# Parallel Replication for Distributed Video-on-Demand Systems

By
Lie, Wai-Kwok Peter

Supervised By :
Prof. John C.S. Lui

A Thesis
Submitted to the Division of Computer Science & Engineering
in partial fulfillment of the requirements for the
degree of Master of Philosophy
at the
Chinese University of Hong Kong
July 1997

香港中文大學
圖書館系統
22 MAY 1998
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

# Parallel Replication for Distributed Video-on-Demand Systems

submitted by

**Lie, Wai-Kwok Peter**

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

## Abstract

Recent technological advances in high speed networking and video compression techniques have made the use of Video-on-Demand (VOD) feasible. A large-scale VOD system imposes a great demand for bandwidth and storage requirements, hence large disk farms are typically needed for VOD systems. Although striping of movie data across a large number of disks balances the disk utilization, it exhibits additional complexities, such as synchronization and homogeneity requirements, and fault tolerance implications. It is thus more practical to limit the extent of striping employed, i.e., arrange the disks in groups and allow intra-group striping only. With multiple nodes of striping groups, however, multiple copies of a movie might be needed for different nodes. This is because if there is a single copy only, the service capacity of the node containing that copy may not be enough to cope with the demand for that movie. The need for multiple copies of movies gives rise to two issues: 1) How many copies of each movie are needed ? 2) How to assign these copies to the nodes ? These issues are further complicated by the fact that the ideal movie replicas placement is time-varying, because movie popularity changes over time. This dissertation describes an approach to reconfigure the movie replicas placement of a VOD system, where both the number of copies of each movie and their placement are adjusted *dynamically*, by high-speed replication of movies across disk nodes via special techniques. It is motivated by the fact that such a system should be more responsive to changes in demand and thus results in better performance. Several policies for the replication techniques and for the support of various components, such as the choice of movies and nodes for replication, in this dynamic replication framework are investigated and compared through a simulation study.

# Acknowledgments

First of all, I would like to thank John C.S. Lui, my supervisor, for his guidance and patience. My research could not have been done reasonably without insightful advices from him, for I had found myself in dark corners with no way out several times. In the past three years under his guidance, which covered an undergraduate final year project in addition to this dissertation, he improved my writing and presentation skills significantly, which I firmly believe is invaluable to the rest of my life. Also, his generous sharing of his computing equipment gave his students, myself included, a good environment for research.

I would also like to thank Leana Golubchik, who gave critical comments on this work and helped me improve it on many technical aspects, even when she was ill. She opened up new angles for me to view my work, thereby allowing me to make a more thorough treatment of the subject.

Finally, I want to express my thanks to my colleagues Cedric C.F. Fong and Michael P.F. Fung. Cedric have always answered my questions on mathematics with patience. Michael enlightened me on some VCR-functionality aspects of this work. Both of them are extremely capable people.

# Contents

# List of Figures

# Chapter 1

# Introduction

Recent technological advances in high speed networking and video compression techniques have made the use of Video-on-Demand (VOD) feasible. A VOD system allows users to playback digital video, which can be chosen from a pool of selections, at any time they want, ideally with functions commonly found in video cassette recorders (VCR), such as fast forward, rewind, pause, etc. A VOD service targeted for movie delivery purposes offer distinct advantages over traditional video rental stores: customers no longer need to leave their home to rent a video. Best of all, the hassle of returning video, disliked by many, is also eliminated. Aside from entertainment uses, VOD can also be considered as an integral part of multimedia databases, which have important uses in various fields, such as, the medical profession, computer aided instruction, etc.

This dissertation provides techniques for designing large-scale VOD systems to allow good scalability. A large-scale VOD system imposes a great demand for bandwidth and storage requirements, hence large disk farms are typically needed. Since there are usually practical limitations to the number of disks that can be attached to a processing node, it would be unrealistic to employ a centralized design of the video server where the whole disk farm attached to a single processing node. A more viable architecture would be a distributed system with multiple nodes, each with its own collection of disks, linked by an interconnect. For this architecture to support a popular movie that has a greater demand for resources than a single node can provide, there are two

approaches:

1. Employ striping of each movie across all nodes/disks, e.g., as in the staggered striping approach[6], such that all the nodes in the system can participate in servicing of each request for a movie (without requiring all of them to be part of the same processing node).

2. Employ limited striping to disks within the same node only, and maintain multiple copies of the same movie in different nodes. This way, each request is basically tied to a specific node for processing.

Although the first approach possesses the benefit of needing one copy of each movie only, it exhibits additional complexity. For instance, there is a requirement of synchronization among the different nodes for supporting continuous playback of a movie. In addition, use of striping techniques across all disks becomes significantly more complex in a heterogeneous disk environment. In a realistic system, heterogeneity is unavoidable due to a need to acquire additional disks as the system operates — this need can arise either due to an increase in user population or disk failures (which results in disk replacements). The newly acquired disks will likely exhibit different performance and reliability characteristics. Finally, addition of new disks will necessitate "restriping" of all movies.

Due to the above problems with the first approach, this dissertation thus focuses on making the second approach viable, which involves replication of movies. Two important issues for this type of architecture involve determining: 1) the number of copies of each movie that should be maintained and 2) assignment of these copies to the nodes in the system. Inappropriate choices of either can lead to poor resource utilization. That is, it can lead to situations where there is existing service capacity available somewhere in the system, but it cannot be used to service an incoming request due to the fact that the capacity is not available on the nodes which contain the movie being requested. Previous literature, such as [56, 13], suggested to determine the number of copies needed for each movie by the demand for that movie, based on collected statistics. In addition, movies are placed on nodes in such a manner as to

spread the (anticipated) load as evenly as possible among the nodes (adjustments to load imbalance can be made during system operation as in [56]). However, popularity of movies changes over time (and thus their demand for resources); hence, the number of copies maintained for each movie should be adjusted as well. One approach to solving this problem is to periodically readjust the number of copies, again based on collected statistics, e.g., as in [56, 13]. In contrast, we consider an approach to reconfigure the movie placement such that both the number of replicas of each movie and their placement on destination nodes are adjusted *dynamically.* This approach is motivated by the fact that such a system should be more responsive to changes in demand and thus result in better performance. Besides, it does not possess the difficulty of determining a good frequency of reconfiguration associated with a periodic scheme.

Once it is determined where a movie replica should be placed, another question on how to do so arises. Previous literature[13, 22] dealing with on-line placement of movies considered the use of a tape library as the source for copying of movie data to place additional copies of movies, when necessary. Due to the slow access rate and latency involved with the tape mechanism, this additional replica will be available for use only after a relatively long period of time, especially considering that movie objects are large, even after compression. In this dissertation, performing replication by transferring data from a disk node to another is considered. This approach shortens replication time and lets additional replicas to become available sooner.

Based on the techniques briefly described above, this dissertation illustrates that it is viable to build a scalable VOD system with a multi-node architecture, by employing dynamic movie placement reconfiguration through replication of movie data across disk nodes. In support of this thesis, this dissertation makes the following contributions:

- Devises and compares various policies to perform movie data replication in a manner that shortens the replication time, by temporarily using resources which are normally left for servicing requests;
- Demonstrates via simulation that the benefits of shortening replication time far outweighs the resource usage, which affect system performance for a short period;

- Proposes a dynamic replication scheme, which determines when movies should be replicated, when replicas should be removed, and how nodes should be chosen for replication; Several options for designing these components are compared via simulation;
- Considers various aggressive strategies to put incompletely replicated copies of movies into service early, on top of the dynamic replication scheme framework, and evaluates their impact to VCR functions;
- Devises a set of protocols for implementing a scalable distributed VOD architecture based on this dynamic replication scheme, with provisions for fault-tolerance.

The remainder of this dissertation is organized as follows. The following chapter surveys literature on multimedia systems, which lay the foundation of VOD service, and describe related work on VOD systems based on replicated movies. In Chapter 3, the system model is introduced. Chapter 4 describes the details of the dynamic replication scheme, and the various strategies that shorten replication times. A set of protocols to implement this scheme in a distributed manner is suggested in Chapter 5. Chapter 6 gives experimental results of employing the dynamic replication scheme, and compares various policies proposed in this dissertation. Chapter 7 compares this scheme with the on-line placement scheme proposed in [13]. Finally, Chapter 8 concludes this dissertation and suggests future research directions.

# Chapter 2

# Background & Related Work

In construction of VOD systems, there are a variety of challenges, most of which are due to the nature of multimedia objects. Digital video and audio differ fundamentally from conventional text-based data in various aspects:[39]

- Multiple data streams: A multimedia object typically consists of multiple data streams, usually involving at least video and audio information. For digital movies, it involves multi-language subtitles as well. If the delivery of data from these streams are not synchronized well, the playback of movies will not make sense, say, voice is heard several seconds later than the moment an actor is seen on screen to have said something. Effective synchronization among multimedia streams is thus an important subject.
- Continuous nature of multimedia streams: Unlike conventional file access, which typically shows up in form of short bursts of random access, multimedia streams are *continuous* and largely sequential in nature. An example would be the playback of a movie, which is likely to last for 90 minutes or more. Delivery of multimedia data that satisfies such continuity requirement can be seen as real-time constraints. This continuous nature necessitates radically different designs for multimedia server architectures and new types of operating system support.
- Large size of multimedia objects: Storing video and audio in digital format needs very much storage for decent quality (high resolution video with 24-bit per pixel

(bpp) color representation, high sampling rate of 44.1kHz for audio). To tackle this problem, extensive research in the field of compression of multimedia data has been made.[15, 36]

## 2.1 Early Work on Multimedia Servers

Early work related to multimedia systems, such as digital audio editing[1], the Diamond system[53], the Muse system[23], Ooi's optical disk-based system[35], the Etherphone project[52] and the VOX audio server[5] focused on still images and/or audio only, but not video data.[39] This is not surprising, considering that at the time these research projects were made, relatively high storage costs would have prohibited the use of a significant amount of digital video.

## 2.2 Compression of Multimedia Data

Most of the pioneering activities in video compression were triggered by teleconferencing and video-telephony applications.[15] The definition and planned deployment of ISDN (Integrated Service Digital Network) led to the CCITT Recommendation H.261[9], which focuses on real-time encoding-decoding with less than 150ms delay. Owing to the necessity of very low bit-rate operation (around 64kbit/s) for ISDN, the amount of overhead information is kept to be very low.

MPEG[15] video relaxes the H.261 constraints on very low delay and extremely low bit rates to achieve increased visual quality. The MPEG video compression algorithm relies on block-based motion compensation for the reduction of the temporal redundancy, and discrete cosine transform[42] (DCT)-based compression for the reduction of spatial redundancy. Three main types of pictures are present in MPEG video:

- Intrapicture (I-frame): it is similar to a JPEG[55]-compressed still picture. It provides access points for random access, but with relatively little compression

only.

- Predicted pictures (P-frame): coded with reference to a past picture (I-frame or P-frame). It provides moderate compression.
- Bidirectional predicted pictures (B-frame): coded with reference to both a past and a future reference. It provides the highest compression.

The logical view of an MPEG-coded video stream looks like this:

I B B P B B P B B P B B I B B P B B P B ...

The MPEG audio compression algorithm[36] targets digital compression of high-fidelity audio. It achieves its compression without making assumptions about the nature of the audio source, and is not tied to speech compression, which many audio compression techniques focused on. It exploits psycho-acoustics that deal with the way human brain perceives sound, such as the *masking effect*, which allows certain signals to be thrown away without being noticeable.

## 2.3 Multimedia File Systems

The Sun Multimedia File System[51] provides a set of library functions for storing audio data as Unix files, which can be shared among workstations with the now widely-used Sun Network File System (NFS)[45]. The Cambridge Pandora project[27] and Matsushita's Real-Time File System[32] investigated storage issues for digital video. Anderson et al.[4] and Gemmell et al.[17] described file system designs for supporting multiple audio channel playback, and proposed techniques for conforming to hard real-time constraints. Rangan et al.[39] presented a model that relates disk characteristics to the continuity requirements of multimedia streams for proper layout and synchronization of data from different multimedia streams, which are stored individually on disk but processed together in a data structure called *multimedia rope*. It also introduced an admission control algorithm that determines whether a new request to

a multimedia server can be accepted without affecting the existing users, with respect to real-time constraints of the access to multimedia data.

In many operating systems, such as Unix, logically contiguous blocks of data files are not guaranteed to be stored in a physically contiguous manner. Such allocation schemes are not suitable for continuous media, because successive blocks may be physically too distant from each other to conform to the real-time constraints, with respect to disk characteristics. Arranging them in a contiguous manner causes problems with fragmentation and copying overheads with insertion/deletion of blocks. Rangan et al.[40] proposed constrained block-allocation schemes, which bound the separation between successive blocks to satisfy the real-time requirements of the multimedia streams.

With concurrent accesses to multimedia streams, the bandwidth requirements can become very high. When such requirements are higher than the amount a single disk can support, multiple disks are needed to provide a large aggregate bandwidth capacity. Techniques like *striping*[44, 37] and *declustering*[43, 30, 18] are clever schemes that layout data blocks in a manner that maximizes the benefits with multi-disk I/O subsystems for text-based data. In particular, striping of multimedia objects in a disk array balances the workload among all the disks. The Streaming RAID[54] scheme and other literature[20, 21] extended these techniques for use with multimedia systems. Another multimedia object placement scheme known as staggered striping[6] allows accommodation of multimedia objects of different bandwidth requirements through a special layout of data blocks.

## 2.4 Scheduling Support for Multimedia Systems

To meet the real-time constraints for concurrent access to different multimedia streams, proper scheduling is required. In the *grouped sweeping* scheduling scheme[11], streams are divided into groups. Within each group, scan scheduling is used. By controlling the number of the groups and the group assignment, it is possible to adjust the trade-off between buffer requirements and the length of each round of access in scheduling. In [19], striping-based video layout schemes were analyzed to minimize the start-up

latency as observed by new requests, where the start-up latency is defined as the amount of time elapsed from when a request is issued to the time that its display starts.

Ramakrishnan et al.[38] suggested that multimedia systems encounter not only real-time streams, but a mix with non-real time traffic instead, due to unsolicited requests from network or background management. They proposed a cpu scheduler, which is a combination of rate-monotonic and weighted round robin scheduling algorithms, and a disk scheduling scheme based on a round-robin servicing of requests, with disk read sizes proportional to the streams' playback rate. Shin et al.[47] proposed a *reservation-based* scheduling algorithm to serve both periodic and aperiodic tasks. Han et al.[26] focused on scheduling MPEG video streams, placing emphasis on I-frames, which are more important than P-frames and B-frames.

## 2.5 Inter-media Synchronization

To deal with inter-media synchronization, a proposal for specifying synchronization requirements at a logical data level while implementing them at a physical data level was described in [34]. In [29] and [48], formal treatments of the synchronization problem were made. Some network protocols for dealing with this problem were presented in [14]. In [3], algorithms were introduced for recovering from loss of synchronization with interrupt-driven I/O devices. In [41], a method for synchronization without requiring a globally synchronized clock was given.

## 2.6 Related Work on Replication in VOD Systems

The following literature in VOD systems involves the use of replicated movies. Some of them are different in their purpose of replication, and thus are different in the ways they handle replication as well, and the aspects they consider.

Stoller et al.[49] examined high-level design of VOD servers. It analyzed the performance and costs of storing movies in a hierarchical storage consisting of memory and disk, and described a method for distributing different objects among the levels of the hierarchy. It considered the use of mirroring-based replication and *parity declustering*[33] to provide fault-tolerance.

Ghandeharizadeh et al.[22] considered a set of VOD servers based on the *shared-nothing*[50] architecture, together with a tertiary storage device (such as a tape library), which stores all the available movies. It treats the disks associated with a server as a cache to store the movies transiently for actual access by the server, by replicating movies from the tertiary storage to the disks. When the disk storage capacity is exhausted, some movies are removed. Multiple copies of the same movie may be present in different servers, for coping with high demands for very popular movies.

Wolf et al.[56] proposed a load balancing scheme for different disk striping groups, since different striping groups carry different movies, skewness of movie accesses may cause load imbalance among the groups. This scheme involves two phases. In the first phase, movie popularities are estimated and the number of replicas needed for each movie, and their placement, is considered. This phase is meant to be carried out periodically. The second phase carries out migration of customers to different replicas for load balancing, while the VOD system is serving customers.

Dan et al.[12] investigated dynamic replication of movies among multiple striping groups to deal with dynamic load fluctuations. Its replication is done on a *segment* basis, where a segment is a fraction of the whole movie. However, since movie accesses in VOD systems would likely be skewed, popular movies, with many viewers' progress distributed throughout the movie, would eventually have all its segments replicated. Therefore, in our work we do not consider this type of segment-based movie replication, but focus on strategies tailored for complete movie replication instead.

Dan et al.[13] proposed an on-line policy for placing replicas of movies in different striping groups. Its choice of placement location is based on the notion of *bandwidth-to-space ratio* (BSR), which relates the bandwidth and storage characteristics of a movie

or a storage device. Its idea is to match the BSR of a movie to the BSR of a device, since unmatched placements would result in underutilized storage or bandwidth. Although this scheme is designed to be run periodically for initial placement only, it possess dynamic control of the replication of movies, and thus shares some similarity with our work. Chapter 7 compares this scheme with ours in detail.

As mentioned before, Ghandeharizadeh et al.[19] analyzed the latency involved with disk scheduling for new requests in a disk array with striped movies. In addition, they proposed the technique of replicating objects within the same disk array, but with a shift in the objects' layout, as one of the means to minimize the start-up latency.

Golubchik et al.[24] proposed the *chained declustering* layout strategy for data placement in a disk array. It provides better fault-tolerance and load balance than mirroring does, when a disk failure occurs. Chen et al.[10] applied this chained declustering technique to VOD system and proposed a variant of it, for the layout of multiple replicas of movies on disks. This variant further improves upon the chained declustering technique to allow for dynamic addition/removal of movies, which causes problems for conventional declustering techniques.

# Chapter 3

# System Model

We consider a VOD system which consists of a set $S$ of $N$ *nodes* linked by an interconnect in a shared-nothing[50] manner. For the purpose of this work, we define a single node as a high-level abstraction over a unit of the storage subsystem, which is likely to be a collection of physically attached storage devices like a disk array. This unit of storage subsystem can be associated with server, or it can be stand-alone as a network attached peripheral[31]. This notion of node is a generalization of the terms *disk striping group* and *logical disk* in [56] and [12] respectively.

