Graduate School ETD Form 9
(01/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis Acceptance

This is to certify that the thesis prepared

By Yung Ryn Choe

Entitled Design and Implementation of a Resource-Efficient Storage Server for VoD

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of Doctor of Philosophy

Final examining committee members

V. Pai , Chair

S. P. Midkiff

M. S. Thottethodi

A. N. K. Yip

Approved by Major Professor(s): V. Pai

Approved by Head of Graduate Program: V. Balakrishnan

Date of Graduate Program Head's Approval: 04/25/2007

DESIGN AND IMPLEMENTATION OF A RESOURCE-EFFICIENT STORAGE SERVER FOR VOD

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yung Ryn Choe

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2007

Purdue University

West Lafayette, Indiana

UMI Number: 3287296

UMI Microform 3287296

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Dedicated to my Savior and Lord Jesus Christ.

ACKNOWLEDGMENTS

I thank my Lord Jesus Christ for the meaning in life and always being with me. This work is for Him. I also thank my wife Chae Hyon for her love and support, and my children, Daniel (Myoungchul) and Grace (DahEun) for their prayers. Thanks to my parents who gave me support and encouragement.

Special thanks to my advisor Vijay Pai for research direction, challenge, and guidance. I also would like to thank Derek Schuff for his joint work on Toast and discussions on many things. It was a pleasure to work with these two people for the past two years. Thanks to Chase Douglas for his work on VoD prototype, Gautam Upadhyaya for his work on VoD server, and Jagadeesh Dyaberi for his work on Toast. Thanks also to my Advisory Committee members Mithuna Thottethodi, Samuel Midkiff, and Aaron Yip for taking the time to consider this work.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

ABSTRACT

Choe, Yung Ryn Ph.D., Purdue University, May, 2007. Design and Implementation of a Resource-Efficient Storage Server for VoD. Major Professor: Vijay S. Pai.

First, this dissertation presents a mathematical model and a prototype of a resource-efficient storage server for high-bitrate Video-on-Demand (VoD) applications. The model is detailed enough to account for the rate-based nature of streaming video, the buffering time allowed by the application, and average-case disk hardware characteristics while remaining simple enough to use for algorithm and system design. This dissertation then describes a prototype storage server designed to serve large video files at the specified bitrates and finds its performance to agree closely with the model.

Second, this dissertation investigates randomization and replication as strategies to achieve reliable performance in disk arrays targeted for VoD workloads. A disk array can provide high aggregate throughput, but only if the server can effectively balance the load on the disks. Such load balance is complicated by two key factors: workload hotspots caused by differences in popularity among media streams, and "fail-stutter" faults that arise when the performance of one or more devices drops below expectations.

This dissertation focuses on the random duplicate assignment (RDA) data allocation policy which places each data block on two disks chosen at random, independent of other blocks in the same media stream or other streams. This strategy is compared to traditional single-disk file allocation, disk striping (RAID-0), disk mirroring (RAID-1), and randomization without duplication. The results indicate that combining randomization and replication allows RDA to effectively tolerate both workload hotspots and fail-stutter faults better than previous schemes.

Third, this dissertation presents and evaluates Toast, a scalable VoD streaming system which combines the popular BitTorrent peer-to-peer (P2P) file-transfer technology with a simple dedicated streaming server to decrease server load and increase client transfer speed.

The results show that the default BitTorrent download strategy is not well-suited to the VoD environment. Instead, strategies should favor downloading pieces of content that will be needed earlier, decreasing the chances that the clients will be forced to get the data directly from the VoD server. Such strategies allow Toast to operate much more efficiently than simple unicast distribution, reducing data transfer demands by up to 70–90% if clients remain in the system as seeds after viewing their content. Toast thus extends the aggregate throughput capability of a VoD service, offloading work from the server onto the P2P network in a scalable and demand-driven fashion.

1. INTRODUCTION

Recent years have seen a dramatic increase in computational capability and network bandwidth available to end-users in commercial, home, and mobile environments. The long-running trend of exponential performance growth per unit cost has now yielded systems ranging from ubiquitous handheld entertainment devices to high-end desktops connected through Gigabit Ethernet. As the computational capability and network bandwidth available to end-users has increased dramatically, all levels have seen demand for richer and more targeted content.

With the proliferation of inexpensive broadband connections, many applications have arisen to take advantage of this widespread bandwidth. One of the most commercially important and technically challenging applications is Video-on-Demand (VoD) service for high-quality, full-length movies. For example, the most recent annual report of the largest cable company (Comcast) starts with the sentence "On Demand is in" and states "Our growing ON DEMAND library attracted more than 1.4 billion views in 2005, nearly a 150 percent jump over the previous year" [1]. This is an average of 62 views per year per customer household. These numbers can only be expected to increase as higher bandwidth links proliferate and library capacity increases, allowing a greater number of high-quality video selections for consumers. Forms of VoD range from news and entertainment clips on the Internet to high-quality full-length movie services offered by ISPs or cable companies.

For example, the mobile space has seen the arrival of the video iPod, as well as services such as Verizon V-CAST that bring video to cell phones. High-end entertainment has seen a rapid increase in pay-per-view and regional satellite channels. Even the educational environment has seen offerings such as iTunes U, which provides video and audio captures of classroom lectures over the Web. All of these trends indicate an increased need for efficiently serving on-demand video.

The key issue in the performance of Video-on-Demand (VoD) storage servers is managing the disk array. Hard disk drive capacity has been improving at a 100% annual rate since the introduction of GMR heads in 1997 [2], now reaching capacities as great as 500 GB with prices under $1 per Gigabyte. At the same time, the internal data transfer rate of disks, measured for sequential access between the magnetic media and the driver's internal buffers has been improving at roughly 40% per year. These improvements have come about primarily through the greater information density on disk. Disk access latency has also improved, but only at about 15% per year. While the capacity improvements in disks certainly enable the storage of an ever-greater number of high-bitrate streams, the performance trends only support substantial improvements if the workload offered to the disks consists primarily of large sequential transfers. Such trends motivate investigations into Video-on-Demand (VoD) servers. Substantial previous work has analyzed, prototyped, and built VoD servers [3–6]. However, more recent advances in storage technologies enable such servers to be built in a cost-effective and storage-efficient manner, requiring substantially different design strategies.

The network server field in general has achieved great performance improvements in recent years, with such strategies as event-driven software architectures and zero-copy I/O [7–11]. However, these works have focused primarily on workloads with small files and high disk cache hit ratios (e.g., web servers). The VoD workload is different, as each stream may be multiple Gigabytes and may by itself overwhelm the disk cache. The web proxy server workload sees substantial disk access, and some software strategies have focused on minimizing seeks [12]. However, those file sizes are still small compared to the VoD workload, and the performance levels reported by proxy benchmarks are less than 300 Mbps [13]. Theoretical computer science has also produced advances in parallel disk models and algorithms, but these do not account for real hardware or software characteristics [14].

Rapid improvements in magnetic and electromechanical technologies have facilitated greater information densities and capacities in hard disk drives [2]. The use

of point-to-point links in modern Serial ATA (SATA) interfaces has also allowed for the construction of cost-effective and resource-efficient disk arrays, since individual disks no longer need to contend for shared bus resources as in the older Ultra ATA and SCSI standards. Such disk arrays are ideally-suited as building blocks for Video-on-Demand (VoD) servers. Since a modern 500 GB commodity disk can hold more than 100 DVD-quality (4.7 GB) multimedia streams, a single 2U Video-on-Demand server could hold 800 movies. A standard 42U rack of such servers could hold over 16000, dwarfing the corner video store. Five such racks would yield a greater selection than Netflix (65,000 movies as of 10/1/2006). Although modern systems can easily accommodate the capacity requirements of VoD storage, it is far more challenging for the server to actually schedule all the distinct request streams to the much smaller number of disks while meeting real-time delivery targets.

Although a disk array can achieve high aggregate throughput by employing parallel disks, such parallelism is only possible if the system achieves good load balance across the disks. Achieving such balance is complicated by workload hotspots and variations, as well as by "fail-stutter faults": performance degradations caused by manufacturing variations, by hardware glitches such as seek errors and bus timeouts, or by geometry-related variations (e.g., slower reads toward the center of the drive platter as a result of lower linear momentum).

One approach to achieving load balance in the presence of workload hotspots is disk-striping, popularly called RAID-0. Although disk striping can provide ideal load balance by parallelizing requests across disks, it may also suffer from poor performance. Reddy and Banerjee observed that disk striping thus increases the effective service time of a request, reducing aggregate throughput of a disk array [15]. In particular, the actual transfer rates of striped data accesses will be accelerated by the degree of striping, but the seeks must take place across all the disks, creating occupancy at the disk heads. Arpaci-Dusseau further showed that the performance of a RAID-0 array degrades to the performance of the slowest disk, crippling performance in the case of a single-disk performance degradation [16]. These problems can be mit-

igated to some extent by choosing disk stripe sizes large enough to accommodate any single request, but this may be difficult when dealing with disk requests of multiple sizes: choosing too large will reduce parallelism for small requests but choosing too small will increase seek overheads for large requests. Although designed primarily for redundancy and availability, RAID-1 (disk mirroring) may also improve performance by splitting a read-dominated workload between two disks [17].

Randomization is a useful alternative to disk striping to reduce the likelihood of hotspotting. In this strategy, data blocks are mapped to disks randomly, with any given block of data equally likely to reside on any disk. Hotspotting is thus eliminated statistically, and this strategy has been adopted in Video-on-Demand systems like RIO [18]. Theoretical results have nevertheless shown that randomization may also suffer from hotspotting. In particular, when D blocks are randomly selected from D disks, it can be expected with high probability that one disk has requests for $\Theta(\frac{\log n}{\log \log n})$ blocks, using balls-and-bins analysis [19]. Even with careful randomization, there are known to be problems such as external sorting which are likely to perform poorly [20].

Sanders et al. gave an elegant technique to avoid hotspotting by placing each logical data block on two randomly selected disks so that at least one copy can be read without I/O bottlenecks [21]. This strategy, called *random duplicate allocation* (RDA) trades off some capacity for more reliable performance, but the current rate of capacity growth may make this approach a reasonable tradeoff.

Numerous systems have targeted VoD through powerful network servers, which must multiplex a large number of distinct request streams across the disks in a storage array while meeting real-time network data delivery targets [3–5]. Potential difficulties in the storage arena stem from problems such as load imbalance across disks and can be solved with various flavors of randomization and striping [18, 22–24]. However, VoD systems may also experience difficulties in achieving high network throughput as a result of their network data delivery method. Whether they use UDP, TCP, or another transport protocol, VoD servers communicate with each client separately since each client may request a different piece of content, and since even multiple

clients requesting the same piece of content may be at different points in the stream. This potentially requires the repetitive delivery of the same content from the server. Although live streaming systems benefit from various flavors of multicast, this is not directly applicable to true Video-on-Demand.

An alternative model for data delivery on high-bandwidth communication channels is peer-to-peer (P2P) communication, which overturns the traditional client-server network model and makes all participants perform both client and server functions. Using P2P technology can greatly reduce the cost of distributing content, because the peers contribute their resources as well. Among various P2P systems, the most popular is BitTorrent. BitTorrent breaks each file into pieces to improve the efficiency and speed of file transfers. Participants seeking a particular file form a *swarm*. Entries in the swarm that have the complete file are called *seeds*. Members that join the swarm obtain data pieces from seeds and other new members depending on data availability, while also providing data to other new members as well. A "tit-for-tat" policy aims to insure that every participant must also provide service to others. Once a member has the full file, it also becomes a first-class seed and remains such until its user closes the BitTorrent session. By allowing an individual client to download pieces of a file simultaneously from multiple sources, BitTorrent improves user latency while also avoiding bottlenecks at a centralized server.

BitTorrent has many uses, such as distribution of Linux ISOs and other software updates. It is also extensively used for video data transfer (including piracy, though this use is officially discouraged) [25]. Despite its use in video environments, however, BitTorrent is fundamentally based on a download model rather than a streaming model: actual end-user access to the media content is not possible until the entire piece of content has arrived. The default BitTorrent policy is for a given swarm member to prefer to get the rarest piece of data available from another swarm member. This is beneficial for keeping more copies of this rare piece in the swarm and allowing other members to pull this data from multiple sources, but is not necessarily well-suited to the goals of the end-user who actually wants to view the data in real-time.

The rest of this dissertation is organized as follows. Chapter 2 shows the background of this research in Video-on-Demand and BitTorrent. Chapter 3 describes a model and prototype of a VoD storage server [26]. Chapter 4 shows achieving reliable parallel performance in a VoD storage server using randomization and replication [27]. Chapter 5 shows improving VoD server efficiency with BitTorrent. Chapter 6 discusses related work, and Chapter 7 concludes this dissertation.

2. BACKGROUND

The network server field in general has achieved great performance improvements in recent years, with such strategies as event-driven software architectures and zero-copy I/O [7–11]. However, these works have focused primarily on workloads with small files and high disk cache hit ratios (e.g., web servers). The VoD workload is different, as each stream may be multiple Gigabytes and may by itself overwhelm the disk cache. The web proxy server workload sees substantial disk access, and some software strategies have focused on minimizing seeks [12]. However, those file sizes are still small compared to the VoD workload, and the performance levels reported by proxy benchmarks are less than 300 Mbps [13].

The key issue in the performance of Video-on-Demand (VoD) storage servers is managing the disk array. Hard disk drive capacity has been improving at a 100% annual rate since the introduction of GMR heads in 1997 [2]. At the same time, the internal data transfer rate of disks, measured for sequential access between the magnetic media and the driver's internal buffers has been improving at roughly 40% per year. These improvements have come about primarily through the greater information density on disk. Disk access latency has also improved, but only at about 15% per year. While the capacity improvements in disks certainly enable the storage of an ever-greater number of high-bitrate streams, the performance trends only support substantial improvements if the workload offered to the disks consists primarily of large sequential transfers. Storage servers achieve high data transfer rates by employing parallel disks, but effectively utilizing parallel disks for large data streams is challenging because of the need to multiplex a large number of distinct request streams onto a smaller number of disks while meeting real-time delivery targets. Additionally, the server must carefully balance the load across the disks to extract the maximum possible parallelism from the disk array.

There are two important ways in which disk arrays can fail to achieve their desired parallelism. One way is due to realistic workloads in which some media objects are far more popular than others. Chesire et al. studied client-based streaming-media workloads and found that the distribution of client requests to objects is Zipf-like with a Zipf parameter of 0.47 [28]. In other words, the number of requests to the object ranked r is roughly proportional to $r^{-0.47}$. Because the workload is Zipf-like, there will be files that are requested more frequently than others, causing hotspotting. If files are not carefully spread across the disks, a workload hotspot will directly lead to disk load imbalance.

A second way for disk arrays to achieve less-than-expected parallelism is through the unexpected underperformance of a component. For example, Remzi and Andrea Arpaci-Dusseau cite several documented cases of disks underperforming when describing fail-stutter faults [29]. In one case, most disks delivered 5.5 MB/s sequential transfer bandwidth but one disk delivered only 5.0 MB/s due to bad block faults that were being remapped elsewhere on disk, introducing seeks. Van Meter pointed out that performance can vary by a factor of 2 across the different zones of a single disk; consequently, a disk reading from a slow zone will underperform another one in the same array that happens to be reading from a faster zone [30]. Other performance degradations occur for reasons that cannot be easily or directly analyzed.

2.1 Video-on-Demand

Video-on-Demand (VoD) has long been a research goal for system architecture, networking, and audio/video coding researchers, and hundreds of systems and solutions have been developed in these areas. A common way of implementing a VoD server is to use unicast and send each client a copy of the media, using one of several protocols designed for this purpose. For example, RTP (Real-Time Transport Protocol) provides end-to-end delivery services to support transmission of real-time data, and RTSP (Real-Time Session Protocol) is a control protocol for initiating and

directing delivery of streaming multimedia [31,32]. However, with hundreds or thousands of clients this unicast approach is inefficient. By taking advantage of the fact that the same files are requested by many of the clients, many techniques have been developed using multicast for nearly on-demand viewing, or using multiple unicast or multicast streams to reduce server load while still providing true on-demand service. Such schemes include patching, staggered broadcasting, hierarchical multicast stream merging, adaptive piggybacking, and periodic broadcast protocols [33–37]. However, IP multicast is rarely seen on the Internet or even intra-ISP networks, so these solutions have not had much impact.

Many newer techniques for video data delivery are based on some form of peer-to-peer or overlay multicast technology, all with their own protocols to manage peer communication and organization [38]. The first such system uses linear chains of clients to achieve functionality similar to IP multicast and uses these chains to implement a generalized batching technique for on-demand video [39]. GloVE combines these chaining and batching techniques, allowing multiple streams of data between different clients, but relies on IP multicast to make these streams efficient [40].

On-demand streaming systems such as CoopNet and PROP are closely related to this work. Both systems seek to support an infrastructure-based system with P2P networks and thus achieve scalability and reliability. CoopNet provides both live and on-demand streaming using a multicast tree rooted at the server and divides the streaming media content into multiple sub-streams using *multiple description coding (MDC)* to provide robustness [41]. When CoopNet is used for on-demand streaming, the P2P network is used only when the server is overloaded. The server is required to keep track of the peers and the content held by them, and redirect requests to peers when it is overloaded. (BitTorrent and Toast clients do this by themselves, preferring to get content from peers rather than the server). PROP is designed for intranets which deploy a proxy server [42]. At any time, the requesting client receives data from either a peer or the proxy server. If data is not available in the peers or the proxy, the proxy server requests the missing data from the media server. The BitTorrent

approach allows each peer to retrieve pieces of the video stream from multiple other clients simultaneously. This provides robustness by allowing multiple sources for each piece in case of node failure, and provides better performance by allowing downloaders to get pieces well in advance, reducing the chance that they will have to get them from the server.

Other works have provided analysis and simulations of proposed peer-to-peer VoD systems backed by servers, in which the peer-to-peer network is used to reduce load from the servers. Cui et al. propose oStream, a system using overlay multicast trees to stream most of the video from peers instead of the server [38]. They offer extensive analysis and some simulation results. Huang et al. describe a peer-assisted VoD service with different fetching policies based on mathematical models of client needs and the capabilities of the server and P2P network [43]. They provide the results of a discrete-event simulation model based on real-world traces of accesses to a video to show the potential of such an approach. Dana et al. propose a system similar to Toast called BitTorrent-Assisted Streaming System (BASS) [44]. Their work uses Torrent trace data from the distribution of Fedora Core 3 and develops a simplified model of BitTorrent client performance, which they simulate for several metrics. None of these approaches describes or benchmarks a system that has actually been implemented. Toast is a real implementation and thus allows for more detailed insight and analysis on actual performance issues. For example, using a real implementation allows this dissertation to explore various modifications to BitTorrent that better target this system for VoD.

2.2 BitTorrent

BitTorrent has become the most popular file distribution protocol on the Internet. This is primarily due to its efficiency and speed in transferring files. Previous P2P systems usually consisted primarily of a method to search for and locate files shared on the network. Once found, a peer simply requested the file from another

peer, which transferred it using HTTP or a similar protocol. These systems were primarily differentiated by their methods of locating content, but were all similar with respect to their transfer methods. BitTorrent on the other hand ignores the search problem. Instead, it relies on web sites or other common distribution methods to distribute small files called *torrent* files (sometimes called "dot torrent" files, due to their filename extension), each of which is essentially a descriptor of a file or group of files to be downloaded. Each file to be distributed has its own torrent file, and the group of clients downloading a particular torrent is called a *swarm*. Each swarm is independent and self-contained, but individual clients may participate in more than one swarm at a time. The swarm is managed by a simple network server called a *tracker*, which is responsible for keeping track of all clients in the swarm, and informing clients about each other. The tracker does not upload or download any file data; to begin file distribution requires at least one client which has the entire file, and which will upload to other clients in the swarm. Such a client is called a *seed*, and downloading clients which do not yet have the whole file are called *peers* or *leechers* (we will use the general term *client* to refer to either a peer or a seed). Peers become seeds once they have the whole file, and there are no distinctions between the seeds. (In particular, there are no differences between the original seed run by the original distributor of the file and other clients that have become seeds and are still participating in the swarm.) In most cases, once they have the entire file, clients will continue to participate as seeds until the user closes them.

