香港浸會大學
HONG KONG BAPTIST UNIVERSITY

FACULTY OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
TECHNICAL REPORT (JNG06-99) JUNE, 1999

## Title:

# A Multi-Server Design for a Distributed Video System that Supports Arbitrary-Rate Playback [1]

## Authors:

Joseph Kee-Yin, NG (Department of Computer Science, HKBU)
Shuhua, XIONG[2] (Department of Electronics, Sichuan Univ.)
Hong, SHEN[3] (School of Computing & Information Technology, Griffith Univ.)

## ABSTRACT:

In this paper, we propose a multi-server design for video-on-demand systems that can provide different Quality of Service (QoS) guarantees by arbitrary-rate video playback. This design chooses a symmetric architecture that consists of a set of identically configured servers and supports arbitrary rates of video playback required by different clients in the following way. The playback requirements requested from all clients are first sent to the master server. If the master server decides to accept a new client, it distributes the task of the new client to all the servers, including itself. Through the pre-processing of the video streams, we can map the data of each frame of every video to a *frame map*. This frame map indicates whether a frame should be sent to the client at a given frame rate. In our multi-server design for a VoD system, using these frame maps and a suitable transmission schedule, a client can demand to play any video, at any playback rate and at any time. Furthermore, the VoD system provides a high degree of fault tolerance. This design not only can tolerate the failure of one or more servers, but also requires relatively small bandwidth capability or reservation for fault tolerance. In addition, this multi-server design can effectively overcome the hot spot problem of popular videos and keep the load balanced at the same time. Finally, we discuss how to generalize the proposed method for a large scale VoD system with heterogeneous architecture.

## Keywords:

Video-on-Demand System Design, Multimedia System, QoS Control, Video Server Design.

[1]This work was supported in part by the RGC Earmarked Research Grant under RGC/97-98/54, and by the FRG under FRG/96-97/II-103.

[2]Visiting research scholar in the Department of Computer Science at Hong Kong Baptist University.

[3]On leave with the Department of Computer Science at Hong Kong Baptist University, in fall 1998.

# 1 Introduction

With the advent of technologies in multimedia computing and high-speed network, Video-on-demand (VoD) system is becoming more and more popular. There have been many methods and system models proposed in the literatures for VoD systems (Bolosky et al., 1996; Chang and Zakhor, 1996; Liao and Li, 1997; Jadav et al., 1997; Orji et al., 1997). In general, they can be classified into two categories: single server VoD system and multi-server VoD system. The single server VoD system has some limitations such as no fault tolerance, limited I/O and network bandwidth and limited computation power. If we have enough network bandwidth, then the I/O bandwidth of the disk and the computation power of the server probably become the bottleneck of the single-server VoD system. Thus, there is a need to design and construct a multi-server VoD system such that the system can scale up nicely. That is, more clients can be served.

In theory, an ideal multi-server VoD design has to address the following key problems:

- Providing different QoS for video playback. In a flexible VoD system, different clients may demand to play any video at any playback rate due to their application requirements, the actual network workload, or the configuration environment.
- Overcoming hot spot of popular video. While some videos are popular at the moment, some videos may become popular over time. A good VoD system should prohibit hot spot problems resulted from popular videos.
- Keeping the load balanced. All the servers in a multi-server VoD system should share the tasks as evenly as possible. The case where some servers are overloaded while others are under used should be avoided.
- Supporting fault tolerance. In a multi-server VoD system, it is likely that some servers may fail to perform properly due to hardware or software failures. In order to keep the guaranteed service to all clients, tasks that are originally assigned to the failed servers should be shared by other active servers without severely affecting the playback of clients.
- Minimizing the storage capacity. A typical VoD system has a large collection of videos. In principle, we hope to store videos as many as possible but consume storage as small as possible. At present, since the capacity of storage devices is becoming larger and the price is getting lower, storage is a problem of less importance.
- Facilitating easy maintenance. After a VoD system is put on-line, it is likely that the videos need to be updated over time and slave servers may need maintenance and hence leave the system occasionally. As slave servers join and leave, scalability becomes a problem to the system. Moreover, it is needed to remove old videos and add new videos into the system from time to time. Therefore, a feasible system should be able to cope with these requirements without severely interrupting its service.

Each of the existing VoD systems has its advantages and drawbacks. In a VoD system of single server with multiple disks, or multiple servers with multiple disks, the most popular design is based on the striping strategy (Wong and Lee, 1997; Patterson et al., 1988; Sinsoskie, 1991). Striping strategy can be divided into two categories: space striping and time striping. One typical example of using space striping is the Tiger video file server (Bolosky et al., 1996). In the Tiger system, files are broken up into blocks of equal size. Every file is striped across every disk and every cub. Thus, it can provide absolute load balancing, a constant bit rate (CBR) transmission, and some limited fault tolerance by employing the mirroring strategy. However, it has several main drawbacks. One of them is that different clients can not demand different levels of QoS for playback. Another problem is the difficulty in adding a new server or a new video to the system. Furthermore, the degree of fault tolerance is very limited, it is because of the tradeoff in the choice of decluster factor between bandwidth reservation for failed mode operation and the decreased fault tolerance. On the other hand, there are many VoD systems that use time striping (Lee, 1998; Kwon et al., 1997; Srivastava et al., 1997; Sengodan and Li, 1997). The principle of time striping is similar to that of space striping. Their difference is that time striping uses time unit or frame to stripe, every unit has equal time interval. Thus, with MPEG-1 or MPEG-2 compression standards, VoD systems using time striping strategy provide a variable bit rate (VBR) transmission at a constant frame rate. It can also provide some limited QoS guarantees. However, the load for each server will be unbalanced. The common disadvantage of striping is that changing the system configuration by adding or removing slave servers requires changing the allocation of all the video data.