Each node $x \in S$ has a finite storage capacity, and a finite service capacity $B_x$, in units of movie-access streams. If a node $x$ exists for the sole purpose of providing data storage only, for example, a disk array with a controller attached to the network, $B_x$ simply represents its bandwidth. If a node $x$ is associated with server hardware, i.e., it is a VOD server with local storage, $B_x$ would indicate the actual throughput of the server, which is usually limited by a combination of the disk bandwidth and the server processing power, etc. Different nodes in this model can be heterogeneous in terms of their storage capacities and service capacities. This feature provides scalability by allowing the system to grow on a node basis, even if the specific type of hardware equipment that was used to make a node is no longer available from the market and has to be replaced by newer types (which have different performance characteristics). In this model, we do not require the presence of a tertiary storage such as a tape library. However, if a tape library is present, it is treated as a special node that has

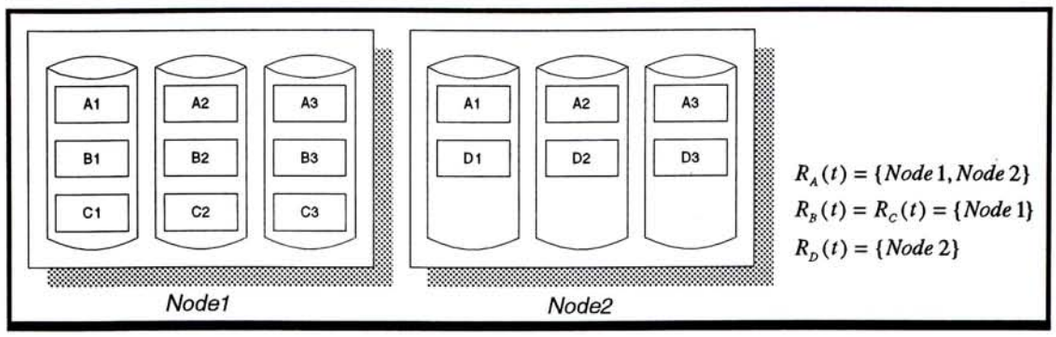


**Figure 3.1**: Example of a Two-Node System

very low service capacity, solely for providing movie data and does not serve customers.

The load on node $x$ at time $t$ is denoted by $L_x(t)$, again in units of movie-access streams. Multiple copies of the same movie $i$ may be stored on different nodes known as *replica nodes*. The set of these nodes is denoted by $R_i(t) \subseteq S$. Note that $R_i(t)$ is a function of time since in a system using dynamic replication, the number of copies of a movie may vary with changes in its popularity. Figure 3.1 shows a system with two nodes and four movies. There are two copies of movie A, each stored on one of the nodes. There is only one copy of movies B, C, and D. Each copy of a movie is striped across the three disks of a node[1].

Customers arrive to the VOD system at an average rate of $\lambda$. Upon a customer arrival at time $t$, there is a probability of $p_i(t)$ that movie $i$ is requested. The admission policy then examines the load of each node in the set $R_i(t)$. If sufficient capacity for servicing the new request is available on at least one of the nodes in $R_i(t)$, then the customer request is accepted by the system and assigned to the least-loaded node in $R_i(t)$. Otherwise the customer is rejected[2].

For ease of reference, the following table summarizes the notation used in this dissertation. Some of the notation given in the table will be defined later in the dissertation, as it is needed.

[1]Note that, the data represented by each of the rectangular blocks in Figure 3.1 is not necessarily logically-contiguous.

[2]Instead of allowing customers to queue for service, immediate rejection is adopted for clearer indication of the performance results as shown in Section 6.

| | |
|---|---|
| $A_i(t)$ | *available service capacity* of movie $i$ at time $t$ (in streams) |
| $B_x$ | maximum service capacity of node $x$ (in streams) |
| $L_x(t)$ | load (service capacity in use) on node $x$ at time $t$ (in streams) |
| $m_i$ | length of movie $i$ (in minutes) |
| $M$ | Set of movies in the system |
| $N$ | number of nodes, which is equal to $\|S\|$ |
| $p_i(t)$ | probability that a customer requests movie $i$ at time $t$ |
| $R_i(t)$ | set of replica nodes for movie $i$ at time $t$ |
| $S$ | set of all nodes in the system |
| $T_i(t)$ | *replication threshold* of movie $i$ at time $t$ |
| $\alpha$ | source node for replication |
| $\beta$ | target node for replication |
| $\lambda$ | average arrival rate to the system |
| $\phi$ | hard limit of replication streams |

# Chapter 4

# Replication Methodology

In this section we consider the problem of dynamic replication, that is, replication of movies on-the-fly when demand for them rises above the level that the existing configuration can support. Replication of a movie involves copying of that movie from a *source node* to a *target node*. There are a number of issues involved in such dynamic replication of movies. These issues include:

- When replication should be invoked.
- Selection of a movie to replicate.
- Selection of a source node and a target node for replication.
- How replication should be performed.
- When and how *dereplication* (removal of a replica) of a movie should be done.

Some of these issues are analogous to those in the field of task migration for processor load balancing[8]. Generally speaking, in the processor load balancing problem, a task can be executed by any node (perhaps at different costs). In VOD systems, however, an additional level of complexity is present due to the fact that only a subset of nodes can serve requests for a movie (i.e., a request for movie $i$ can only be serviced by the nodes in $R_i(t)$). In this chapter the primary focus is on dynamically changing the membership of this subset through replication and dereplication.

## 4.1 Replication Triggering Policy

We adopt a threshold-based policy for triggering movie replication. The main motivation for using a threshold-based approach is that there is a non-negligible cost for creating or removing a replica. And, as in most practical situations, an important concern is not only the system performance but rather its cost/performance ratio. More specifically, under light loads, it is not desirable to create a replica unless there is a sufficient demand that would justify the cost of its creation. One approach to improving the cost/performance ratio of a system is to react to changes in workload through the use of threshold-based policies.

For the following discussion, we define the *available service capacity* for movie $i$ at time $t$, $A_i(t)$, to be:

$$A_i(t) = \sum_{x \in R_i(t)} (B_x - L_x(t))$$

We would use the notion of available service capacity in defining a threshold-based replication triggering policy.

The aim of replication is to allow more nodes to handle requests for viewing particular movies when necessary, thereby decreasing the number of customer rejections. Therefore, when the available service capacity $A_i(t)$ for movie $i$ offered by its replica nodes $R_i(t)$ is too low, it is time to create an additional copy of that movie. Thus we define the following policy for triggering replication.

When a customer request arrives for movie $i$ at time $t$, its replication is to be initiated if and only if all of the following criteria are met:

- $A_i(t) < T_i(t)$, where $T_i(t)$ is a *threshold* parameter in our replication algorithm. We propose a heuristic for setting $T_i(t) = \min(m_i p_i(t)\lambda, h\bar{B})$ in which $p_i(t)\lambda$ is an estimate of the arrival rate of movie $i$ according to statistics taken in the period of a certain length just before time $t$, and $\bar{B}$ is the mean service capacity of a node. The rationale behind the use of $m_i p_i(t)\lambda$ in the heuristic is that we need at most $m_i$ time to replicate a movie to gain additional service capacity for movie $i$.

If the current available service capacity is lower than $m_i p_i \lambda$, the expected number of arrivals during replication, there may be some rejections[1]. Since this heuristic is designed to trigger replication early enough to reduce customer rejections, for popular movies it may yield threshold values that are larger than the full service capacity of a single node, preventing the addition of available service capacity brought by replication from satisfying the needs of service capacity for a movie. Therefore, the *threshold limit* parameter $h$, where $0 < h < 1$, is introduced to control excessive replication of very popular movies.

- The system is not currently replicating movie $i$.
- Suitable source and target nodes can be found according to the policies described in the following section.

## 4.2 Source & Target Nodes Selection Policies

The source node selection policy is simple — we choose the least-loaded node that contains a copy of the movie to be replicated. The rationale for this policy is that some of the replication policies (to be described in the following section) may result in an additional load on the source node. Thus to reduce the probability of replication interfering with the normal workload (i.e., with future requests for movies), we choose the least-loaded of the possible source nodes.

The target node selection policy is somewhat more complicated. A target node should be a *lightly-loaded* one, otherwise there might be little additional service capacity to be gained for a movie after it has been replicated. Since we classify a node into two states of lightly-loaded and heavily loaded, we divide the load information of a node into two halves equally to determine which state a node belongs to, i.e., we define a node $x$ to be lightly-loaded when its load $L_x(t) < \frac{B_x}{2}$. Three target node selection policies are studied in this section. All of them choose a node from the set

[1]The actual situation is much more complicated, due to customer departures, the existence of other movies in the same node and that the replication time can usually be reduced (to be described in Section 4.3). As a tradeoff for simpler computation, the proposed heuristic does not take these factors into account.

of lightly-loaded nodes, which do not contain a copy of the movie being replicated to avoid duplicating movies in the same node:

1. Randomly choose a node from this set.
2. Choose the least-loaded one. This policy suffers from a problem of *competition for service capacity* by multiple movies which have just been replicated, because simultaneous replication of different movies to the same target node would often occur under this policy.
3. Choose one which has the highest *estimated residual capacity*, which is defined to be $\frac{B_x - L_x(t)}{1+\sigma_x(t)}$, where $\sigma_x(t)$ is the number of movies being replicated to node $x$ at time $t$. Intuitively, this policy avoids simultaneous replication of different movies to the same target node, if possible. It thus reduces the likelihood of competition for service capacity. The constant 1 in the denominator is needed due to the possibility of $\sigma_x(t) = 0$, for a node which does not have any replication being carried out.

## 4.3 Replication Policies

The basic mechanism for performing replication is to use residual service capacity at a node to *inject* additional streams for reading/writing of data. These replication streams read/write movie data at the same rate as normal playback streams read movie data. They differ from the playback streams only in two aspects:

- They are for the VOD systems' internal use instead of serving the customers.
- Some replication streams write to the nodes, while all normal playback streams only read from the nodes.

Aside from these two differences, no extra support is needed from the VOD system for scheduling of resources for these replication streams.

As replication of movie $i$ would be triggered by a customer arrival for movie $i$, the resources allocated for servicing that customer request can be used for replication as well by multicasting the movie data to the customer and the target node, i.e., we can *piggyback*[33] upon a customer which is currently viewing the movie we want to replicate. This way, the reading part of the replication does not require injection of an additional stream at the source node, and thus replication through piggybacking does not create much additional load at the source node.

Replication of a movie through piggybacking or injection of a single stream at the source node, termed as *sequential replication*, can take on the order of 1-2 hours (i.e., the duration of the movie). In what follows, we consider approaches to reducing of movie replication time. Reduced replication time should lead to a reduction in the number of occurrences of customer rejection. To this end, we propose *parallel replication* strategy — such strategies utilize multiple concurrent streams for replicating different parts of the same movie so as to shorten the replication time. For example, if the system can support three additional streams at both the source node and the target node, a 90-minute movie can be replicated by injecting three concurrent streams for replicating three different 30-minute portions. The overall replication time is thus reduced to 30 minutes.

In the following sections, several policies for carrying out the actual replication process are described, in ascending order of complexity:

1. Injected Sequential Replication — replication is performed by injecting a single stream at both the source and the destination nodes.

2. Piggybacked Sequential Replication — replication is performed by piggybacking on an existing stream at the source node (i.e., the one that triggered the replication) for reading of the movie data, and injecting a stream to the target node for the writing part.

3. Injected Parallel Replication — replication is performed by injecting multiple streams at both the source and the target nodes; the number of injected streams is limited by the available capacities of the source and the target nodes. This

parallelism reduces replication times needed.

4. Piggybacked Parallel Replication — replication is performed by piggybacking on multiple existing streams at the source node and injecting a single stream at the target node.

5. Injected & Piggybacked Parallel Replication — this policy is a hybrid of the two preceding ones; replication is performed by piggybacking on as many streams as possible (using the algorithm from the previous policy), and the remainder is replicated through injecting addition streams at the source node.

6. Multi-Source Injected & Piggybacked Parallel Replication — this policy is an extension to the previous one, by letting multiple source nodes to participate in a replication session. This policy offers the greatest parallelism, and thus yielding the shortest replication times among others.

These replication policies, based on different combinations of the techniques, can be classified as follows:

| | Sequential | Single-Source Parallel | Multi-Source Parallel |
|---|---|---|---|
| Injected | Policy 1 | Policy 3 | - |
| Piggybacked | Policy 2 | Policy 4 | - |
| Hybrid | - | Policy 5 | Policy 6 |

The overhead of each of these policies, in terms of the service capacity involved, and the time needed for replication are evaluated, along with considerations for VCR functionalities (pause, fast forward, backward, etc). In addition, some strategies for utilizing nodes which have not completed replication are discussed.

### 4.3.1 Policy 1: Injected Sequential Replication

The basic replication policy injects one movie data reading stream to the source node, and one movie data writing stream to the target node. They copy the movie data from

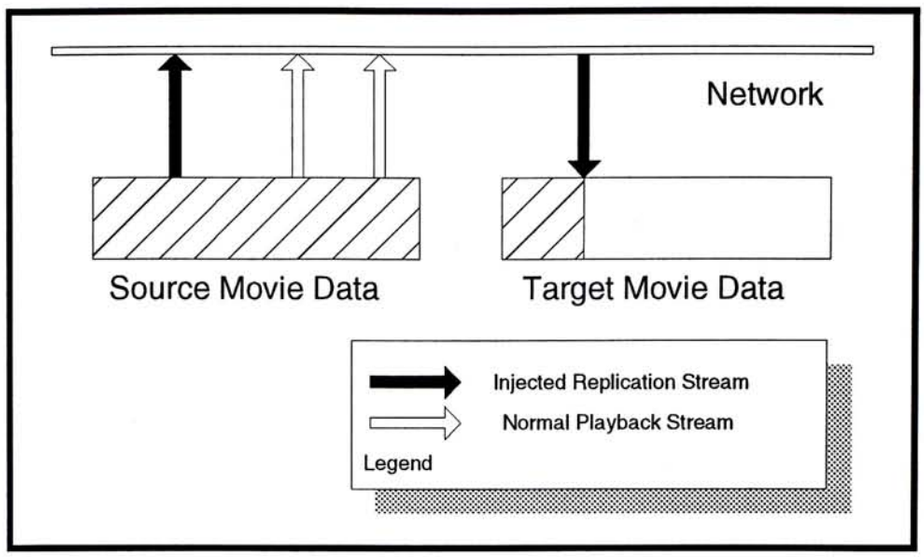


**Figure 4.1**: Injected Sequential Replication

the beginning to the end sequentially at the same rate as a customer views a movie. The overhead would thus be one stream for each of the two nodes. The replication time is simply $m_i$, the length of movie $i$. Figure 4.1 shows a movie at the source node, which is being viewed by two customers. An injected replication stream reads data from the source node. The movie data is transferred across the network and written to the replica copy at the target node by another injected replication stream there.

A variation of this policy is to allow the replication streams to proceed faster than the normal playback speed. This would shorten the replication time, but with a tradeoff of increased overhead. For a data access speed of $r$ times the normal rate, the overhead would be $r$ streams to both nodes, with the replication time shortened to $\frac{m_i}{r}$. This increased rate may cause a significant problem for disk scheduling, however. Usually, scheduling for concurrent streaming access to movies stored on a disk array would depend on the access speed, to satisfy the real-time constraints of delivery of the movie data. When the access speed of a movie is changed, it may disrupt the scheduling of other movies being accessed in the same disk array. Of course, the scheduling algorithm may be changed to cope with such high-speed replication streams, but such changes bring some costs. In Section 4.3.3, we study a different replication policy that provides the same performance benefits as this high-speed sequential replication does, but does not necessitate a change in the scheduling algorithm in use.

Assuming the customers playback movies sequentially from the beginning to the end, it is possible to direct new arrivals for a movie which is being replicated to the target node, even if the replication has not been completed yet. This is because the replication has been started from the beginning of the movie, and both the replication and the playback proceed at the same rate. By the time the customer needs to read a block of movie data, the block would have already been replicated under the sequential viewing assumption. We call this *early acceptance* by the target node. With early acceptance, the system can make use of a replica quickly.

Early acceptance has a drawback, however. Most of the time early accepted customers would be viewing incomplete copy of movies, so no fast forward beyond the replicated portion is possible for them. We call the position beyond which no fast forward can be made a *fast forward limit*. Figure 4.2 shows two early accepted customers viewing an incomplete copy of movie, which is being written to the storage by an injected replication stream. It can be seen from the figure that the two early accepted customers cannot fast forward to the second half of the movie, because that part has not been written to the storage yet. This fast forward limit is time-varying because the replication process will continue until it is completed. A solution to this problem is to migrate the customer, who has issued a fast forward request, to another node which has a complete copy of the movie. If no other node can serve it, the fast forward request has to be rejected.

Other VCR functions such as backward and pause operations do not cause problems for early accepted customers, under this replication policy.

### 4.3.2 Policy 2: Piggybacked Sequential Replication

By making use of *piggybacking*[33] and exploiting the sequential pattern of movie viewing, movie data can be read from the source node, but without incurring extra load to the source node as in the previous policy, in terms of movie-access streams. With this policy, the system piggybacks upon the the customer who has triggered the replication, i.e., when some movie data is read for playback by the customer, it is multicasted to

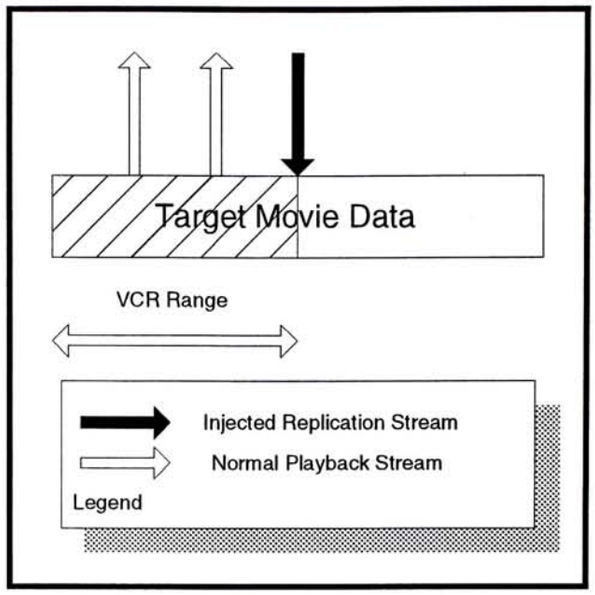


**Figure 4.2**: Early Acceptance

the target node for replication as well. Figure 4.3 shows four customers viewing a movie, one of which has been chosen for piggybacked replication of that movie to the target node.

Piggybacking has one drawback, however. If the piggybacked customer performs any VCR function (pause, fast forward, etc). the replication will be affected. For example, if the customer performs a pause or a rewind operation, the replication should be paused, until the customer resumes normal playback. If the customer performs a fast forward operation, we may setup the target node to switch to a fast writing mode, provided that the target node has enough residual service capacity for this. This way, the replication will be done sooner than the normal case. If the target node does not possess enough service capacity for replication in a fast forward mode, or the customer stops viewing the movie, the most sensible way to deal with such situation is to cancel the piggybacking operation, and inject a replication stream for the reading. Actually, this approach may be employed for the occurrence of all other VCR functions that act on a piggybacked customer as well. In case the source node runs out of service capacity, there are two recovery procedures to choose from:

- Cancel the replication altogether. Obviously, the effort previously spent on repli-

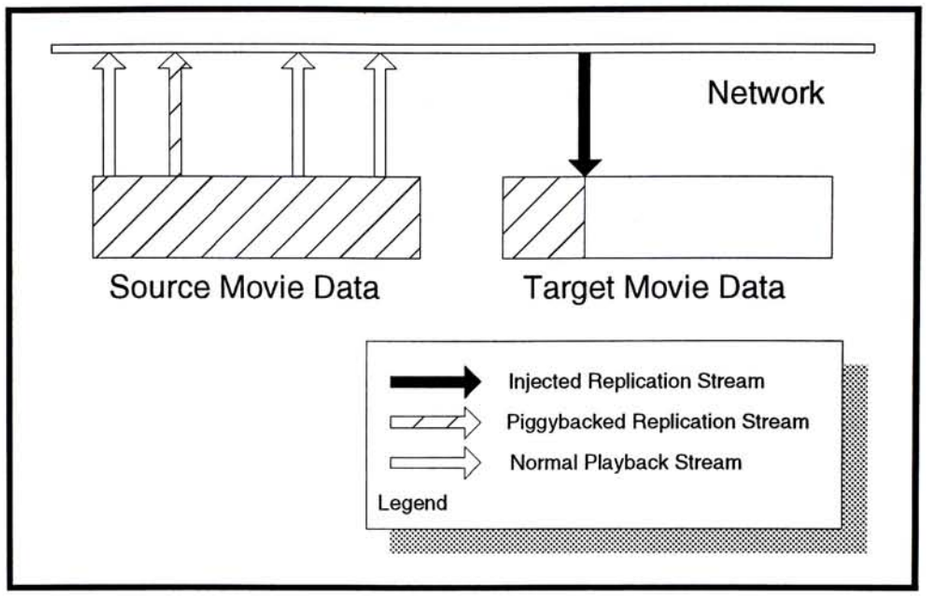


**Figure 4.3**: Piggybacked Sequential Replication

cation would be wasted.