There are two major innovations in the BitTorrent approach. The first is that each file is split into a number of small pieces (often 256 kB), and these pieces are transferred out of order. This means that peers that have different pieces of the file can exchange them, and that a peer can download different pieces from several other peers at once. In fact, since transfers are made at a granularity even smaller than the piece size (usually 16 kB), even a single piece can be downloaded from several peers at once. This can greatly increase the speed at which a file is transferred compared to simply downloading all of it from a single peer.

The second important feature of BitTorrent is an incentive strategy designed to reduce the impact of "freeloaders" (clients who download a lot of data but rarely or never upload anything) [45]. Such freeloaders are common in many P2P networks and do not contribute positively to overall system performance. In the BitTorrent system, each client maintains connections with many others, but is not necessarily willing to upload to all of them. Remote clients which have active connections with the local client, but to which the local client is not willing to upload, are said to be *choked*; all remote clients start out choked. A remote client may be randomly selected to be unchoked (called "optimistic unchoking"), or may be unchoked when the local client receives file data from it. Thus, sending pieces of the file to remote clients increases the performance seen by the local client. This strategy is called *tit-for-tat* because of its reactive nature; similar strategies have been shown effective in solving a variety of optimization problems in game theory [46].

For BitTorrent to operate effectively, peers communicating in a swarm need to have different sets of pieces so that they can exchange them. The choice of which piece to request from another peer can thus be critically important. Selecting pieces from the other peer's set uniformly at random generally does a good job of maintaining this "piece diversity" throughout the system. However, the standard BitTorrent client uses a "rarest-first" policy in which the client keeps track of how many copies of each piece exist among its peers, and selects the pieces with the fewest copies. Preferentially choosing the rarest piece has three benefits. First, it helps to ensure that all the pieces will still be available if all the seeds leave the network. Second, this scheme also improves the aggregate upload bandwidth available for the chosen piece since this peer will now be able to provide the piece to other swarm members. Third, it can maximize the possibility that a peer has something to exchange with other peers since it is unlikely that other peers have this content as well. Otherwise, other peers do not have the incentive to serve this peer due to the tit-for-tat nature of the algorithm.

To maintain piece diversity, the standard client uses random selection until a specified number of pieces have been downloaded (4 by default), and then switches to rarest first, randomly selecting among pieces with the same rarity.

3. A MODEL AND PROTOTYPE OF A RESOURCE-EFFICIENT STORAGE SERVER FOR HIGH-BITRATE VIDEO-ON-DEMAND

The goal of this work is to model and prototype a resource-efficient storage server for VoD applications with bitrates ranging from 1 Mbps (roughly the quality viewable on an iPod) to 25 Mbps (HDTV-quality), with a key operating point at 6 Mbps (DVD-quality). The server should be built using commodity hardware and as few disks as possible while also serving as many high-bitrate streams as allowed by the network links. The design should be based on sound principles resulting from the model-based analysis.

The contributions of this work are twofold. First, this work presents a performance model of a VoD storage server that accounts for the rate-based nature of streaming video, the buffering time allowed by the application, and the disk performance characteristics. The model is a substantial improvement over existing parallel I/O models both for its ease of use and for its ability to capture realistic hardware and application characteristics. Second, this work describes a prototype storage server designed to serve large video files at specified bitrates and finds its performance to agree closely with the model, with an average discrepancy of only 11% for high-bitrate streams. The system uses up to 8 SATA-300 disks and can simultaneously serve 290 distinct DVD-quality (6 Mbps) streams or 74 distinct HDTV-quality (25 Mbps) streams from disk, achieving an aggregate network throughput of 1.85 Gbps.

The remainder of this chapter proceeds as follows. Section 3.1 gives a model for predicting the performance of a VoD storage server. Section 3.2 describes a prototype VoD implementation, while Section 3.3 presents and analyzes the resulting performance. Section 3.4 summarizes this chapter and discusses possible extensions.

3.1 Modeling a VoD Storage Server

System design is a challenging process and can be aided by the use of models. Models are ideally detailed enough to capture important performance effects while still simple enough to use easily. The following gives a model for designing a parallel I/O system suited for delivering VoD from a modern disk array.

Parallel I/O. A popular model for theoretical analysis of parallel disk I/O systems is the parallel disk model (PDM) [14]. PDM accounts for a fixed-size memory buffer and assigns a unit cost for each disk access, casting the optimization problem as one of minimizing the number of atomic I/O steps. This model has been extensively used for analysis and optimization of external-memory applications under adversarial request models [20, 47–49]. Further, Barve et al. have provided a competitive online algorithm for retrieving a single multimedia reference stream from multiple disks [50]. However, the general problem of optimally scheduling multiple streams on parallel disks has been proven NP-Complete [51]. As a further complication, the model does not account for the variable latencies of real disks that consist of seek time, rotational latency, and transfer time, or for realistic application characteristics such as real-time delivery.

Disk characteristics. Seek time and rotational latency are independent of transfer size, but the transfer time depends on the transfer size used to access the disk and the disk's sequential transfer rate. Consequently, disks can be accessed more efficiently if the block sizes are large enough to amortize the overhead of seeking and rotational latencies. However, using exclusively large block sizes could be extremely wasteful when storing small pieces of data (e.g., typical Unix files with a median size of 10 KB [52]).

Rotational latencies between discontiguous disk accesses are uniformly distributed with an average of $\frac{30000}{R}$ milliseconds, where R is the rotational speed of the disk stated in RPMs. However, neither seek time nor sequential transfer rate is a constant; seeks that must travel a short distance are much faster than those that span the

disk, and transfers at outer tracks provide higher bandwidth because of the higher linear velocity of the magnetic medium. Properly considering the distributions of these terms in detail requires extensive knowledge of the layout and workload that may not be available at design time. Ruemmler and Wilkes have designed a disk simulation model and explain how various disk drive performance components can affect its accuracy [53]. They quantify the effectiveness of their model by comparing performance distribution curves for a real drive and the model output; they use the root mean square of the horizontal distance between these two curves as a metric called the *demerit* figure. Achieving a truly detailed model with a very low demerit figure requires a detailed understanding of the seek time, rotational positioning, disk data layout, and data caching characteristics of the disk. Although this level of information may be possible under certain circumstances, it is often difficult or impossible to extract from commodity operating systems and disk drives. Such detailed information is also difficult to acquire before building the actual system, making such a model less valuable for performance-oriented design. Consequently, the analysis presented here chooses to abstract out those details and only considers the average seek times and sequential transfer rates. (Note that even these measures incorporate some short seeks, as these are present in sequential transfers of large blocks.)

If the average access time (seek plus rotational latency) is termed t_a and the average sequential transfer rate is termed r_d, the average time to access a contiguous block of disk with size B is $t_a + \frac{B}{r_d}$. Typical values of t_a and r_d range from 17 milliseconds and 440 Mbps for high-capacity SATA drives to 6 milliseconds and 800 Mbps for high-performance SCSI drives. These values may be measured using tools such as lmbench [54].

Multimedia performance model. Real multimedia servers such as Video-on-Demand must schedule N simultaneous connections onto their D disks. Each connection is transferring content from a large logically contiguous file with a target bitrate (r_c). Additionally, the server may only force each connection to buffer data for a certain amount of time, as the buffer time affects the initial startup delay of

viewing the stream or the amount of time required for recovery after a fast-forward or rewind operation. We call this time t_b; based on user perceptions of Web quality of service, this time should be no more than 5–10 seconds [55].

We may define a measure of *disk availability* as the number of disks available over a period of time. Because the server has D disks, there are simply Dt_b units of disk availability during t_b amount of time. At the same time, each of the N simultaneous streams must retrieve $r_c t_b$ data in that time period to keep its client connection active; $r_c t_b$ is the conceptual block size of logical data retrieval in this system. Using the above formula for disk access time, each stream requires $t_a + \frac{r_c t_b}{r_d}$ units of disk availability in t_b time. We then solve for N by comparing the required disk availability and the total disk availability.

$$N(t_a + \frac{r_c t_b}{r_d}) = Dt_b \quad \Rightarrow \quad N = \frac{Dt_b}{t_a + \frac{r_c t_b}{r_d}} = \frac{Dt_b r_d}{t_a r_d + r_c t_b}$$

This represents the maximum number of simultaneous distinct streams available from the disk subsystem, assuming no contention for resources when accessing the disks. Alternatively, solving for D gives $D = N\frac{t_a r_d + t_b r_c}{t_b r_d}$ as the number of disks required to support such a number of streams. Since the streams must then be delivered on the network, N is also bounded by $\frac{r_l}{r_c}$ where r_l is the achievable link-level bitrate (949 Mbps for TCP/IP over Gigabit Ethernet.)

Accounting for RAID. A common approach to improving disk bandwidth for certain classes of applications is to use RAID to stripe data across disks [17]. In particular, each large sequential transfer will be striped across several disks depending on the RAID level used. The disk access (with cost t_a) will take place simultaneously across disks, but the transfer rate will be multiplied by the number of disks across which it has been striped, termed D_s. Note that D_s only considers the number of disks across which a given transfer has been striped, not the total number of disks in the array. In total, a sequential transfer of size $r_c t_b$ takes $t_a + \frac{r_c t_b}{r_d D_s}$ time, using $t_a D_s + \frac{r_c t_b}{r_d}$ units of disk availability. The disk usage corresponding to the access time has been multiplied but that from the transfer time is unchanged. Solving for N now gives:

$$N = \frac{Dt_b}{t_a D_s + \frac{r_c t_b}{r_d}} = \frac{Dt_b r_d}{t_a D_s r_d + r_c t_b}$$

Since this is strictly less than the non-striped version (and degrades as D_s increases), this model predicts better peak performance without RAID than with it. This is somewhat contrary to conventional wisdom but jibes with the observation of Reddy and Banerjee that disk striping increases the effective service time of a request [15]. A possible solution to avoid this increase is to make the disk stripe size at least as large as typical requests, thus giving a D_s of 1 for such transfers [56]. However, a server that supports multiple bitrates invariably has multiple reasonable request sizes; choosing the smallest leads to a large D_s for larger transfers, while choosing the largest gives no bandwidth improvement for small transfers. Consequently, the prototype server studied here manages the disks in the array as independent units rather than as a RAID.

3.2 Prototype Implementation

We have built and started to experiment with a prototype streaming storage server. The following discusses the hardware and software used in this study.

Hardware platform. This study uses two hardware systems, one based on a SATA-1 disk array and another based on a SATA-300 disk array. The SATA-1 platform uses a Promise FastTrak TX4200 4-port SATA RAID controller, a motherboard with a 2-port Intel SATA controller, 1–6 Seagate SATA-1 disks with capacity 400 GB and 7200 RPM rotation speed, two 2.8 GHz Pentium-4 Xeon hyperthreaded processors, two Gigabit Ethernet links, and 4 GB of DRAM. The SATA-300 platform uses a Promise FastTrak SX8300 8-port SATA-300 RAID controller, 1–8 Seagate SATA-300 disks with capacity 500 GB and 7200 RPM rotation speed, two 2.0 GHz dual-core Opteron processors, two Gigabit Ethernet links, and 4 GB of DRAM. As discussed in Section 3.1, this study configures the disks independently rather than as a RAID.

The SATA-1 disks achieve a disk access time (t_a) of 17 ms and a sequential transfer rate (r_d) of 525 Mbps. Although SATA uses a point-to-point link for each disk, the

controller is on a PCI bus that is limited to 2.1 Gbps. Consequently, inter-disk contention becomes an issue with 4 or more disks. With 4 disks simultaneously running, the per-disk r_d is 431 Mbps; with 6 disks (placing the last two on the motherboard controller), the per-disk r_d is 458 Mbps. The SATA-300 disks see t_a of 16 ms, r_d of 446 Mbps (because of their larger size), and negligible inter-disk contention because the controller has an 8 Gbps PCI-X host interface.

Server software. The software platform consists of a standard Linux 2.6.8 kernel using the ext3 filesystem for all data partitions. Because the operating system is unchanged, it may still allocate disk blocks at a minimum granularity of 4K; in practice, the ext3 filesystem tends to allocate file blocks sequentially. The content is stored on disk as large files, with one stream data partition per disk. The SATA-1 disk array is large enough to hold over 500 4.7 GByte DVD-quality streams, while the SATA-300 array can hold over 800. However, even these numbers are not enough to fully exercise the system at low rates. Consequently, these tests use an even larger number of 1 GByte streams. No stream data is reused in the filesystem cache across performance tests because each test overflows the cache in less than one minute.

Figure 3.1 represents the flow of operations in the application-level server software. The server responds to a client connection by creating a new thread and then sending $r_c t_b$ data as an initial buffer. It then enters a repeated pattern of sending $r_c t_b$ data, seeing how long it took to perform that transmission, and sleeping that connection for the remainder of t_b. If a transmission takes longer than t_b, the server sends the next chunk at a higher rate (by sleeping for less time) until the client side catches up. Each time the transmission takes longer than t_b, the server adds the excess time to a per-connection deficit variable (t_d) and sets the target time for the next transmission of $r_c t_b$ data to be $t_b - t_d$. This deficit is reduced (or reset) if the next transmission completes before the target elapses. If the client side repeatedly fails to keep up, its buffer will be exhausted and it will starve; the server will continue to send at a higher rate until its target has been met.

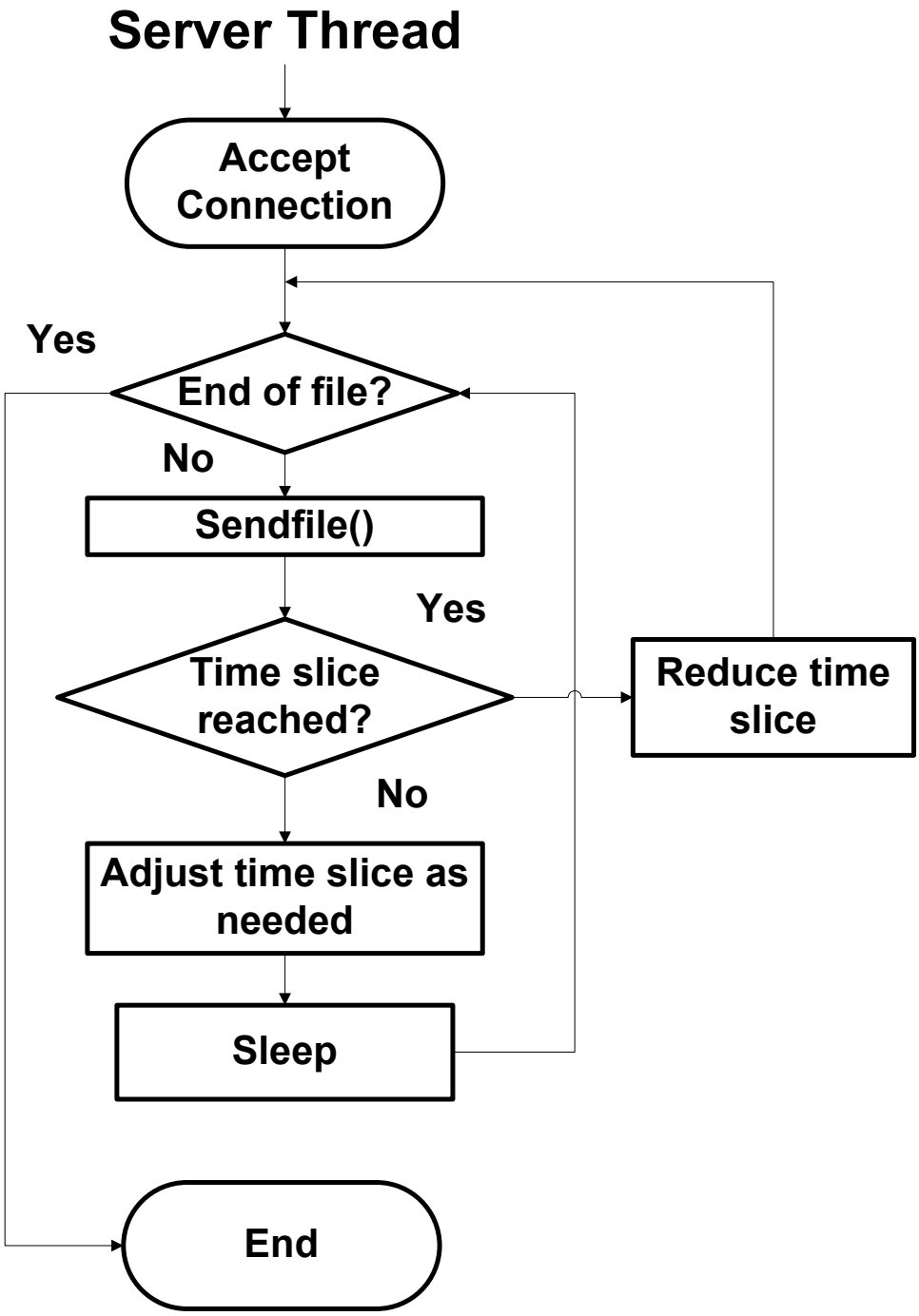

Fig. 3.1. Flowchart describing operations in server thread for processing individual Video-on-Demand (VoD) client connection

The performance tests use a t_b of 5 seconds and report a prototype N value as the maximum number of simultaneous connections to distinct streams that can be

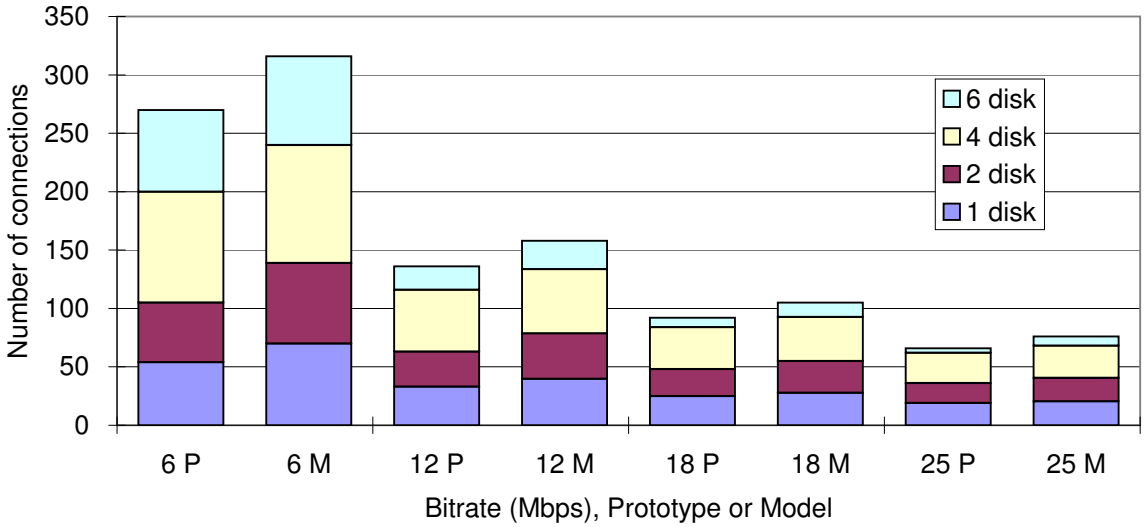

Fig. 3.2. Performance of high bitrate SATA-1 system according to model and prototype.

successfully supported by the server. This maximum value is determined by binary search; any test in which any client connection starves even once after the initial buffering (namely, in which its data buffer empties because of slow server responses) is considered to have failed. The client connections come from one or two client machines on the same LAN as the server, each initiating parallel requests for distinct content. The prototype delivers the content via HTTP over TCP/IP using the application-level rate-pacing described above; realistic multimedia protocols would add some network overhead but would not change the disk access pattern.