In addition to the above VoD systems, there are also systems that are designed to improve the tradeoff between time striping and space striping (Kwon et al., 1997; Wolf et al., 1997). For example, Chen et al. (Chen et al., 1994) proposed a segment selection scheme to support arbitrary-rate playback, which consists of

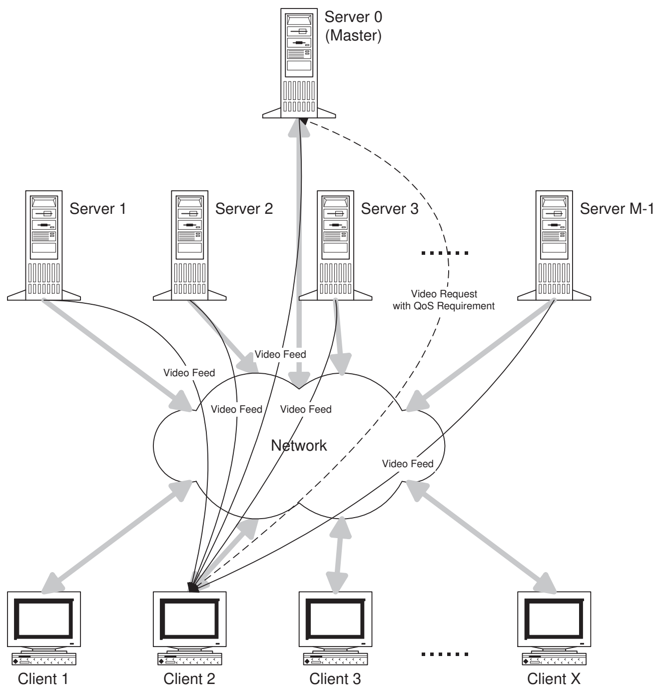


Figure 1: The basic architecture of our VoD system.

two alternative methods: a segment sampling method (SSM), and a segment placement method (SPM). Taeck-Geun Kwon et al. (Kwon et al., 1997) proposed the prime round-robin (PRR) scheme, which provides uniform load balance of disks for arbitrary retrieval rate, without additional disk bandwidth. However, the concept "arbitrary-rate" in these papers only refers to arbitrary multiples of the normal playback rate. In addition, Sincoskie (Sincoskie, 1991) has proposed a design based on interleaving the video frames, which provides very limited remote-control functionality and is non-scalable. Alok Srivastava et al. (Srivastava et al., 1997) have also proposed another interleaved video storage, which can provide the complete functionality of a remote control. All of them did nothing to support a high degree of fault tolerance and care less about maintaining the system dynamically.

In summary, almost all of the existing VoD systems are unable to provide different QoS for video playback without degrading the total performance of the VoD system. In practice, at present, there are many users who own computer with low computing power and only have a link with limited bandwidth, such as POST link. Thus it is likely that different clients may request to play a video at different levels of QoS. Moreover, many existing systems and models ignored the feature of the MPEG compression standard, and did not incorporate the MPEG characteristics in their design. Thus, in our VoD system design, we focus on supporting different levels of QoS under the MPEG compression standard. We propose a simple master election algorithm, load balancing algorithm, fault-tolerance algorithm, and the algorithm for maintaining the system in the presence of dynamic join and leave of servers.

The remainder of this paper is organized as follows. Section 2 describes the design of the system architecture and its working principle. Section 3 discusses the principle and the method used to support different levels of QoS. Section 4 presents algorithms for providing different levels of QoS, for load balancing, for fault-tolerance and the dynamic join algorithm for the new or repaired servers. Section 5 generalize our design for a large scale VoD system with heterogeneous architecture. We conclude the paper with a summary and discuss our future work in Section 6.

## 2 A Multi-Server System Architecture

In this paper, we assume that the I/O bandwidth and the computation power of the servers are the bottleneck, rather than the storage capacity. We also assume that there is sufficient network bandwidth from the servers to the clients. Since multiple servers are sending different video frames to a client, we assume that these frames carry unique frame numbers and the client has a large enough buffer to reconstruct the video stream for display. This can be achieved by using a sliding window protocol on the client side. On the other hand, the clients might have other constraints such as graphic display limitation, computation power of the local computers, price factor, and other special requirements for specific applications. Because of these limitations, the clients need an arbitrary-rate playback to meet their QoS requirements. Thus, we propose a multi-server VoD system design, which not only overcomes the limitation of I/O bandwidth and computation power caused by the single server and single disk system, but also achieves an overall improvement in system performance.

The basic architecture of our multi-server VoD system is shown in Figure 1. In order to avoid the catastrophe of single point failure, we chose a symmetric architecture in designing our multi-server VoD system. The VoD system consists of a set of identically configured servers. They have the same type of network hardware, computation power and the same type of storage devices, etc. Moreover, each video has a copy in each server and a network connects all the servers and clients. At any moment, among these servers, there is one and only one server acting as a master. In this paper, we call this master server the *master*, and other normal servers *server*. Since all servers have the same configuration, any one of them can be elected as the master. The master serves as a contact point where the QoS requirements from all clients are received. The master also provides the system clock for synchronization. In the mean time, it is also responsible for admission control, task assignment, fault-tolerance and the dynamic join of new or repaired servers. It must ensure that a new client will be admitted into the system only when the system has enough capability and this new admission will not violate any guarantees already made to previous clients. On the other hand, all the servers, including the master, are responsible for sending the required video stream to the clients at an appropriate time, adding and removing clients from the transmission schedule, and compensating for the failure of system components.