- Wait for availability of service capacity for injection of the extra replication stream, before continuing the replication process. To avoid the resources, which are being held at the target node for replication, from becoming unavailable for too long, this waiting time should be bounded. The replication is canceled only after the waiting period is over.

This drawback of piggybacking causes another problem with early acceptance. In the event that the system is waiting for service capacity availability, or the replication has been canceled, if there are some early accepted customers at the target node, the customers may reach the end of the incomplete copy of the movie being replicated. Beyond this point no movie data is available to continue the movie playback. Although migrating the customer to other nodes would allow the continuation of the movie playback, migration cannot be done when all replica nodes of the movie run out of residual service capacities. To prevent this serious problem, the system may reject requests for VCR functions from customers who have been piggybacked upon for replication. Since there can be a potentially large number of customers who have been piggybacked, allowing such rejection is going to affect a lot of customers, so it is not desirable. Therefore, under this replication policy, it would be better to simply

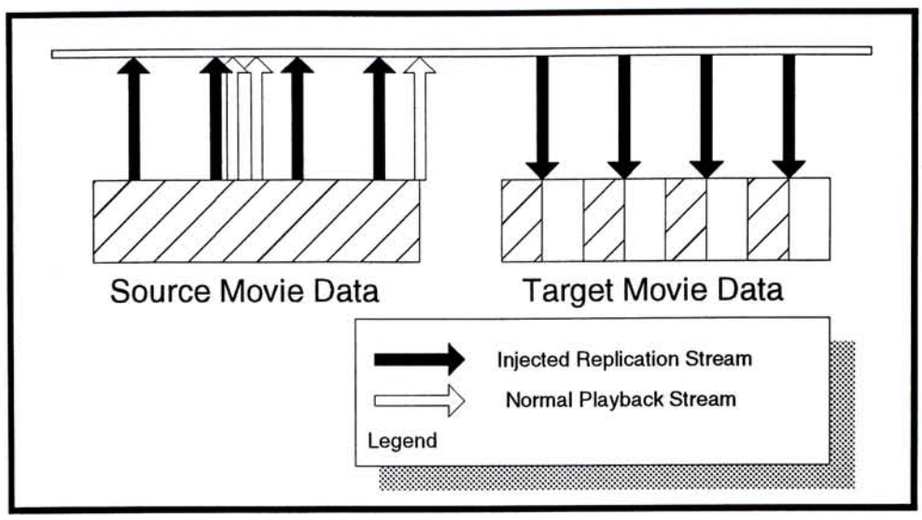


**Figure 4.4**: Injected Parallel Replication

abandon the use of early acceptance.

### 4.3.3 Policy 3: Injected Parallel Replication

The motivation behind *parallel replication* is that by using multiple streams for replication of a movie, the replication time can be reduced greatly. In Figure 4.4, there are four pairs of injected replication streams for the reading/writing of movie data, along with three customers viewing the movie at the source node. Since these four replication streams perform the copying process concurrently, it only takes one-fourth the time of the movie duration for the replication.

To carry out parallel replication, consider that if both the source node $\alpha$ and the target node $\beta$ can support $r$ additional streams (i.e., $r = min(B_\alpha - L_\alpha(t), B_\beta - L_\beta(t), \phi)$, the data of a movie is divided into $r$ portions regularly. Then $r$ streams are injected to both nodes, with each stream responsible for one of the portions. The parameter $\phi$ is a hard limit of the number of replication streams, which allows control of the tradeoff between using many resources to shorten replication time and leaving more resources for servicing customers in the near future, while replicating for a long time.[2] The introduction of parameter $\phi$ allows us to perform experiments to find out

[2]We assume there is a non-negligible cost to transfer resources previously allocated to replication of a movie to a newly arrived customer for a different movie.

whether spending resources on shortening replication times is worthwhile or not, in Section 6.3.3. Note that when $\phi = 1$, this policy falls back to become a sequential policy.

The $r$ added streams represent overhead to both nodes, and by the time $\frac{m_i}{r}$ all streams would have completed their replication for movie $i$ simultaneously. This amount of overhead and the time needed for replication are the same as that of sequential replication proceeding at a rate of $r$ times the normal speed. However, as explained in Section 4.3.1, high-speed sequential replication might incur scheduling complexities with disk arrays. Parallel replication avoids this problem while maintaining the same performance, because the multiple streams involved in parallel replication are replicating non-contiguous parts of the movie, thus it is very unlikely for them to incur extra load to the same disk in a disk array with striped[6] movie placement.

With multiple replicas of the same movie in the system, it is possible to perform *load balancing* by migrating customers from one replica to another. With parallel replication, there can be two types of customer migration:

- Regular migration — migration of customers to a complete copy of movie replica.
- Early migration — migration of customers to an incomplete copy of movie replica.

The subject of regular migration has been explored in [56]. So we would consider the issues of adopting early migration here only. Early migration possesses some restriction on when the migration can take place, because only certain parts of the movie is present at the target node, at a given time before the replication is completed. The only moments, when such migration is possible, are during the times existing customers viewing the movie at the source node reach the replication portion boundaries. They can then be migrated to the target node to continue the movie playback, because the beginning of the succeeding portion have already been replicated, if such a move is determined to be desirable (say, when the source node is highly loaded and the target node is not). In this way, a lot of existing viewers can be migrated to the replica node, thus quickly reducing the source node load.

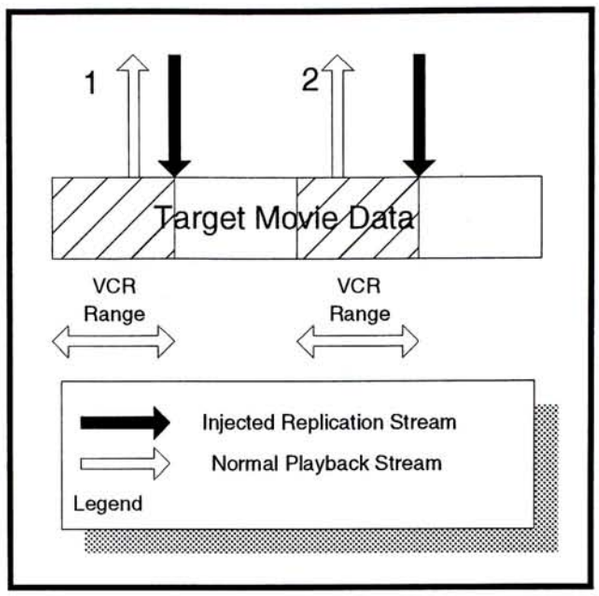


**Figure 4.5**: Early Migration

To estimate how many customers can be early migrated, consider that over a period of $T$, a single replication stream would have replicated $\frac{T}{m_i}$ fraction of the movie i. Since the replication rate is equal to the customer's viewing rate, this expression is also equivalent to the fraction of the customers who have reached the portion boundaries over the period $T$, assuming the customers are spread more or less evenly through the movie, or $T$ is sufficiently large. In other words, there would be $\frac{rT}{m_i}$ fraction of customers reaching the portion boundaries. However, for the first replication stream, since it is replicating from the beginning of the movie, customers reaching its starting boundary are actually new arrivals (under the early acceptance strategy) instead of existing ones. Therefore, the customers reaching the beginning boundary (under early acceptance) should not be counted as customers that can be early migrated, thus on average $\frac{(r-1)T}{m_i}$ fraction of the existing customers viewing the movie at the source node can be migrated, after the replication process has been started for a duration of $T$.

Early acceptance behaves like a special case of early migration, since early accepted customers are also viewing incomplete copies of movies. Recall that early accepted customers will cause problem if they try to issue VCR functions. Likewise, early migrated customers would face similar problems as well. In early migration, the problems are more serious, because early migrated customers would not only see a fast forward limit,

but also a *backward limit* as well. This limit is the beginning boundary of the replication portion they belong to, because the ending part of the preceding boundary would not be available until all the streams have completed their replication. This situation is depicted in Figure 4.5, in which there is an early accepted customer number 1, and an early migrated customer number 2, while two replication streams carry out parallel replication. Customer number 1 has a fast forward limit. Customer number 2 has both the fast forward limit and the backward limit. In these cases, the only solution would be migrating the customers to another node with a complete copy of the movie. If no node can serve them, their VCR requests have to be rejected. Note that there is no problem for pausing or stopping by migrated customers.

### 4.3.4 Policy 4: Piggybacked Parallel Replication

By applying the piggybacking strategy to parallel replication, reading of the movie data can be done without injecting replication streams to the source node. To adopt this strategy, a number of existing customers viewing the movie to be replicated are chosen for piggybacking. Instead of having uniform splitting of movie into replication portions for different streams as in the previous policy, dependence on existing customers for reading implies that the customers' progress define the boundaries among the different replicating portions in this policy. The chosen set of customers are responsible for reading of movie data from their current positions up to those of the succeeding ones (or the end of the movie for the final one). For complete replication of the movie, this set of customers must include the one who has triggered the replication, since it is trying to playback the beginning part of the movie. In Figure 4.6, there are six customers viewing a movie at the source node, three of which have been chosen for piggybacked replication. Note that the piggybacked customers' distribution are much less uniform than the injected streams in the previous policy (Figure 4.4).

A difficulty with piggybacked parallel replication lies in the selection of customers to be piggybacked. In the ideal case, customers viewing the same movie would be spread out evenly with respect to their progress, because it would then be possible to choose customers in a way that mimics the regular splitting strategy as used in the previous

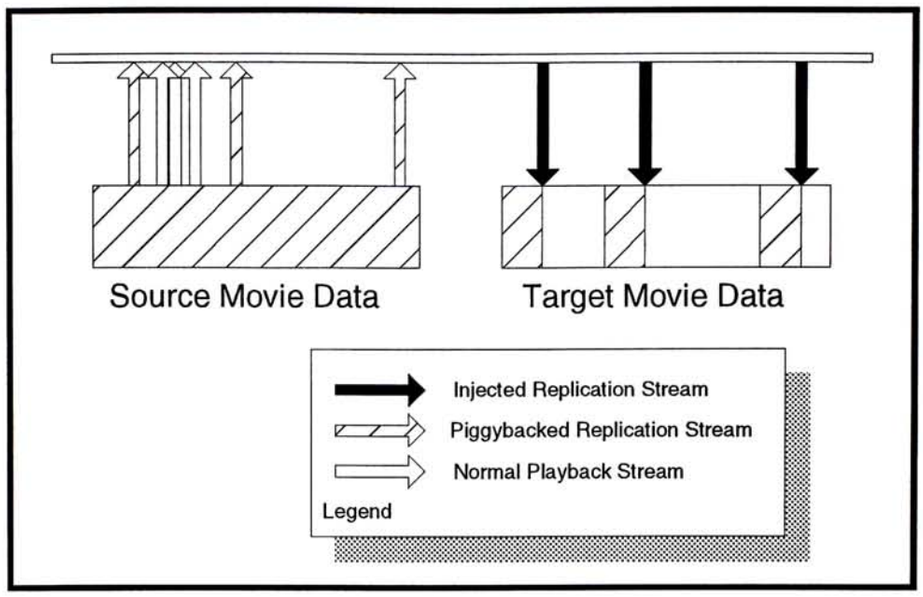


**Figure 4.6**: Piggybacked Parallel Replication

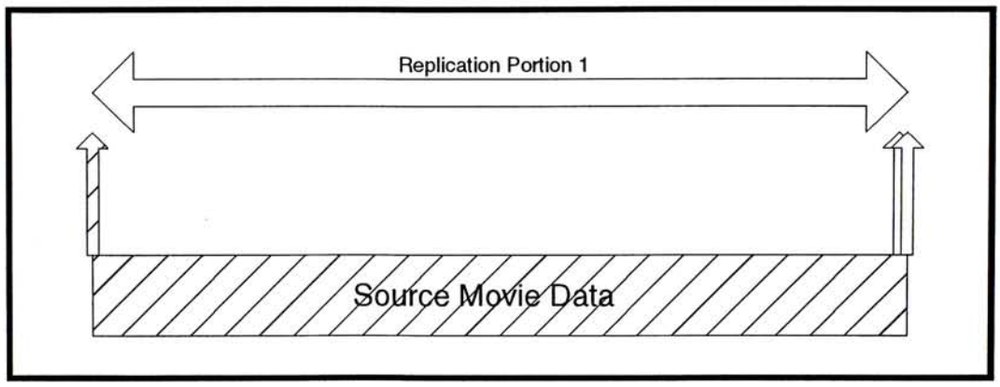


**Figure 4.7**: Worst Case for Piggybacked Parallel Replication

policy, i.e., all streams replicate the same amount of the movie and complete at the same time. In the worst case as depicted in Figure 4.7, the customer who triggered the replication would be at the beginning, but all remaining customers would be near the end of the movie. Thus one piggybacked stream would be responsible for nearly the whole movie, and other streams would be responsible for negligible portions only. This is the worst scenario because even if there are many replication streams, one of them dominates the replication time. Effectively it falls back to the performance level of a sequential replication process. Another difficulty lies with the use of a least-loaded source node selection policy described in Section 4.2. The least-loaded node inherently possesses fewer customers that can be chosen for piggybacking.

As the replication time is determined by the longest portion that needs to be replicated by one stream, the customers should be chosen in a way that they are as evenly spread as possible, yielding portions of similar lengths. To achieve this, it is necessary to find a set of customers in a way that the differences among the positions of consecutive ones are more or less similar, or in other words, a way that reduces the maximum of such differences.

**Lemma 1** An exhaustive search for the set of $r$ customers among $n$ customers who are viewing a movie, such that the $r$ customers are most evenly spread, takes $O(\frac{n!}{r(r-2)!(n-r)!})$ time.

**Proof** There are $C_r^n = \frac{n!}{r!(n-r)!}$ possibilities of choosing $r$ customers among $n$ customers. For each of these possibilities, it takes $r-1$ subtractions to determine the distance among all pairs of consecutive customers. It thus takes $O(\frac{(r-1)n!}{r!(n-r)!})$ to perform an exhaustive search. ∎

By Lemma 1, an exhaustive search for the best choices of piggybacking customers is computationally intensive. For example, if there are 25 customers viewing a movie at a node, and we want to choose 10 customers for piggybacking, it would need more than 29 million subtractions to find out the best choices. Given that an exhaustive search for customer selection for replication possesses a high complexity, it may violate the real-time constraints of delivery of movie data in a VOD system. Therefore, we propose an algorithm for customer selection that is effective but possesses a much lower complexity:

$r^* := \min(B_\beta - L_\beta(t), \phi)$
Divide the movie to be replicated into $r^*$ regions regularly
For each region
    if the region is not empty of customers
        choose the earliest customer in that region

The variable $r^*$ represents the maximum number of possible streams that can be gen-

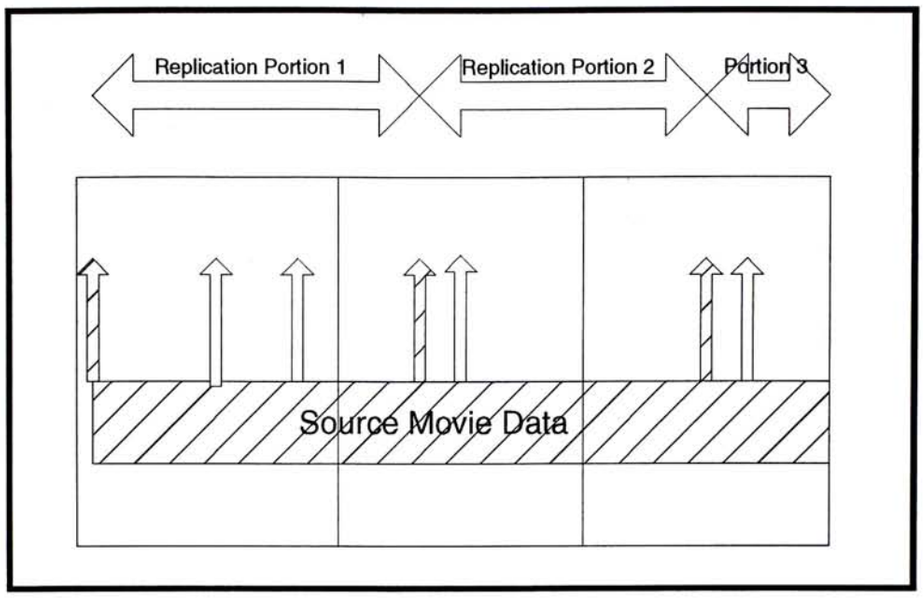


**Figure 4.8**: Choice of Customers for Piggybacking

erated for replication, since a target node $\beta$ for replication cannot support more than $B_\beta - L_\beta(t)$ additional streams, and we impose the hard limit $\phi$ on the number of replication streams, as in the previous policy. Note that the number of customers chosen by the above algorithm, $r$, always satisfies $r \leq r^*$, such that the replication process would not overload the nodes. The customer who has triggered the replication is the earliest one in the first region, so this customer would always be chosen for piggybacking. A straight forward implementation of the selection scheme would involve checking the progress of all the $n$ customers viewing the movie to be replicated at the source node, this algorithm possesses a lower complexity of $O(n)$, as compared to that of an exhaustive search. The effect of this algorithm is illustrated in Figure 4.8. In this figure, the movie is equally divided into $r^* = 3$ regions. In the first region, the customer who has triggered the replication replication is chosen for piggybacking. In the other two regions, the earliest customers in their respective regions are also chosen.

