MCMD,

```
$n$: the total number of batches currently in the
    video session

at the beginning of every time slot
1. If InputQ != empty
2.     create a new batch, $batch_{n+1}$
3.     $batch_{n+1}.leader \leftarrow$ InputQ[1]
4.     For each node, $n_k$, in InputQ
5.         apply $SearchForParent$( $n_k$, $n+1$ )
           to find a parent
8.         If $SearchForParent$ terminates with
           failure
9.             apply $SearchForFutureParent$( $n_k$,
               $n+1$ ) to find a future Parent
10.            If $SearchForFutureParent$
               terminates with failure
11.                duplicate $n_k$ from $TotalQ$ to
                   $MCMDQ$
```

Fig. 9. The join algorithm

MEDF and LPF are 32.29%, 80.63% and 83.18% respectively at the arrival rate of 100 (requests/min). MCMD performs 59.95% and 61.18% better than MEDF and LPF respectively. This great improvement in the rejection rate is due to the adoption of the dynamic multicast by MCMD. In MCMD, whenever the server is about to deliver a video segment, it identifies all the requests lacking the segment to be delivered, and groups them as a temporary multicast group. Conversely, MEDF and LPF specify that only the group triggering the delivery of a segment can receive the delivered segment. Therefore, in MCMD every channel can reach the maximum utilization. With the aid of P2Deliver, the performance of MCMD can be further enhanced. For example, the rejection rates of MCMD and P2MCMD are 32.29% and 26.39% respectively at the arrival rate of 100. P2MCMD gives a relative improvement of 18.27% over MCMD. Moreover, the improvement over MEDF and LPF can be boosted to 67.27% and 68.27% respectively. Such an improvement made by P2MCMD results from the contributions of peers. In P2MCMD, Early peers can serve as the parents of late peers and provide them with the video segments they lack, which can keep the server from retransmitting duplicate video segments and thereby achieves more efficient utilization of channels.

TABLE I

PARAMETERS USED FOR THE FIRST EXPERIMENT.

PARAMETERS	VALUE
Number of videos, N	500
Video length, L	120 (minutes)
Server bandwidth (channels)	100
Arrival rate, λ, (requests/minute)	0-100
Plan-ahead time, N(μ,σ)	$\mu = 180$, $\sigma = 60$
Skew factor, α	0.7