Each server is assigned an identifier (ID) at a time. The ID number of a server is dynamically assigned according to the current status of the system. All servers have continuous ID numbers starting at 0 and form a logic ring. In general, the server, whose current ID is 0, is the master. The master receives all the QoS requirements requested from all the clients, and assigns every task of every client to all the servers according to a proper assignment algorithm. The master periodically broadcasts current global information such as the number of servers, the number of clients, and the current position of every video file. After each server receives the periodic broadcast information, it saves the information into their local memory and responds with an acknowledge (ACK) to the master to indicate that it is still alive. Once the master does not receive the ACK from a server within a specific time interval, it knows that the server has failed. Then it re-assigns ID numbers to all active servers to ensure that the ID numbers are continuous to form a new logic ring, and re-assigns all the current tasks to these servers according to the appropriate algorithm. On the other hand, if a server has not received the broadcast information from the master within a specific time interval, then it assumes that the master server has failed. Therefore, this server will start an election algorithm to elect a new master out of the current active servers. Once the new master is elected, it will take over the job which was taken care of by the old master originally. Then it will set its ID to 0, and re-allocate a new ID to each active servers to form a new logic ring. When there are new or repaired servers that want to join the system, they apply to the master. The master then allocates the next ID numbers in sequence to them so as to ensure that all the ID numbers in the system are continuous, finally it re-assigns all current tasks to all servers, including the new servers.

In addition, all the servers are synchronized at some degree. The service process is divided into many service rounds. In each service round, all servers must serve each of the clients once. This synchronization ensures that each client needs to have only a medium size buffer and provides simple algorithm for fault tolerance.

Although the storage capacity required in our VoD system is larger than that in some of the existing VoD systems (Bolosky et al., 1996; Srivastava et al., 1997; Patterson et al., 1988), with the advance in storage technology and capacity, storage devices are getting faster, larger, and cheaper, hence, for a medium size VoD system, it is justifiable to achieve the many good performance gains such as arbitrary-rate playback with QoS, high degree of fault tolerance, load balancing, at the cost of increased storage capacity. To the best of our knowledge, the proposed system is the first one that provides client specified arbitrary-rate playback and thus supports different levels of QoS.

# 3 A Method to Support Different Levels of QoS

## 3.1 The Principle of Frame Mapping Selection

In a VoD system, different clients can have different constraints, such as limited network bandwidth and limited computation power. On the other hand, some clients may have special requirements on the QoS for their special purposes. Thus, when they are playing the video, they probably require different levels of QoS according to their specific resource or requirement constraints. Recently, pre-recorded videos are usually compressed according to MPEG-1 or MPEG-2 standards. Normally, the playback rate is 30 frames per second. But in practice, clients may want to play a video at a lower frame rate. Note that this does not mean the video is playing in a slow motion manner. There are many factors affecting video's QoS, such as frame rate, resolution, color depth, and dithering. Among all these factors, frame rate is the most critical one. In this paper, in order to simplify the problem, different levels of QoS only means different frame rates. For example, if a client demands to play a video at $m(m \leq 30)$ frames per second (fps), when the slave server retrieves the video stream from its hard disk, it retrieves only $m$ frames out of 30 frames for every second of the pre-recorded video and sends these $m$ frames to the client. That is, it skips $(30-m)$ frames per second intelligently. In MPEG-1, MPEG-2 standards, there are three different types of frame: I-, P-, and B- frame. The P- and B- frames are dependent upon the previous or subsequent frames. Furthermore, the P- frame located at different positions within a Group of Picture (GoP) pattern has different degree of importance, a later P- frame can be decoded correctly only if all the previous P- frames have been decoded correctly. Therefore, for MPEG-1 video data, the ways of selecting which $m$ frames are to be sent to the client are not arbitrary. The principle of selection must conform to the followings:

1. Select the I- frames first, then select the first P-, the second P-, $\cdots$, until the last P- frame, and finally select the B- frames.

2. The selection of frames should be distributed as evenly as possible within a GoP and among the video stream.

Thus in a VoD system, we should pre-process all frames of all videos. By using the above rules, we can gain the information that indicates whether a frame should be retrieved and be sent to the client at the rate requested by the client. A client can demand any video at frame rates from 1 to 30 fps[4], hence, every frame has 30 different choices.

We define a frame map for each frame in a video to be such a map that consists of 30 bits of mapping data, each indicating that, at a given rate, whether this frame should be retrieved from the hard disk and sent to the client.

## 3.2 Constructing the Frame Maps

Consider that there are $N$ video streams in our VoD system, denoted as $V_0, V_1, \cdots, V_{N-1}$, respectively. The total number of frames for each of these $N$ streams are denoted as $F_0, F_1, \cdots, F_{N-1}$, respectively. The symbol $f_{ij}$ refers to the $j^{th}$ frame of the $i^{th}$ video, where $i = 0, 1, \cdots, N-1$ and $j = 0, 1, \cdots, F_{i-1}$. In addition, we use another symbol $M_{ij}^m$ to represent the frame map of frame $f_{ij}$, which indicates whether the $j^{th}$ frame should be selected for transmission when the client demands to play the $i^{th}$ video at the rate of $m$ fps, where $1 \leq m \leq 30$, that describes the levels of QoS. If $M_{ij}^m = 1$, it indicates that when a client demands to play the $i^{th}$ video at $m$ fps, and the $j^{th}$ frame should be selected and be sent to the client. Otherwise, if $M_{ij}^m = 0$, the corresponding frame should be skipped to meet the QoS of $m$ fps requested by the client.

Therefore, in order to support different levels of QoS, the goal of pre-processing is to obtain the frame maps of all frames, that is, to calculate all the values for the $M_{ij}^m$ flags. In addition, according to the MPEG-1 and MPEG-2 compression standards, the display frame sequence is different from its encoded frame sequence, but there is a fixed relationship between them. In this paper, we use the display frame sequence to analyse the problem. Furthermore, the GoP pattern in a video might change from time to time. If there is only one type of GoP pattern for the whole video stream, we can easily find a formula to calculate the $M_{ij}^m$ values. Even if there are more than one type of GoP pattern within a video file, we can also calculate the $M_{ij}^m$ values during the pre-processing period. Normally, for a particular video file, the GoP patterns of a video does not change. In this case, we calculate the $M_{ij}^m$ values with the following rules.