With the use of piggybacking, this policy does not incur extra load to the source node, in terms of movie-access streams. The overhead for the target node would be $r$ initially for the writing of movie data. It will decrease over time when some streams complete earlier than others do. For example, in Figure 4.8, the third piggybacked replication stream would finish earlier than the first two, because the third replication

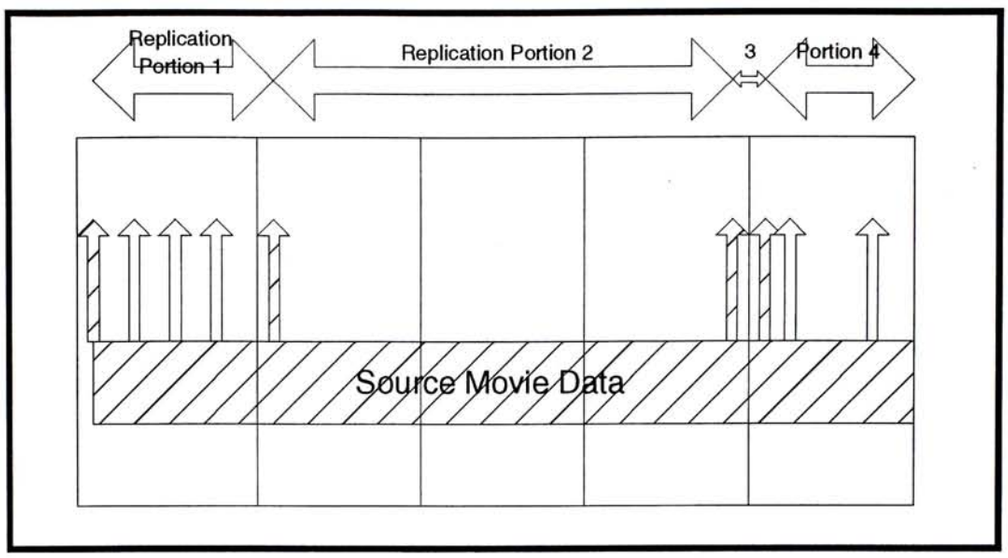


**Figure 4.9**: Illustration for Lemma 2

portion is much smaller than the others. Unlike the previous injection-based policy, there is no way a high-speed sequential replication process can achieve the same low level of overhead offered by piggybacking, for the same target performance level with respect to replication time.

The replication time depends on the largest replication portion, which depends on the progress of the piggybacked customers at the time they are chosen, since the longest portion would dominate the replication time. To find out the replication time needed, consider a snapshot of customers' progress viewing movie $i$ at node $x$ at time $t$, in which there is $U_{i,x}(t)$ empty consecutive regions.

**Lemma 2** The replication time of a movie $i$ under the piggybacked parallel replication policy, $\Upsilon_i(t)$, satisifies $\Upsilon_i(t) \leq (U_{i,x}(t)+2)\frac{m_i}{r^*}$, where $U_{i,x}(t)$ denotes the largest number of consecutive regions present in the replica of movie $i$ residing at node $x$ that are empty of customers, $m_i$ denotes the length of movie $i$, $r^*$ is the maximum number of possible streams that can be generated for replication, in consideration of a snapshot of customers' progress viewing movie $i$ at time $t$, the instant when replication is triggered.

**Proof** In the worst case, the piggybacked customer preceding the empty regions would be at the beginning of the (non-empty) region it belongs to, and the succeeding

customer would be at the end of its region. The replication stream that is responsible for the largest part of the movie thus involves the $U_{i,x}(t)$ empty regions, plus two regions for the two customers bounding these regions. Each region takes $\frac{m_i}{r^*}$ time to replicate. Therefore the total replication time is bounded by $(U_{i,x}(t)+2)\frac{m_i}{r^*}$. ∎

For example, in Figure 4.9 four customers have been chosen for piggybacked replication of a movie divided into $r^* = 5$ regions, one $(= U_{i,x}(t))$ of which is empty of customers. The second piggybacked customer is responsible for replicating the greatest portion, which spans its own region, the empty region, and almost all of the succeeding region, because the succeeding piggybacked customer is located near the end of its own region. In fact, the third piggybacked customer in this figure is responsible for a negligible portion only.

It can be seen that the actual replication time depends on the customers' progress distribution, which affects $U_{i,x}(t)$, at the instant replication is invoked. Therefore, a disadvantage inherent to pure piggybacking is that it is not always possible to have enough suitable customers to be piggybacked upon to create a great number of replication streams, even if there is enough service capacity at the target node for injection of corresponding replication streams for writing of the movie data. This limitation is immediately apparent when we consider that no more than one piggybacked stream can be created, if there is only one customer viewing the movie. It implies that under this policy, replication time cannot be made as short as possible with uneven customers' progress distribution.

Just like that piggybacked sequential replication policy causes problems with early acceptance, this policy causes problems with early migration. When the customers, who have been chosen for piggybacking, issue VCR function requests, the early migrated customers may experience problems of movie data unavailability. The considerations in this case are essentially the same as the case for early acceptance in piggybacked sequential replication. Therefore, under the piggybacked parallel replication policy, it is not desirable to perform early migration.

### 4.3.5 Policy 5: Injected & Piggybacked Parallel Replication

In view of the inherent disadvantage associated with piggybacking in the previous piggybacked policy — possible lack of customers or uneven distribution of them to limit the number of replication streams, and hence the replication speed, we combine the piggybacked policy with the injected policy to create a hybrid one. The advantage of this hybrid policy is that when the piggybacking component is ineffective, due to uneven distribution of customers' progress of viewing a particular movie, this policy can make use of the available service capacity to speed up replication with the injection component. In such situations, it injects replication streams to the source node for reading, instead of depending completely on piggybacking.

This combined policy is carried out in two stages. In the first stage, we follow the piggybacked policy to obtain a set of $r$ customers for piggybacking. The second stage will be carried out if there are residual service capacities left at both the source and the target nodes. For the largest replication portion formed by the piggybacked customers' progress, an extra replication stream is injected into the middle for reading the movie data, cutting the length of the longest portion for replication by half. Figure 4.10 shows the effect of this policy. In this figure, four customers are viewing a movie, and two of whom have been chosen for piggybacked replication. The system has determined that it can support one extra injected replication stream, so one is injected at the middle of the two piggybacked replication streams, finally yielding three streams for replication.

The injection process in the second stage of this policy is repeated to generate no more than $\tau$ number of streams, where $\tau$ is:

$$\tau = \min(B_\alpha - L_\alpha(t), B_\beta - L_\beta(t) - r, \phi - r)$$

The number of injection streams, $\tau$, at the source node is limited by the available residual service capacities at the source node, that at the target node, and the hard limit of replication streams respectively.

The overhead for the source and the target node would be $\tau$ and $\tau + r$ streams

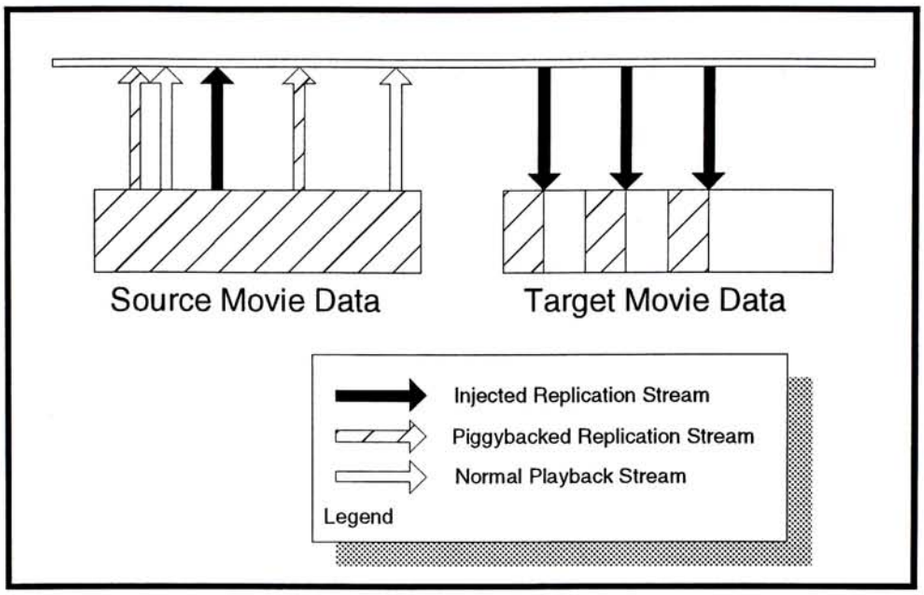


**Figure 4.10**: Injected & Piggybacked Parallel Replication

respectively initially. As the replication streams would be responsible for different lengths of portions of the movie, they would complete at different times and thus the overhead would decrease gradually.

Regarding the fast forward/backward problem associated with early migration used together with piggybacking, as described in the previous section, this policy provides an opportunity to address the problem. A few injected replication streams can be reserved for the situation when some piggybacked customers request VCR functions. In this case, the reserved streams then replace the piggybacked customers in their role of replicating movies. Basically, this strategy represents a tradeoff between the highest possible degree of concurrency offered this replication policy and an ability to deal with VCR functions of piggybacked customers. However, this strategy does not completely eliminate the problem, when the number of piggybacked customers who issue VCR functions is larger than the number of reserved injected replication streams.

Since this replication policy also solves the uneven customers' progress problem by allowing a fall-back to injection-based replication, its success is dependent on the presence of residual service capacities at the source node. However, since the need of replicating a movie implies that its available service capacity is low, it is quite likely that the source node would have very limited amount of residual service capacity only,

especially if the movie being replicated is very popular. In such cases, the parallelism offered by this policy will be limited, and thus the replication time cannot be decreased very much. The next policy attacks this low residual service capacity problem.

### 4.3.6 Policy 6: Multi-Source Injected & Piggybacked Parallel Replication

In order to have higher parallelism in the replication process even with limited residual service capacities at a single source node, we extend parallel replication to make use of multiple source nodes, such that many more replication streams can be created. This policy achieves shorter replication times than using a single node does, in two ways:

- The combined resources available in multiple nodes allow for more injection streams to be created. The increased degree of parallelism allows for shorter replication times.
- Use of multiple source nodes allows for all customers viewing the movie to be considered for piggybacking as compared to using a single node only, yielding more even distribution in terms of their progress. It thus also alleviates the problem of that a least-loaded-node source node selection policy does not favor piggybacking due to the the relatively low number of customers present in the least-loaded-node, as explained in Section 4.3.4. This better distribution of piggybacked customers would yield more uniform replication portions, giving shorter replication times.

To carry out replication of a movie $i$ with this policy, some or all nodes in $R_i(t)$, the set of nodes which contain the movie, are chosen to be source nodes. Piggybacking of customers from multiple source nodes is a direct extension of the original single-source method: instead of considering the customers in a single node for piggybacking only, all customers viewing the same movie in different nodes are considered. Then, replication streams are injected to all the nodes in $R_i(t)$, in a round-robin manner. The round-robin injection sequence has the advantage of spreading out the injection

load evenly, so the nodes have more residual service capacities for normal workload arriving during replication.

## 4.4 Dereplication Policy

Performing *dereplication* (removal of replicas) before the system runs out of storage capacity is necessary. The reason is that by the time the system runs out of storage capacity but finds that additional copies of a movie is needed, it may not be possible to begin replication immediately by simply overwriting an existing replica. This is due to the possibility of that some customers are viewing the replica to be removed. In such cases, those customers must be migrated to other nodes first. In general, two issues must be addressed: which movie to dereplicate and when to do it.

In our work, dereplication is invoked when the residual storage space in the whole system drops below a predefined level. When the popularity of a certain movie decreases, causing the number of replicas to be more than the currently needs, that movie becomes a suitable candidate for dereplication. Therefore, we first choose movie $i$, with the greatest value of $\frac{A_i(t)}{T_i(t)} > 1$. Then, we proceed to choose a replica of that movie to satisfy the following conditions:

- The remaining replica nodes possess sufficient residual service capacity to allow migration of all the existing customers viewing the replica at node $x$, i.e.,

$$\sum_{y \in (R_i(t) \setminus x)} (B_y - L_y(t)) > C_{i,x}(t)$$

  where $C_{i,x}(t)$ denotes the number of customers viewing movie $i$ at node $x$. As migration of customers could be expensive, it is desirable to choose a replica that possesses a lower value of $C_{i,x}(t)$.

- The system should not oscillate between replication and dereplication:

$$A_i(t) - (B_x - L_x(t)) - C_{i,x}(t) > T_i(t) + D$$

where $D$ denotes the *dereplication threshold* parameter. With the removal of a replica at node $x$, $A_i(t)$ would be decreased by $(B_x - L_x(t))$, the residual service capacity of node x. Since the existing customers viewing that replica to be removed would need to be served by other replicas, $A_i(t)$ would be further decreased by $C_{i,x}(t)$. The resulting $A_i(t)$ must be greater than the threshold $T_i(t)$, otherwise the system might in the near future replicate a movie that has just been dereplicated. To prevent the system from oscillating between replication and dereplication, a dereplication threshold, $D > 0$, is introduced to impose a margin between the triggering threshold for replication and that for dereplication.

# Chapter 5

# Distributed Architecture for VOD Server

In this chapter, we introduce a distributed architecture for VOD that is designed for scalability, and discuss the protocols for implementing VOD services and parallel replication on this architecture, with provisions for failure handling.

In VOD systems, any increase in the number of supported users/streams would incur significant impact to the VOD service, because support of video streams inherently need relatively large amount of resources. The hardware used by the VOD system imposes different kinds of limits, many of which actually directly influence the actual number of streams that can be supported, e.g. storage subsystem bandwidth, server-network interface bandwidth, server processing power, etc. Although these hardware limits can often be overcome by replacing the hardware with more advanced counterparts, such replacement is expensive in that the replaced equipment is wasted. The key to cost-effective scalability would thus be an architecture that enables the use of multiple server nodes. [1] However, with multiple server nodes, coordination and management becomes difficult, as compared to a centralized design.

To addresses this coordination problem among multiple shared-nothing[50] server

[1] For clarity of presentation, in this chapter we assume each node consists of a server which delivers movie data and local disk array for storage.

nodes, we describe the following architecture, which is scalable in terms of the increasing numbers of supported streams and that of movies. This is made viable by dynamic reconfiguration of the movie placement, through the parallel replication scheme discussed in the previous sections. To deal with the coordination problem in a scalable manner, our architecture employs a hierarchy (Figure 5.1) of:

- server node — at the bottom of the hierarchy, server nodes deliver movie data to clients;
- Movie Manager — at the middle of the hierarchy, Movie Managers manage information specific to a particular movie and control the server nodes which carry their movies;
- Metadata Manager — at the top of the hierarchy, Metadata Manager manages global information and all Movie Managers, and acts as a name service for clients to look up the suitable Movie Manager they need.

In the following sections, these components are described in more detail. Then, several protocols, comprising the basic operations of VOD services and parallel replication, designed for this architecture are presented.

## 5.1 Server Node

To achieve scalability and fault-tolerance in VOD service, a shared-nothing[50] architecture can be employed, i.e., we use a set of *server nodes*, each with its own storage subsystem such as a disk array. When the service capacity and/or storage capacity of the system needs to be increased, it can be done by adding a node to the system. The dynamic replication approach makes this architecture viable by on-the-fly readjustment of the placement of movies among the nodes. It also allows the different nodes to be heterogeneous in terms of service and storage capacity. Thus commodity products, which typically have relatively short life time in the market, can be used to build cost-effective VOD servers.

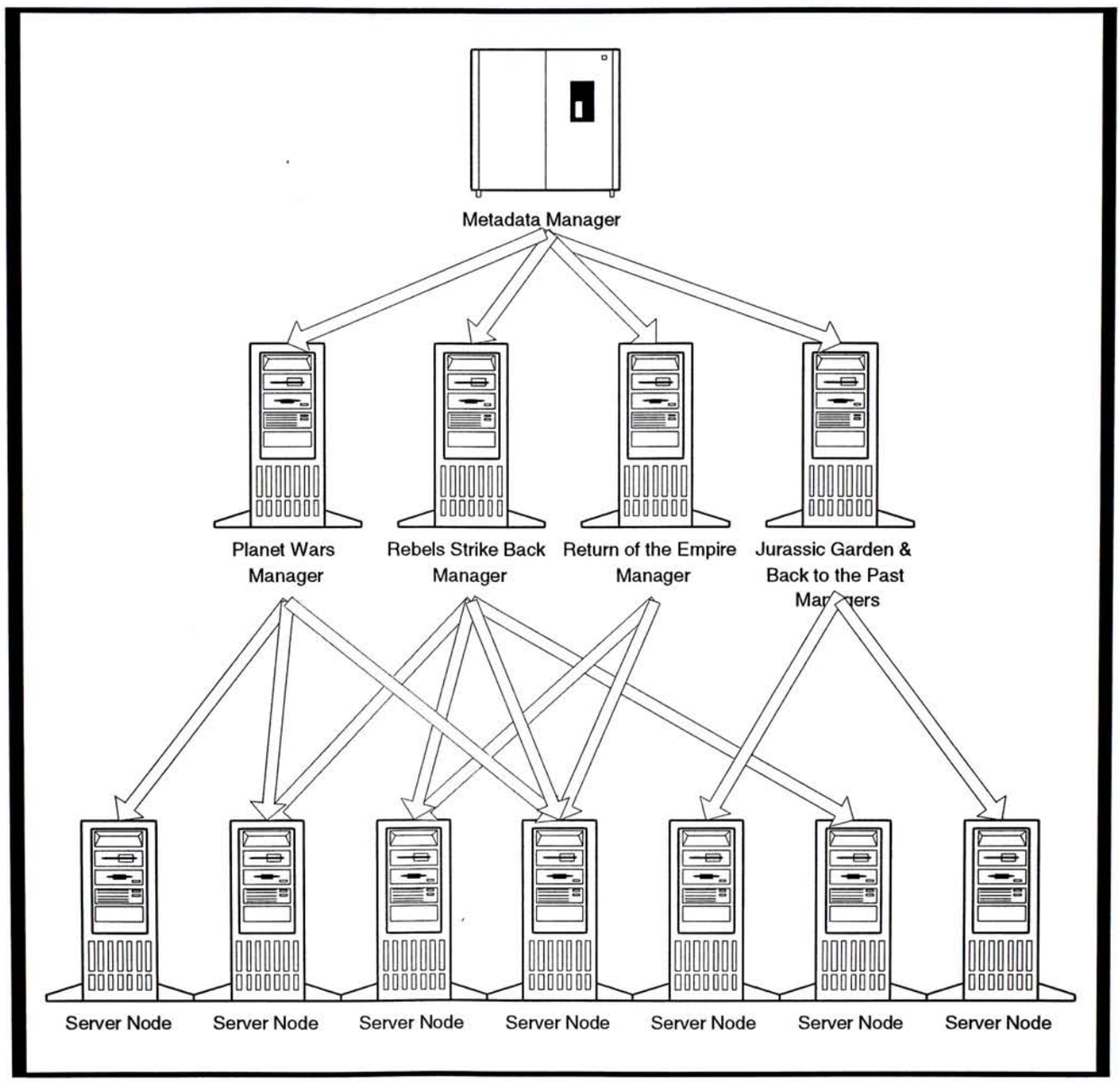


**Figure 5.1**: Hierarchy of VOD Server Components

## 5.2 Movie Manager

Since there may be multiple copies of the same movie residing in different nodes, there must be some sort of management structure that keeps track of movie replica location information to allow proper assignment of nodes to serve new requests. Also, such information is needed for making use of the redundancy to provide tolerance to server node crashes. On the other hand, a single centralized structure to keep track of all these information for all movies would limit scalability, and becomes a single point of failure. Therefore, we propose the concept of a per-movie-based logical component called *Movie Manager*. A Movie Manager for a particular movie is responsible for:

- keeping track of the number of replicas of movies, and the location of those replicas;
- keeping track of the load of the replica nodes;
- making server node assignment decisions for new customers;
- managing replication of the movie by sending control messages to server nodes;

By "logical component" we mean that we are concerned with the functionality of a Movie Manager only, not its physical location. Therefore, Movie Managers for all movies may physically reside in the same machine in a small-scale VOD system, or they may even reside on a server node. For a large-scale VOD system, each of them may reside in different machines, distinct from the server nodes, for good scalability and tolerance to machine failures. This location independence is achieved via the Metadata Manager component.