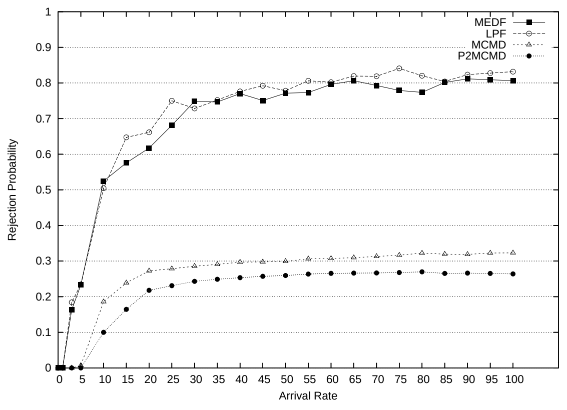


Fig. 10. The rejection rates of MEDF, LPF, MCMD and P2MCMD

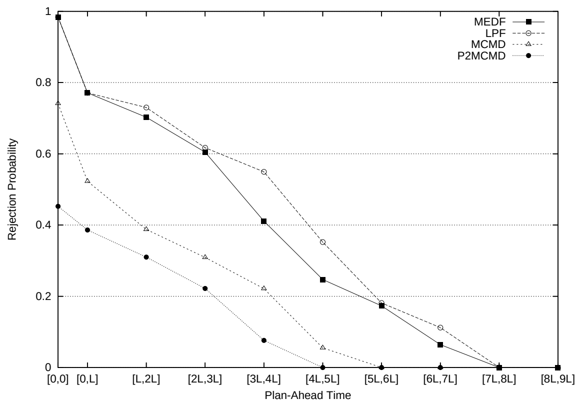


Fig. 11. The rejection rates of MEDF, LPF, MCMD and P2MCMD

In the second experiment, we evaluate the performance of P2MCMD in terms of the requirement of plan-ahead times for no rejection. The average arrival rate, λ, is fixed at 100 (requests/minute). The plan-ahead times are set to be 0 and then follow the normal distribution, N(μ, σ), where μ is varied from 60 to 1020 with a step of L = 120 and σ is fixed at 20. Accordingly, the plan-ahead times fall within the ranges: [0, 0] and [$a * L$, $(a+1) * L$] respectively, where a runs from 0 to 8. The other parameters remain unchanged as in the first experiment. The results are shown in Figure 11. It can be observed that MEDF and LPF experience the same rejection rates when the plan-ahead times of requests are not more than L. This result is consistent with the implications of LPF. In scheduling, LPF first applies MEDF to obtain an initial schedule and then modifies the initial schedule by postponing the groups with demand for more popular videos and advancing the groups with demand for less popular videos. If the room made in the schedule through the postponement of the aforementioned groups is not enough to accommodate the groups to be advanced, the modified schedule will remain the same as the initial groups and thus LPF will give the same performance as MEDF. Besides, Figure 11 reveals that the minimum requirements placed by P2MCMD, MCMD, MEDF and LPF on the plane-ahead times for no rejection are $[4L, 5L]$, $[5L, 6L]$, $[7L, 8L]$ and $[8L, 9L]$ respectively. P2MCMD demands the least plan-ahead times among the four approaches, which demonstrates the contribution from P2Deliver and the ability of P2MCMD to efficiently utilize the channels of the servers.

V. Conclusion

In this paper, we deal with the problem of providing VoD services under SVD paradigm, in which users are allowed to issue their requests for instantly viewing or later viewing. We propose a novel P2P scheme, P2Deliver, to augment our previous work. P2deliver includes an administrative organization to represent and manage the relationships among peers, a control protocol to keep the information stored in the server and peers up-to-date, a join algorithm. With the two-tier architecture, a new node or an orphaned node only contacts two node, the server and a batch leader, to determine whether there is an appropriate parent for it. Therefore, P2Deliver enables peers to join and recover quickly. The experimental results show that with the aid of P2deliver, P2MCMD outperforms the existing works on SVD in terms of the rejection rate and the requirement of plan-ahead times. Our future work is aimed to address the issue of incentive for sharing in P2P networks.

References

[1] A. Dan, D. Sitaram, and P. Shahabuddin, ”Dynamic Batching Policies for Video-on-Demand Storage Servers” Multimedia System, 4(3):112-121, June 1996.

[2] Chow-Sing Lin, Tzong-Yao Chang, and Fang-Zhi Yen, ”Perfirmance Study of Multi-Channel Multicast Delivery for Scheduled Videos,” IEEE APCCAS’04, Tainan, Taiwan, 2004.

[3] D. A. Tran, K. Hua, and T. Do, ”A peer-to-peer architecture for media streaming” IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks, January 2004.

[4] D. A. Tran, K. A. Hua, and T. T. Do, ”Zigzag: An efficient peer-to-peer scheme for media streaming” IEEE INFOCOM, 2003.

[5] G. Zipf, Human Behavior and the Priciple of Least ERort, Addison-Welsley, 1999.

[6] Kien A. Hua, Ying Cai, and Simon Sheu, ”Patching: a multicast technique for true video-on-demand services,” ACM Multimedia’98, Bristol, U.K., 1998, pp. 191-200.

[7] Kien A. Hua and Simon Sheu, ”Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems,” ACM SIGCOMM’97, Cannes, France, 1997, pp. 89-100.

[8] Kien A. Hua, Ying Cai, and Simon Sheu, ”Exploiting Client Bandwidth for More Efficient Video Broadcast,” IEEE IC3N’98, Lafayette, Louisiana, U.S.A, 1998, pp. 848-856.

[9] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu and Bharat Bhargava, ”PROMISE: peer-to-peer media streaming using CollectCast” ACM, MULTIMEDIA ’03

[10] Min-You Wu, Su-Jun Ma, and Wei Shu, ”Schduled Video Delivery for Scalable On-Demand Service,” ACM NOSSDAV’02, Miami, Florida, USA, May, 2002, pp. 167-175.

[11] Munindar P. Singh, ”Peering at Peer-to-Peer Computing” IEEE Internet Computing, Vol. 5, No. 1, January/February 2001.

[12] Schollmeier, R., ”A Definition of Peer-to-Peer Networking for the Classification of Peer-to-peer Architectures and Applications,” IEEE P2P’01, Linkoping, Sweden, 2001, pp.101-102

[13] Tai Do, Kien A. Hua, and Mounir Tantaoui, ”P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment” Proceedings of the IEEE International Conference on Communications (ICC 2004), June 20-24 2004, Paris, France

[14] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, ”Resilient peer-to-peer streaming” IEEE International Conference on Network Protocols, Atlanta, USA, Nov.2003.

[15] Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley, ”P2Cast: Peer-to-peer Patching Scheme for VoD Service” Proceedings of the 12th World Wide Web Conference (WWW-03), Budapest, Hungary, May 2003.