[4] We assume all the videos are encoded at 30 fps.

Consider a client who demands to play the $i^{th}$ video at $m$ fps, and the GoP pattern of the $i^{th}$ video is $IB\cdots BP_1B\cdots BP_2B\cdots B\cdots P_nB\cdots B$. Let $num(GoP)$ be the number of frames in one GoP and let $I(x)$, $P_1(x)$, $P_2(x)$, $\cdots$, $P_n(x)$, $B(x)$ be the numbers of I-, $P_1$-, $\cdots$, $P_n$- and B- frames within $x$ frames, respectively.

While the QoS requirement from the client is $m$ out of 30 frames, and the number of frames in a GoP is $num(GoP)$, in order to find the regular formula for all $M_{ij}^m$, we should find the least common multiple period of the two periodic requests, which is denoted as $T_{min}$ and defined as:

$$T_{min} = LCM[num(GoP), 30] \tag{1}$$

where the function $LCM[x, y]$ returns the least common multiple (LCM) between $x$ and $y$. Hence, within the $T_{min}$ frames, the numbers of I-, $P_1$-,$\cdots$, $P_n$-, B- frame are:

$$I(T_{min}) = P_1(T_{min}) = \cdots = P_n(T_{min}) = \frac{T_{min}}{num(GoP)} \tag{2}$$

$$B(T_{min}) = T_{min} - (n+1) \times I(T_{min}) \tag{3}$$

When the client demands $m$ frames out of 30 frames ($m$ fps), this requirement is equivalent to the demand of $m'$ frames out of $T_{min}$ frames. Thus, $m'$ is defined as:

$$m' = m \times \frac{T_{min}}{30} \tag{4}$$

Now, our goal is to select $m'$ frames out of $T_{min}$ frames so that we can guarantee the QoS requirement of $m$ fps . The selection principles are described in Section 3.1, and the detailed selection for the first $T_{min}$ frames of the video stream is shown as follows:

- Case 1: If $m' \leq I(T_{min})$, then we only need to select $m'$ I- frames from $I(T_{min})$ I- frames so that the corresponding $M_{ij}^m = 1$, where $j$ should satisfy:

$$j = num(GoP) \times \lfloor K \times \frac{I(T_{min})}{m'} \rfloor \tag{5}$$

$$K = 0, 1, 2, \cdots, m' - 1$$

- Case 2: If $(a+1) \times I(T_{min}) < m' \leq (a+2) \times I(T_{min})$, where $a = 0, 1, \cdots, n-1$, then we should select all I- frames and all $P_1$-, $P_2$-,$\cdots$,$P_a$ frames within the $T_{min}$ frames so that their mapping data is set to 1, i. e., $M_{ij}^m = 1$. In addition, we should select $m' - (a+1) \times I(T_{min})$ $P_{a+1}$- frames from $I(T_{min})$ $P_{a+1}$- frames such that their mapping data is set to 1, and $j$ satisfy:

$$j = num(GoP) \times \lfloor K \times \frac{I(T_{min})}{m' - (a+1) \times I(T_{min})} \rfloor + l \tag{6}$$

$$K = 0, 1, 2, \cdots, m' - (a+1) \times I(T_{min}) - 1$$

  where $l$ refers to the absolute frame number of the first $P_{a+1}-$ frame in the video stream.

- Case 3: If $m' > I(T_{min}) + n \times P_1(T_{min})$, then all the I- and P- frames are selected and their corresponding mapping data is set to 1, i. e., $M_{ij}^m = 1$. Besides, we still should select $m' - (I(T_{min}) + n \times P_1(T_{min}))$ B- frames from a total number of B- frames, $B(T_{min})$. Since all the B- frames are of the same degree of importance, the number and layout of B- frame are different in different types of GoP pattern, we should select all B- frames from the first $T_{min}$ frames and put their absolute frame number $j$ into an array. The size of this array is $B(T_{min})$. The relative frame number $j'$ we should select in this array conforms to:

$$j' = \lfloor K \times \frac{B(T_{min})}{m' - (n+1) \times I(T_{min})} \rfloor \tag{7}$$

  where $K = 0, 1, 2, \cdots, m' - (n+1) \times I(T_{min}) - 1$

  Thus we can find the absolute frame number $j$, which is the data of $j'^{th}$ array element. We introduce a function $Array[x]$ to describe this relation. So, we still select some B- frames $f_{ij}$ such that their *mapping data* is set to 1, where $j$ conforms to:

$$j = Array[j'] \tag{8}$$

Through the above selection steps, we can find the mapping data for a particular value of $m$ within the first $T_{min}$ frames. It is easy to extend this method and apply it to the whole video stream: vary the value of $m$ (from 1 to 30), and then find out the mapping data for each value of $m$. Therefore for every frame in every video, there is a frame map of 30 bits to indicate whether this frame when retrieved at $m$ fps ($1 \leq m \leq 30$) playback rate should be sent out or not. Note that the frame map for each frame requires only 30-bit in space, so the total size of all frame maps for a video, which is the extra space required by our proposed scheme, is very small when compared to the file size of the video. This overhead is worthwhile because it enables the system to supports different levels of QoS according to the specifications from clients.