## 5.3 Metadata Manager

The Metadata Manager is conceptually located at the top level of our hierarchy of components. It keeps track of the location of all the Movie Managers present in the

system in form of a Movie Manager-address mapping table, and other global information. Whenever a new customer arrives for service, it sends a request to the Metadata Manager first. The Metadata Manager then assigns the proper Movie Manager for the requested movie. In this respect, the Metadata Manager behaves like a name server, and should have a well-known address for clients to contact. The Metadata Manager also maintains some information about the VOD system which is not specific to any particular movie. For example, the aggregate residual storage capacity of the system, which affects the triggering of dereplication, and the billing of customers, belongs to this type of information.

Although the Metadata Manager is a centralized structure, it does not hinder scalability in VOD applications because all the normal workload, i.e., movie data delivery, is done by the server nodes instead of the Metadata Manager. In normal situations, the Metadata Manager is mostly reponsible for name service and log for new customers (for billing, if applicable), which is negligible when compared with streaming data delivery by the server nodes.

## 5.4 Protocols for Distributed VOD Architecture

With the distributed architecture presented in the previous section, it is clear that a set of protocols for the interaction among different components of the architecture is needed to carry out various aspects of the VOD service, such as normal playback, VCR functions, parallel replication, etc. In this section, we will first describe the protocol for servicing a new request to illustrate how the different components tie together. Next, support for VCR functions is considered. Then, the protocol for dynamic parallel replication will be presented, along with that for dereplication.

### 5.4.1 Protocol for Servicing New Customers

When a customer wants to view a movie, essentially it issues a new request through the set-top box to the Metadata Manager, which forwards the request to the Movie

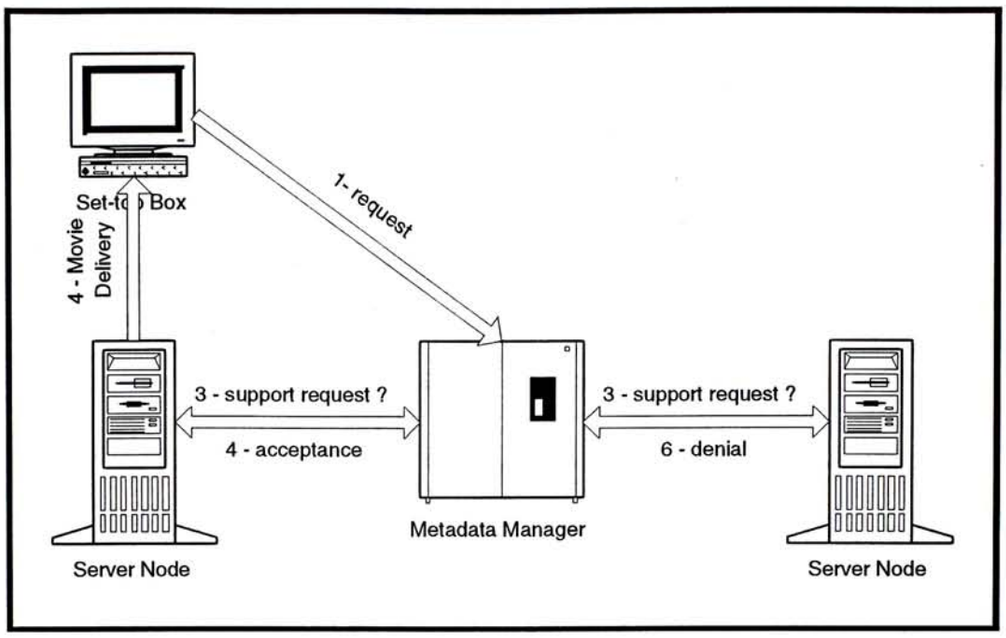


**Figure 5.2**: Servicing of a New Request

Manager corresponding to the requested movie. The Movie Manager then assigns a server node to the client. The followings describe this process (Figure 5.2) in more detail, with one message or one round of messages depicted as one point:

1. When a client issues the request, which specifies the movie to be viewed and the maximum waiting time for service availability tolerated, to begin viewing a movie, the request is sent to the Metadata Manager, which has a well-known address.

2. The Metadata Manager looks up the Movie Manager for the requested movie, then forwards the request there.

3. The Movie Manager maintains a cache for the load information about the replica nodes for its supported movie. With this knowledge, the Movie Manager chooses the least-loaded server node. To have resources allocated for the request and validate the cached load information of the chosen node, the Movie Manager sends a request for immediate viewing to the node.

4. If the node finds that it still has remaining service capacity for the request, it will reply with the updated load information and a globally and temporally unique

*key*. The key allows the client to be authenticated during subsequent contact of the server node (without going through Metadata Manager or Movie Manager again). The node can then start delivering the movie data to the client.

5. If it turns out that the server node has already run out of service capacity, on the contrary to the cached state known by the Movie Manager, the server node responds Movie Manager with a denial, together with an estimate of when service capacity will become available again when some streams finish.

6. The Movie Manager will then try other server nodes in ascending order of the cached load information.

7. When no acceptance is granted, the Movie Manager will choose those server nodes that offer waiting times acceptable by the client's acceptable limit. These nodes will be polled for reservation, but now in ascending order of the waiting times. The Movie Manager needs to poll these nodes, instead of just choosing the one which offers the shortest waiting time, because these server nodes' states could have changed just after they had sent their replies. Use of locking could have eliminated the need for this round of polling, but it would have restricted concurrency of handling of other requests by the server nodes, thus it is not preferred.

8. If an acceptance is granted from a node, the node carries out movie delivery like in step 4, after the needed waiting time expires. If the request is not fulfilled by this round of polling, or none of the server nodes offered acceptable waiting times, a rejection will be issued to the client.

### 5.4.2 Protocol for Servicing Existing Customers

When an existing customer performs a VCR function, or decides to stop the movie playback, it contacts the server node directly. This does not need to go through the Metadata Manager again, because the client set-top box already knows the address of the server node it is using. The actual process is:

1. The customer sends its request and key to the server node which has been serving it.

2. The server node authenticates the customer by looking up its hash table of valid keys.

3. If the authenticiation is successful, the server node checks whether the request can be fulfilled and performs accordingly.

4. If the request is to stop the movie playback, or the authenticiation fails, the Metadata Manager is informed to log this event.

### 5.4.3 Protocol for Single/Multi-Source Injected & Parallel Replication

The Movie Manager is responsible for replication, if needed. Basically, it performs the following tasks:

1. triggers replication when necessary;

2. chooses target & source nodes;

3. determines the number of streams used for replication;

4. chooses customers for piggybacking;

5. instructs the chosen nodes to carry out the replication accordingly.

The first step in this sequence, the replication triggering, is best done by the Movie Manager, since the Movie Manager for a particular movie would be contacted whenever that movie is requested. The Movie Manager is also the ideal candidate to record the statistical data of the requests and thereby estimate the movie popularity. The replication threshold is then calculated from the popularity estimate. Whenever the Movie Manager processes a new request in the aforementioned manner, it examines

the conditions for trigger replication, namely, comparing the threshold with the *available service capacity*, which is calculated from the load information cache. The cache contents will be updated by the server nodes, whenever their load changes by a certain amount, and also at the times the server nodes reply to the Movie Manager's requests with their updated load information. If replication is desired, the Movie Manager will proceed to initiate the replication sequence:

1. The Movie Manager first chooses a target node using a target node selection policy as described in Section 4.2, and finds out the number of streams the target node can support for parallel replication. The target node reserves those capacities for replication.

2. The Movie Manager then multicasts a load information query to the replica nodes of its movie. Depending on whether a multi-source replication or a single-source replication is used, one of the following ways is adopted to continue the replication initiation sequence:

   - Single-source replication: Among the set of server nodes which do not contain a copy of the movie being replicated, the Movie Manager chooses the least-loaded one as the source node, according to the target node selection policy. It instructs that node to begin replication, by informing it the maximum number of streams, as limited by the residual service capacity of the target node, to be used. The chosen source node carries out piggybacking customers selection, and injects an appropriate number of replication streams. Then it contacts the target node to prepare receiving of the replication streams.
   - Multi-source replication: The Movie Manager multicasts a query to all replica nodes of its supported movie, to obtain their viewers' progress information. With these information combined, the Movie Manager selects customers for piggybacking. Then it instructs the server nodes to piggyback the chosen customers, and inject an appropriate number of replication streams Then the Movie Manager contacts the target node to prepare receiving of the replication streams.

3. When all streams, which have been injected to the target node, finish their replication, the target node would notify the Movie Manager, such that the Movie Manager can add the target node to its list of replica nodes. The new replica node can then be included in the Movie Manager's decision of server node assignment to new requests.

4. The Metadata Manager would also be notified, such that it keeps a correct count of the aggregate residual storage capacity of the system.

### 5.4.4 Protocol for Dereplication

When the Metadata Manager finds the residual storage capacity of the system to be too low, it needs to dereplicate some movies. The responsibility for dereplication lies in the Metadata Manager because the aim of dereplication is to free up some service and storage capacity by removing some movie replicas. As any one of all the movies can be a possible candidate for dereplication, a movie-specific Movie Manager is not suitable to deal with dereplication. Besides, dereplication is dependent on some global information such as the aggregate residual storage capacity, as explained in Section 4.4, therefore only the Metadata Manager would be suitable to carry it out. The actual dereplication process is as follows:

1. First, the Metadata Manager multicasts a query to all Movie Managers to find out the available service capacity and the threshold of each movie.

2. From these information some movies can be chosen for dereplication. Then the corresponding Movie Managers will be asked to carry out dereplication, if possible.

3. Upon receiving an instruction to dereplicate a movie, a Movie Manager will multicast a query to its replica nodes to obtain the updated load information and the number of streams viewing the movie in consideration.

4. With the replies from the replica nodes, the Movie Manager determines if there exists a replica that can be dereplicated.

5. If no replica is found to be suitable for dereplication, this process is terminated. This process will be repeated after a certain amount of time, when a number of customers would have departured from the system.

6. If a replica is found to be suitable for dereplication, the Movie Manager, as an coordinator, tries to migrate all the customers viewing that movie in the chosen node to other nodes. This migration is done by reserving the required amount of service capacity from other nodes, with an atomic commitment protocol[25], We cannot assume this reservation will always be successful, even if the Movie Manager has previously obtained updated load information from the nodes, because the load information of the nodes could have changed just after their report to the Movie Manager.

7. Finally, the chosen node would mark the storage space previously allocated to the dereplicated movie as available for future use, and the node would be removed from the list of replica nodes maintained by the Movie Manager.

## 5.5 Failure Handling

A large scale VOD service would be costly to build by today's standard. Providing fault-tolerance by using a duplicate set of the whole system is thus not economically desirable. As a tradeoff between cost and the level of fault tolerance, this section explains how to exploit the inherent redundancy present in the architecture described earlier, i.e., existence of multiple copies of movies at different server nodes. We present a scheme to provide fault tolerance to the Metadata Manager, server nodes, and the Movie Managers, under the assumption that the no network failure occurs.

Fault-tolerance of a Metadata Manager, which mainly provides name service and triggers dereplication, can be provided by adopting a primary-backup[2] or state-machine[46] approach.

With a large amount of disks attached to server nodes, disk failures could be quite common. Berson et al.[7] investigated several different layouts of striped movie data,

and parity information for reconstruction after failure, to address the problem of disk failures. Their techniques can be directly applied to the disks within each node.

Our main focus is on how to deal with crashing of server nodes without adding significant costs. Due to a possibly large number of server nodes, we consider an alternative to the fault-tolerant measures of the Metadata Manager, due to cost considerations. Basically, when a server node has failed, we rely on the Movie Manager to migrate customers from the failed node to other replica nodes. A difficulty with this approach lies in that the remaining nodes might not possess enough residual service capacity to support all customers. In order to minimize the possibility for this, it is necessary to provide "true" redundancy of the movies by replicating them for more copies than actually needed. This can be achieved by increasing the replication thresholds of the movies. Also, a minimum of two replicas for each movie would be required, otherwise when the node containing the only replica of a movie crashed, no recovery can be done. Extra care should be exercised by the admission policy to reserve service capacities for migration of customers in the event of server node crashes.

### 5.5.1 Handling of Server Node Failures

During playback of movies, the client set-top boxes are able to detect failures of server nodes when the expected movie data does not arrive by a certain time limit. In this case, the following sequence (Figure 5.3) is invoked to resume playback with a different server node.

1. The client set-top box sends a failure signal to the Metadata Manager. It does not directly send the request to the Movie Manager because the Movie Manager could have been migrated elsewhere, after its first contact with the client, to be explained later.

2. The Metadata Manager forwards the request to the Movie Manager, which then verifies whether the server node has really failed by sending it a liveness detection message. This verification is necessary because the customer could have sent

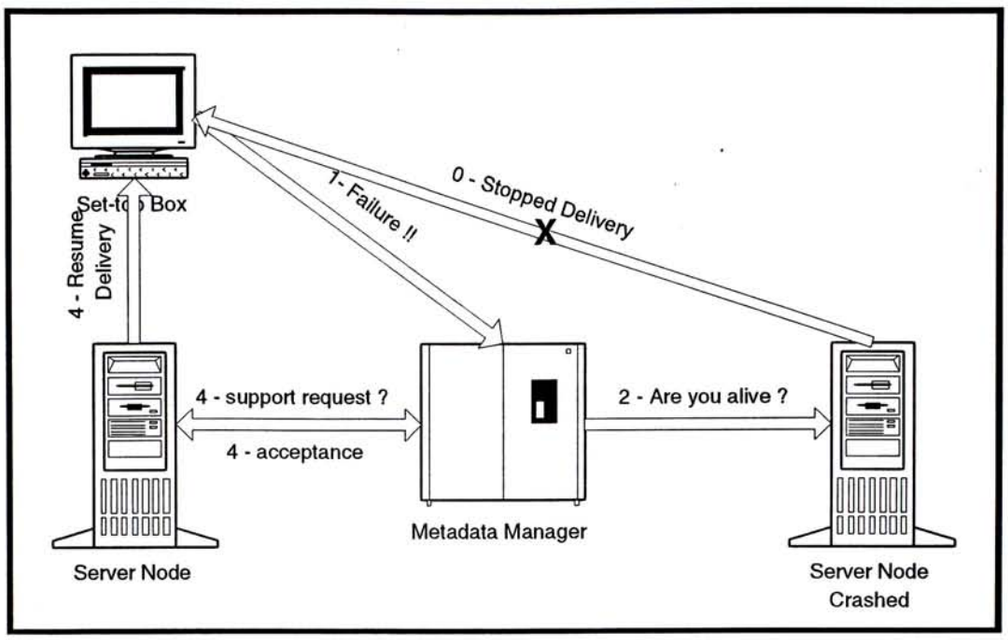


**Figure 5.3**: Handling of a Server Node Failure

an untrue signal of server node failure for malicious purposes. The Metadata Manager does not authenticate the client at this point because it does not keep track of the keys of all customers, which would consume too much resources, and that the verification process is sufficient for this failure handling process.

3. If what the client has noted is true, the Movie Manager will remove the failed node from its list of replica nodes.

4. The Movie Manager then selects a replica node for the client, as if it is treating a new viewing request, with the exception that the movie data delivery starts from the position where the client has not received from the former server node being used. The new node issues new keys to the customer for subsequent contact authentication. Since the Movie Manager adopts a least-loaded node assignment policy, as stated in Chapter 3, the migrated customers would be distributed in a balanced manner, with respect to the load of alive replica nodes.

During a movie assignment process, The Movie Manager may detect server node failures. Nothing needs to be done if this happens, since affected customers would eventually ask the Movie Manager for reassignment of new server nodes with the above

process. The Movie Manager cannot initiate this reassignment process on behalf of the customers, because it does not keep track of the customers' progress.

During a replication process, a server node can also detect other server nodes' failure, when they are participating in movie replication. When a server node, which has been acting as a target for replication of a particular movie, has failed and is detected by a source node, the source node will inform the Movie Manager to cancel that replication. The Movie Manager then multicasts a replication abortion signal to other source nodes to remove the relevant replication streams are removed. If the failed server node was acting as a source node, upon detection of its failure by the target node, the Movie Manager will again be informed. The Movie Manager then attempts to resume replication by choosing other source nodes. If such resumption is determined to be impossible, the replication is cancelled in a similar mannner.

### 5.5.2 Handling of Movie Manager Failures

To tolerate Movie Manager failures, existence of multiple machines which are capable of running a Movie Manager is needed. We propose a mechanism to replace the failed Movie Managers by newly created instances, via rebuilding of the state that the failed Movie Managers previously held, which is coordinated by the Metadata Manager (Figure 5.4). This process involves:

1. Failure of Movie Managers can be detected by the Metadata Manager, or a server node when the Movie Manager was relied upon to carry out replication. In case it is detected by a server node, the Metadata Manager is informed.
2. When such a failure happens, the Metadata Manager instructs all the machines that are capable of running Movie Managers to hold an an election[16] to recreate new Movie Manager(s).
3. The new Movie Manager(s) reports to the Metadata Manager.
4. The Metadata Manager creates the list of movies previously supported by (various Movie Manager(s) in) the failed machine by a reverse lookup operation on

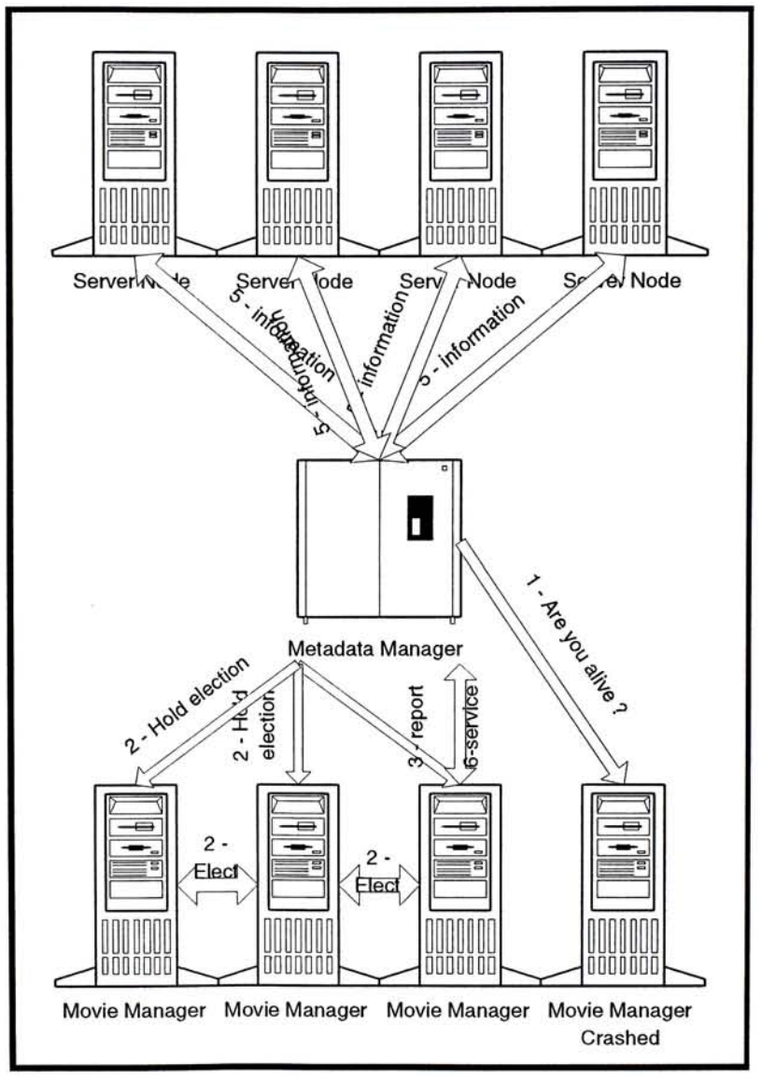


**Figure 5.4**: Handling of a Movie Manager Failure

its internal Movie Manager-address mapping table.