3.3 Experimental Results

Figures 3.2 and 3.3 show the number of simultaneous distinct streams that can be served by the SATA-1 and SATA-300 systems for high bitrates. From bottom to top, the graphs represent the performance of systems with 1, 2, 4, 6, and (for SATA-300) 8 disks. In each case, one set of bars shows the prototype at each of the target bitrates: 6, 12, 18, and 25 Mbps, while the other set shows the performance predicted by the model (saturating at 1.9 Gbps because of link-level bandwidth limits). The

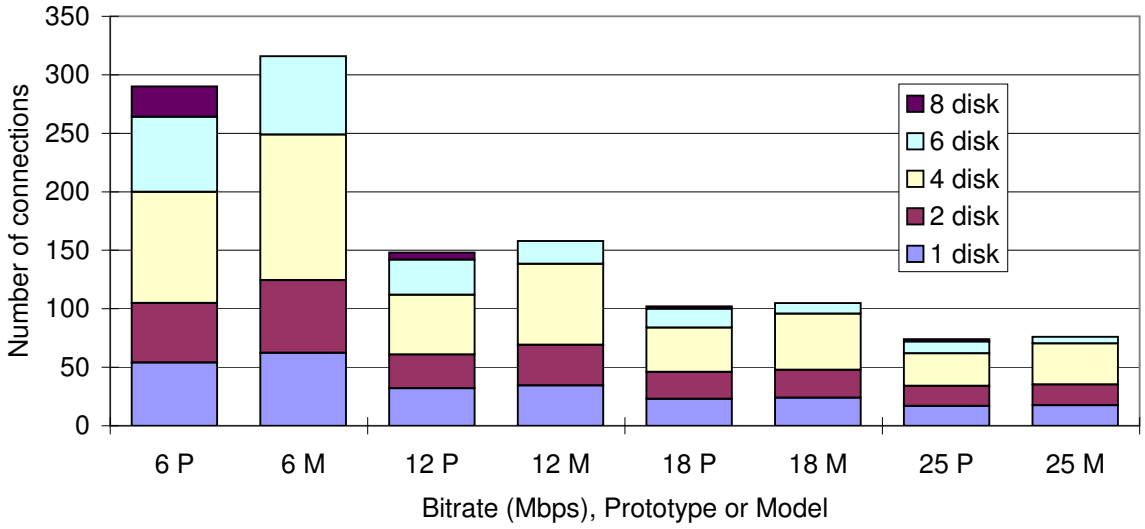

Fig. 3.3. Performance of high bitrate SATA-300 system according to model and prototype.

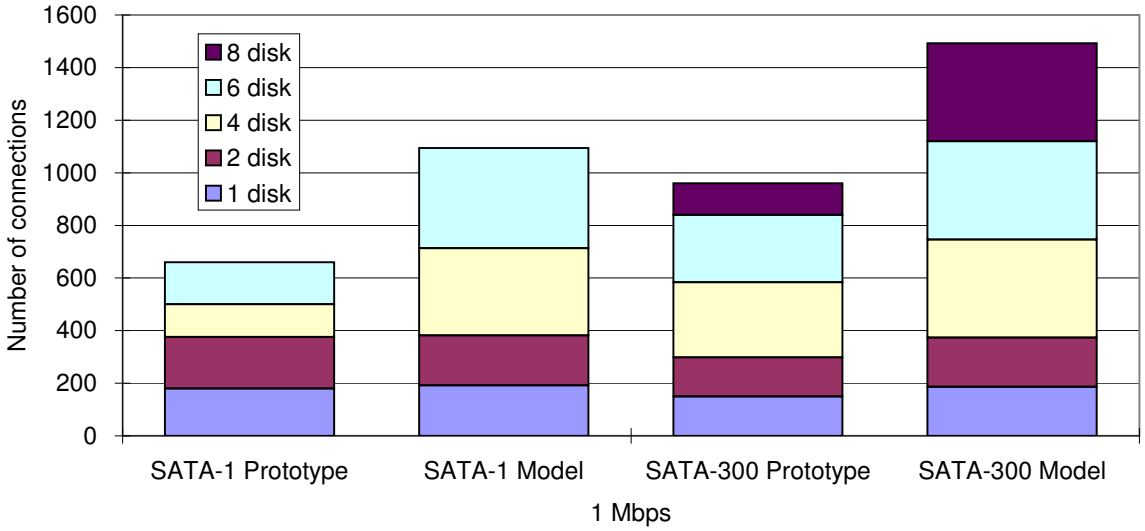

Fig. 3.4. Performance of 1 Mbps SATA-1 and SATA-300 system according to model and prototype.

model graphs are next to the prototype result graphs for comparison. The model graphs assumes the same r_d as in the 1 disk case except for SATA-1 with 4 and 6 disks. In these two cases, the model graphs account for inter-disk contention by using

the adjusted r_d values given in Section 3.2. None of the other configurations see noticeable inter-disk contention.

For the bitrates shown in Figures 3.2 and 3.3, the model closely predicts the performance actually achieved by the prototype. The average discrepancy is only 11% across all combinations of disk array size and bitrate. These results serve to validate the model and show that it can be used for effective system design.

Further analysis shows that in both systems, limitations from the operating system and PCI bus cap the actual network throughput to 1.85 Gbps; this measurement was taken by sending data from memory rather than accessing disk. This limit explains why the SATA-300 prototype does not scale to the expected 1.9 Gbps link-level bandwidth; the 8-disk SATA-300 system actually saturates the network for 18 and 25 Mbps streams. The remaining discrepancies may stem from other limitations not considered by the model, such as memory transfer bandwidth, I/O bus contention from other devices, CPU utilization, or seek time and transfer rate variations across the disk.

Figure 3.4 shows the performance achieved by the prototype and predicted by the model for 1 Mbps streams. The bars are similar to those used in Figures 3.2 and 3.3, representing the number of simultaneous connections to distinct streams as the number of disks is varied. The prototype values are within 15% of the model when only 1 or 2 disks are used in the SATA-1 system or with 1–4 disks in the SATA-300 system. However, the numbers start to diverge substantially as more disks are used. This discrepancy arises because the prototype uses a software concurrency architecture with one thread per connection, and CPU utilization becomes saturated with about 400 threads per processor. We are currently investigating alternative software architectures to achieve sufficient concurrency without a large number of threads.

3.4 Summary and Discussion

This work contributes to the state-of-the-art in parallel I/O performance modeling by providing a model that is simple enough to use for algorithm and system design while also being detailed enough to account for streaming application characteristics and disk hardware trends. We then describe a prototype storage server designed to serve large video files at bitrates ranging from 1–25 Mbps and finds its performance to agree closely with the model (with an average discrepancy of only 11% for high-bitrate streams). The prototype can deliver 1.85 Gbps of aggregate network throughput when serving distinct streams from disk, using a commodity PC-based server, an unmodified operating system, and inexpensive SATA disks. These results are promising by indicating that commodity equipment can be used to support emerging high-bitrate streaming media and that such systems can be designed and evaluated analytically.

One workload-related limitation of the model of Section 3.1 is that it targets only the maximum possible performance for delivering data from a VoD storage server's disk array. This performance level is achieved when N distinct streams are uniformly spread across D disks. The model should be expanded to consider other workloads. For example, some reuse patterns would allow successful file caching, reducing the disk utilization. On the other hand, hotspotting could reduce the effective parallelism of a disk array. Analysis has shown that possible solutions to avoid hotspotting such as RAID or random placement of disk blocks may still lead to a loss of concurrency and performance under certain workloads [20, 57]. Sanders et al. gave an elegant technique to avoid disk hotspotting by placing each logical disk block (in the case of this work, a chunk of at least $r_c t_b$) on two randomly selected disks so that at least one copy can be read without I/O bottlenecks [21]. Such a strategy trades off some capacity for more reliable performance, but the current rate of capacity growth may make this strategy a reasonable tradeoff.

4. ACHIEVING RELIABLE PARALLEL PERFORMANCE IN A VOD STORAGE SERVER USING RANDOMIZATION AND REPLICATION

In this chapter we explore the performance of various data allocation policies when applied to a disk array for serving DVD-quality Video-on-Demand streams. The policies of naive single-disk allocation, disk striping, disk mirroring, randomization, and random duplicate allocation are implemented on an actual prototype VoD storage server, using application-controlled mapping of blocks to disks. These strategies are tested under uniform workload distribution, various workload hotspot models, and single-disk fail-stutter fault models caused by degraded throughput from a disk. The results show that random duplicate allocation is the only strategy studied that performs well under each of the workload hotspotting models. RDA also suffers fewer and shorter client starvation incidents than most of the other schemes when exposed to single-disk fail-stutter faults. These results from an actual prototype storage server complement and experimentally confirm the prior theoretical analysis showing the advantages of RDA [21].

The remainder of this chapter proceeds as follows. Section 4.1 describes strategies for achieving reliable disk array performance. Section 4.2 describes the prototype server hardware and software, while Section 4.3 covers the experimental methodology. Section 4.4 presents the experimental results. Section 4.5 summarizes this chapter.

4.1 Achieving Reliable Performance

Reliable performance in a parallel I/O system requires the system to support a high level of concurrency even in the presence of workload hotspots and hardware performance variations. Effective mapping of request data blocks to disks can be

critically important in determining performance. Temporal variations in the workload and unpredictable hardware performance degradations make it impossible to find an ideal solution for all situations. However, some basic principles can help:

1. The entire contents of a stream should not be on one disk. Otherwise, severe disk load imbalances could arise for workloads with requests for multiple streams that happen to use only the same disk.
2. The granularity of allocating data contiguously should be at least as large as a single time slice in the stream. Otherwise, a single time slice from a single stream could require service from multiple disks, activating simultaneous seeks on each of those disks and thus reducing the aggregate throughput of the system. The remainder of this dissertation refers to such allocation units as *blocks*.
3. The performance of the system should not be dramatically impacted by the underperformance of a single disk.

Traditional file allocation in an operating system meets none of the above goals: a file is usually on a single disk, allocated with data blocks of 4 kB (compared to over 4 MB of data in a 5-second time slice of a DVD-quality video), and an underperformance on a single disk would impact any attempt by the system to access files on that disk. Modern extent-based file systems typically meet goal 2 above since they aim to put successive writes to a file in contiguous blocks on disk [58]. This dissertation refers to schemes that allocate each stream entirely on one disk as *naive*.

Disk striping (RAID-0) meets goal 1 by spreading data across all disks and may meet goal 2 by choosing the stripe size appropriately [24, 56, 59]. However, goal 3 is not met since prior work has shown that RAID-0 array performance tracks the slowest disk [16]. Disk mirroring (RAID-1) meets goal 2. Additionally, it may meet goal 3 by splitting the workload between the two disks in the mirror-pair intelligently [17]. However, RAID-1 does not meet goal 1 since hotspotting is only spread across a mirror-pair; this particular mirror still sees more load than the other mirrors. The RIO system stores media blocks on randomly selected disks to support multiple access

patterns and stream popularity variations [18]. Such randomization can enable goal 1 by distributing popular content across the disks; goal 2 can be met by choosing the granularity appropriately. However, randomization alone may actually exacerbate the problem of single-disk underperformance since every stream access will now touch the underperforming disk.

To address limitations in randomization alone, the random duplicate allocation (RDA) strategy replicates every content block on two randomly-chosen disks [21]. Figure 4.1 illustrates how the blocks of a file are replicated and scattered across the disks under RDA. Although replication in RDA was proposed to help improve load balance and tolerate workload variations, it may also be useful to tolerate single-disk fail-stutter disk faults since the more properly functioning replica may be chosen preferentially over the degraded one. In general, replication requires resolving coherence issues on updates; this is not relevant here since the Video-on-Demand workload is write-once, read-many. RDA trades off some capacity since twice as much storage is required compared with the naive allocation. In order for the naive allocation to have equal number of and size of files as RDA, naive would require double the number of disks. This would require more space, power, and cost.

4.2 Prototype Implementation

Server hardware. This dissertation reports the performance of a prototype streaming storage server that employs modern multicore processors and a SATA disk array. The system specifically includes two 2.2 GHz dual-core AMD Opteron processors, four Gigabit Ethernet links, 4 GB of DRAM, two Promise FastTrak TX4200 4-port SATA RAID controller, and 8 Seagate SATA disks with 400 GB capacity and 7200 RPM rotation speed. The SATA controller is configured in JBOD (Just a Bunch Of Disks) mode for all tests. Performance tests using lmbench indicate that the disks see an average access time of 17 ms and a peak sequential transfer rate of 525 Mbps;

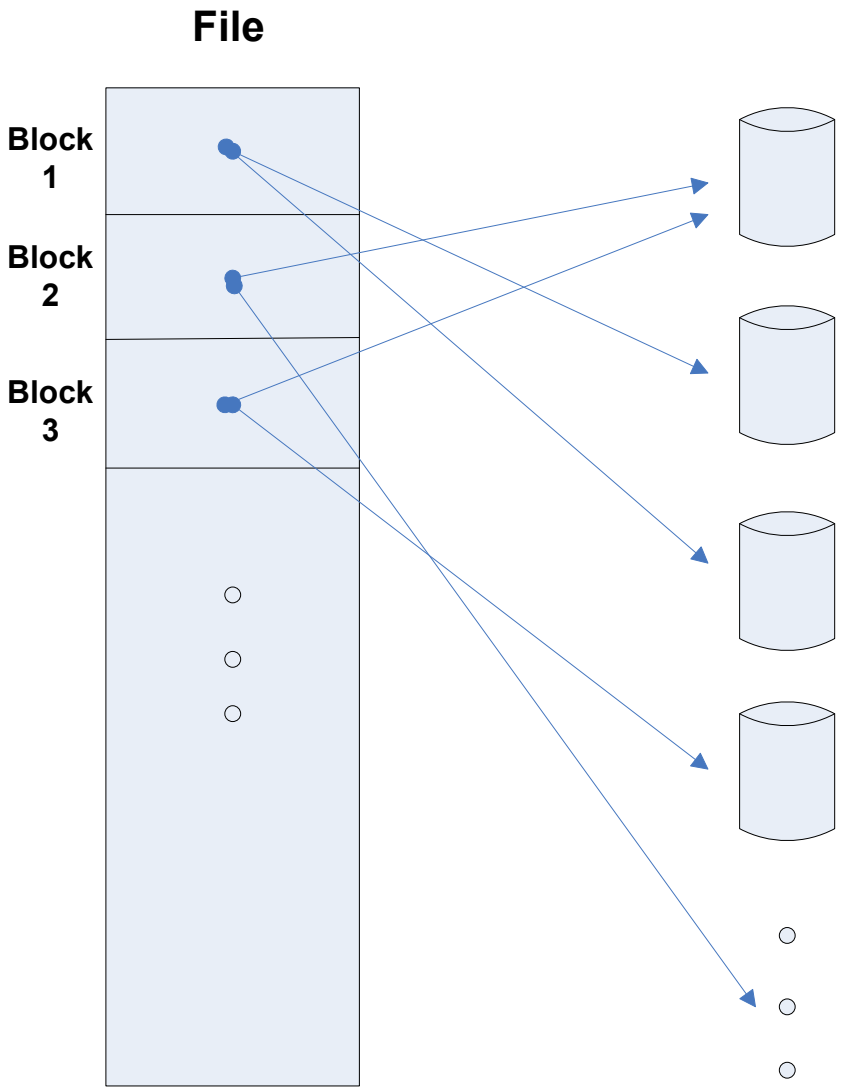

Fig. 4.1. Depiction of assignment of data blocks to disks using the random duplicate assignment strategy.

there is negligible inter-disk contention since each disk has a 3 Gbps point-to-point link to the controller and the controller itself sits on an 8 Gbps PCI-X bus.

Server software. The software platform consists of a standard Linux 2.6 kernel. Because the operating system is unchanged, it may still allocate disk blocks at a minimum granularity of 4K; in practice, the filesystem tends to allocate file blocks sequentially. The disk array can hold over 650 4.7 GByte DVD-quality streams. However, these tests use an even larger number of 1 GByte streams to represent a

broader selection of available content. Further, the tests only access the first 300 MB in order to allow the tests to complete more quickly. There is no substantial difference in performance between accesses to 300 MB and 4.7 GB streams; stream data is extremely unlikely to be reused in the filesystem cache in any case since each test overflows the cache in less than 30 seconds.

Stream blocks are mapped to the disk array using the allocation strategies discussed in Section 4.1: naive, RAID-0, RAID-1, random, and RDA. The mapping of stream blocks to disks is controlled through user-level code for all allocation strategies tested, allowing full flexibility in allocation without the complexities of operating-system coding. Test results show that user-level data management has less than 1% overhead compared to OS-level implementation of basic policies (such as naive file allocation). This is possible because no actual file data is passed to the user-level; the user-level only records disk numbers and offset positions for the stream data blocks. All interactions with the file data are performed using the Linux `sendfile` system call which directly dispatches data from a disk file to a network socket. The system also does not use hardware-controlled RAID because the controller does not support sufficiently large disk stripes and does not expose how it balances load in a disk mirror.

Figure 3.1 represents the flow of operations in the application-level server software. The server responds to a client connection by creating a new thread and then sending a media data block as an initial buffer. The media data block size is determined by the target bitrate and the time slice used for allocation. In this system, the target bitrate is 6 Mbps for DVD-quality streams and the time slice is 5 seconds. It then enters a repeated pattern of sending a data block, seeing how long it took to perform that transmission, and sleeping that connection for the remainder of the time slice represented by the block. If a transmission takes longer than that time slice, the server sends the next chunk at a higher rate (by sleeping for less time) until the client side catches up. Each time the transmission takes longer than the time slice, the server adds the excess time to a per-connection deficit variable and sets the target

time for the next data block transmission be the standard time slice less the deficit. This deficit is reduced (or reset) if the next transmission completes before the target elapses. If the client side repeatedly fails to keep up, its buffer will be exhausted and it will starve; the server will continue to send at a higher rate until its target has been met. The prototype delivers the content via HTTP over TCP/IP using the application-level rate-pacing described above; realistic multimedia protocols would add some network overhead but would not change the disk access pattern.

4.3 Experimental Methodology

This dissertation evaluates the resiliency of various file allocation to workload hotspotting and fail-stutter faults. Each of these types of tests is conducted by measuring the performance of the prototype VoD storage server described in Section 4.2. Four client machines on the same LAN as the server initiate parallel requests for content from the server. The results report the maximum number of simultaneous connections that can be successfully supported by the server for the given workload. This maximum value is determined by binary search; any test in which any client connection starves even once after the initial buffering (namely, in which its data buffer empties because of slow server responses) is considered to have failed.

This dissertation explores three workload models. The first is uniform popularity, in which streams are chosen with equal likelihood from the collection of available streams and the workload is exactly balanced across the disks in the naive allocation scheme.

The second workload model uses Zipf-like request distributions, with a Zipf popularity parameter of 0.5 to represent workload hotspotting akin to that seen by Chesire et al. [28] and a Zipf parameter of 1.0 to model greater request hotspotting. Each of these tests also uses a Poisson request arrival model, with a mean interarrival time of 1 second. In each of these Zipf popularity models, the naive allocation strategy is split into "naive expected" and "naive worst". In the former, the files are allocated

across the disks with no respect for their popularity ranking. In the latter, the files are allocated across the disks in such a way that the most popular files are clustered together on one disk, the second most popular set of files on the second disk, and so forth down the disk array. We do not consider "naive best" since the best possible placement of files on disks would depend greatly on ever-changing popularity and would thus require continual reshuffling.

For the third workload model, files known to be allocated on a specific disk or disk mirror are chosen with an extra likelihood in addition to a base uniform distribution; this distribution can be thought of as having between 20–80% overloading of one of the disks. This workload model only applies to naive and RAID-1 since RAID-0, random, and RDA all scatter accesses to all disks.

The dissertation also considers the impact of single-disk fail-stutter faults in which a disk achieves degraded sequential transfer rates, as in the examples discussed in Chapter 2. To model the impact of a single underperforming disk, each server access to that disk also invokes an additional transfer that reads in an amount of useless data proportional to the amount of requested data. For example, a 1 MB read combined with a 20% additional read will actually read 1.2 MB for every 1 MB of useful data. The VoD server is then subjected to a uniform request workload at the throughput level that was achieved by the performance tests. The results report the number of times that client connections starve for the given fault model, the average duration of each starvation incident, and the maximum duration of the starvation incidents seen for each allocation policy.