# 4 Algorithms Adopted by the VoD System

## 4.1 Algorithms for Providing Different Levels of QoS

With the frame map for every frame of every video stream, a client can demand any video at arbitrary rate. When a client demands to play the $i^{th}$ video at $m$ fps, the server will find out the corresponding frame map. If $M_{ij}^{m} = 1$, then the $j^{th}$ frame is retrieved from the secondary storage and is sent to the client. Otherwise the $j^{th}$ frame is skipped. Thus, arbitrary-rate playback can be achieved by utilizing these frame maps in our VoD system design.

## 4.2 Load Balancing Algorithm

Load balancing is closely related to arbitrary-rate playback because the playback of each client is provided by all servers collectively at a particular time within each round and therefore assigning appropriate tasks to the servers and balancing the load among them so that all playback requests can be completed satisfactorily within each round is an important function of the proposed system. Thus the proposed system should employ an appropriate algorithm for assigning and balancing load among all servers for each playback request.

Once the master receives a request from a new client, it decides whether to accept the new client according to the current load of the system. It must assure that the system has sufficient capacity to serve the new client without violating the guarantees it has already made to the previous clients. If the master decides to accept the new client, it will assign the task of the new client to all servers.

Consider that there are $M$ servers in a VoD system, denoted as $S_0, S_1, \cdots, S_{M-1}$, with their identifier numbers ($ServerIDs$) being $0, 1, \cdots, M-1$, respectively. For a client who demands to play the $i^{th}$ video at $m$ fps, the sender task can be divided into $M$ sub-tasks, $\tau_0^{mi}$, $\tau_1^{mi}$,$\cdots$, $\tau_{M-1}^{mi}$. Without loss of generality, we introduce an offset factor $Q$, which indicates the start of the video fro each sub-task begins from the $Q^{th}$ frame. This offset is in particular useful for our fault-tolerance and dynamic join algorithms in sub-tasks re-assignments. If a client demands to play a video from the first frame, then $Q = 0$. Thus, sub-task $\tau_k^{mi}$, ($k = 0, 1, \ldots, M-1$) includes such frames $f_{ij}$ that satisfy $(j - Q) \bmod M = k$. Because of the different sizes of I-, P- and B- frames and the different frame maps at different frame rate, the workload of a sub-task differs from those of the others. Therefore, the load balancing problem becomes a task assignment procedure: assign $\tau_0^{mi}$, $\tau_1^{mi}$, $\ldots$, $\tau_{M-1}^{mi}$ to $S_0, S_1, \ldots, S_{M-1}$ for video $i$ playing at $m$ fps. After the assignment, every server is responsible for a sub-task for every client in a round-robin manner. At any instance, if the mapping data of frame $f_{ij}$ is one, i.e., $M_{ij}^{m} = 1$, the corresponding server should retrieve and send the frame to the client. Otherwise, the server will skip it, and do nothing for this client until the next service round.

The simplest task assignment algorithm is to assign $\tau_k^{mi}$ to server $S_k$, where $k = 0, 1, \cdots M-1$. This algorithm can overcome the hot spot problem of popular video. However, in our VoD system, since the clients are permitted to play the videos at an arbitrary rate, it is likely that many clients may request the same video at the same playback rate. In such case, for some values of $m$, the number of frames being sent by a sub-task, $\tau_k^{mi}$, could be very much different. This will lead to an unbalanced load among the $M$ servers such that some of them are overloaded while others are underloaded. In order to keep the load balanced and provide arbitrary rate playback, we propose another task assignment algorithm, which is also called the load balancing algorithm. The primary rule is: the sub-task of the client with the lightest workload is assigned to the server with the heaviest workload, the sub-task with the second lightest workload is assigned to the server with the second heaviest workload, and so on, until at last, the sub-task with the heaviest workload is assigned to the server with the lightest workload. Through the pre-processing of the video data, we have collected all the data of frame maps and the size of every frame of every video. Thus we can pre-calculate the workload of sub-task

$\tau_k^{mi}$, $W_k^{mi}$ as follows:

$$W_k^{mi} = \sum_{j=Q}^{F_i} \xi(i,j,k,m)/(F_i - Q), \tag{9}$$

where

$$\xi(i,j,k,m) = \begin{cases} sizeof(f_{ij}), & \text{if } (j-Q) \bmod M = k \text{ and } M_{ij}^m = 1, \\ 0, & otherwise. \end{cases}$$

Assume that there are Y clients in the VoD system, then the current workload of a server, denoted as $L_k$, is the sum of the workload of Y sub-tasks, which are assigned to that server according to the task assignment algorithm.

According to above rule and definition, we describe task assignment algorithm in two cases. The first case is when the VoD system just starts to work, all the clients are new for the system. Every video needs to be played from the first frame, so $Q = 0$. Therefore, the master divides the total video requested by the first client into M sub-tasks, and assigns them to the servers in turn. Later the current workloads of all servers can be calculated and be indexed in increasing order. Consequently, the workloads of all sub-tasks of the second client can be calculated and be indexed in decreasing order. Then the sub-tasks are assigned to the appropriate servers one to one according the above rule. For the third client, the current workloads of all servers and workloads of its all sub-tasks can be calculated, indexed and assigned. In this way, all task of all clients can be assigned appropriately such that load balancing can be achieved. The second case is when the system is running and there is a new client requesting to be served, the method on how to divide the new task, the indexing and assignment algorithms are the same as in the first case. However, when calculating the current workload for every server, the part of the video which has been played, should be ignored. That is, we only need to calculate the workloads of the remaining part of all videos, and the value of $Q$ is the current frame number of the video being played.

Compared our method with the other methods in which tasks are assigned in term of the entire video file (Wong and Lee, 1997; Sincoskie, 1991), our method has the advantages of relatively small granularity and better load balancing.