5. The Metadata Manager broadcasts a query to all server nodes for all information related to the movie(s) which the Movie Manager(s) should be responsible for. Such information includes: which server nodes carry replica of the movie(s) in consideration, their replication status, if any, etc.

6. The Metadata Manager assigns responsibilities for different movies to the Movie Manager(s) with the information collected from the server nodes, and updates its own mapping table.

# Chapter 6

# Results

Experiments were carried out to compare the performance of the different policies proposed in Chapter 4. In particular, we would like to seek answers to the following important questions from these experiments:

- How much does dynamic parallel replication benefit VOD systems ?
- Is it worthwhile to consume large amount of resources for shortening replicating time with parallel replication, with the tradeoff of not being able to use those resources for new arrivals ?
- Given all the proposed replication policies in Chapter 4, which of them performs the best ?
- How does the movie popularity skewness affect dynamic replication ?
- How does the threshold limit parameter involved in dynamic replication affect system performance ?

In this chapter, we will first describe our performance metric to compare different policies meaningfully. Then, the simulated system is introduced. The experiments were separated into two groups. The first group was done with dereplication disabled for isolating the performance among different policies, while the second group was done

with dereplication enabled to obtain a more realistic picture of the benefits of dynamic replication.

## 6.1 Performance Metric

To measure how successful our dynamic replication scheme deals with the problem of movie placement configuration among different nodes at a high-level, it would be appropriate to focus on what happens when customers arrive. This is because if the movie placement reconfiguration is not done fast enough to match the demand of the movies, it is likely that the nodes that carry a very popular movie would be fully loaded, while other nodes may be lightly loaded, creating a load imbalance. To quantify this effect, there are several related aspects that can be measured:

- Nodes' Service Capacities Utilization — Although this gives a view of how much of the system's capacity is effectively utilized, it is a meaningful measure of the movie placement reconfiguration speed only if the customers arrive in a way that, in an ideal situation, would precisely saturate the system. Consider that when customers' arrival rate is significantly higher than what the system can support, the system will still give a near 100% utilization, in spite of poor reconfiguration.

- Waiting time of the customers — The average waiting time of customers, focusing on customers instead of the system, correctly correlate to the movie placement reconfiguration regardless of the arrival rate. However, it is not quite clear that for how long a customer would tolerate for service. Besides, it is unlikely that all customers would have the same degree of patience. Therefore, we adopt a similar customer-focused notion but without waiting customers:

- *Acceptance Rate* — In a system that would immediately reject a customer request when the system cannot accept (service) it immediately, we define the acceptance rate to be the number of accepted customers divided by the number of arrivals, during a certain period. This metric is merely used to compare the different policies in an environment set up for benchmarking. Of course, a real-life system

should allow the customers to wait for service, if they intend to.

Low-level measurement of how well a replication policy performs, without being affect by other components in the replication algorithm, can be done simply by checking measuring the average replication time. This is because the replication policies are designed specifically to shorten replication times.

However, a low-level measurement methodology of how well a target-node selection policy performs is not as obvious as that of replication policy. A good target-node selection policy ultimately should yield high acceptance rates, which is a high-level performance metric tied to other components in the replication scheme. To isolate the performance of a target node selection policy, we measure how close the actual number of copies of movies generated is to the expected number of copies, based on the movie popularity distribution. This measure, termed *replication quality*, properly reflects how good a target node selection policy is, because a bad target node selection causes replicas of different movies to be created in the same nodes, instead of spreading the replicas throughout the system, causing the problem of competition of service capacity. This competition prevents movies to obtain sufficient service capacity, and in turn causing further replication and generating more replicas than necessary. Therefore, only a good target node selection policy can yield a good result with this measure. By Little's result[28], the expected number of copies is $E_i(t) = \lceil \frac{m_i p_i(t) \lambda}{B} \rceil$. We define replication quality of a snapshot of the system containing a set $M$ of movies at time $t$ as:

$$\frac{\sum_{i \epsilon M \wedge (|R_i(t)|>1 \vee E_i(t)>1)} E_i(t)}{\sum_{i \epsilon M \wedge (|R_i(t)|>1 \vee E_i(t)>1)} (E_i(t) + \left| \, |R_i(t)| - E_i(t) \, \right|^2)}$$

The summation in this expression is performed only over those movies with more than a single replica. This is motivated by a desire to reduce the effect of a large number of non-popular movies, which do not need to be replicated, on the overall measure of the system's quality of replication. The choice of $t$ is associated with the benchmarking methodology, which is discussed in Section 6.3. In the ideal case, the replication quality should be equal to 1, representing that the actual number of replicas is precisely equal to the expected number of replicas. If the discrepancy between these two numbers is too large, say, a movie is unnecessarily replicated to all nodes in the system, the

replication quality will be close to 0.

## 6.2 Simulation Environment

The simulated system consisted of 20 nodes, each of which has a storage capacity of 50 movies, and service capacity of 250 movie access streams. In our study, we used 500 movies, each 90 minutes long. Initially, there was one copy of each movie in the system. The movies were assigned to nodes in a round-robin manner, in order of descending popularity.

The following values for system parameters are used as defaults for the experiments, except for the experiments that investigate the effect of varying a particular parameter. For replication, a threshold limit $h$ (refer to Section 4.1) of 0.7 was used. No early acceptance or early migration was employed unless otherwise stated. A Poisson arrival process with an average rate of $\lambda = a * \frac{NB}{m}$ was used to model customer arrivals, where $N$, $B$, $m$, and $a$ are the number of nodes, node service capacity, the length of a movie, and the *relative* arrival rate with respect to the aggregate system service capacity, respectively. Although a Poisson arrival process may not be the closest approximation to real-world uses of VOD (which may occur as bursts), it is still chosen for our simulation because it would allow the system to experience an amount of arrival which just matches the full capacity of it (with $a = 1.0$) over a long period of time, stressing the need of dynamic reconfiguration of movie replicas placement. The skewness of movie popularity was modeled by a geometric distribution with $p_{i+1} = \chi p_i$, where $\chi = 0.618$ is the degree of uniformity of movie popularities.

The results presented here do not include the effect of any event occurred during the initial 90 minutes, since during that period the system possesses a much lower load than usual. Also, replication was suppressed during that period to avoid affecting subsequent acceptance of customers. In the first half of this study, we will focus on the behavior of various policies in a benchmark setting and find out the best policy. In the second half of this study, we employ the best policy in a more realistic setting to give a clear view of how dynamic replication impacts a VOD system.

## 6.3 Results of Experiments without Dereplication

In this section, we will compare the performance of various policies, and examine the effect of tuning various parameters of the replication algorithm. To isolate the performance of the various policies discussed in Chapter 4, we disable the dereplication policy and keep the popularity distribution constant in this section, such that all policies will reach stable movie placement configuration eventually, allowing fair comparison.

To compare the performance of different replication policies in Section 4.3, we measure the acceptance rates during the transient period of movie placement reconfiguration, i.e., from the initial one-copy-per-movie configuration at the 91th minute to the stable movie achieved by the replication policies, we would know how good the replication policies are. It would not be meaningful to measure the acceptance rates achieved after the system has reached a stable movie placement configuration, because that configuration is, regardless of which replication policy is used, tuned to the constant movie popularity distribution and thus show high acceptance rates.

To compare how well different target node selection policies in Section 4.2 perform, we focus on the snapshot of the instant at time $t$ when all replication have finished, i.e., the movie placement configuration has reached a stable state: $R_i(q) = R_i(t)\ \forall q > t$. Since the popularity distribution is constant, the expectation of the number of copies of movie $i$, $E_i(t)$, is constant regardless of time $t$. It is then possible to compare the target node selection policies with the replication quality metric defined in Section 6.1.

The multi-source replication policy is not included in these benchmarks because with the one-copy-per-movie initial configuration, a multi-source replication policy will not be able to find more than one source for replication policy for the crucial initial replication period. We leave the evaluation of the benefits of a multi-source replication policy to a later section, in a different simulation setting.

For ease of illustration, the lines in the legend of each figure are labeled in the same vertical order as they appear in the graph. The line labeled ``No Replication'' represents a system in which replication was suppressed, thus in this case there was

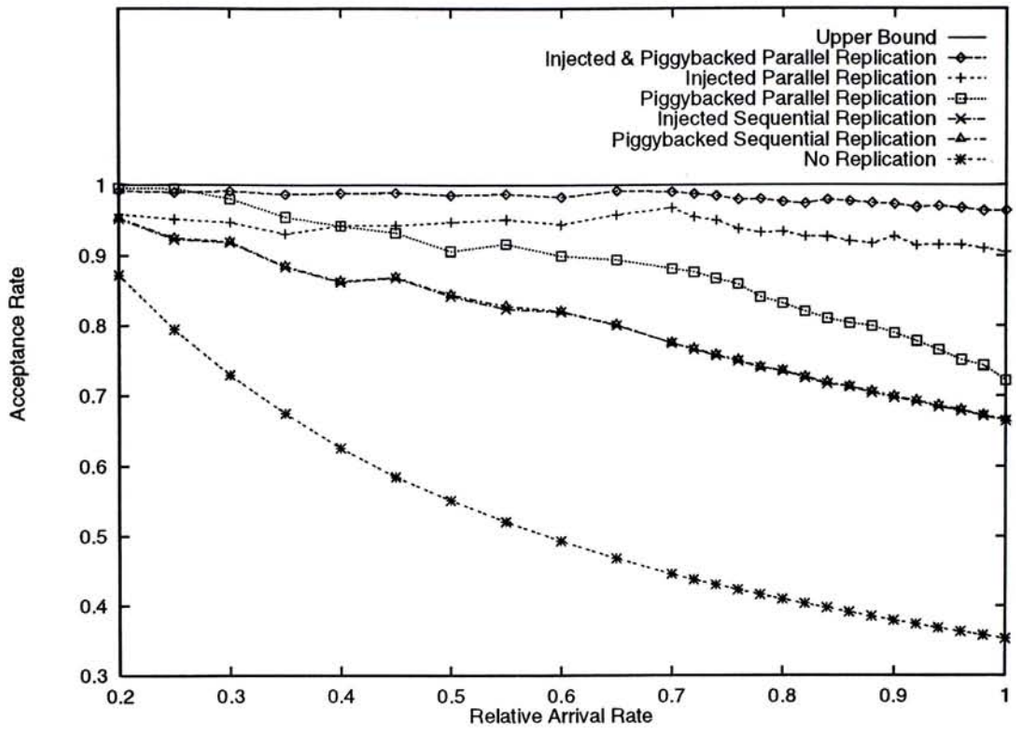


**Figure 6.1**: Comparison of Different Replication Policies

only a single copy of each movie throughout the simulation. The line labeled ''Upper Bound'' refers to an upper bound on our performance metric, namely, acceptance rate of 1.0. Although this upper bound is *unachievable* in practice, it is placed in the graphs for clearer comparison of how close the simulated system performance is to the ideal situation.

### 6.3.1 Comparison of Different Replication Policies

Figure 6.1 illustrates the acceptance rate corresponding to the different replication policies described in Section 4.3. A good replication policy should be able to perform movie placement reconfiguration in such a way that customer acceptance rate is reasonably high.

Generally, the injected & piggybacked parallel policy exhibits the best performance among the single-source policies we studied. This is a natural result, given that this particular policy is most aggressive in shortening the replication time, and a shorter

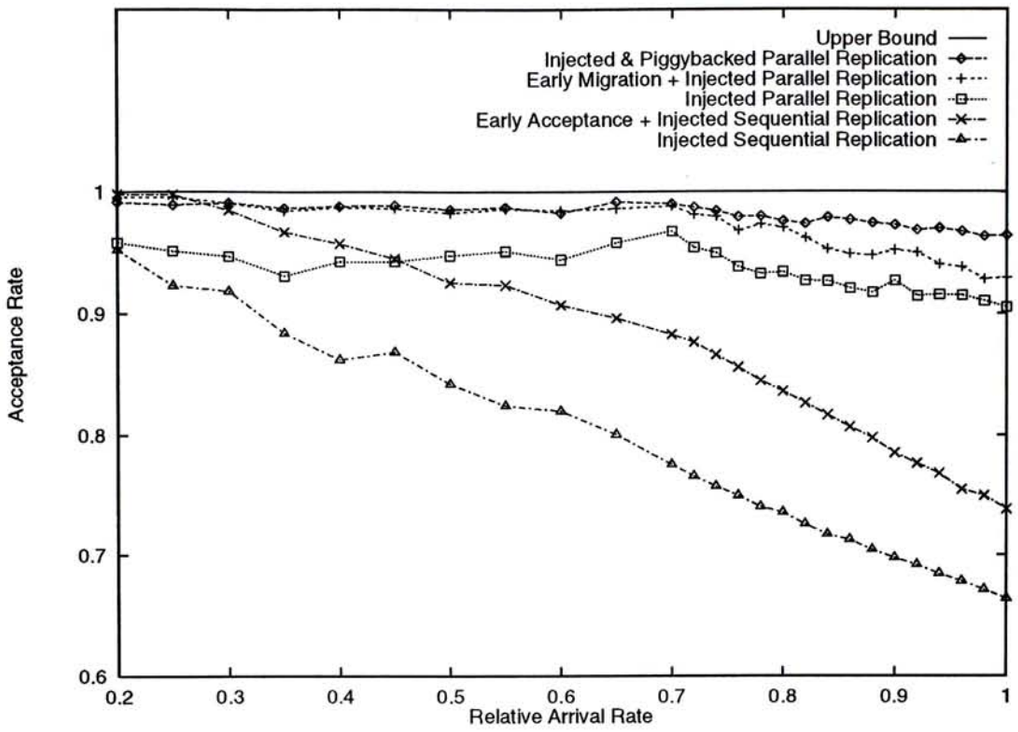


**Figure 6.2**: Effect of Early Acceptance/Migration

replication time means shorter time for a new replica to be available to serve users. The injected parallel policy is a close second. The piggybacked parallel policy lagged behind because piggybacking would not have allowed as many replication streams to be created, as explained in Section 4.3. As expected, the sequential policies perform poorly because they exhibit significantly longer replication times and therefore, are not as responsive to demand. The injected sequential policy and piggybacked sequential policy did not show any difference at all, because the large-scale environment masked the minimal overhead saving with piggybacked sequential policy. When the arrival rate is low, the differences among the different policies are not too large, because the system was not under stress.

### 6.3.2 Effect of Early Acceptance/Migration

Figure 6.2 shows the benefits of employing early acceptance or early migration, i.e., allowing customers to view incompletely replicated copies of some movies. As explained in Section 4.3.2 & Section 4.3.4, piggybacking-based replication policies are not good

for early acceptance/migration due to complications with VCR functions, unless some stand-by streams are reserved for recovery as mentioned in Section 4.3.5. Therefore, early migration was applied to the injected parallel replication only, and early acceptance was applied to the injected sequential replication only. It can be seen that in both cases, the acceptance rates improved. These cases are compared to the injected & piggybacked parallel replication policy without early migration, which was found to be slightly better.

### 6.3.3 Answer to the Resources Consumption Tradeoff issue

Figure 6.3 depicts the replication time reduction achieved through parallel replication. The x-axis in this graph denotes the hard limit of the number of replication streams, $\phi$, which has been introduced in Section 4.3.3. That is, even if the system could support a higher number of replication streams, only the amount as defined by the hard limit would be used for replication. In particular, the data points corresponding to the hard limit of 1 represents the case for sequential replication. It is clear that a greater number of (allowed) replication streams yields significantly shorter replication times.

The short replication times achieved with higher number of replication streams translate to better acceptance rates as illustrated in Figure 6.4, which gives an answer to the question on tradeoff between using resources for shortening replication time and leaving them for normal customer service. Since the hard limit controls the maximum amount of resources that could be allocated to parallel replication, that the acceptance rate grows with increasing limit implies that spending resources to shorten replication time is better. This is because addition of replicas have a long-term benefit over the short-time disadvantage of consuming resources.

There are diminishing returns in both figures because when the system is heavily loaded, there might be few residual capacity in the nodes that could be used for replication, thus limiting the number of replication streams.