4.4 Experimental Results

4.4.1 Uniform Workload Distribution

Figure 4.2 shows the performance of the various stream allocation strategies when subjected to a uniform workload distribution. In this distribution, the clients request distinct streams that are chosen in such a way as to provide a completely

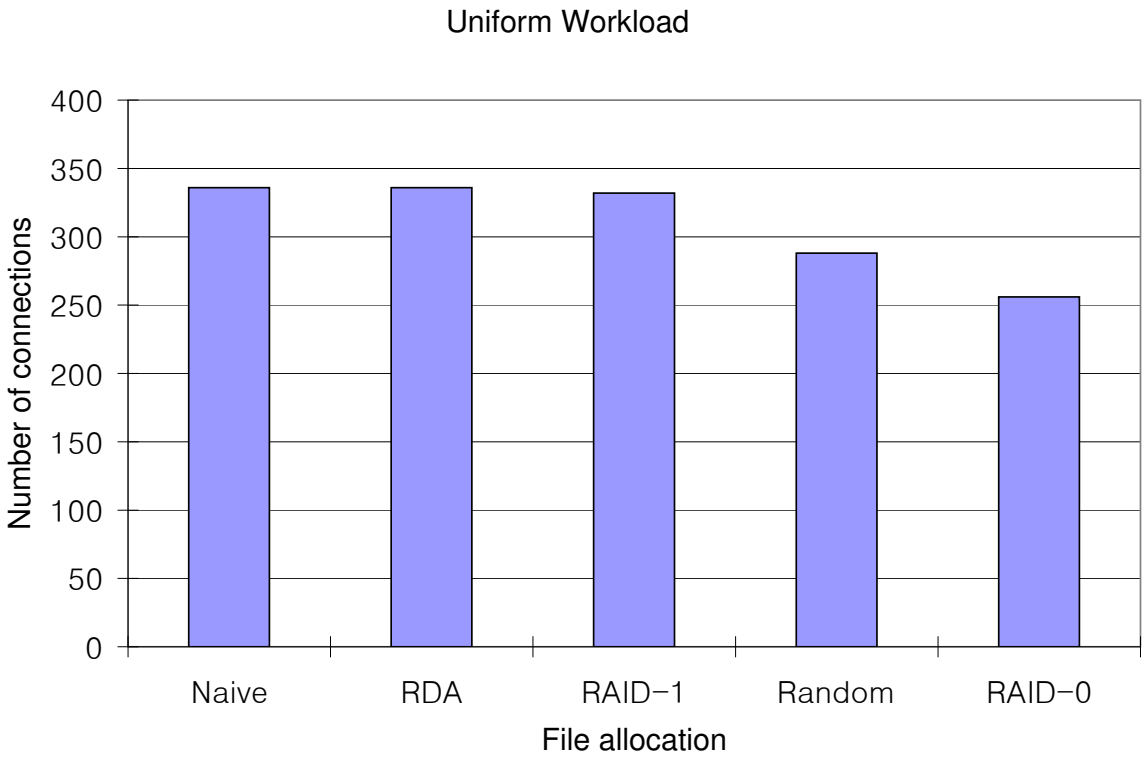

Fig. 4.2. Performance under uniform workload measured in terms of number of simultaneous DVD-quality streams supported by storage server under various allocation policies.

balanced workload even with the naive allocation strategy. Under this workload, each of naive allocation, random duplicate allocation, and RAID-1 perform nearly identically, achieving 332–336 simultaneous 6 Mbps DVD-quality connections without ever starving the client connections. It is worth noting that these results, with an aggregate throughput of over 2 Gbps, exceed any previously reported numbers from single-machine academic or industrial VoD servers of which we are aware. (Specific performance citations are provided in Chapter 6.)

The random allocation strategy achieves 288 connections; randomization may actually underperform naive allocation in a uniform workload since the effects of randomness may actually create some load imbalances as some disks are randomly favored over others. RDA avoids this problem because it always chooses the less heavily-loaded replica for a given data block. RAID-0 performs the worst, at 256

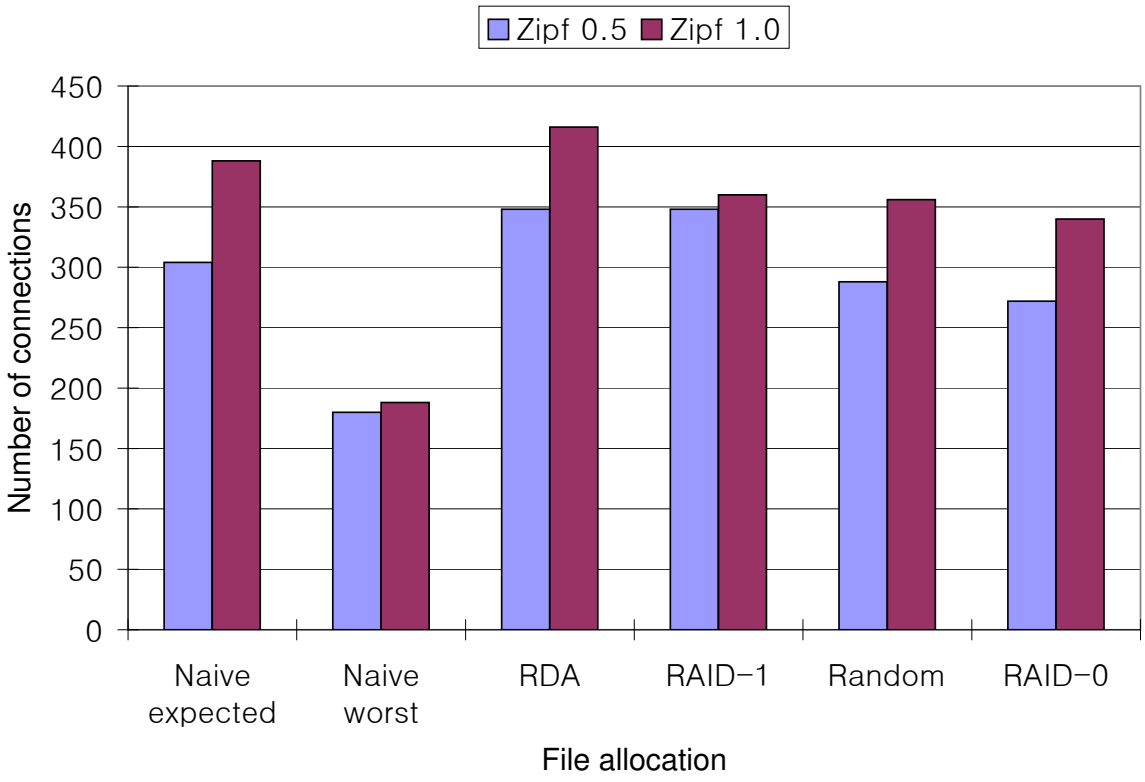

Fig. 4.3. Performance under Zipf workload with parameter 0.5 and 1.0, measured in terms of number of simultaneous DVD-quality streams supported by storage server under various allocation policies.

connections. RAID-0 actually sees load imbalance for this workload because the connections start nearly simultaneously and all the streams start their disk striping on disk 0. Similar problems have been observed with disk striping in problems such as external sort [57].

4.4.2 Workload Hotspotting

Figure 4.3 presents tests where the clients choose from the set of streams randomly with a Zipf distribution using a popularity parameter of 0.5 and 1.0. A parameter of 0.5 is similar to the actual popularities seen by previous work analyzing multimedia workloads [28]. The performance of naive allocation degrades by about 10% in the expected case and by nearly 50% in the worst case. Recall that naive allocates each

file on a single disk, with the popular files split among the different disks randomly in the expected case and clustered onto the same disks by order of popularity in the worst case. The other schemes perform similarly to, or slightly better than, their performance in the uniform distribution. Skewing may degrade performance by introducing load imbalance but may also help performance by allowing some file cache effectiveness; these effects seem to be roughly in balance for all but the naive allocations. The benefits of RDA and RAID-1 over the other schemes are slightly more pronounced here than in the other cases, with a 14% lead over the nearest competitor (naive expected).

A Zipf distribution parameter of 1.0 represents a still more skewed distribution. Here the positive effects of skewing in improving cache reuse are much more evident, as all but the naive worst-case allocation perform better than in the uniform distribution. RDA outperforms naive-worst by more than double, outperforms naive-expected by 5%, and outperforms RAID-1, Random, and RAID-0 by 15–20%.

Figure 4.4 examines another workload hotspotting model, where specific disks or disk mirrors are overloaded with a greater request rate. This hotspot model only applies to naive and RAID-1, since the other schemes already distribute files across all the disks. The number on the X-axis represents the overload percentage applied to the target disk or disk mirror above and beyond the base uniform distribution. For example, an overload of 40% indicates that the target disk will see 40% more load than the other disks. Note that the overload rates are not exactly 20, 40, 60, and 80% because the number of streams applied in overload must be an integer and may not divide nicely with the base value. The performance of the naive allocation strategy drops nearly 40% when an 82.5% overload is applied to one disk, while the RAID-1 performance drops 30% when a 75% overload is applied to one disk. This sort of overloading does not apply to RDA, random, or RAID-0, and thus does not affect their performance.

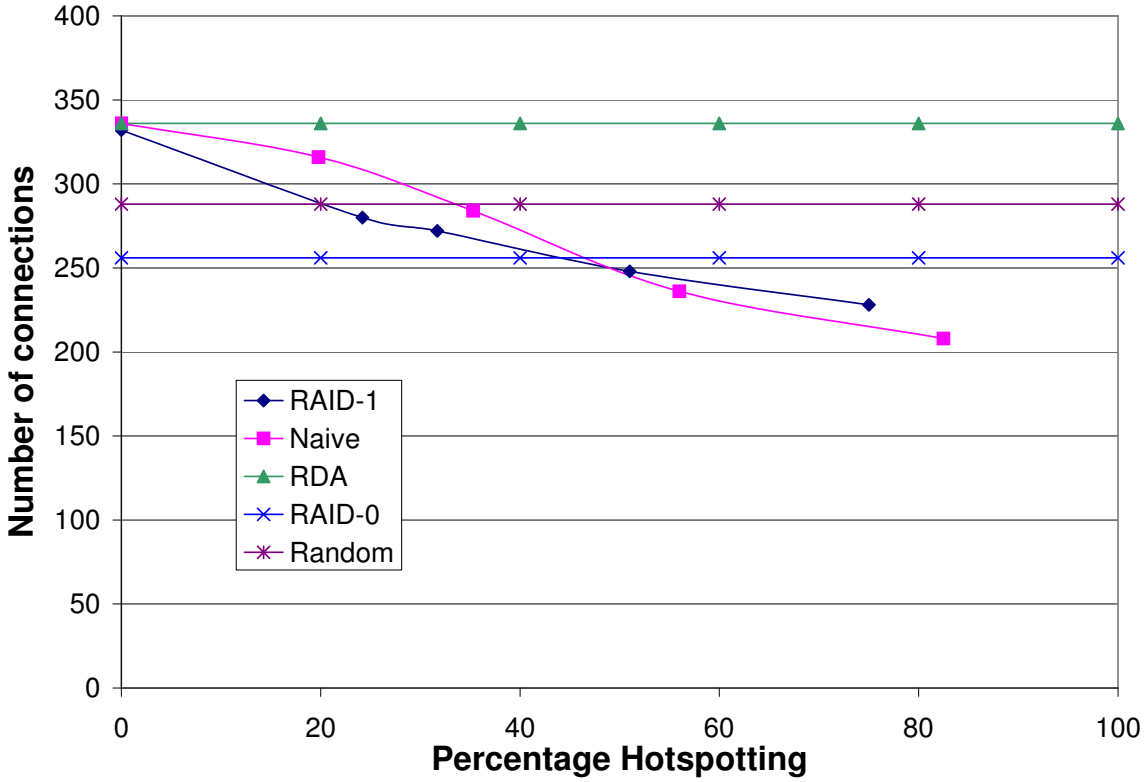

Fig. 4.4. Performance of stream allocation policy when streams allocated on one disk are subject to an additional load beyond the uniform workload distribution.

4.4.3 Fail-Stutter Faults

Figure 4.5 represents tests that model single-disk fail-stutter faults that lead to reduced sequential transfer throughput. As discussed in Section 4.3, these tests model a disk with degraded bandwidth by having the server read additional useless data on every transfer from disk. The X axis represents the percentage of additional data transferred read from disk along with the request. The Y axis represents the number of times that client connections starved as a result of this bandwidth degradation (called client stoppages). The bars for each bandwidth degradation represent RAID-1, RDA, Naive, RAID-0, and random, from left-to-right. Confirming the observations of Arpaci-Dusseau, RAID-0 sees a large number of faults when one disk is slowed down, as it offers no protection against a single slow disk [16]. Random file allocation behaves

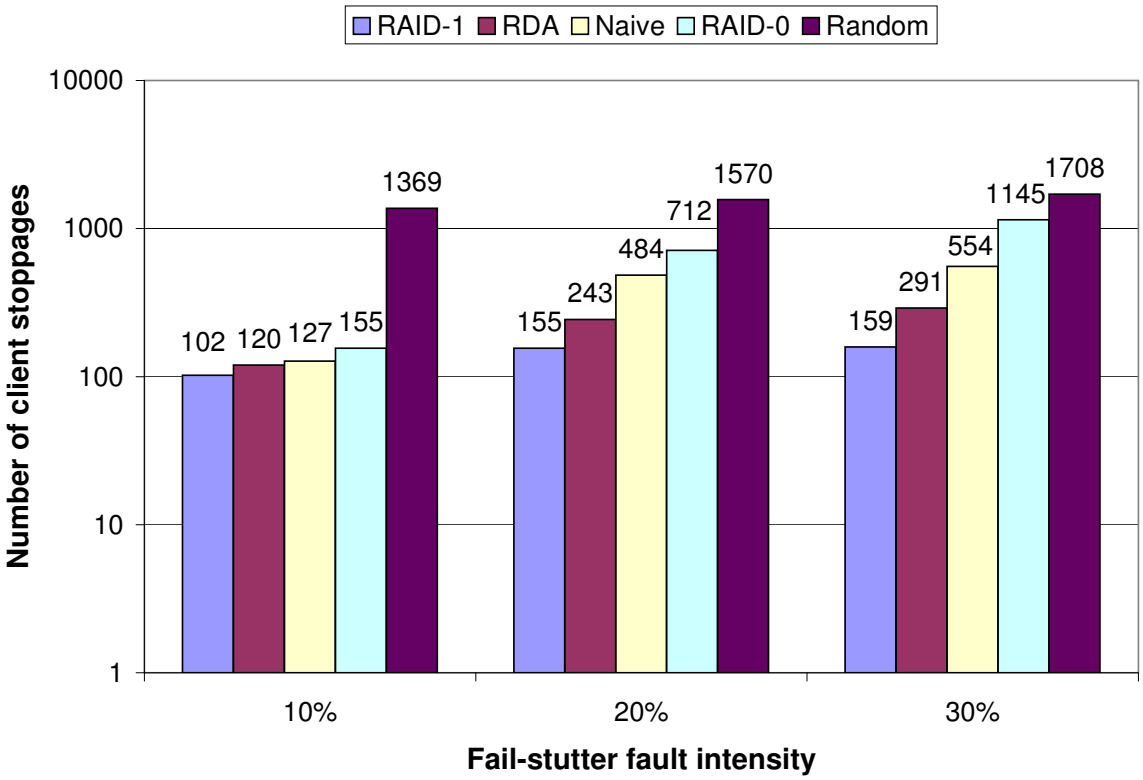

Fig. 4.5. Number of times that client connections starve when one disk in the system is subject to fail-stutter faults that lead to degraded sequential transfer throughput. The X-axis represents the intensity of the fail-stutter faults modeled, while the Y-axis represents the number of times that client connections starve.

even more poorly as bandwidth degrades since all blocks of all files are potential candidates for being allocated on the wrong disk. Interestingly, naive allocation does not perform as poorly as RAID-0 or random. This is because only those files that are actually allocated on the faulting disk will see timeouts; the other connections behave completely normally. In contrast, RDA always has the choice to avoid the faulting disk, which it generally learns since the faulting disk will tend to have longer work queues than the other replica. This explains RDA's performance superiority to random, RAID-0, and naive. RAID-1 has slightly fewer connection stoppages than RDA because of a fault-isolation effect akin to that seen with naive (only files allocated to the bad mirror will be effected by the fault).

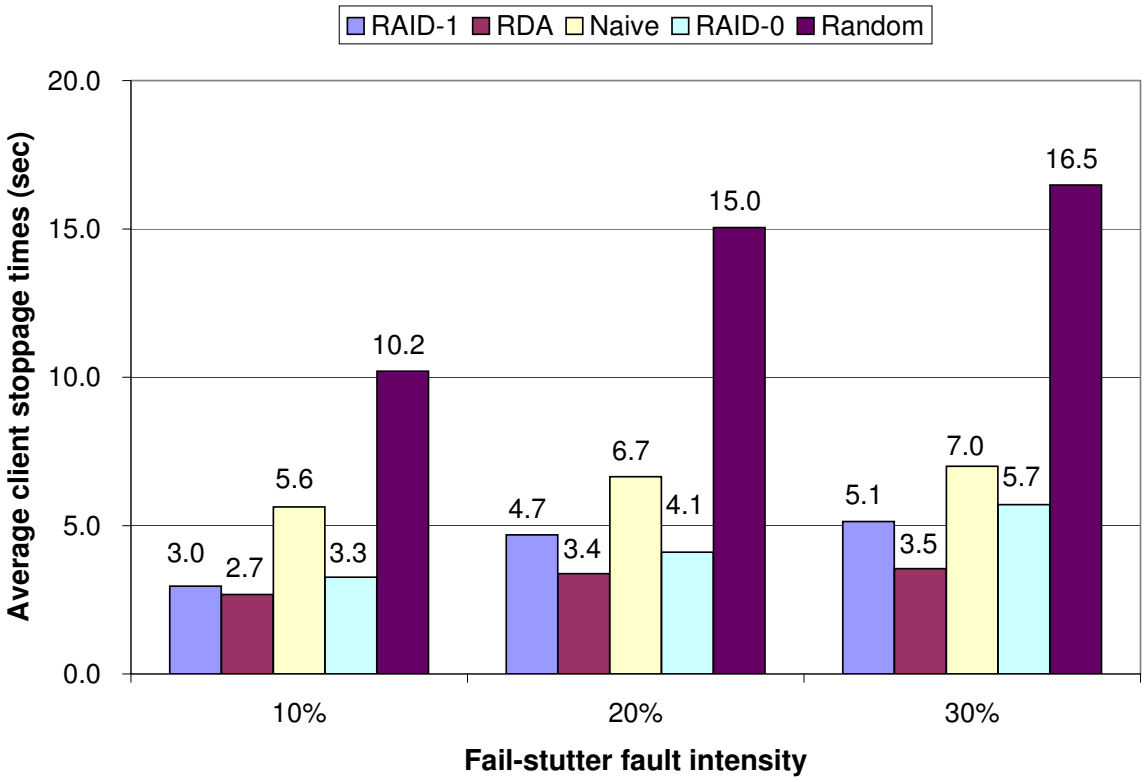

Fig. 4.6. Average time in seconds of a client connection stoppage when one disk in the system is subject to fail-stutter faults that lead to degraded sequential transfer throughput. The X-axis represents the intensity of the fail-stutter faults modeled, while the Y-axis represents the average time of a client connection stoppage .