### 4.3 Master Election Algorithm

In our design, the system architecture is symmetric and distributed. Among the identically configured servers, one server need to be elected as the master, which acts as the coordinator, sequencer, and performs some special role. There are some famous traditional election algorithms (Tanenbaum, 1995) such as the *bully algorithm* and *ring algorithm* for electing a co-ordinator within a group of processes. Coping with the characters of algorithms and system, we propose a simple and effective master election algorithm.

Initially, before the VoD system starts running, we can specify any one server as the master, and assign unique ID number to the master and to each servers. The ID of the master is always assigned to 0. The ID numbers of other servers are assigned starting from 1 in a continuous increasing order. In addition, we assume that the server with smaller ID number has higher priority to be elected as the master. As described in Section 2, the master needs to broadcast information periodically to all the servers, such as the number of current active servers, the number of clients and the current frame position of each video. This enables the servers to know the global information so that they can work correctly to complete the job even if the master fails to operate. After a server receives the broadcast message, it responds with an ACK to the master to indicate that it is alive and pass the status information back to the master. If the master has not received an ACK from one or more normal servers within a specific time interval, it assumes that these servers have failed, then it starts to run the fault-tolerance algorithm, which will be described in detail in next section.

On the other hand, if any server does not receive the broadcast message within a specific time interval, it knows that the master has failed and the VoD system needs a new master to keep the system operational. Hence, the master election algorithm should be invoked. Consequently, any server can start the election algorithm to elect a new master. If the ID number of a server is equal to 1, that shows this server has the highest priority to be elected as the new master. Thus, this server will act as the new master immediately, and set its ID number to 0. In the mean time, it will inform all other servers that it is the new master from now on. If the ID number of a server is not equal to 1, that shows that there maybe some other active servers who have higher priorities to be elected as the new master. Therefore, this server will send an election message to all the servers whose ID numbers are smaller than its ID number. If it receives a reply from any one of these servers, it knows that there is at least one active server who has a higher priority to be elected as the new master. Then this server will give up the election. If there is no reply returned, it knows that all the servers who have the higher priorities have failed, so this server will act as a new master, and change its ID number into 0. Once an active server receives a election message came from the server who has a larger ID number, it will send a reply to the sender to indicate it is alive, and it will send election messages to check whether all the servers whose ID numbers are smaller than itself and see if they are active or not. If all of these servers are dead, then it will act as the new master. Otherwise it will end its election job to let other servers go on electing until a new master is elected. Once the new master is elected, it will take over all jobs of the old master. That is, it changes its ID number into 0 and checks whether other servers are still alive, assigns new ID numbers to the current active servers in a continuous manner to form a new logic ring. In addition, the new master should have obtained the current status of the VoD system according to the latest broadcast information came from the original master. So it can re-divide the tasks according to the number of current active servers, calculate workload for each sub-task which depends on the current frame position, and re-assign the sub-tasks of every client to the appropriate servers according to the load balancing algorithm.

## 4.4 Fault-tolerance Algorithm

Being able to tolerate faults is a desirable property of any practical system. For the completeness of the proposed system, we therefore need to discuss issues on fault-tolerance and present an effective method to achieve fault-tolerance.

In our design, each server keeps a full set of videos and their corresponding mapping data. The advantage of this redundant storage is that the degree of fault tolerance is high while the algorithm is simple. When the VoD system is operating, it is unavoidable to have some failed severs from time to time. In general, there are two kinds of failure. Either the master server fails or one or more servers ceased to operate. Since the master broadcast the current global information to all servers, it is easy to check and find out the occurrence of server failures. When the master has not received the ACK from a server within a specific time interval, it assumes that this server has failed. After having identified all active servers, the master will re-assign a new ID to each active server in a continuous sequence starting from the number 1, and tells them the total number of active servers and clients. On the other hand, when any server has not received the broadcast message from the master, it assumes the master is dead and can start the fore-mentioned election algorithm to elect a new master. The new master at first identifies whether all other servers are active and identify the number of active servers. Then it re-assigns a new ID number to each server in sequence, the ID number of itself is 0. No matter whether the servers or the master fail, once the new logic ring is formed, the master must re-divide the remaining tasks among the currently active servers. Assuming that at some moment, $n$ servers have failed, then the tasks, which are originally taken care of by $M$ servers, that is $M-1$ servers and one master, should now be taken care of by $M-n$ active servers. It is worthy to note that the part of the video which has already been played should be ignored when the master is redistributing the tasks of the system. Thus, only the remaining part of the video for each client needs to be divided into $M-n$ sub-tasks instead of $M$ sub-tasks. That is, the master will re-divide all remaining tasks and re-assign them to the active servers. This task assignment procedure is very similar to that when the VoD system starts to operate. The difference is that the former sets the $Q$ value of each video according to its current frame position, whereas the latter sets $Q = 0$ for all videos. Finally, the master re-assigns all the remaining tasks to each server one by one according to the load balancing algorithm.

From the above discussion, we can see that in our VoD system, the communication overhead for providing fault tolerance is small, the fault-tolerance algorithm is simple and the calculation is not very complicated. In addition, the system must reserve some bandwidth or capability in order to support a certain degree of fault tolerance. There is no doubt that we shall reserve the smallest amount of resources but attain the highest degree of fault tolerance. In our VoD system, when there is a server failure, all the other servers share the tasks that are originally handled by the failed server. For this scheme, the reserved bandwidth and computation power required are relatively small. When network bandwidth is large enough, our design can tolerate the failure of one or more servers at the same time. Theoretically, as long as the bandwidth and computation power are sufficient, our system can still operate correctly even if there is only one server left. Therefore, the degree of fault tolerance is very high.

## 4.5 Server Dynamic Join Algorithm

A dynamic join algorithm is needed in order to allow new servers to join the system and to bring the repaired faulty servers on-line. Moreover, it is necessary to remove old videos and add new videos from time to time. Therefore, a flexible design should be able to cope with these changes without severely degrading its current service. Many other multi-server VoD systems may suffer from these kinds of dynamic changes of configuration.