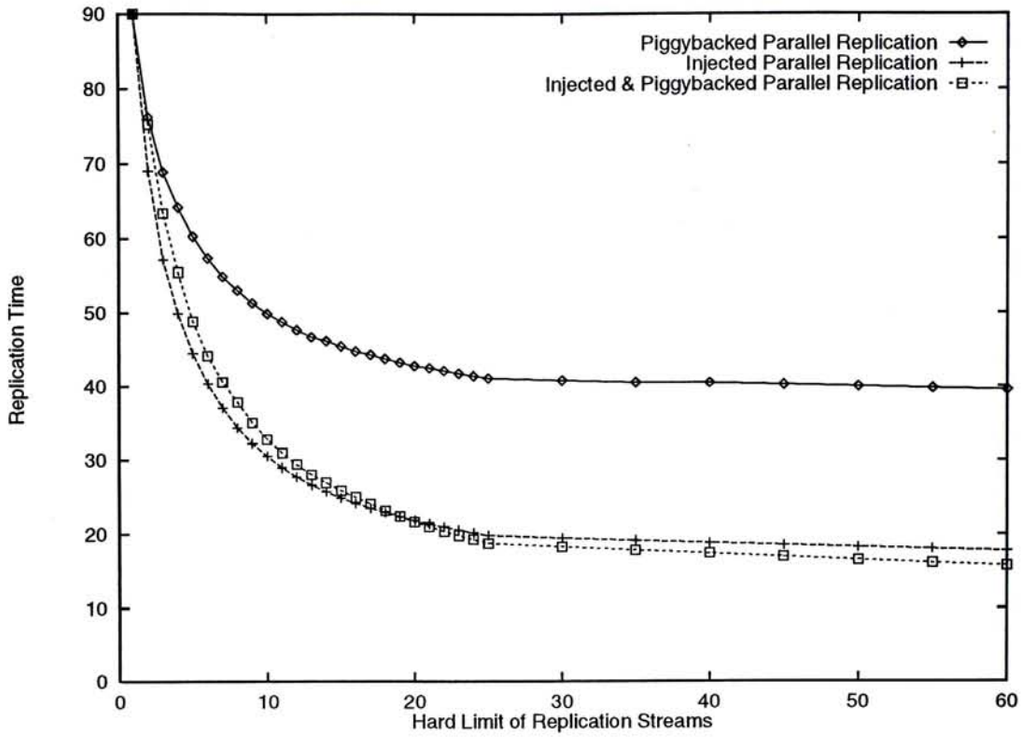


**Figure 6.3**: Replication Time

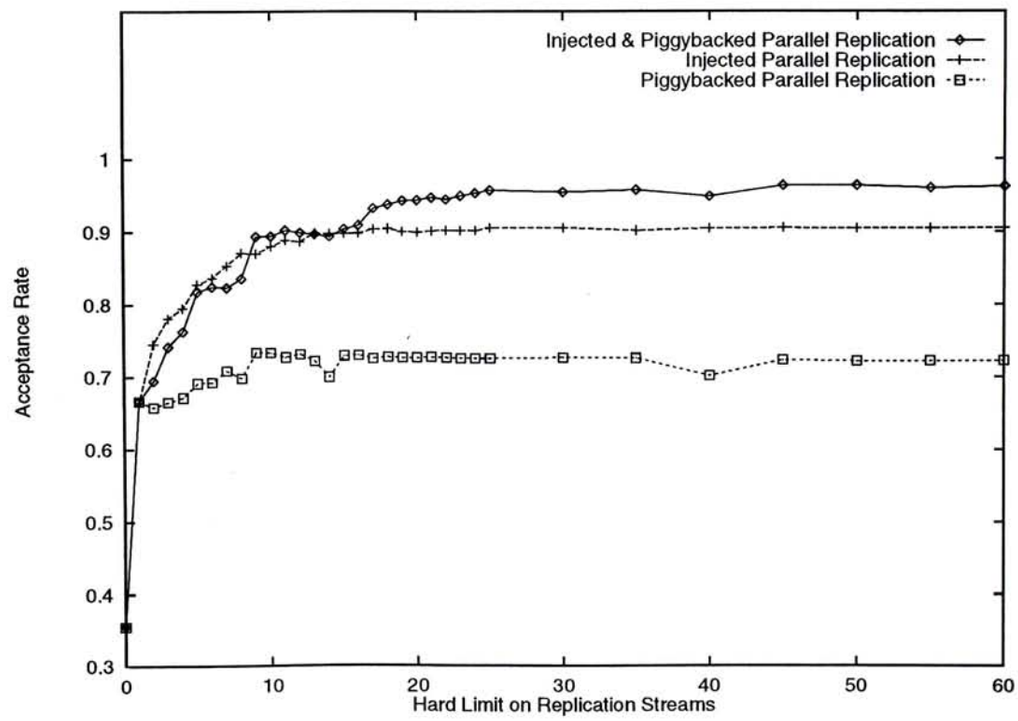


**Figure 6.4**: Significance of Parallel Replication

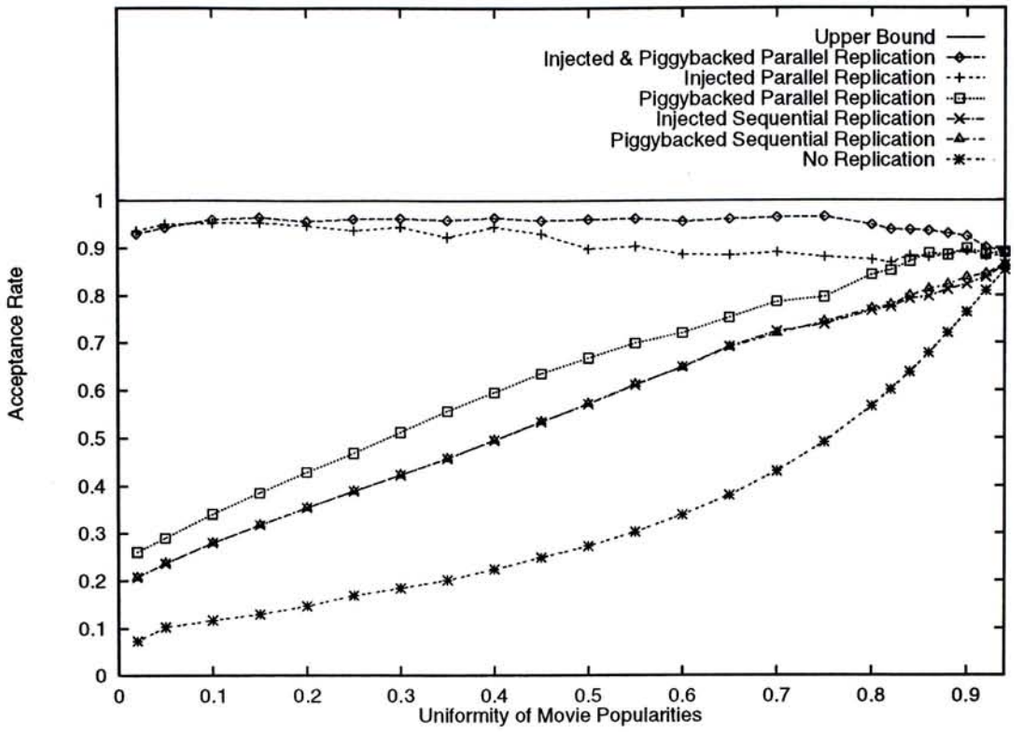


**Figure 6.5**: Effect of Varying Movie Popularities Distribution

### 6.3.4 Effect of Varying Movie Popularity Skewness

Figure 6.5 illustrates how the skewness of movie popularity, adjusted by the parameter $\chi$ (described in Section 6.2) affects dynamic replication. When the movie popularity is uniform, i.e., $\chi = 1$, all movies have equal bandwidth requirements and in this case, the system performance is not very sensitive to the choice of replication policy. Therefore, the curves converge when $\chi$ increases. The distinction among the replication policies is more evident when $\chi$ is low, i.e., the skew in movie popularity is high. The policies that are able to replicate movies faster, for instance, the injected & piggybacked parallel replication, perform better and result in higher acceptance rates.

### 6.3.5 Effect of Varying Replication Threshold

Figure 6.6 illustrates the effect of varying the replication threshold limit, $h$, which controls the responsiveness of the replication algorithm as described in Section 4.1. When the threshold limit is too low, the responsiveness decreases and leads to lower accep-

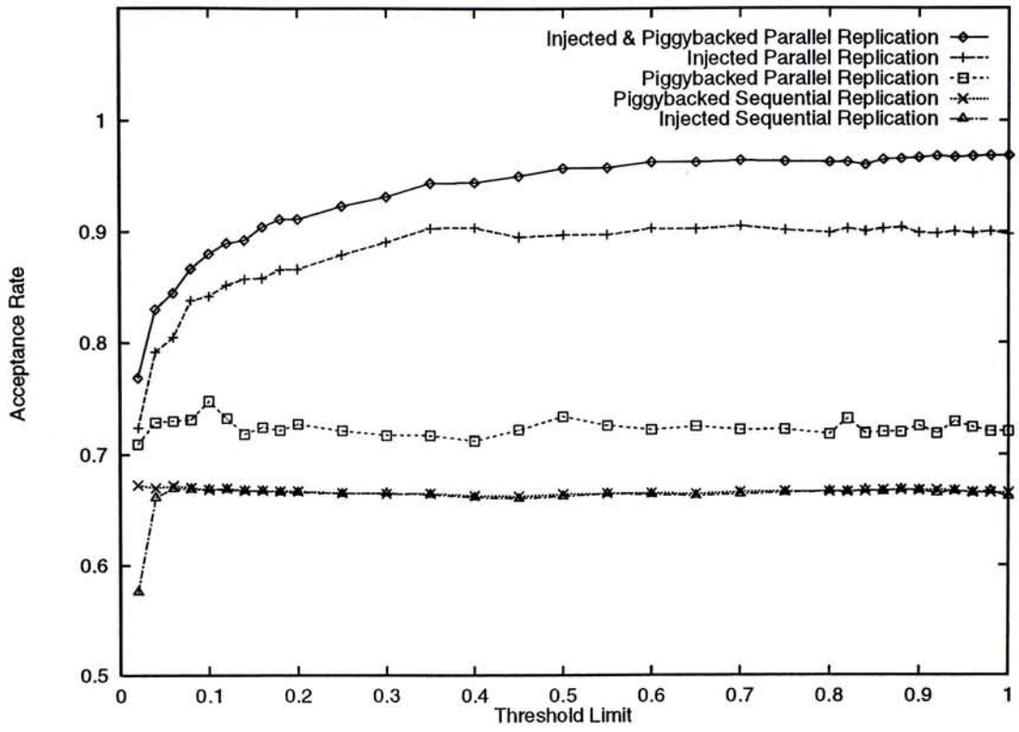


**Figure 6.6**: Effect of Varying Threshold Limit

tance rates. However, for policies which exhibit high replication time, the acceptance rate is fairly insensitive to that since the long replication time dominates the performance of the replication policy. These results suggest that the choice of a replication threshold is not a critical issue, since over a large range of values, the threshold limit does not severely affect the performance of the replication algorithm.

### 6.3.6 Comparison of Different Target Node Selection Policies

Figure 6.7 compares the three target node selection policies described in Section 4.2 as a function of the replication threshold limit, in terms of the replication quality measure we have introduced.

The estimated residual capacity-based target node selection policy consistently performed better than the random and the least-loaded node policies. All replication policies exhibit similar behavior with respect to the use of the three target node selection policies, but only two of them are illustrated here for clarity of presentation.

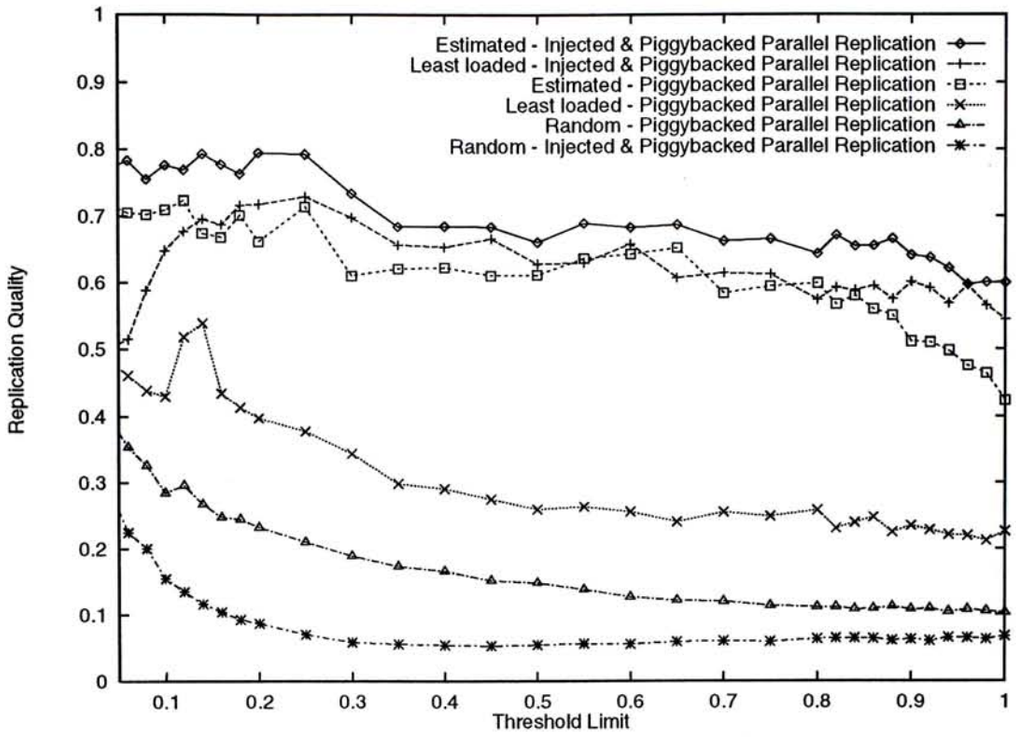


**Figure 6.7**: Comparison of Three Target Node Selection Policies

When the threshold limit increased, i.e., the responsiveness of the replication algorithm increased, the system tended to generate more copies of movies (than necessary), so in general the replication quality decreased.

## 6.4 Overall Impact of Dynamic Replication

In the simulation studies described earlier, the movie popularities were kept constant, and no dereplication was allowed, such that the performance benefits yielded by different replication policies can be compared fairly. To see exactly how dynamic replication benefits a VOD system, another experiment was done with changing movie popularities. The changing movie popularities necessitated the use of dereplication, otherwise the system storage would be filled up by replicas of previously popular movies, leaving no storage for currently popular ones over time. Also, we analyze the performance of the single-source injected & piggybacked policy versus its multi-source counterpart.

Specifically, we would like to investigate the effect of dynamic replication under

the increase of movie popularities. Failure to cope with it will result in increased rejection rate (which translates to waiting time in a real-world system), making users unhappy. Although increase of popularity of some movies is always coupled with decrease of popularity of other movies. The former change is more directly related to the acceptance rate. Therefore the following analysis will focus on the aspect of popularity increase instead of decrease. We consider two kinds of scenarios:

1. Gradual increase of popularities among movies — Initially, a group of movies are the most popular. However, it becomes less popular gradually over time, while another group of movies becomes gradually more popular, i.e., the popularity of one group is continually replaced by another one gradually. We believe that this pattern corresponds to the change of types of audiences (and hence their interests) at different times throughout a day.

2. Drastic increase in popularities of movies — This scenario is mainly intended to test dynamic replication under heavy stress. Although this could be thought as addition of new and popular movies, in reality, a VOD service provider could usually anticipate the demand of new movies and prepare several copies before announcing their availability. Nevertheless, it is still interesting to see what would happen if there is unanticipated drastic increase of popularity of a movie, when only one copy of which is available originally. We believe this would occasionally occur in reality, for example, the death of an actor/actress might lead to viewing of his/her old movies.

To simulate these scenarios, we "rotate" the movie popularities over time. With time $t = 0$ denoting the initial instant, we define a *rotation period*, $\gamma$, and shift the popularity of a movie to the next one every $\gamma$ units of time, with the 500 movies arranged in a circular fashion:

$$p_i(t) = p_{(i-\lfloor \frac{t}{\gamma} \rfloor + 500) \bmod 500}(0)$$

1. When $\gamma > 0$, the movie popularities increased gradually. To understand why, consider the initial condition ($t = 0$): $p_0(0) > p_1(0) >> p_{499}(0)$. At time $t = \gamma$,

$p_0(t) = p_{499}(0), p_1(t) = p_0(0), p_2(t) = p_1(0)$, causing $p_1(t) > p_2(t) >> p_0(t)$. This means that movie 1, originally the second popular movie, has its popularity increased slightly to become the most popular one. Movie 2, originally the third popular movie, become the second popular movie, and so on. Movie 0, originally the most popular one, becomes unpopular. Generally, with a positive $\gamma$, movies become more popular in a gradual fashion, although some popular movies suddenly become unpopular.

2. When $\gamma < 0$, one movie will have its popularity drastically increased every time the popularities are rotated. Consider that at time $t = |\gamma|$, $p_{499}(t) = p0(0), p_0(t) = p_1(0), p_1(t) = p_2(0)$. Movie 499 is originally an unpopular movie, but it suddenly becomes the most popular one, by gaining the popularity that originally belonged to movie 0.

Aside from the incorporation of this rotating popularities model, dereplication of movies was also enabled in this simulation. Dereplication would be triggered when the system-wide aggregate residual storage capacity dropped below 30 movies, with a dereplication threshold D (described in Section 4.4) of 5. Other simulation parameters were the same as before, except that the storage capacity of a node was decreased to 30 movies from 50. Since 25 different movies would be stored in each node initially, this change reduced the initial free storage from 25 movies to 5 movies. If the VOD system can still offer good acceptance rates with this small amount of free storage under changing movie popularities, it would imply that dynamic replication is effective and practical. The statistics was averaged over a 7-day simulation period, to take account of the effects of both replication and dereplication, instead of the short transient period involving replication only. Again replication (and dereplication) were disabled for the first 90 minutes, and the calculation of acceptance rates did not include that period.

This simulation was done with employing the single-source and multi-source injected & piggybacked parallel replication policy with different values of $\gamma$. The results are shown in Figure 6.8, which shows the acceptance rates plotted against the absolute values of the rotation period, $|\gamma|$, in units of minutes. An increasing rotation period means that movie popularities changed less rapidly. This allowed for higher accep-

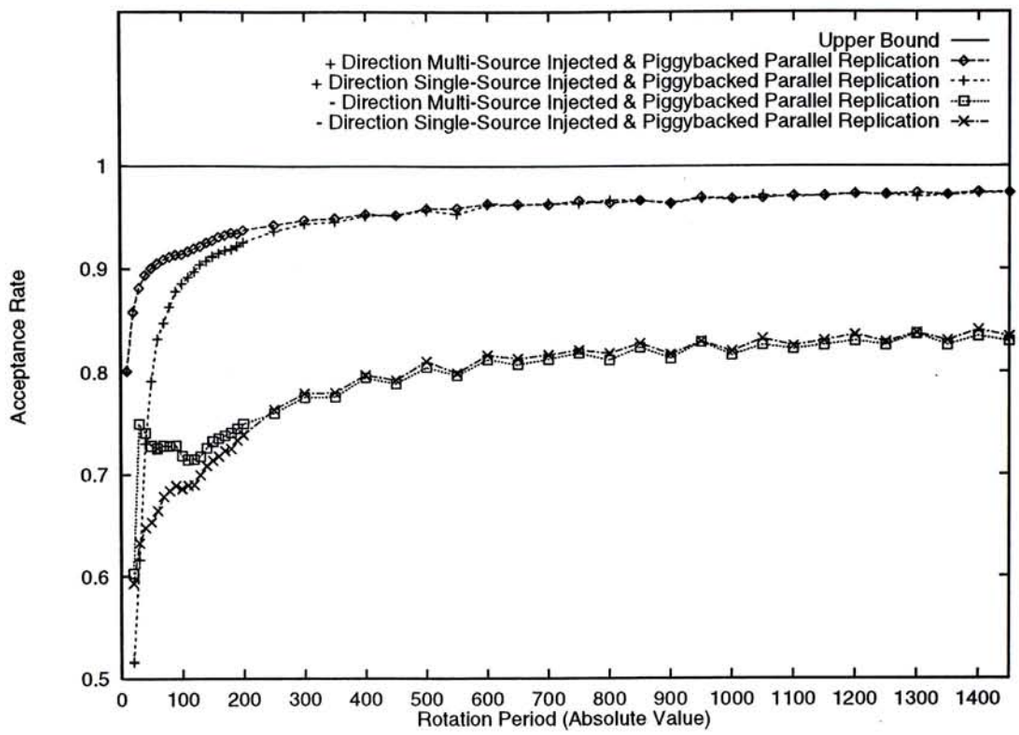


**Figure 6.8**: Dynamic Movie Popularities

tance rates because it gave more time for dynamic replication to adjust the movie configuration to match the demand well.

The lines labeled "+ Direction" and "- Direction" show the acceptance rates achieved with dynamic popularities with positive values and negative values of $\gamma$ respectively. With positive values of $\gamma$, i.e., the movie popularities are increased gradually, the system generally offered high acceptance rates. As expected, the acceptance rates are generally lower with negative values of $\gamma$. This is because the system was under higher stress when unpopular movies would suddenly become very popular, making the movie configuration mismatch the demand by a large extent for much of the time. Nevertheless, even at a rate as high as two movies with unanticipated drastic popularity increase per day ($\gamma = -720$), the system still offers very reasonable acceptance rates, which are greater than 80%.