Figure 4.6 reports the average duration in seconds of a client connection starvation incident. RDA is the fastest to recover from client starvations, on the average, with RAID-1 and RAID-0 about equal, and naive and random substantially worse. RDA recovers promptly from client starvations since even if a particular block is chosen on the faulting disk, the very next block will probably not even have the faulting disk as a choice and thus is certain to be delivered without delay. RAID-1, on the other hand, repeatedly sees the same choice of disks for every successive block in a stream, increasing the likelihood of one delay following another. RAID-0, naive, and random suffer because of a lack of choice in where to access any given block, causing the faulting disk to experience the same number of requests per unit time as any of the

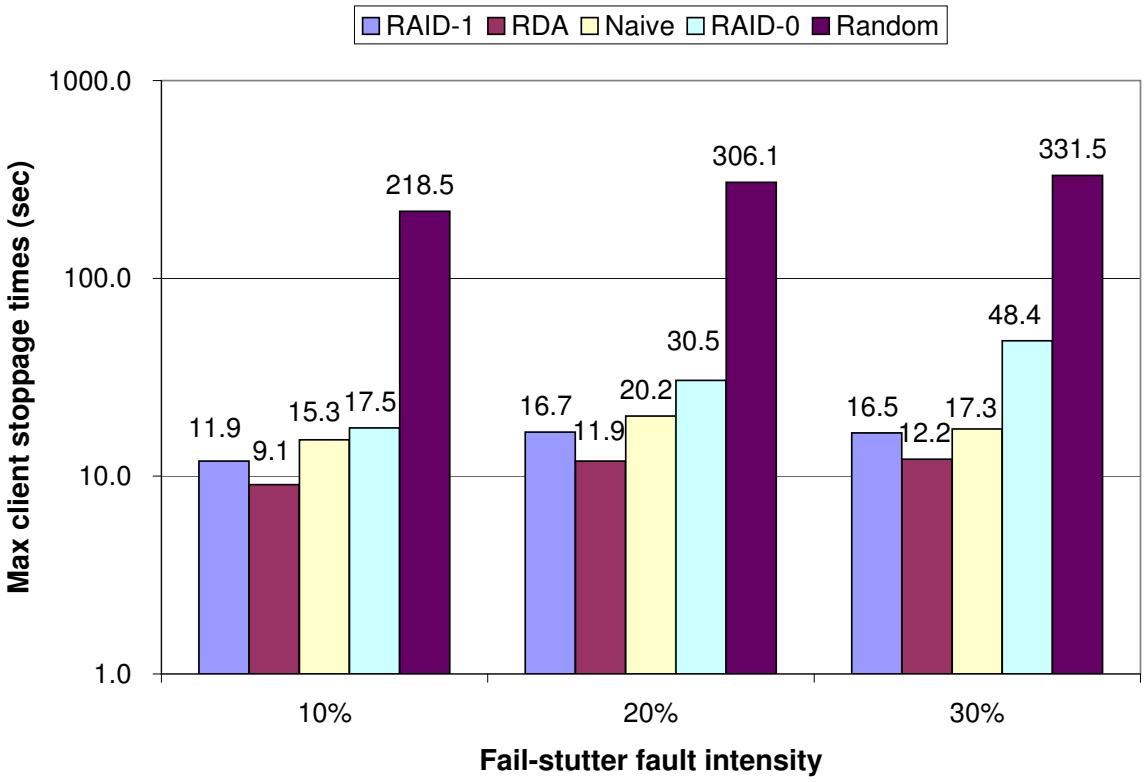

Fig. 4.7. Maximum time in seconds of a client connection stoppage when one disk in the system is subject to fail-stutter faults that lead to degraded sequential transfer throughput. The X-axis represents the intensity of the fail-stutter faults modeled, while the Y-axis represents the maximum time of a client connection stoppage .

disks in normal operation. Since the faulting disk cannot keep up with this request load, it is likely to cause large delays. The same basic trends are seen for maximum client stoppage times in Figure 4.7, with the exception that naive is noticeably better than RAID-0 in the worst case.

4.4.4 Results Summary

Random duplicate allocation (RDA) is the only strategy studied that performs well in each of the workload hotspotting models when applied to disk arrays for DVD-quality Video-on-Demand storage servers. It also suffers less average and maximum

client starvation time than any of the other schemes when subjected to single-disk fail-stutter faults that lead to bandwidth degradation, with a smaller number of total starvation incidents than all but one other allocation scheme (RAID-1). These results from an actual prototype storage server complement, experimentally confirm, and extend the prior theoretical analysis showing the advantages of RDA under simulated workload models [21]. We intend to extend this work by testing Video-on-Demand servers for various qualities of streams (e.g., 1 Mbps to represent iPod video quality), by building larger disk arrays, by experimenting with other workload models, and by examining other cases of fail-stutter faults.

4.5 Summary

A disk array can provide high aggregate throughput, but only if the server can effectively balance the load on the disks. This dissertation investigates the performance of disk arrays for Video-on-Demand (VoD) workloads when subjected to workload hotspots and single-disk fail-stutter faults. This dissertation focuses on the random duplicate assignment (RDA) data allocation policy which places each data block on two disks chosen at random, independent of other blocks in the same media stream or other streams. This strategy is compared to traditional single-disk file allocation, disk striping (RAID-0), disk mirroring (RAID-1), and randomization without duplication. The various allocation schemes are implemented and tested using a prototype VoD server with 2 dual-core Opteron processors, 8 SATA disks, and 4 Gigabit Ethernet interfaces running the Linux 2.6 kernel.

The results indicate that combining randomization and replication allows RDA to effectively tolerate workload hotspots better than previous schemes. When exposed to a single-disk fail-stutter fault that results in bandwidth degradation, RDA sees fewer client starvation incidents than all allocation policies except for RAID-1. Although RDA sees somewhat more client starvation incidents than RAID-1, these starvation incidents tend to be shorter in duration because randomization implies that

subsequent blocks from the same stream are unlikely to even be on the same faulting disk. These results from an actual prototype storage server complement and experimentally confirm the prior theoretical analysis showing the advantages of random duplicate allocation.

5. IMPROVING VOD SERVER EFFICIENCY WITH BITTORRENT

In this chapter we present a new system called Toast (Torrent Assisted Streaming) which aims to improve the efficiency of VoD servers by using direct P2P delivery through a video-targeted version of BitTorrent. The BitTorrent version in Toast has been adapted to support streaming real-time data delivery and to communicate with a traditional back-end VoD server when none of the swarm participants is able to provide the desired data in time for the user to view it. In essence, a P2P network acts as a distributed cache for a VoD server, offloading work from the server in a scalable and demand-driven fashion. The VoD server remains in the system, however, as a backup mechanism to allow clients to receive data pieces in real-time even when BitTorrent swarms cannot provide those pieces efficiently. Though Toast could be used on the Internet for the type of content seen on popular video sites such as YouTube, the focus of this work is higher-quality streams and a faster network; for example, a cable company could deploy a system like Toast on its set-top boxes (STBs).

The contributions of this work are as follows. First, we describe the design and implementation of Toast, providing experimental verification of its basic principles. Second, we explore the strategies by which BitTorrent picks the next piece of content to download. The results show that the default BitTorrent policy is not well-suited to the VoD environment. Instead, policies should favor downloading earlier pieces of content from other swarm participants to make it more likely that the end clients will not have to instead get the data from the VoD server. Third, we explore the impact of seeding on the effectiveness of Toast. The experimental results show that intelligent piece picking and persistent client seeding allow Toast to be quite effective,

offloading 70–90% of the network traffic from the VoD server onto the P2P network (depending on the upload bandwidth available at the clients).

Section 5.1 presents the overall Toast system and discusses the client and server implementations, along with the various client options that are tested. Section 5.2 discusses overall evaluation goals, the testbed and methodology used to investigate Toast. Section 5.3 presents and discusses the findings and Section 5.4 summarizes this chapter.

5.1 Implementation

5.1.1 Overview

Figure 5.1 shows an overview of the Toast system, and some details of the client. The Toast system is essentially a hybrid of a modified BitTorrent and a simple unicast VoD system. The tracker and and the seed client (which is optional in Toast), are unmodified. The Toast clients implement standard BitTorrent functionality, but also have an additional component. As in BitTorrent, each file has its own instance of the system, although some components may be shared. However, Toast also includes a VoD media server that can satisfy any client's request for a piece of the file.

The fundamental problem with BitTorrent for VoD applications is that the file pieces are downloaded out of order. This means that if a user is watching a video while it is being downloaded, it may come to a piece of the file that has not been downloaded yet (even though still later pieces of the file may have already been downloaded). If this situation arises or will arise soon, the client simply makes an extra request to the dedicated VoD server, outside of the BitTorrent system, to prevent interruption of the video stream. In this way, the VoD server serves as a kind of backup for the BitTorrent system. The "on-demand" nature of VoD is still satisfied, but with much lower overhead on the server, compared to systems where the server sends the whole file itself.

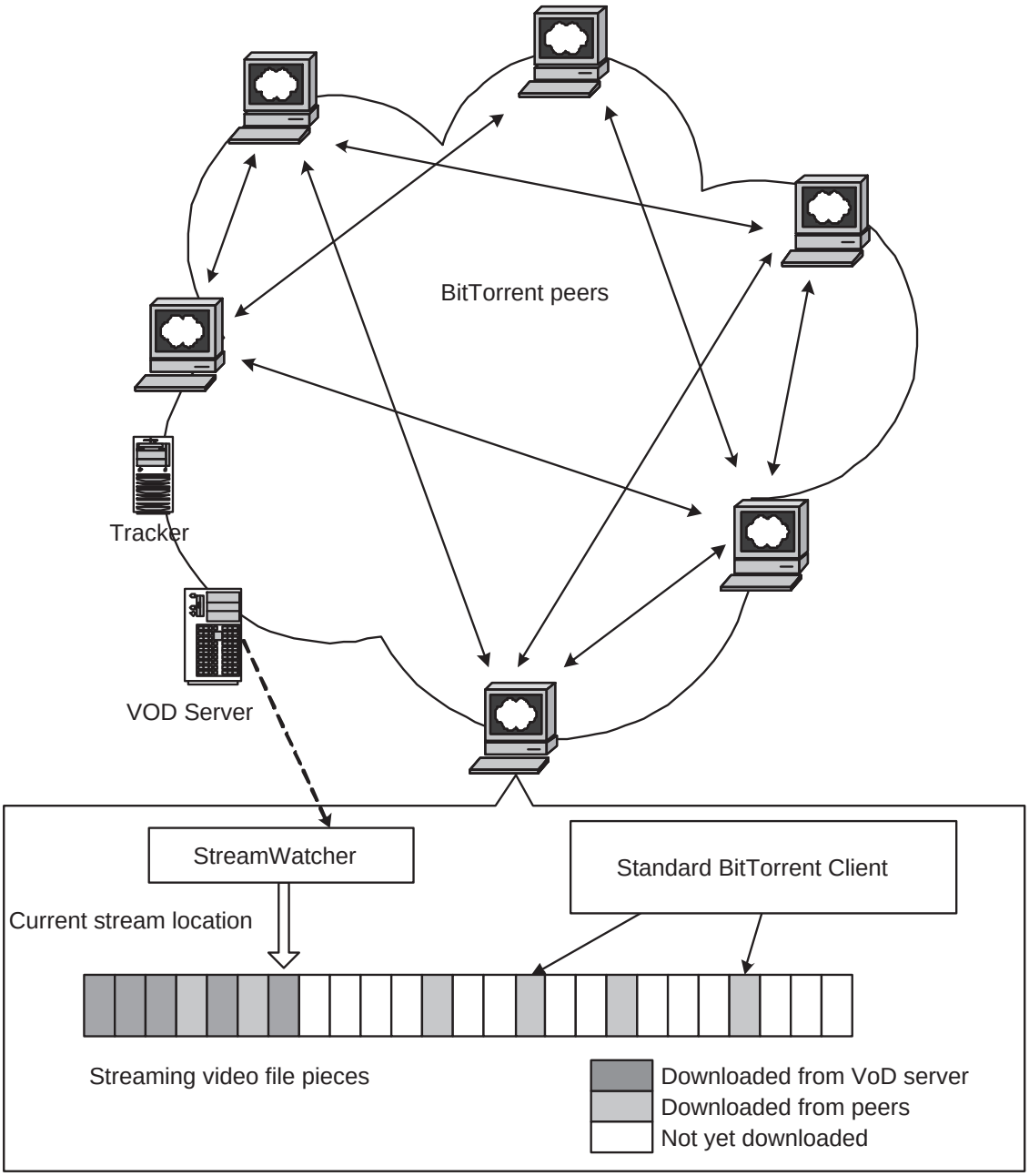

Fig. 5.1. System and client overview

5.1.2 Modified Client

The Toast client is based on version 4.4 of the official BitTorrent client from `http://www.bittorrent.com`, also known as the "mainline" client. This implemen-

tation was chosen from the many that exist because it was the original implementation by the author of the protocol (and so is a popular de-facto standard), and is written in Python, which facilitates relatively easy reading and modification of its code.

StreamWatcher. The primary addition is a new module called StreamWatcher, which tracks the progress of a conceptual video stream that the user is watching. (No actual video functionality is implemented; the StreamWatcher merely tracks a file position.) This is done by simply keeping track of time and multiplying the elapsed time by the bitrate of the video stream (assuming no use of VCR operations such as pause, fast-forward, and rewind). Alternatively, if an interface to a movie viewer were implemented, the viewer could inform the StreamWatcher of the current position in the file, updating this position as time elapses or as VCR operations are invoked. By knowing the file position, the StreamWatcher knows which block will be next needed by the viewer, and it uses the interfaces to BitTorrent's internal components to keep track of the download in progress.

Whenever the stream reaches a new piece of the file, the StreamWatcher checks the file store to see if the new piece has been downloaded yet. If it has, then nothing needs to be done until the next piece is needed. If the piece is not present, then it needs to be downloaded immediately or the video stream will suffer delays or breaks. So, the StreamWatcher sends a request to the VoD server (which is carried out in a separate thread) and downloads the piece. This download is carried out effectively out-of-band with respect to the BitTorrent swarm, but when it is complete, the piece is written into the file using the standard internal BitTorrent component which manages the downloaded file. In addition, the internal PiecePicker (which keeps track of which pieces have arrived and is responsible for selecting which piece to request next) is also updated. These measures ensure that the piece will not be selected and downloaded again from another peer. If there are any outstanding requests to other peers for parts of this piece, they are cancelled. Finally, this new piece is advertised to adjacent peers (just as if it had been downloaded from a peer) so that they can request it from the

local client. Taken together, these procedures effectively add the new piece to the existing swarm.

The BitTorrent portion of the client operates as usual, downloading pieces of the file from peers and writing them into the BitTorrent file store. The StreamWatcher keeps track of the current stream position, and downloads missing pieces from the VoD server when necessary, writing them into the same file store.

PiecePicker. The other major modification to the client is to the policy that determines which piece will be selected when the local client is ready to request a new piece from a peer. Unlike the standard BitTorrent, Toast has less reason to favor rarest-first since there is no danger that the file will become unavailable. Instead, the primary goal should be to reduce load on the VoD server as much as possible. This means that there is a potential trade-off in piece selection: biasing selection toward pieces that will be needed sooner would make it more likely that pieces are present when they are needed by the StreamWatcher. However, good piece diversity is still required for efficient BitTorrent operation, and more even distribution would facilitate this. It should be noted that the piece selection policy is only a preference. The PiecePicker is called when a remote client has indicated its willingness to upload (that is, it has unchoked the local client), but the remote client may not have the picker's preferred piece. In this case, the picker chooses the next most desirable piece, according to the policy. It is also possible that there are no pieces available that the client needs; in this case it must simply wait and try again later.

The Toast client implements three piece selection schemes in addition to the default. All policies share the common feature of "giving up" and not selecting pieces that are too close to the current stream position to download on time. For example, if a missing piece that represents 4 seconds of video will be needed in less than 4 seconds, and the remote client's upload rate is less than the video bitrate, then the download cannot finish before the piece is needed, so the picker will skip it and choose a piece further ahead. The amount of this lookahead is estimated based on the length of the piece, the client's upload rate, and the video bitrate. The first selec-

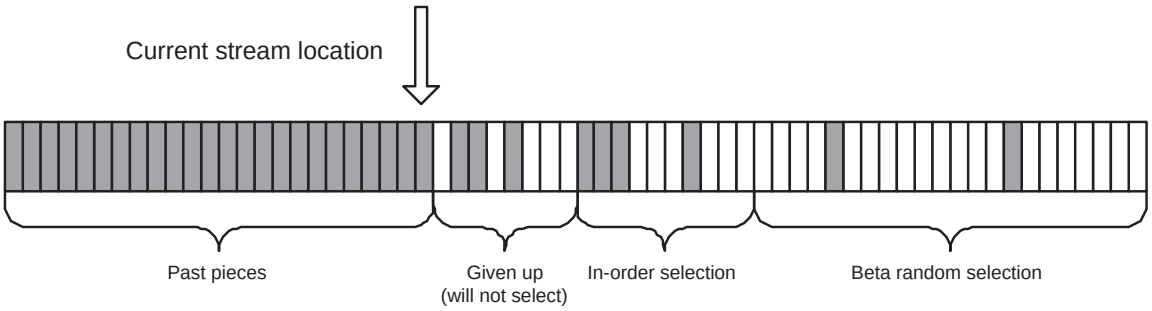

Fig. 5.2. Hybrid piece picking policy

tion policy, called "in-order," simply selects the piece that will be needed soonest by the StreamWatcher; that is, the lowest-numbered available piece. This shows a sort of greedy approach where piece diversity is ignored in favor of pieces which will be needed soon. The second policy, called the "beta" policy, selects randomly among all needed pieces using a distribution that favors earlier pieces over later pieces. This is implemented using Python's generator for a beta distribution with an α parameter of 1.0 and a β parameter of 2.0. This gives a probability density function that decreases linearly as piece number increases. This is an attempt to strike a balance between getting pieces that will be needed soon, and maintaining piece diversity.

Figure 5.2 shows the third policy, a hybrid of the previous two. Aside from the pieces in the past and those given up, It maintains two ranges of pieces: those that will be needed "soon" (e.g., in the next few minutes), and those that will not. It first attempts to ensure that it has all the pieces that will be needed soon by choosing in-order in that range. Then, if it has all the available pieces in this range, it chooses with the beta distribution among the remaining pieces. The size of the in-order range can be specified in seconds because the video has a known bitrate.

Local buffer size limit. In order to model situations in which the local client has only a limited amount of hard disk space to devote to the file, Toast has an option to simulate a limited local buffer. The size of the downloaded data is tracked, and when the size limit is reached, one of two actions are taken. First the client attempts

to reduce the local data size by ejecting pieces which are no longer needed (i.e. pieces earlier than the current file position, which have presumably already been watched). To simulate ejection of pieces, the client simply sends a new "bitfield" message to all of its peers, informing them that it no longer has the ejected pieces. If there are no pieces available to eject, the client then suspends all download activity by informing all peers that it is not interested in any of their pieces and by not requesting any new pieces until it can free space by ejecting pieces (new pieces will become eligible for ejection as the current file position passes them).

5.1.3 VoD Server

The server is a modified HTTP server. Instead of streaming the movie from start to finish, the server simply sends the pieces that are requested by the clients. The clients send an HTTP request with the name of the file, and the starting and ending byte (using the HTTP/1.1 Content-Range entity-header [60]) that correspond to the piece they need. The piece is sent using as much bandwidth as the server has available (rather than just at the video bitrate) to ensure that the client has the piece as soon as possible for its own playback and to share the piece with other clients.

5.2 Experimental Methodology

Toast is implemented in version 4.4 of the official (mainline) BitTorrent client and is tested using an actual peer-to-peer network and VoD server. To model the impact of a large number of peers, many instances of the client can be run on each of a small number of machines. The clients are managed by a script that ensures that each client runs in its own directory. Each client also uses a different TCP port for incoming connections. To keep connection speeds realistic, a client's upload bandwidth is limited by the client itself (in fact, this is an existing feature of the mainline client). The client uses only a small amount of CPU time, and provided that bandwidth limits can keep the network interface from being saturated, many

clients can be run on a single machine. Our testbed consists of 6 dual-processor 2.8 GHz Intel Pentium 4 systems and 4 4-processor 2.0 GHz AMD Opteron systems, all with 4 GB of RAM and connected by a gigabit ethernet switch. For all tests shown, 300 clients are distributed across 9 systems, and the server has its own Intel-based system. A seed client also runs on the server system during these tests, though Toast does not require this.

To emulate a cable-LAN, a Linux tool called NetEm was used to add a fixed amount of delay to all packets as they are sent, including those destined for other ports on the same machine [61]. NetEm is used to control traffic characteristics like delay and packet loss, allowing a lab environment to simulate a wide-area network. It is part of the Linux kernel version 2.6 and is controlled by "tc" (traffic control), a tool which is part of the iproute2 package. All machines used for testing (including the server machine) were assigned a one-way send-side delay of 4 ms using NetEm. The combination of this delay with processing time and propagation delay gives an average round-trip time of about 10 ms, corresponding to the latency characteristics of a metro-area LAN [62].

While there are many types of VoD services on many types of networks, one of the main challenging goals remains to serve high quality full-length movies, for example as an ISP service for its customers. In this type of situation, clients often have low latency and high download bandwidth to the ISP's network, though upload rates are more limited. A typical DVD quality movie in MPEG format would be about 4 GB streamed at a rate of 6Mbps. Due to disk space and bandwidth capacity, the number of clients that can be run on a single machine is limited when using such large files. Instead, the experiment here scales down the system to use 2 Mbps streams, each of which has a capacity of 858 MB. Note that even this lower bitrate exceeds the quality of one of the most popular legal video distribution schemes (iTunes Store) and is thus still a realistic operating range. Toast is evaluated using client upload rates of 1 Mbps, 1.5 Mbps, and 2 Mbps.