In a striping VoD system, if there is a new server that wants to join the system, the join procedure is very difficult and complicated. All video data must be stripped or re-arranged again. The whole system service have to be suspended and have to take a long time to do the re-striping work. Thus the service for all the clients have to be interrupted.

While in our VoD system, when one or more new servers or repaired servers want to join the system, no task should be interrupted. Assuming that there are $n$ severs requesting to join the system, they need to ask the master for joining. After having decided to accept the join request of new or repaired servers, the master allocates a ID number to each of the new server, in an increasing order after the ID numbers allocated to the original ones, thus ensuring that all the ID numbers are continuous. That is, the last server joining has the largest ID number and it will have the lowest priority to be elected as the new master. Assuming that there are already $M$ servers in the VoD system, and there are $n$ new servers, then the master needs to re-assign all the remaining tasks, which were taken care of by $M$ servers originally, now will be taken care of by $M+n$ servers. For every video requested by any client, the part that has been played should be ignored for the calculation of the workload. That is, the value of $Q$ in the load balancing algorithm is determined by the current frame number. According to the load balancing algorithm, the master can re-assign all the remaining tasks to all the servers, and keep the load balanced.

From the above algorithm and the fault-tolerance algorithm, it is clear that when some servers fail or join the system, due to the re-assignment of tasks by the master, all current clients will be affected. However, since the algorithm is simple and the overhead is low, together with the probability of a server failure or rejoining is not high, the time interval during which all clients are affected is very short and the overall performance can be sustained without severe performance degradation.

# 5 Generalization

As described in previous sections, the basic VoD system design not only can provide arbitrary rate playback, it also can achieve good load balancing, high degree of fault-tolerance, and allow the dynamic joining of servers. Moreover, the architecture of the basic VoD system is symmetric, the master can be changed dynamically from time to time according to the current status of system. Thus the single point failure of the whole VoD system can be overcome. These advantages are achieved at the cost of the large redundancy of storage and the overhead of information broadcast. If the size of VoD system, the number of servers and clients are medium, our basic design proposed can tolerate the above disadvantages. Thus, the basic design is applicable and has good performance when it is used on a medium size VoD system.

For a large scale VoD system with large numbers of servers and videos, the drawbacks of redundant storage and communication overhead can not be tolerated. Under this condition, we have to make some compromise. In this section, we generalize the basic theory proposed in the previous sections to the design of a very large and heterogeneous system. For this purpose, we propose a grouping architecture as shown in Figure 2. That is, when there is large number of servers in the VoD system, we divide them into several groups. In the meantime, all the videos in the whole VoD system are also divided into the same number of groups. Therefore, different groups have different videos. For each group, the basic design theory as described in previous sections is applied. Thus, within each group, all the good performance as described in the previous sections can also be achieved.

Assume that there are $M$ servers and $N$ videos in the VoD system, if we only use one group, then the amount of storage is $N \times M$. If we divide the servers and the videos into $H$ groups equally, then the amount of storage in each group is $M \times N/H^2$, the amount of storage for the whole system is $M \times N/H$ times of that for the basic architecture. The amount of storage for the grouping architecture is only $1/H$ of that for the basic architecture. In the meantime, because the master only needs to broadcast the current information within its group, the amount of broadcast messages is also $1/H$ times of that for the basic architecture. It is obvious that the larger the $H$, the smaller the redundancy and the overhead of communication. On the other hand, a larger value of $H$ leads to a lower degree of fault-tolerance. Since the redundancy of storage is limited within a group, once a server has failed, only those servers within in that group can be used to compensate this failed server, thus the bandwidth reservation for the failed mode is larger than that for the basic architecture. Furthermore, if all the servers within a group failed, then the tasks of the clients who are served by this group can not be taken care of by other servers in other groups, because those videos are only stored within that group. Thus, the failed servers in one group can not be compensated by those in other groups. Therefore, in an actual VoD system, we should consider all the factors and find out a suitable value for $H$.

Based on the assumption that all servers are identically configured, the most simple grouping method is to divide all servers and videos into equal groups. When all the videos have the same peculiarities, this simple grouping method is applicable and effective. However, videos can get popular from time to time. If we group

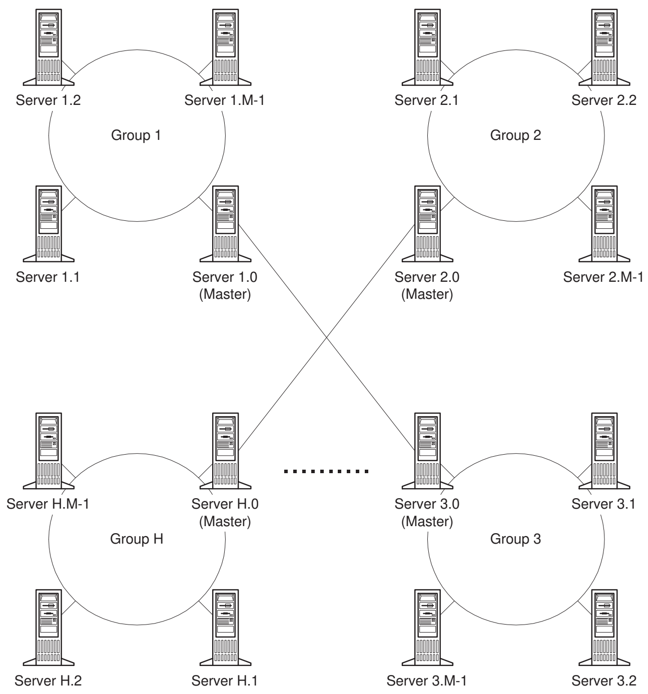


Figure 2: A group architecture for a large scale VoD system.

the servers and videos without considering the popularity of each video, it is likely that some groups would be overloaded while other groups are under used. Thus, in order to overcome the problem of unbalance load among the groups resulted from the popularity of some videos, we'd better predict the popular levels of the videos. According to the levels of popularity of all videos, we can employ two ways to achieve a balance load among the groups. One way is to distribute all the popular videos into every group as evenly as possible, this can ensure that there are most popular, popular, and not so popular videos in each group, and therefore, load balancing can be achieved. Another way is to divide the servers into different size groups. Each group has its capability that is different from that of the other groups. We put the most popular videos in the largest group, the second most popular videos in the second largest group, and so on. Load balancing can also be achieved effectively.

