Simulation of a Video-on-Demand System

Song Bac Toh

Senior Thesis in Computer Science
Dartmouth College
Hanover, New Hampshire 03755
June 6, 1995

Abstract

This paper presents a simulation study of a video-on-demand system. The focus of the study is the effectiveness of different caching strategies on a video-on-demand system with two levels of cache, RAM and disks, in front of a tape library. Using an event-driven simulator, I show that the service capacity of the system benefited only marginally from caching if no other information was available. I find that on-demand caching was only useful if movies to be shown clustered around a few popular titles (in other words, there was temporal locality).

1 Introduction

A video-on-demand system is a multimedia application with immense storage requirements. Even with the advances in storage architecture, the stringent real-time demand of a video-on-demand (VoD) system can still stretch the technology to its limit. In such a system, multiple streams of audio and video data have to be delivered uninterrupted to the clients. When the system fails to meet the requirement of on-time error-free delivery, "glitches" result on the clients' end. These glitches are unpleasant, noticeable breaks in the continuity of the audio or video stream.

The video-on-demand system also has to offer enough advantages over the conventional video rental stores to be economically viable. One advantage of a video-on-demand system is that it offers customers the ability to order a movie at any time, without leaving their homes. To exploit this advantage, the system should be able to respond to a request in a reasonable time, and the quality of the movie delivered

should be at least as good as that offered by VHS tapes if not better. Video-on-demand systems are also useful in environments like hotels, schools and libraries. Schools may also be interested in implementing a video-on-demand system to serve classroom materials and lectures on demand. Libraries, which must increasingly archive and provide access to non-print media, can use such a system to present multimedia items on demand.

This paper describes a simulation of a video-on-demand system, to study the effect of different caching strategies on a multi-level cache system using RAM, a disk array, and a tape library, to find a configuration that yields the lowest storage requirement for the largest number of concurrent movies.

The next section covers the background information in video-on-demand applications and describes the architectural requirements. Section 3 reviews related work on video-on-demand systems and continuous media. Section 4 describes the design of the simulator and some software-engineering issues that had a large impact on the correctness and speed of the simulator. Section 5 of the paper presents some results from experiments. Finally, in section 6, I lay out some possible topics for future work in this particular area of study.

2 Background

Due to the enormous bandwidth and storage requirements of digital multimedia data streams, most applications in this field work only with compressed data streams. An uncompressed video stream with NTSC quality[1] consumes as much as 27 Mbytes per second of network bandwidth and a 90-minute uncompressed movie can take up to 146 Gbytes of storage. HDTV[2] requires up to 81 Mbytes per second of bandwidth for video only. The storage requirement is simply too big for any conventional storage device to handle efficiently.

---

1 NTSC is the standard for television displays in the United States and Japan. The video specification is 640x480 pixels/frame and 24 bits/pixel at 30 frames per second.

2 HDTV stands for High-Definition TV. Its specification is 1280x720 pixels/frame, 24 bits/pixel at 30 frames per second.

There are a few standards for compressing video data, including the Motion Picture Group's MPEG, AVI, Apple Computer's QuickTime, and others. In this paper, I work with MPEG-1 compressed movies. MPEG compression is lossy; in this application, however, the high compression ratio offered by MPEG is more important than the slight quality degradation.

Even though MPEG-1 allows up to 2 megabits per second (Mbps[1]), I use 1.5 Mbps (MPEG-1's Constraint Parameters Bitstream, CPB rate, which is also the data rate of CD-quality audio [9]) as the maximum stream rate. The decompressed video quality will match VHS video-tape playback. As MPEG is a scalable compression algorithm, one should bear in mind that the option to provide better quality video is there, but the demand on the hardware will scale up as well.

Based on a paper by O. Rose [7], the parameter I chose for the MPEG compression is 12 frames per group of pictures (GOP). Each stream plays 30 frames per second. I also set the size of a movie segment to be the same as the size of a GOP. Each segment, and thus cache slot, was fixed at 76.8 kilobytes, which is $\frac{12}{30}$ of the 1.5 Mbps bitstream rate. All GOPs are assumed to be at most 76.8 kbytes to facilitate network and device read and write operations. Any GOP of the size greater than 76.8 kbytes will have to drop some bits to fit into a video segment. However, this should not be a major problem since 1.5 Mbps is a rate supported by MPEG-1.

Since network design is not part of this study, the network is assumed to not be a bottleneck. The network within the system should not saturate since the simulation runs with less than 30 concurrent 1.5 Mbps streams, while there are already system buses (e.g., PCI) that support up to 132 Mbytes per second transfer rate. Even though each movie stream takes more bandwidth internally than it does externally (1.5 Mbps), the simulated runs should still be well within the limit.

3 Related Work On Video-On-Demand Systems

[1] Throughout this paper, bps stands for bits per second while bytes per second is written as Bps.

There is much related work about video-on-demand systems. Most studies that used real hardware, however, were limited to systems with only RAM and disks, possibly because high-speed tape libraries were not readily available.

Oyang et al. [8] did a high-level study on the storage system design for a multimedia system with on-demand playback. The study concentrated on calculating the upper bounds on seek time for various disk-seek algorithms, and using the knowledge to enhance the performance of a multimedia server. The study, however, did not concern itself with the interaction between the disk and other levels of cache.

Furht et al. [4] studied the display of HDTV and MPEG-2 (2-20 Mbps) videos from distributed servers. Due to the bandwidth requirement of MPEG-2 movies, the authors recommended using highly parallel servers with high I/O throughput. Storage architecture is not a chief concern in that paper, but it provides insightful information on the set-top-box (STB) used in video-on-demand applications.