The testing scenario models a period in which clients come and go over time. For example, a set of 300 clients might watch the same 2 hour movie over a period of 8 hours, arriving at different times during the first 6 hours. To keep the test runs shorter but still model this scenario, the actual tests were run over a period of 4 hours, with clients starting in the first 3 hours, and all being finished by the end of the 4th. The clients arrived every 36 seconds at a constant rate. Three different client behaviors are tested: in the "download-only" behavior the client quits and leaves the swarm as soon as all the pieces are downloaded (which is before the stream reaches the end of the file). This is the behavior of many Internet users of BitTorrent who do not wish to share their upload bandwidth more than necessary. The streaming behavior downloads the file and continues to seed it on the network until the stream reaches the end of the file (1 hour from when the client started), which might be the behavior of a user who kept the BitTorrent client open while they watched the movie and then quit. The seeding behavior does not exit the client but seeds the file continually until the end of the test, resulting in all 300 clients being active at the end. Because the default behavior of most BitTorrent clients is to seed the file until the client is closed, this behavior is also commonly seen on Internet swarms, and might also be used if the VoD distributor controls all of the clients (for example, in cable set-top boxes).

The key evaluation metric is reduction of load on the VoD server. A simple unicast server would have to send the entire file to every client separately, with a total data transfer equal to the file size times the number of clients. Allowing the clients to send data to each other reduces this load on the server, ultimately allowing it to serve more clients. All traffic sent by the server machine (including the VoD server, BitTorrent tracker, and BitTorrent seed) was tracked at the network interface and compared to the amount of file data that would have been sent by a unicast server, equal to the size of the file multiplied by the number of clients.

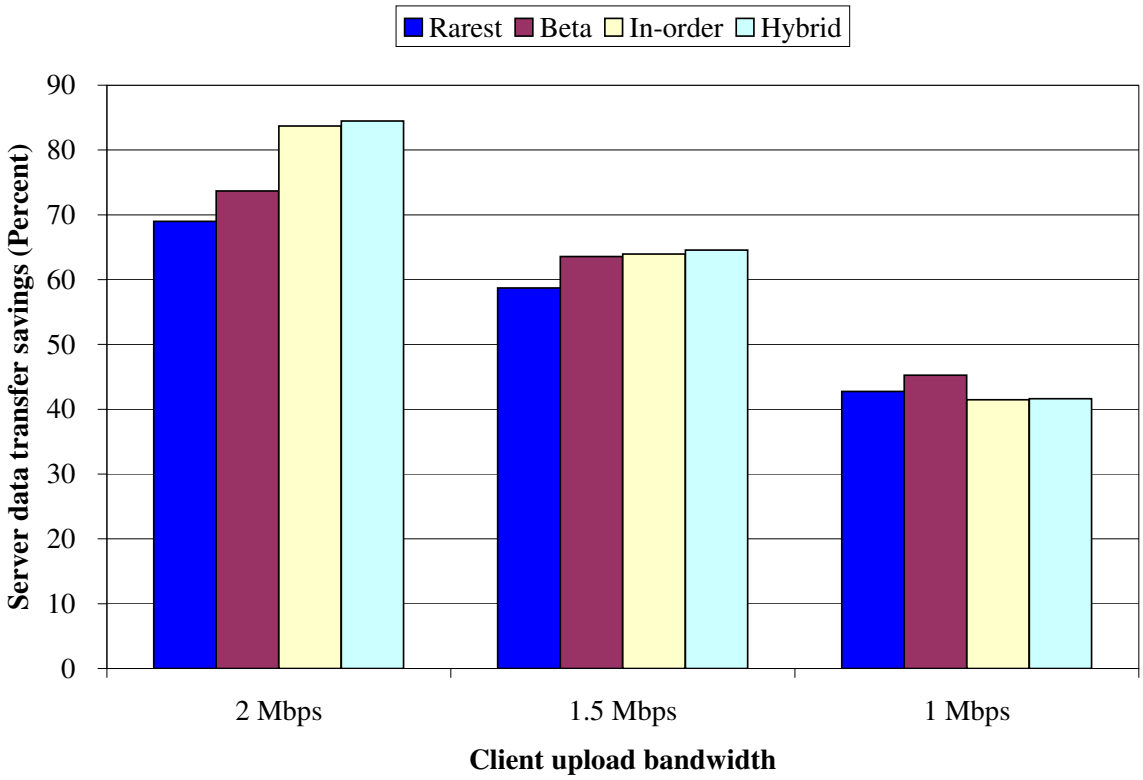

Fig. 5.3. Comparison of picker policies at different maximum client upload rates using streaming behavior

5.3 Results and Discussion

Impact of piece selection policies and upload rates. Figure 5.3 shows the test results as a percent reduction in total data uploaded by the server machine (including the VoD server, BitTorrent tracker, and the seed client) compared to a simple unicast server which sends the entire file to each client. In other words, it shows the percentage of total data transfer that was transferred between peers instead of from the server. Each client upload bandwidth rate is grouped together, comparing each of the four piece selection policies. The 2 Mbps upload bitrate matches the client upload bitrate with the video stream rate, while the 1.5 Mbps and 1 Mbps upload rates represent more constrained environments ($\frac{3}{4}$ and $\frac{1}{2}$ of the video bitrate, respectively). In these tests, all clients use the streaming behavior as described in

Section 5.2: they leave the swarm once the StreamWatcher reaches the end of the video content.

For upload rates lower than the video bitrate, it is impossible for the percent reduction to exceed the ratio of the upload rate to the video bitrate with the streaming client behavior. For example, if the upload rate is half the video stream rate, and each client uploads data for exactly the length of time of the video, then it can only contribute half of the total size of the video into the swarm. The same is true of each client, and the remaining data must be contributed by the VoD server. There was also a greater difference among piece picking policies for greater upload bandwidth, with the better overall performance giving them more ability to differentiate. As expected, the piece selection policies which are designed with streaming in mind almost always perform better than the default rarest-first policy. In addition, the in-order and hybrid policies performs much better with large upload bandwidth, whereas the beta policy performs better with low upload bandwidth. When most of the downloaded pieces can be successfully delivered by peers, it is most profitable to use the available bandwidth to deliver those pieces that will be needed soon rather than attempt to maintain diversity or fetch far into the future.

The 1 Mbps upload case is the only case where rarest-first performs slightly better than in-order and hybrid policies. Since rarest-first has no bias for earlier pieces, it is more likely than the others to select further in the future. This divergence arises as follows. If a piece in the near future is selected and the system has a low upload rate, there is a higher probability that the data will not come fast enough and will need to be downloaded from the VoD server anyway, despite the attempt to compensate for this effect by not selecting on nearby pieces as described in Section 5.1. Predicting the right number of nearby pieces to avoid is difficult because a client does not know what percent of the remote peer's upload bandwidth it is getting, or how many remote peers it can download a single piece from at a time. Consequently, the actual number of pieces that cannot be obtained from peers successfully will vary dynamically. Because the pieces in the expected in-order range are also less likely

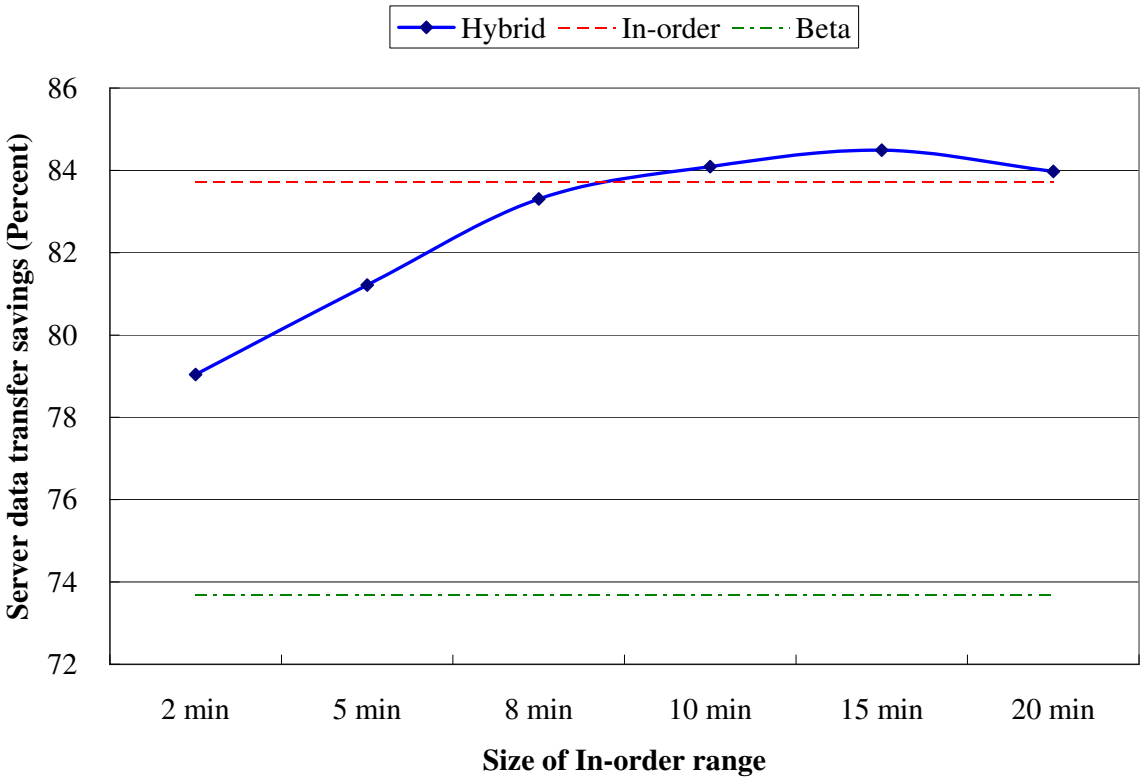

Fig. 5.4. Effect of in-order range size on hybrid policy using streaming behavior and 2 Mbps maximum client upload rate

to be available, the hybrid policy never gets a chance to escape to the randomized region. Consequently, it behaves just like the in-order policy.

Figure 5.4 shows the effect of varying the in-order range of the hybrid piece picking policy. This range is tested with values from 2 to 20 minutes. In these tests, the client upload rate was equal to the video bitrate and again the streaming client behavior is used. In-order and beta policy results are included for reference. Hybrid outperforms beta throughout the range considered; it also outperforms in-order from approximately 9 minute mark on, peaking at around 15 minutes. (The 15 minute in-order range was used in Figure 5.3.) This result, as well as the other results in this section show that the hybrid policy achieves its goal of capturing the benefits of early piece selection and piece diversity over a wide range of conditions and represents a

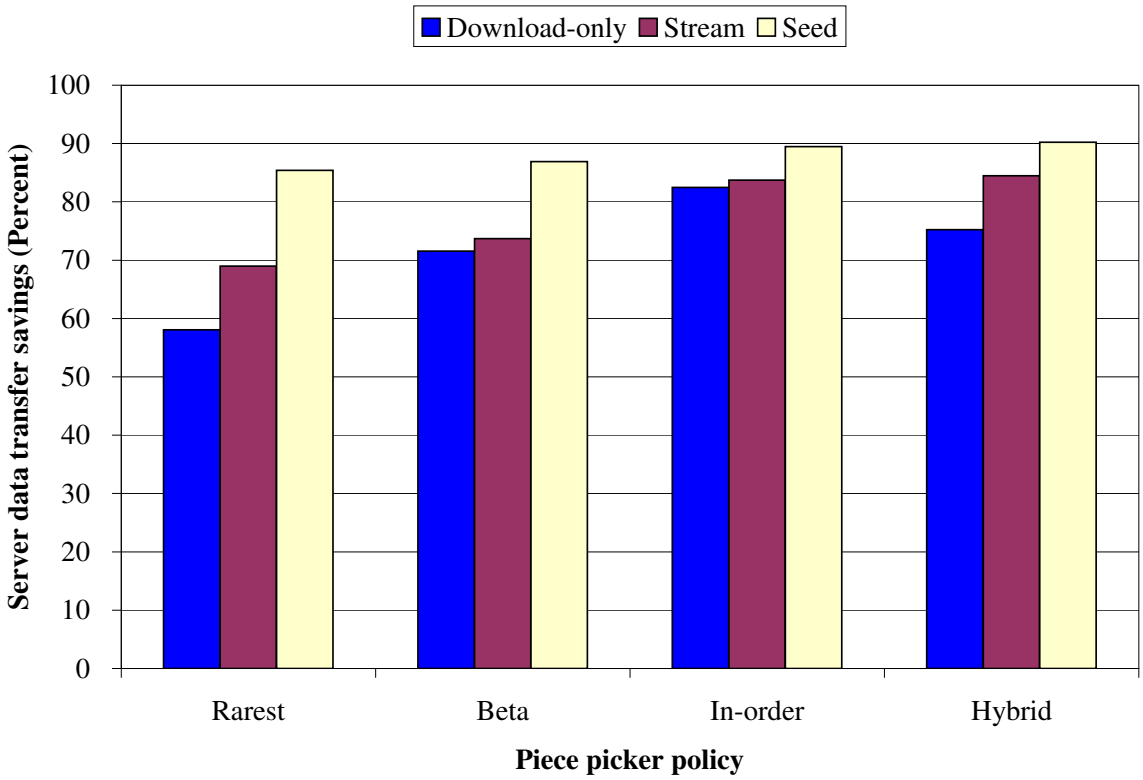

Fig. 5.5. Effect of client sharing behavior in 2 Mbps maximum client upload rate

good compromise between the more randomized rarest first and beta policies and the stricter in-order policy.

Client sharing/termination behavior. Figure 5.5 shows the effect of three testing scenarios when the client upload bandwidth is 2 Mbps. The clients are tested using the download-only, stream, and seed termination conditions described in Section 5.2 for each of the piece picker policies. As expected, the seeding behavior performs the best with server bandwidth reduced by up to 90% in the hybrid policy. However, even the download and stream scenarios see at least 70% savings with the three new piece picker policies. With the exception of in-order, the download-only scenario performs the worst since clients stop sharing the pieces when they are done getting the movie. The actual sharing behavior is likely to depend heavily on the client population. The stream behavior models a likely PC-based VoD scenario where the viewers will close the program upon completing the movie. The seed behavior aims to

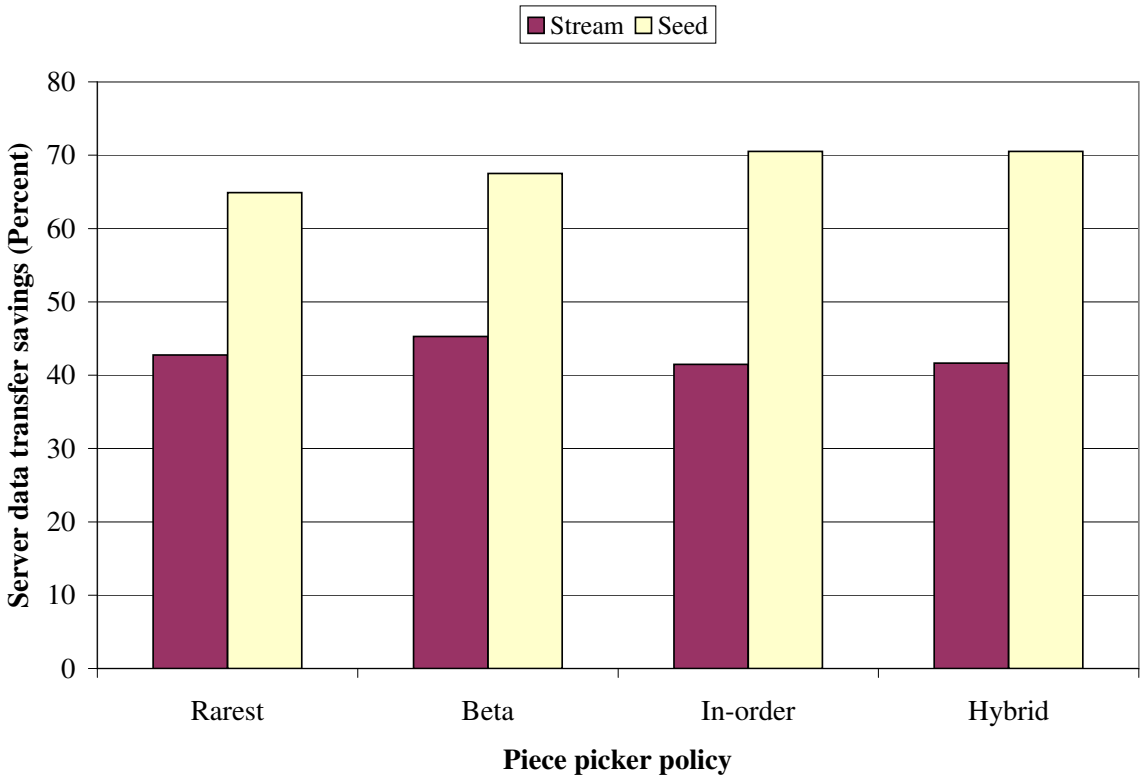

Fig. 5.6. Effect of client sharing behavior in 1 Mbps maximum client upload rate

model a situation where the clients are always connected and responsive, such as one where the client machines are controlled by an external system administrator. One possible deployment that could exhibit such behavior would be set-top boxes in a cable system; thus, the addition of a BitTorrent peer-to-peer network among nodes in a given area has the potential to greatly reduce the server infrastructure requirements of cable-based VoD.

Figure 5.6 shows the impact of the seed scenario when the maximum client upload rate is just 1 Mbps (half of the video streaming rate). As described above, the stream scenario would be theoretically limited to only reducing half the load; in practice, the server savings only reaches 45% at most. However, with the seed scenario, the savings reach 71%. This is actually superior to the savings that the stream scenario achieves with 1.5 Mbps upload bandwidth. This result tells us that even with the upload rate

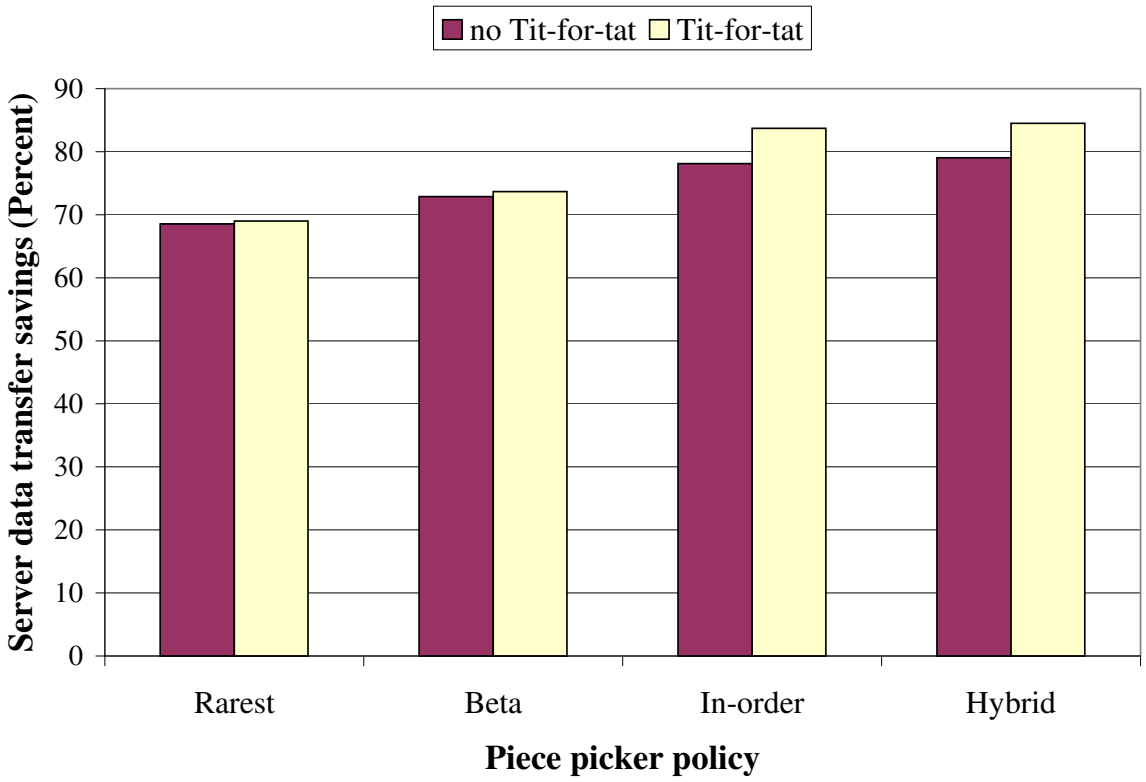

Fig. 5.7. Effect of peer choking policy using streaming behavior and 2 Mbps maximum client upload rate

only half of the video bitrate, Toast can be effective in reducing the server load if there are enough peers contributing.