Finally, the discussion in the previous sections is based on a homogeneous system, which has identically configured servers. In real life, on one hand, some old computers are low in performance but they are still operational. On the other hand, there is a need to add new and high speed computers into the system. In order to make full use of all the existing resources and computation power, we can extend our grouping architecture to heterogeneous systems. Our grouping method now groups the computers of same type and same configuration into one group. Thus, it is very likely that different groups have different types of computers and different computation powers. In this condition, through storing different popular levels of videos into appropriate groups, as described before, load balancing can also be achieved.

# 6 Conclusions

In this paper, we discussed the main drawbacks of current VoD system designs, and found that almost all of them are incapable of providing arbitrary-rate playback to meet the QoS requirements from the clients. Therefore, we proposed a multi-server VoD system that focuses on the support of arbitrary-rate playback. By means of pre-processing the video streams, we can get all the frame maps which indicates whether a frame should be retrieved and sent at a given frame rate. Since the characteristics of MPEG and the smoothness of the playback are being preserved when calculating these frame maps, a client can demand to play any video at any frame rate, and the playback should be smooth and without violating the MPEG compression and decompression standard. In addition, through the use of these frame maps, we proposed simple algorithms to achieve many favorable features to improve the performance of the system. These include the high degree of fault tolerance, load balancing and the flexibility to cope with the dynamic changes of the system configuration and scalability. In addition, we extend our basic theory to the case of large scale systems by employing the grouping architecture to form groups of different sizes to match the popular videos, which overcomes the problem of unbalance load among the groups. Furthermore, we also show how to use this grouping architecture in heterogeneous system to make full use of the current resources.

When comparing our proposed system with those in the literature, we found that only the Tiger (Bolosky et al., 1996) system is close to ours. But unlike ours, the Tiger system does not support dynamic server join and leave. Since all existing systems are different in certain aspect(s) in their design, the performance data in one system cannot be used for comparison with those in another. As a result, it is difficult to justify quantitative performance comparisons of our system with existing ones.

As for the future work, we will elaborate research work in the large scale system which has many servers and videos, and in heterogeneous system which has different types of servers. We plan to extend our basic theory to the actual VoD environments, and hope to find better algorithms to improve the system performance.

# References


[1] Bolosky, W.J., Barrera, J.S., et al., "The Tiger Video File server", *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, IEEE Computer Society Press, Los Alamitos, California, 1996.

[2] Chang, E., and Zakhor, A., "Cost Analysis for VBR Video Servers", *IEEE Multimedia*, No.3, Vol.4, 1996, pp.56-64.

[3] Liao, W., and Li, V.O.K., "The Split and Merge Protocol for Interactive Video-on-Demand", *IEEE Multimedia*, Vol.4, No.4, 1997, pp.51-62.

[4] Lee, J.Y.B., "Parallel Video Servers: a Tutorial", *IEEE Multimedia*, Vol.5, No.2, 1998, pp.20-28.

[5] Kwon, T.G., Choi, Y., and Lee, S., "Disk placement for arbitrary-rate playback in an interactive video server", *Multimedia Systems*, 1997, pp.271-281.

[6] Jadav, D., Choudhary, A., and Berra, P.B., "An Evaluation of Design trade-offs in a High-performance, Media-on-Demand Server", *Multimedia Systems*, 1997, pp.53-68.

[7] Wolf, J.L., Yu, P.S., and Shachnai, H., "Disk Load Balancing for Video-on-Demand systems", *Multimedia Systems*, 1997, pp.358-370.

[8] Srivastava, A., Kumar, A., and Singru, A., "Design and Analysis of a Video-on-Demand Server", *Multimedia Systems*, 1997, pp.238-254.

[9] Orji, C.U., Bobbie, P.O., and Nwosu, K.C., "Spatio-temporal Effects of Multimedia Objects Storage and Delivery for Video-on-Demand Systems", *Multimedia Systems*, 1997, pp.39-52.

[10] Chen, M.S., Kandlur, D., and Yu, P., "Support for Fully Interactive Playback in Disk-Array-Based Video Server", *Proc. ACM Multimedia*, 1994, pp 391-398.

[11] Wong, P.C., and Lee, Y.B., "Redundant Array of Inexpensive servers (RAIS) for On-Demand Multimedia Services", *Proc.ICC 97*, IEEE Computer Society Press, Los Alamitos, Calif., 1997, pp.787-792.

[12] Patterson, D., Gibson, G., and Katz, R., "A Case for Redundant Arrays of Inexpensive Disks(RAID)", *ACM SIGMOD*, 1988, pp.109-116.

[13] Sengodan, S., and Li, V.O.K., "A Shared Buffer Architecture for Interactive VoD Servers", *IEEE Multimedia*, 1997.

[14] Sinsoskie, W.D., "System Architecture for a Large Scale Video-on-Demand Service", *Computer Networks and ISDN Systems*, Vol.22, 1991, pp.155-162.

[15] Tanenbaum, A.S., "Distributed Opening Systems", Prentice-Hall, Inc. 1995.