As shown in the figure, the multi-source version of the replication policy offered higher acceptance rates when the change of popularities was very rapid, regardless of the direction of the popularity rotation. This is because with such rapid changes, dy-

namic replication could keep up with the demand only if the replication of movies could be finished in a short enough time, which would be a limiting factor for acceptance rates. Multi-source replication policies allow more replication streams to be utilized for replication, so replication time can be shortened considerably. As a result, they coped with rapid changes better. When the changes became less rapid, the replication time was no longer the limiting factor, thus they no longer showed improvement over their single-source counterparts.

# Chapter 7

# Comparison with BSR-based Policy

The online Bandwidth to Space Ratio (BSR)-based video placement policy[13] is designed to provide a good initial placement of movies among a set of nodes. The basic premise of allocating the placement of multiple replicas based on BSR is that different movies and storage devices have different service capacities (bandwidth) and storage capacity requirements. The BSR placement policy thus chooses nodes for placing replicas in a way that reduces the amount of mismatch between the BSR of the storage devices and that of the movies.

The operation of the BSR movie placement is divided into four phases:

1. Periodically, for each movie, it estimates the expected load in the future and determines whether additional replicas is necessary.
2. If the additional replica is necessary due to increased demand, it will be placed in a node chosen according to the BSR metric by retrieving the movie data from a tape library.
3. If the demand of the movie has decreased, the system balances the replica nodes in a way that minimizes the BSR mismatch among the devices and the movies, which is possible because the number of replicas exceeds the actual needs.

4. If the system fails to allocate expected load to replica nodes due to lack of service or storage capacity, it redistributes the expected load for some movies by consolidating their replicas into fewer ones.

The BSR policy shares some similarities, along with some important differences, with our dynamic replication scheme. The BSR Phase 1 is analogous to the replication triggering policy. Its Phase 2 and Phase 4 are analogous to replication and dereplication. The BSR Phase 3 is not needed for the dynamic replication scheme, because the underlying system models for the two schemes are very different. The model employed by the BSR scheme has the system pre-allocate service capacity according to estimated demand. Therefore, with decreased demand of some movies, it is necessary for the BSR scheme to re-adjust the pre-allocation to avoid wastage of pre-allocated service capacity. In contrast, the system model employed by dynamic replication does not employ pre-allocation, it just directs new arrivals to the least-loaded replica node. The ways the two schemes handle replication are also different. Whereas the BSR policy is designed to replicate movies from a tape library to a (disk striping group) node, our scheme creates the replicas by copying movie data from other nodes. The BSR policy chooses a node for movie placement according to the BSR metric, which is different from the three target node selection policies discussed in Section 4.2.

The dynamic replication scheme utilizes service capacity better than the BSR placement scheme does, when the movie popularity changes. This is because of a fundamental problem that lies with the pre-allocation model, which the BSR placement scheme is based on. This problem shows up when the service capacity pre-allocated to the movies becomes different from the actual needs, due to movie popularity changes. In this case, some of the pre-allocated service capacity would be wasted, because they are unusable for other movies. In fact, the pre-allocation model is inherent to the BSR placement metric. To see why, consider that if the pre-allocation model is replaced by our model (which does not employ the concept of pre-allocation), the movies no longer possess a constant portion of the nodes' service capacities. Thus a placement configuration with matched BSR of movies to the BSR of storage devices would soon have them mismatched. To illustrate this point on the inherence of pre-allocation

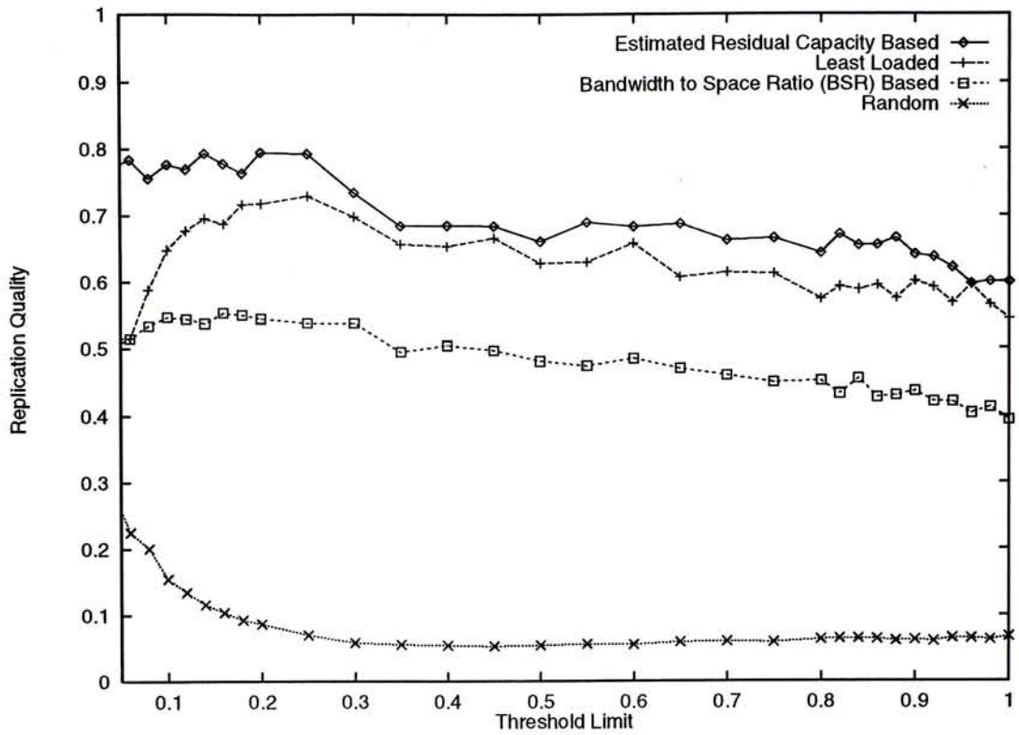


**Figure 7.1**: Employing the BSR metric in Dynamic Replication

model with the BSR metric, we performed an experiment on isolating the BSR metric from the pre-allocation model. This experiment is similar to the one in Section 6.3.6, but with the addition of employing the BSR metric as a target node selection policy. The result is shown in Figure 7.1. From this figure, we see that the BSR metric does not work well if it is not based on the pre-allocation model. Although we believe that the BSR metric should work well on top of the pre-allocation model, the fundamental problem of the pre-allocation model — wastage of service capacity when movie popularity changes, makes this model, and therefore the BSR metric, undesirable.

The dynamic replication scheme necessitates less customer migration than the BSR placement scheme does. Customer migration is needed in the BSR Phase 4 for consolidation of replicas, and is also need for dereplication in the dynamic replication scheme. In addition, the BSR placement scheme needs to migrate customers in Phase 3, which balances the pre-allocation amount across different replica nodes according to the BSR metric. A problem with this redistribution process is that it may require migration of some customers viewing the affected movie, when the new amount of pre-allocation of a node is lower than the number of these customers. Such migration brings extra

overhead to the system.

In conclusion, the dynamic replication scheme utilizes service capacity better, and needs to migrate customers less than the BSR placement scheme does.

# Chapter 8

# Conclusions

In this chapter, we briefly summarize the major work in this dissertation for designing scalable VOD systems, along with some conclusive remarks. Then some future research directions on this work are discussed.

## 8.1 Summary

In Chapter 1, we explained that striping[6], a commonly used technique in VOD systems (and some database systems), across all the nodes/disks in a distributed VOD server architecture hinders scalability, due to synchronization and homogeneity requirements. An alternative architecture, without striping across nodes, is thus more desirable. This architecture necessitates multiple copies of movies. So there is a problem of movie placement configuration, which changes with time when popularity of movies changes.

To deal with this problem, a dynamic replication scheme is thus presented in Chapter 4. This scheme consists of several components, which concern different aspects of the replication algorithm. First of all, the notion of triggering replication by checking the aggregate residual service capacities of all replica nodes of a movie is introduced. A heuristic is proposed to control the triggering time. When a replication session is

triggered, source and target nodes will be chosen to carry out the replication. Several different schemes for the choice of target node are investigated. When the demand for a movie decreases, the system will need to remove some replicas of (dereplicate) a movie to free storage capacity for other movies. The conditions on proper dereplication, without turning the system into an oscillating state between replication and dereplication, are analyzed.

In Section 4.3, six replication policies are discussed. The most advanced one employs multiple streams from multiple source nodes for reading different parts of the data of a particular chosen movie. This multiple-stream strategy is termed *parallel replication.* Combined with *injection* of replication streams and *piggybacking*[33], which attempts to make use of data being reading for normal movie for replication as well, this strategy allows for high degree of concurrency and thus shortens replication time significantly. Although such replication consumes some resources and would affect system performance temporarily, Chapter 6 shows that the benefits of parallel replication far outweighs its resource consumption. It also demonstrates that overall system performance is good with dynamic replication scheme under various different environmental conditions, indicating that the movie replicas placement is being reconfigured well. Chapter 5 provides a set of protocols to implement this dynamic replication algorithm on a scalable distributed VOD server, with provisions for fault-tolerance of all the system components.

In conclusion, the dynamic replication scheme and the parallel replication strategies provided in this dissertation addresses the problem of movie replicas placement reconfiguration in a multi-node VOD system, by allowing relatively quick replication of movie data across disk nodes. It is thus viable to employ a multi-node architecture, without data striping across nodes, as a means to achieve scalability on a node-basis.

## 8.2 Future Research Directions

So far, the performance results of employing dynamic replication, and comparative study of different replication policies, are based on simulation. Reasonable settings

for the parameters that control the replication algorithm are found experimentally. Creating an analytical model for a distributed VOD system with dynamic replication would be beneficial to clearer understanding of how the different components and parameters of the dynamic replication scheme, and various policies, impact high-level VOD system performance.

The dynamic replication scheme presented in this dissertation is primarily designed for disk-to-disk replication of movie data. A tape library, if present, is treated as a special low-service-capacity node which never services customers (directly) or acts as a target node for replication. It might be interesting to combine staging policies[22] with the dynamic replication in a coherent way so as to exploit the full benefits of hierarchical storage subsystem for VOD. Since popular movies should always be placed on disks for efficient accesses, a major concern would be determining when to remove all copies of a non-popular movie from disks. One possible way could be extending the dereplication scheme into differentiating two levels of low popularity: the first level removes some, but not all, replicas of the movie from disks; the second level removes all replicas from disks, requiring future access to the movie to be staged from the tape library.

Finally, the work presented here is targeted at a large-scale pure VOD system. When the VOD system becomes an integral part of a multimedia database, some usual assumptions in this work, and other literature on VOD, made by analogy to conventional video rentals, e.g. popularity skewness, frequency of occurrence of VCR functions (fast forward, pause, etc.), may no longer hold. With different assumptions, considerably different techniques and designs for scalability may be needed.

# Bibliography


[1] C. Abbott. Efficient Editing of Digital Sound on Disk. *Journal of Audio Engineering*, 32(6):394–402, June 1984.

[2] P. A. Alsberg and J. D. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the Second International Conference on Software Engineering*, pages 562–570, 1976.

[3] D. Anderson and G. Homsy. A Continuous Media I/O Server and its Synchronization Mechanism. *IEEE Computer, Special Issue on Multimedia Information Systems*, 24(10):51–57, October 1991.

[4] D. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Transactions on Computer Systems*, 10(4):311–337, November 1992.

[5] S. Angebranndt, R. L. Hyde, D. H. Luong, N. Siravara, and C. Schmandt. Integrating Audio and Telephony in a Distributed Workstation Environment. In *Proceedings of Summer 1991 Usenix Conference*, pages 419–436, June 1991.

[6] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of ACM SIGMOD '94*, pages 79–90, 1994.

[7] S. Berson, L. Golubchik, and R. R. Muntz. Fault Tolerant Design of Multimedia Servers. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, volume 24, pages 364–375, 1995.

[8] T. L. Casavant and J. G. Kuhl. "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems". *IEEE Transactions on Software Engineering*, 14(2):141–154, February 1988.

[9] CCITT. *Video codec for audio visual services at px64 kbits/s. CCITT Recommendation H.261*, 1990.

[10] M. S. Chen, H. I. Hsiao, C. S. Li, and Philip S. Yu. Using Rotation Mirrored Declustering for Replica Placement in a Disk-Array-Based Video Server. In *Proc. of the ACM Multimedia Conference*, pages 121–130, 1995.

[11] M. S. Chen, D. D. Kandlur, and P. S. Yu. Optimization of the Grouped Sweeping Scheduling with Heterogeneous Multimedia Streams. In *Proceedings of ACM Multimedia*, pages 235–242, 1993.

[12] A. Dan, M. Kienzle, and D. Sitaram. A Dynamic Policy of Segment Replication for Load-Balancing in Video-on-Demand Servers. *ACM Multimedia Systems*, 3:93–103, 1995.

[13] A. Dan and D. Sitaram. An Online Video Placement Policy Based on Bandwidth to Space Ratio (BSR). In *Proceedings of ACM SIGMOD'95*, 1995.

[14] J. Escobar, D. Deutsch, and C. Patridge. A Multi-Service Flow Synchronization Protocol. Technical report, BBN Systems and Technologies Division, March 1991.

[15] D. Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM*, 34(4):47–58, April 1991.

[16] H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE Transactions on Computers*, C-31(1):48–59, January 1982.

[17] J. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, 1992.

[18] S. Ghandeharizadeh and D. DeWitt. A Multiuser Performance Analysis of Alternative Declustering Strategies. In *Proceedings of Data Engineering*, 1990.

[19] S. Ghandeharizadeh and S. H. Kim. An Analysis of Striping in Scalable Multi-Disk Video Servers. Technical Report USC-CS-TR95-623, University of Southern California, 11 1995.

[20] S. Ghandeharizadeh and L. Ramos. Continuous Retrieval of Multimedia Data using Parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 8 1993.

[21] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi. Object Placement in Parallel Hypermedia Systems. In *Proceedings of Very Large Databases*, pages 243–254, 9 1991.

[22] S. Ghandeharizadeh and C. Shahabi. Management of Virtual Replicas in Parallel Multimedia Systems. In *Proceedings of the 1993 Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[23] S. Gibbs, D. Tsichritizis, A. Fitas, D. Konstantas, and Y. Yeogaroudakis. Muse: A Multi-Media Filing System. *IEEE Software*, 4(2):4–15, March 1987.

[24] L. Golubchik, J. C. S. Lui, and R. R. Muntz. Chained Declustering: Load Balancing and Robustness to Skew and Failure. In *Proceedings of the 2nd International Workshop on Research Issues in Data Engineering: Transaction and Query Processing*, pages 89–95, February 1992.

[25] J. N. Gray. Notes on Database Operating Systems. In *Operating Systems — An Advanced Course, Lecture Notes on Computer Science*, volume 66, pages 393–481, 1978.

[26] C. C. Han and K. G. Shin. Scheduling MPEG-Compressed Video Streams with Firm Deadline Constraints. In *Proceedings of ACM Multimedia*, pages 411–422, 1995.

[27] A. Hopper. Pandora - An Experimental System for Multimedia Appplications. *ACM Operating Sytems Review*, 24(2):19–34, April 1990.

[28] J. D. C. Little. A Proof of the Queueing Formula $L = \lambda W$. *Operations Research*, 9:383–387, May 1961.

[29] T.D.C. Little and A. Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, April 1990.

[30] M. Livny, S. Khoshafian, and H. Boral. Multi-Disk Management Algorithms. In *Proceedings of SIGMETRICS*, 1987.

[31] R. V. Meter. A Brief Survey of Current Work on Network Attached Peripherals. *ACM Operating Systems Review*, January 1996.

[32] Y. Mori. Multimedia Real-Time File System. Technical report, Matshushita Electric Industrial Co., February 1990.

[33] R. R. Muntz and J. C. S. Lui. Performance Analysis of Disk Arrays Under Failure. In *Proceedings of the 16th VLDB Conference*, pages 162–173, 1990.

[34] C. Nicoloau. An Article for Real-Time Muiltimedia Communication System. *IEEE Journal on Selected Areas in Communications*, 8(3):391–400, April 1990.

[35] B. C. Ooi, A. D. Narasimhalu, K. Y. Wang, and I. F. Chang. Design of a Multi-Media File Server using Optical Disks for Office Applications. In *IEEE Computer Society Office Automation Symposium*, pages 157–163, April 1990.

[36] D. Pan. A Tutorial on MPEG/Audio Compression. *IEEE Multimedia*, 2(2):60–74, 1995.

[37] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *International Conference on Management of Data (SIGMOD)*, pages 109–116, June 1988.

[38] K. K. Ramakrishnan, L. Vaitzblit andC. Gray, U. Vahalia, D. Ting, P. Tzelnic, S. Glaser, and W. Duso. Operating System Support for a Video-on-Demand File Service. *ACM Multimedia Systems*, 3:53–65, 1995.

[39] P. V. Rangan and H. M. Vin. Designing File Systems for Digital Video and Audio. In *Proceedings of the 13th Symposium on Operating Systems*, pages 81–94, October 1991.

[40] P. V. Rangan and H. M. Vin. Efficient Storage Techniques for Digital Continuous Multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5:564–573, August 1993.

[41] P. V. Rangan, H. M. Vin, and S. Ramanathan. Designing an On-Demand Multimedia Service. *IEEE Communications Magazine*, 30(7):56–65, July 1992.

[42] K. R. Rao and P. Yip. *Discrete Cosine Transform - Algorithms, Advantages, Applications.* Academic Press, Inc., London, 1990.

[43] D. Ries and R. Epstein. Evaluation of Distribution Criteria for Distributed Database Systems. Technical Report M78/22, UC Berkeley, May 1978.

[44] K. Salem and H. Garcia-Molina. Disk Striping. In *Proceedings of Data Engineering*, 1986.

[45] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network File System. In *Proceedings of the Summer USENIX Conference*, pages 119–130, 1985.

[46] F. B. Schnedier. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[47] K. G. Shin and Y. C. Chang. A Reservation-Based Algorithm for Scheduling Both Periodic and Aperiodic Real-Time Tasks. *IEEE Transactions on Computers*, 44(12), December 1995.

[48] R. Steinmetz. Synchronization Properties in Multimedia Systems. *IEEE Journal on Selected Areas in Communications*, 8(3):401–412, April 1990.

[49] S. D. Stoller and J. D. DeTreville. Storage replication and Layout in Video-on-Demand Servers. In *Proceedings of NOSSDAV '95*, 1995.

[50] M. Stonebraker. The Case for Shared Nothing. *Database Engineering Bulletin*, 9(1):4–9, 1986.

[51] Sun Microsystems. *Multimedia File System*, August 1989.

[52] D. B. Terry and D. C. Swinehart. Managing Stored Voice in the Etherphone System. *ACM Transactions on Computer Systems*, pages 3–27, February 1988.

[53] R. H. Thomas, H. C. Forsdick, T. R. Crowley, R. W. Schaaf, R. S. Tomlinsin, V. M. Travers, and G. G. Robertson. Diamond: A Multimedia Message System Built on a Distributed Architecture. *IEEE Computer*, 18(12):65–78, 1985.

[54] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID — A Disk Array Management System for Video Files. *ACM Multimedia*, pages 393–400, 1993.

[55] G. K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4):31–44, April 1991.

[56] J. L. Wolf, P. S. Yu, and H. Shachnai. DASD Dancing: A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems. In *SIGMETRICS'95*, pages 157–166, 1995.

CUHK Libraries
003589577