Impact of choking policy. As discussed in Chapter 2, BitTorrent employs a tit-for-tat choking policy, in which peers prefer to upload to other peers from which they have already received data. This strategy is designed to provide incentives for the peers to serve data to others rather than merely acting as freeloaders. In a more controlled environment such as an ISP or cable set-top box, however, this may not be necessary, and a more equal scheme might be more efficient. The tit-for-tat choking policy is compared against a policy that does not consider peer download rates. This is the same policy that is used by a peer when it finishes downloading and becomes a seed. Instead of using download rates (which are no longer relevant for a seed), it uses upload rates and prefers peers which are not downloading from other

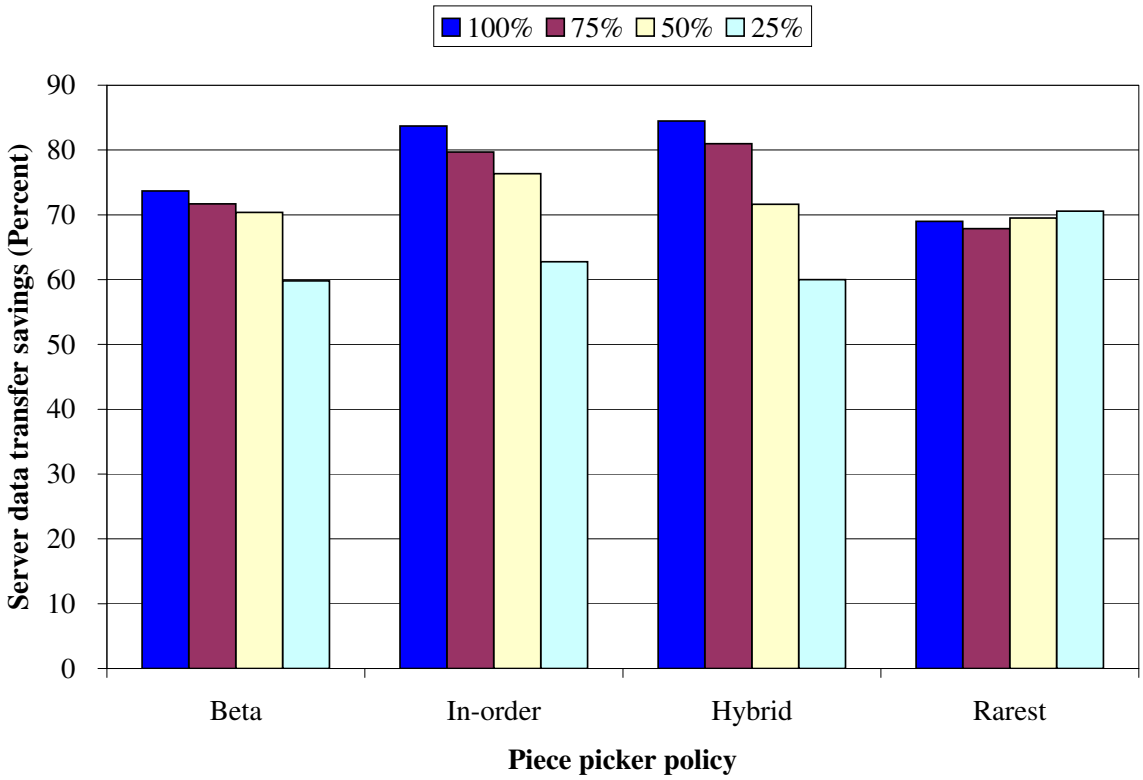

Fig. 5.8. Effect of limited local storage on clients

peers [45]. Figure 5.7 shows the results achieved by both this seed choking strategy and the standard tit-for-tat for various piece-picker policies with the stream sharing behavior. The experimental setup for these tests is the same as for the previous tests. The results indicate that the standard tit-for-tat policy performs better for all piece picker policies studied, with substantial differences in the in-order and hybrid cases. Thus, even in a controlled environment, cooperation between clients plays an important role in the performance of the system.

Impact of limited local storage. Figure 5.8 shows the results when each of the new piece picking policies is subjected to a limited storage space. Such a situation may arise if the storage space at the client machine is shared among multiple swarms or is used for other purposes (such as digital video recording). The buffer space in each test is limited to the percentage of the video file size indicated in the legend. Unsurprisingly, the overall performance is reduced when clients are forced to eject

pieces from their local buffers, since the ejected pieces are then unavailable to send to other peers. This effect is particularly marked when the size is reduced from 50% to 25% because the clients are forced to stop and start the download process more often, an operation which carries significant overhead. The behavior of the hybrid piece picking policy is also notable in that it performs closely to the in-order policy for large buffer limits but performs much more like the beta policy for small storage capacities. When clients are forced to eject pieces they eject the earliest pieces first, thus making it more likely that the pieces in the in-order range are unavailable for delivery to other peers. As a result, those other peers must instead choose pieces further in the future more often. In the rarest-first policy (the default BitTorrent policy), there are only minor differences in performance with respect to the storage space. Interestingly, the performance is slightly better with only 25% of the storage than with more available. Rarest-first is more effective than the other policies with very limited local storage, but cannot take advantage of additional storage to reduce server bandwidth further. For the other policies to be effective, a storage space equivalent to 50% or more of the total stream length should be available.

Impact of latency. Figure 5.9 shows the effect of network latency as described in Section 5.2 on the Toast system, comparing the 1 Mbps and 2 Mbps results using additional network delay (10 ms round-trip time) to the base results with no added delay. These tests use the seed termination policy. The server transfer reduction is almost identical, differing by less than 2.1% in all cases, indicating that the additional delay has almost no effect on performance. Two additional delay scenarios were tested, in which not all clients had the same delay. In the first additional scenario, 4 machines were assigned no delay, 4 were assigned 1 ms, one was assigned 2 ms, and one was assigned 5 ms of send-side delay, resulting in overall round-trip times of up to 7 ms. A second additional scenario used 3 machines with no delay, 2 machines with 4 ms, and the rest with 5 ms, resulting in round-trip times of up to 10 ms. These tests differed by less than 2.8% in all cases from the base results.

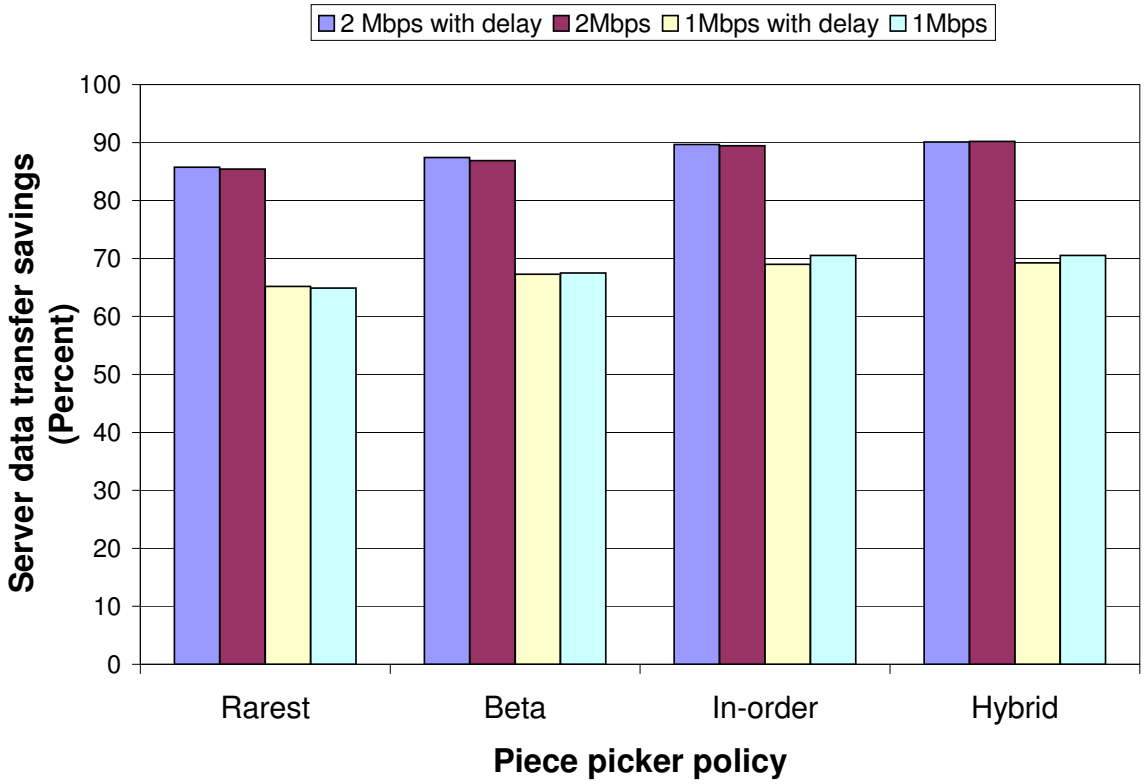

Fig. 5.9. Effect of latency on server bandwidth reduction using seeding behavior

Total system network traffic generated. Besides server data transfer savings, another important measure of the efficiencies of various Toast policies is the overall network traffic generated by the system. Differences in total network traffic may arise because of duplicate network traffic generated by the peers due to inefficiencies in the policy or because of clients fetching data from the server after already requesting the same data from a peer. Figure 5.11 shows the total network traffic in Gigabytes generated by the system for various piece-picker policies, at both 2 Mbps and 1 Mbps client upload rates, and for both stream and seed sharing patterns. This traffic metric includes both incoming and outgoing network traffic generated by the server, the seed client, and all the BitTorrent clients (including intra-machine transfers in the emulation testbed). These tests were done with network delay added as described in Section 5.2. This figure shows only small differences in total network traffic despite using different piece-picker policies, client upload bandwidths, and sharing behaviors.

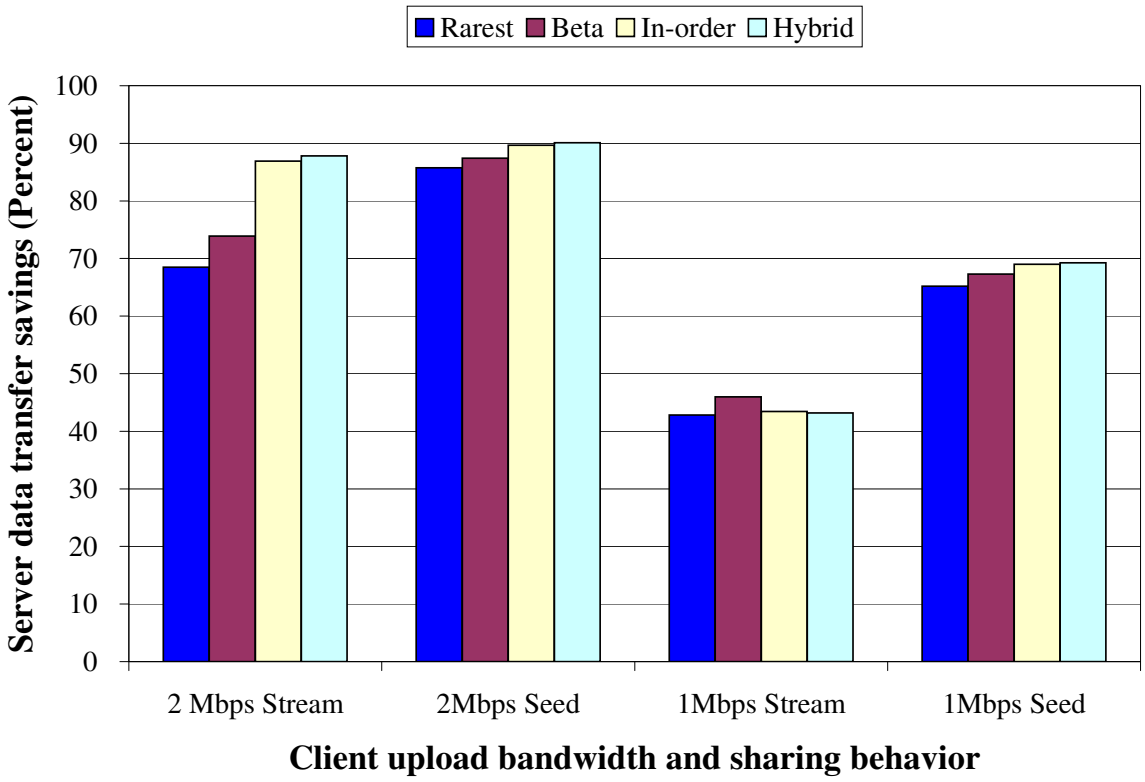

Fig. 5.10. Server data transfer savings

This result indicates that Toast has few inefficiencies in total network traffic; the idea of "giving up" on the data pieces closest to the current stream position makes it unlikely that the client will have to fetch data from the server after first requesting it from a peer. The only exceptions are slight degradations in the case of in-order or hybrid piece-picking policies with 1 Mbps upload bandwidth and stream sharing behavior; this result follows the earlier discussion about the slightly degraded performance of these policies in this client configuration. Figure 5.10 shows the server data transfer savings for comparison.

The small overall differences in total traffic indicate that seeding is advantageous, that rarest-first generates more traffic in the 2 Mbps client upload case despite having poorer performance, and that in-order and hybrid perform slightly worse in the 1 Mbps client upload case.

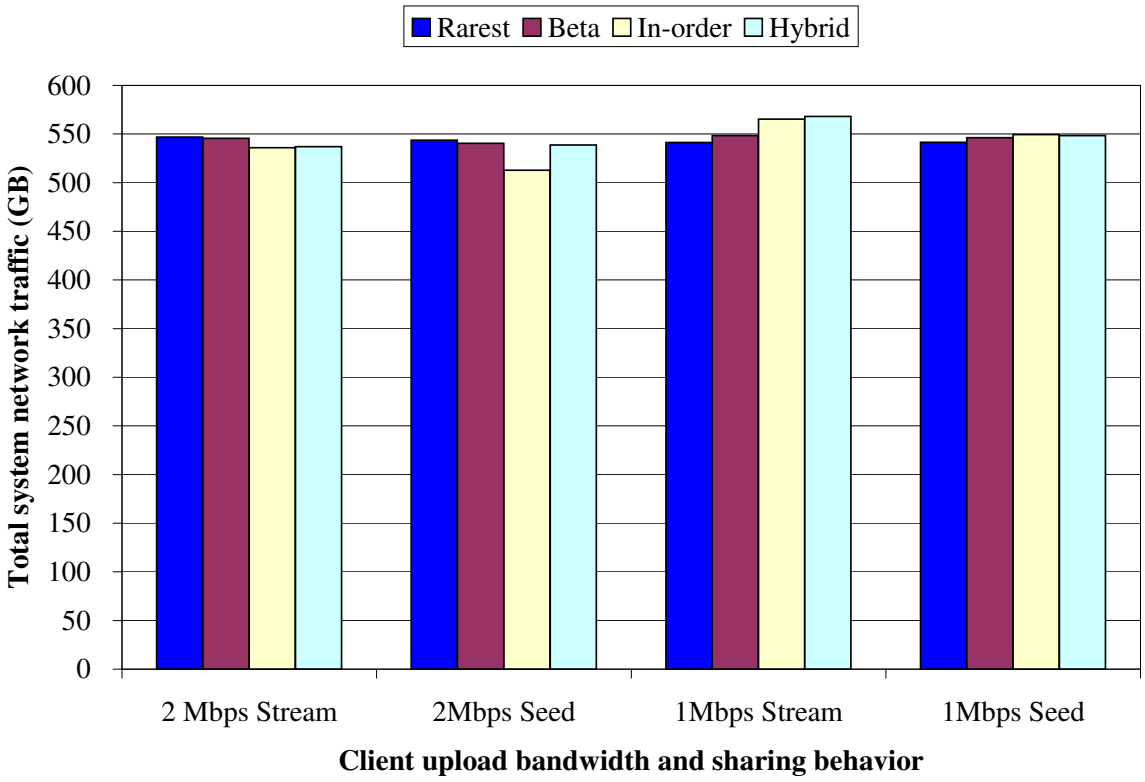

Fig. 5.11. Total network traffic generated

We can see that seeding behavior not only helps the VoD server, it also reduces the overall system network traffic. In the 2 Mbps client upload rate tests, we can see that rarest-first policy generates most network traffic and performs the worst; whereas in-order policy performs near best and generates the least overall network traffic. We can also observe in the 1 Mbps streaming behavior, hybrid policy generates 5% (or 27 GB) more network traffic than rarest-first policy, but the performance is similar in terms of server data transfer savings.

Impact of client arrival behavior. For the primary tests, clients used a uniform arrival rate, arriving every 36 seconds. In a scaled down version of the tests using 90 clients and a smaller video file, two additional arrival processes were considered. In the "all-at-once" arrival behavior, all clients entered the swarm at approximately the same time, simulating the release time of a popular movie or a scheduled showing. In the Poisson arrival behavior, clients arrived according to a Poisson process with an

average interarrival time of 2 minutes. These behaviors were compared to uniform arrivals with 2 minute intervals. The all-at-once behavior performed 21% worse on average than uniform. Because all clients were at approximately the same point in the stream, fewer future pieces were available in the swarm. Poisson arrival behaviors performed very similarly to uniform, within 2% on average.

Discussion. The results presented here show that Toast can be an effective solution for improving the efficiency and aggregate throughput of a Video-on-Demand service. Toast achieves these results using a video-targeted adaptation of BitTorrent backed by a full-fledged VoD server. The results show that the default BitTorrent policy is not well-suited to the VoD environment since it makes no attempt to use available bandwidth to achieve real-time delivery constraints. Instead, policies that favor earlier content pieces make it more likely that clients can avoid using the backing VoD server. Further, allowing clients to remain in the swarm as seeds can help improve VoD server efficiency even for limited client upload bandwidth. Combining BitTorrent, a backing VoD server, a hybrid piece-picking policy, and client seeding allows Toast to offload as much as 90% of the network traffic from the VoD server onto the P2P network.

There are many other useful ways Toast could be evaluated and improved. Many other testing scenarios are possible, depending on potential usage scenarios other than the testing network considered. For example, it may be useful to consider the impact of limited download bandwidth on the clients. This would penalize clients that partially downloaded a piece from a peer and then had to redownload the full piece from the VoD server. Additionally, testing for a situation such as Internet distribution would require studying a realistic distribution of client connection capabilities rather than assuming homogeneity.

Further modifications to the client could include having the PiecePicker dynamically adapt policies based on the performance of the swarm.

Lastly, an analysis that includes more traditional VoD evaluation metrics such as waiting time before startup and service interruptions could be useful, as could

examining the time-varying resource requirements and behavior of the the VoD server. For both of these, evaluation could be done under limitations such as restricted server bandwidth or network congestion.

5.4 Summary

In the chapter we present and evaluate Toast, a system to improve the efficiency of a Video-on-Demand service using the popular and proven BitTorrent peer-to-peer technology. Toast includes a modified version of BitTorrent that supports streaming data delivery rather than just downloads. The BitTorrent client is adapted to communicate with a traditional VoD server when the desired piece of data cannot be delivered by the swarm participants in a timely fashion. Toast thus extends the aggregate throughput capability of a VoD service, offloading work from the server onto the P2P network in a scalable and demand-driven fashion.

This work evaluates Toast under several usage configurations. The results show that the default BitTorrent piece selection policy is not well-suited to the VoD environment. Instead, policies should favor downloading earlier pieces of content from other swarm participants to make it less likely that the end clients will have to get the data directly from the VoD server. Further benefits are possible if clients remain in the system as seeds after they are done viewing their content. Such configurations allow Toast to operate very efficiently, reducing data transfer demands by up to 70–90% compared to simple unicast distribution. Toast is thus an attractive choice to serve as the substrate of a VoD data delivery system, effectively utilizing client resources and network bandwidth while also reducing server infrastructure requirements.

6. RELATED WORK

There have been numerous commercial implementations of VoD servers, as well as academic research on the subject. For example, Kasenna's Media Server uses a high-performance SCSI disk array to support 400 streams of rate 3 Mbps, while Bitband's Vision 680 supports 257 streams at 3.5 Mbps [3, 5]. Yang et al. have built a prototype server using custom hardware, achieving 550 simultaneous streams of rate 1.3 Mbps [6]. Kukol and Gray have studied transferring large data using ten Gigabit links but their focus was on disk configuration not on networks [63]. And it was not rate paced. Shenoy and Vin studied storage techniques such as static and dynamic load balancing, retrieval techniques such as caching and batching, and disk striping policies based on detailed models of large disk arrays (16+ disks) with variable numbers of clients [23,24]. Chang and Garcia-Molina focused on the best memory use techniques under different disk management policies [64]. There have also been numerous studies in disk-scheduling algorithms such as EDF, CScan, and Real-Time [65–67]. Other works have considered VoD storage performance with disk striping and interleaving [67–69]. Wright describes and analytically evaluates a scheme called one-way elevator with interleaving and delayed start (OEID); this scheme interleaves data over all disks and delays the start of each new stream to reduce random fluctuations in disk load caused by new arrivals [67]. In general, prior studies have generally focused on bitrates below 5 Mbps or systems that support only a small number of distinct streams. More recent developments in link-level bandwidths, CPU performance, and disk technology trends require a careful investigation of faster bitrates, more connections, and more distinct streams.

In networks, Rejaie et al. have provided a transport protocol for real-time multimedia streams [70], while Almeida et al. used multicast to service media workloads in education [71]. Issues related to network data delivery are important in the VoD

context. Such protocol issues are orthogonal to the design studied in this dissertation and may be combined productively with this work.

Other works have considered VoD storage performance with disk striping and interleaving [67,69]. There also have been several studies analyzing the characteristics of streaming media workloads [28, 72].

Other works targeting VoD workloads have studied striping, replication, and randomization. Chua et al. propose a phase-based striping method and develop a data replication scheme to further reduce the average waiting time [73]. Using simulations, they showed that their striping method reduces waiting time and that replication further improves the performance by 25%. Chou et al. focus on the scalability of continuous media servers as a function of tradeoffs between striping and replication [74]. Fu et al. compared the traditional block level randomization (BLR) techniques to what they call packet-level randomization (PLR) techniques using analytical models of server behavior [22]. They showed that PLR, which randomly allocates data on disks at the granularity of network packets (500–1400 bytes), can achieve superior short-term load balancing. However, PLR requires additional memory buffering since data must be simultaneously retrieved from all disks at the granularity of an allocation block (as much as 1 MB per disk). Santos et al. analytically compared random data allocation with traditional data striping techniques and found that random data allocation is at least as good as striping in streaming workloads [18]. Although all of the above works contribute to various aspects of Video-on-Demand storage design, this work extends upon the prior state-of-the-art by experimentally prototyping various file allocation strategies (including random duplicate allocation) and observing how those strategies respond to both workload hotspots and single-disk fail-stutter faults.

Chapter 2 covered the projects and systems that most closely relate to this dissertation. The following describes alternative means of solving problems similar to those studied in this dissertation.

A related problem to on-demand video streaming is that of live streaming. Live streaming accomplishes a similar goal (distributing video or other content to large number of people) but does not have the requirement that the user be able to skip to arbitrary points in the stream (for example, in the future). However this is the primary advantage that BitTorrent exploits: the ability of clients to exchange pieces from anywhere in the file. Thus, any BitTorrent-based system is unsuitable for live streaming. Overlay or peer-to-peer networks designed for live streaming have been presented in [75–77].

Numerous web caching systems and content distribution networks (CDNs) have sought to offload the delivery of streaming video content onto geographically-distributed servers. This dissertation presents an alternative approach through the use of peer-to-peer systems. P2P has advantages in dynamically provisioning resources as demand rises since each requester node must also be a provider. If the peer nodes are actually controlled by a system administrator (e.g., cable company or PCs on a LAN), they can be set to remain as background seeds long after their viewing is complete, thus offloading work from remote network servers without requiring additional infrastructure support through a CDN.

There have been several characterization studies focusing on streaming media workloads [28, 72, 78]. These studies all focus on public Internet servers; there have been no published studies on workloads for movie or television distribution on demand. These studies analyze the relative numbers of accesses to different files in a server. This dissertation uses BitTorrent to optimize the distribution of a single file. However, the scheme described here could be deployed using separate swarms for distributing different popular files.

7. CONCLUSION

First, Chapter 3 presents a mathematical model and a prototype of a resource-efficient storage server for high-bitrate Video-on-Demand (VoD) applications. Rapid exponential growth of disk capacity enables the storage of high-bitrate VoD streams; however, a server system must be carefully designed to allow those streams to be retrieved from disk and delivered to the network efficiently. Additionally, a cost-effective server should be implemented using only commodity components, such as standard PCs, SATA disks and controllers, and Gigabit Ethernet links.

Previous parallel I/O performance models have been either oversimplified theoretical models that ignore hardware and application characteristics or complex hardware models that consider detailed disk behaviors such as inter-track variations. This dissertation presents a model between these extremes: detailed enough to account for the rate-based nature of streaming video, the buffering time allowed by the application, and average-case disk hardware characteristics while remaining simple enough to use for algorithm and system design. This dissertation then describes a prototype storage server designed to serve large video files at the specified bitrates and finds its performance to agree closely with the model (with an average discrepancy of 11% for high-bitrate streams). The system uses up to 8 SATA-300 disks and can simultaneously serve 290 distinct DVD-quality (6 Mbps) streams or 74 distinct HDTV-quality (25 Mbps) streams from disk, achieving an aggregate network throughput of 1.85 Gbps.

Second, Chapter 4 investigates randomization and replication as strategies to achieve reliable performance in disk arrays targeted for Video-on-Demand (VoD) workloads. A disk array can provide high aggregate throughput, but only if the server can effectively balance the load on the disks. Such load balance is complicated by two key factors: workload hotspots caused by differences in popularity among

media streams, and "fail-stutter" faults that arise when the performance of one or more devices drops below expectations due to manufacturing variations, hardware problems, or geometry-related variations.

Chapter 4 focuses on the random duplicate assignment (RDA) data allocation policy which places each data block on two disks chosen at random, independent of other blocks in the same media stream or other streams. This strategy is compared to traditional single-disk file allocation, disk striping (RAID-0), disk mirroring (RAID-1), and randomization without duplication. The various allocation schemes are implemented and tested using a prototype VoD server with 2 dual-core Opteron processors, 8 SATA disks, and 4 Gigabit Ethernet interfaces running the Linux 2.6 kernel. The results indicate that combining randomization and replication allows RDA to effectively tolerate both workload hotspots and fail-stutter faults better than previous schemes.

Third, Chapter 5 presents and evaluates Toast, a scalable Video-on-Demand (VoD) streaming system which combines the popular BitTorrent peer-to-peer file-transfer technology with a simple dedicated streaming server to decrease server load and increase client transfer speed. Toast includes a modified version of BitTorrent that supports streaming data delivery and that communicates with a VoD server when the desired data cannot be delivered in real-time by other peers.

The results show that the default BitTorrent download strategy is not well-suited to the VoD environment because it may fetch pieces of the desired video from other peers without regard to when those pieces will actually be needed by the media viewer. Instead, strategies should favor downloading pieces of content that will be needed earlier, decreasing the chances that the clients will be forced to get the data directly from the VoD server. Such strategies allow Toast to operate much more efficiently than simple unicast distribution, reducing data transfer demands by up to 70–90% if clients remain in the system as seeds after viewing their content. Toast thus extends the aggregate throughput capability of a VoD service, offloading work from the server onto the P2P network in a scalable and demand-driven fashion.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Comcast Corporation, "Annual Report to Shareholders," 2005.

[2] E. Grochowski and R. D. Halem, "Technological impact of magnetic hard disk drives on storage systems," *IBM Systems Journal*, vol. 42, pp. 338–346, April 2003.

[3] Bitband Technologies Ltd., "Vision 680." Data Sheet.

[4] Entone Technologies, Inc., "Entone Video Server Architecture." White paper, 2005.

[5] Kasenna, Inc., "Kasenna Media Servers." Data Sheet, August 2003.

[6] S. Yang, H. Yang, and Y. Yang, "Architecture of high capacity vod server and the implementation of its prototype," *IEEE Transactions on Consumer Electronics*, pp. 1169–1177, November 2003.

[7] J. R. Larus and M. Parkes, "Using Cohort Scheduling to Enhance Server Performance," in *Proceedings of the 2002 USENIX Annual Technical Conference*, pp. 103–114, June 2002.

[8] E. M. Nahum, T. Barzilai, and D. Kandlur, "Performance Issues in WWW Servers," *IEEE/ACM Transactions on Networking*, vol. 10, pp. 2–11, February 2002.

[9] V. S. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An Efficient and Portable Web Server," in *Proceedings of the USENIX 1999 Annual Technical Conference*, pp. 199–212, June 1999.

[10] V. S. Pai, P. Druschel, and W. Zwaenepoel, "I/O-Lite: A Unified I/O Buffering and Caching System," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation*, pp. 15–28, February 1999.

[11] M. Welsh, D. Culler, and E. Brewer, "SEDA: An architecture for well-conditioned, scalable internet services," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pp. 230–243, October 2001.

[12] E. P. Markatos, M. Katevenis, D. N. Pnevmatikatos, and M. Flouris, "Secondary Storage Management for Web Proxies," in *USENIX Symposium on Internet Technologies and Systems*, pp. 93–104, October 1999.

[13] A. Rousskov and D. Wessels, "The Third Cache-off." Raw data and independent analysis at http://www.measurement-factory.com/results/, October 2000.

[14] J. S. Vitter and E. A. M. Shriver, "Optimal algorithms for parallel memory I: Two-level memories," *Algorithmica*, vol. 12, no. 2-3, pp. 110–147, 1994.

[15] A. N. Reddy and P. Banerjee, "An Evaluation of Multiple-Disk I/O Systems," *IEEE Transactions on Computers*, vol. 38, pp. 1680–1690, December 1989.

[16] R. Arpaci-Dusseau, "Performance availability for networks of workstations," *PhD thesis, University of California, Berkeley*, 1999.

[17] D. A. Patterson, G. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," in *Proceedings ACM SIGMOD Conference*, (Chicago, Illinois), June 1988.

[18] J. Santos, R. Muntz, and B. Ribeiro-Neto, "Comparing random data allocation and data striping in multimedia servers," *Performance Evaluation Review*, vol. 28, no. 1, pp. 44 – 55, 2000.

[19] G. H. Gonnet, "Expected length of the longest probe sequence in hash code searching," *Journal of the Association for Computing Machinery*, vol. 28, pp. 289–304, April 1981.

[20] R. D. Barve, E. F. Grove, and J. S. Vitter, "Simple randomized mergesort on parallel disks," *Parallel Computing*, vol. 23, no. 4, pp. 601–631, 1997.

[21] P. Sanders, S. Egner, and J. Korst, "Fast concurrent access to parallel disks," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, vol. 11, (San Francisco), pp. 849–858, January 2000.

[22] K. Fu and R. Zimmermarm, "Randomized data allocation in scalable streaming architectures," *Database Systems for Advanced Applications. 10th International Conference, DASFAA 2005. Proceedings*, pp. 474 – 86, 2005.

[23] P. Shenoy and H. Vin, "Multimedia storage servers," in *In Readings in Multimedia Computing and Networking* (K. Jeffay and H. Zhang, eds.), Morgan Kaufmann Publishers, 2002.

[24] P. Shenoy and H. M. Vin, "Efficient Striping Techniques for Variable Bit Rate Continuous Media File Servers," *Performance Evaluation Journal*, vol. 38, pp. 175–199, December 1999.

[25] C. Thompson, "The BitTorrent Effect," *Wired Magazine*, January 2005.

[26] Y. R. Choe, C. Douglas, and V. S. Pai, "A Model and Prototype of a Resource-Efficient Storage Server for High-Bitrate Video-on-Demand," in *IPDPS Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems*, March 2007.

[27] Y. R. Choe and V. S. Pai, "Achieving Reliable Parallel Performance in a VoD Storage Server Using Randomization and Replication," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium*, March 2007.

[28] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming-media workload," *Proceedings of 3rd USENIX Symposium on Internet Technologies and Systems. (USITS'01)*, pp. 1 – 12, 2001.

[29] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, "Fail-stutter fault tolerance," *Proceedings of the Workshop on Hot Topics in Operating Systems - HOTOS*, pp. 33 – 38, 2001.

[30] R. V. Meter, "Observing the effects of multi-zone disks," in *Proceedings of the USENIX 1997 Annual Technical Conference*, pp. 19–30, January 1997.

[31] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications." IETF RFC 1889, January 1996.

[32] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)." IETF RFC 2326, April 1998.

[33] S. Carter and D. Long, "Improving video-on-demand server efficiency through stream tapping," in *Computer Communications and Networks, 1997. Proceedings., Sixth International Conference on*, pp. 200–207, 22-25 Sept. 1997.

[34] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*, (New York, NY, USA), pp. 15–23, ACM Press, 1994.

[35] D. Eager, M. Vernon, and J. Zahorjan, "Minimizing bandwidth requirements for on-demand data delivery," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 13, pp. 742–757, Sept.-Oct. 2001.

[36] L. Golubchik, J. C. S. Lui, and R. Muntz, "Reducing i/o demand in video-on-demand storage servers," in *SIGMETRICS '95/PERFORMANCE '95: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 25–36, ACM Press, 1995.

[37] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 4, pp. 197–208, August 1996.

[38] Y. Cui, B. Li, and K. Nahrstedt, "oStream: asynchronous streaming multicast in application-layer overlay networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 91 – 106, 2004.

[39] S. Sheu, K. Hua, and W. Tavanapong, "Chaining: a generalized batching technique for video-on-demand systems," *Proceedings IEEE International Conference on Multimedia Computing and Systems*, pp. 110 – 17, 1997.

[40] L. de Pinho, E. Ishikawa, and C. de Amorim, "GloVE: A distributed environment for scalable video-on-demand systems," *Int. J. High Perform. Comput. Appl. (USA)*, vol. 17, no. 2, pp. 147 – 61, Summer 2003.

[41] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pp. 177–186, 2002.

[42] L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang, "PROP: a Scalable and Reliable P2P Assisted Proxy Streaming System," *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pp. 778–786, 2004.

[43] C. Huang, J. Li, and K. Ross, "Peer-Assisted VoD: Making Internet Video Distribution Cheap," *Proceedings of Sixth Internaltional Workshop on Peer-to-Peer Systems*, 2007.

[44] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, "BASS: BitTorrent assisted streaming system for video-on-demand," *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, October 2005.

[45] B. Cohen, "Incentives build robustness in bittorrent," tech. rep., May 2003.

[46] R. Axelrod, *The Evolution of Cooperation*. Basic Books, 1984.

[47] A. Aggarwal and J. S. Vitter, "The Input/Output complexity of sorting and related problems," *Communications of the ACM*, vol. 31, no. 9, pp. 1116–1127, 1988.

[48] M. Kallahalla and P. J. Varman, "Optimal read-once parallel disk scheduling," in *In Proc. of Sixth ACM Workshop on I/O in Parallel and Distributed Systems*, pp. 68–77, 1999.

[49] R. Shah, P. J. Varman, and J. S. Vitter, "Online algorithms for prefetching and caching on parallel disks," in *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures*, 2004.

[50] R. Barve, M. Kallahalla, P. Varman, and J. S. Vitter, "Competitive Parallel Disk Prefetching," *Journal of Algorithms*, pp. 152–181, 2000.

[51] A.Gulati and P. Varman, "Scheduling Multiple Flows on Parallel Disk Systems," in *Proceedings of the International Conference on High-Performance Computing (HiPC '05)*, December 2005.

[52] S. J. Mullender and A. S. Tanenbaum, "Immediate files," *Software: Practice and Experience*, vol. 14, pp. 365–368, April 1984.

[53] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *IEEE Computer*, vol. 27, pp. 17 – 28, March 1994.

[54] L. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis," in *Proceedings of the 1996 USENIX Technical Conference*, pp. 279–295, January 1996.

[55] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating User-Perceived Quality into Web Server Design," in *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[56] G. R. Ganger, B. L. Worthington, R. Y. Hou, and Y. N. Patt, "Disk subsystem load balancing: Disk striping vs. conventional data placement," in *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, vol. I, pp. 40–49, January 1993.

[57] J. S. Vitter, "External memory algorithms and data structures: Dealing with massive data," *ACM Computing surveys*, vol. 33, pp. 209–271, June 2001.

[58] L. W. McVoy and S. R. Kleiman, "Extent-like Performance from a UNIX File System," in *Proceedings of the USENIX Winter 1991 Technical Conference*, (Dallas, TX, USA), pp. 33–43, 1991.

[59] S.-H. Lim, Y.-W. Jeong, and K.-H. Park, "Interactive media server with media synchronized raid storage system," in *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, (New York, NY, USA), pp. 177–182, ACM Press, 2005.

[60] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP 1.1." IETF RFC 2616, June 1999.

[61] S. Hemminger, "Network Emulation with NetEm," in *Proceedings of the 2005 Linux Conference Australia (LCA-2005)*, April 2005.

[62] Aspera Inc., "Overcoming the challenges of network data delivery." White paper, 2006.

[63] P. Kukol and J. Gray, "Performance considerations gigabyte per second transcontinental disk-to-disk file transfers," May 2004.

[64] E. Chang and H. Garcia-Molina, "Effective memory use in a media server," *Proceedings of the Twenty-Third International Conference on Very Large Databases*, pp. 496 – 505, 1997.

[65] P. Bosch and S. Mullender, "Real-time disk scheduling in a mixed-media file system," *Proceedings Sixth IEEE Real-Time Technology and Applications Symposium. RTAS 2000*, pp. 23 – 32, 2000.

[66] A. Narasimha Reddy and J. Wyllie, "I/o issues in a multimedia system," *Computer*, vol. 27, pp. 69 – 74, March 1994.

[67] W. E. Wright, "An efficient video-on-demand model," *IEEE Computer*, pp. 64–70, May 2001.

[68] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, "Staggered striping in multimedia information systems," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 23, no. 2, pp. 79 – 90, 1994.

[69] B. Özden, R. Rastogi, and A. Silberschatz, "Disk striping in video server environments," in *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, pp. 580–589, June 1996.

[70] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *Proceedings of IEEE INFOCOM '99*, pp. 1337–1345, March 1999.

[71] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon, "Analysis of educational media server workloads," in *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, pp. 21–30, June 2001.

[72] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," *Proceedings of the 2004 ACM SIGCOMM Internet Measurement Conference, IMC 2004*, pp. 41 – 54, 2004.

[73] T. S. Chua, J. Li, B. C. Ooi, and K.-L. Tan, "Disk striping strategies for large video-on-demand servers," *Proceedings ACM Multimedia 96*, pp. 297 – 306, 1996.

[74] C.-F. Chou, L. Golubchik, and J. Lui, "Striping doesn't scale: how to achieve scalability for continuous media servers with replication," *Proceedings 20th IEEE International Conference on Distributed Computing Systems*, pp. 64 – 71, 2000.

[75] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," in *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 1–12, ACM Press, 2000.

[76] D. Tran, K. Hua, and T. Do, "ZIGZAG: an efficient peer-to-peer scheme for media streaming," *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 1283 – 92, 2003.

[77] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," *Proceedings IEEE Infocom 2005 (IEEE Cat. No. 05CH37645)*, vol. vol. 3, pp. 2102 – 11, 2005.

[78] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon, "Analysis of educational media server workloads," June 2001.

VITA

VITA

Yung Ryn (Elisha) Choe received a Bachelor of Science degree in Computer Engineering with minors in Computer Science and Mathematics from Virginia Polytechnic Institute and State University (Virginia Tech) in May 1994. He received a Master of Science degree in Electrical Engineering from Purdue University in May 1996. He joined Digital Signal Processor Development Tools group in Analog Devices, Inc in June 1996 and was a Senior Software Engineer until August 2003. He was a Graduate Teaching Assistant in Mathematics Department at Purdue from August 1995 to May 1996 and from August 2003 to May 2005. He was an instructor for courses such as MA 151, 153, 223, and 224, responsible for teaching the entire class, writing midterms, and assigning final letter grades. He was a Graduate Research Assistant from June 2005 to May 2007.