Gemmell [5] studied the disk-head scheduling policy, the use of contiguous placement, the use of a smaller file-index table with greater allocation sizes, and data striping across disks, all in the context of continuous-media servers. I have borrowed some of Gemmell's concepts, for example, the need to prefetch enough video segments before starting the stream, and the advantage of keeping the beginning of a movie on the disks, instead of waiting for the whole movie to come in from the tape library. Gemmell also considered the use of tertiary storage devices, e.g., tape libraries. The author acknowledged that it is not practical to play the movie streams directly from a tertiary storage device due to its slow random-access time, which concurs with my finding.

Rowe et al. [2] presented a larger scale video-on-demand system, although it is not conceptually different from the one in this paper. Rowe had the resources to actually implement the system. Rowe's studied a distributed hierarchical storage system. The paper describes how to improve the performance of a video server by intelligent caching, i.e., by using the knowledge of which movies are popular and which may become popular in the near future.

4 The Simulator

The simulator in this paper is based on Simpack [11], a free simulator package distributed by Paul Fishwick from University of Florida. Simpack contains several

simulation models, from which I chose the queuing event model. There are several "facilities" in the video-on-demand server simulator, namely the customer, the switchboard, the movie manager, a jukebox, a disk array, and the RAM server [Figure 1]. Each facility serves as a potential consumer as well as producer of events. The jukebox, disk array, and the RAM server should each be controlled by a fast and intelligent I/O controller, so that they can handle incoming events efficiently. However, it is also plausible to put each of these servers on a separate CPU and then link the CPUs together with a high-speed ATM or FDDI network.

Each facility in the system should service incoming events immediately to beat the deadline. Therefore, the facilities wake up to check their incoming event queues both after handling an event and when awakened by another facility. This kind of aggressive handling, however, will cause problems with the single event queue in Simpack, i.e., the facility may dequeue an event before it is ready for consumption. After a facility receives an event, it needs a certain amount of time to handle it, before it sends an event to another facility, based on the result from the incoming event. To simulate this lag time, I had to implement a separate internal event queue for each facility so that each can keep outgoing events in its internal queue until they are ready to be sent out (or seen by their destinations).

Customers interact with the switchboard through their set-top-boxes, which also serve as MPEG-decoders [Figure 1 & 2]. The switchboard and the other managers run as processes on the CPU of the server. The CPU, the disk array, and the jukebox are then tied together with a high-bandwidth system bus. The RAM is connected to the CPU through a memory bus.

The video-on-demand system is designed so that it needs no warm-up time, i.e., as soon as the system is turned on, it should be able to handle movie requests (given a small amount of wait time between the time a request comes in and the play time of the requested movie). As the cache fills up, the cache replacement strategies will take effect at each level of cache.

As mentioned in Section 2, the network bandwidth is not considered as a bottleneck in this simulation. All transmission through the network, both the internal LAN and the external cables are assumed to be error-free and encounter no saturation. This is, of course, an area for future study.

Following is a description of the logical model of each of the facilities in the simulation [Figure 1].

4.1 Customer

The customer is represented by a request generator that reads requests from a precomputed request file and generate requests to the video server. The request file is generated by a separate program. In this paper the distribution of movies is pseudo-randomly chosen from a normal distribution [Figure 3]. I compare the behavior of the system to one facing a uniform request distribution [Figure 4].

We assume that each customer takes 30 seconds to choose a movie, and each incoming request has a tolerance of 60 seconds for the movie to start. Therefore, any request will not be officially given to the movie manager until 30 seconds after the request time. When the first 30 seconds is over, the switchboard will send the request to the movie manager, so it can start the movie.

Since everybody takes the same amount of time, this delay is only deliberately inserted to achieve more realistic effects; it should not affect the working of the system. From the video-on-demand system's stand point, it is as if the service for every request has to commence 30 seconds after the customer has finished issuing the request, though a grace period will be granted if the beginning of the movie is not on disk already, as mentioned in Section 4.2 below.

Two request distributions are used. In the uniform distribution, each movie has the same probability of being requested next. In the normal distribution, we begin with normal distribution of integers with a mean at 188 and a range of [1..375] (there are 375 movies with IDs from 1 through 375). The standard deviation is 6, so most of the requests will be for movies in the range of [182..174]. Then I map each of the [1..375] number to a unique number also in the range of [1..375] to make sure not all the requests are located on one tape, to avoid spatial locality (each tape contains 75 movies; therefore movie 1 through movie 75 is on the first tape, and so on). The normal distribution ensures temporal locality, of course.

4.2 Switchboard

The switchboard interacts with the customers directly. It is the facility that dispatches movie requests to the movie manager. Like a telephone operator, the only thing the switchboard does is to process a movie request. It denies requests for unavailable movies, and inserts a 2 minute delay for movies whose beginning[1] is not currently on disk. Thus, a request for a movie not on disk may take up to 2.5 minutes (30 seconds from the original 60 second tolerance time and 2 minutes for the not-on-disk grace period) before showing. The switchboard also refuses to take requests when the system is already running at maximum number of streams (a configuration parameter).

4.3 Movie Manager

The movie manager manages every movie stream that is running through the system and keeps track of the locked movie segments, which are the segments that the cache should not throw out. Before a movie segment is played, the movie manager issues a prefetch request to the RAM to make sure the next segment is there. If the segment is already in the RAM, the RAM will do nothing further until it is requested to send out the segment. If it is not, the RAM will reply to the movie manager, which will then send a prefetch request to the disk. If the disk does not have the segment, a request will be sent back to the movie manager by the disk. Then the movie manager will issue a read request to the jukebox for the segments. As soon as a prefetch request is issued to the RAM, the movie manager locks the segment to prevent it from being replaced in cache-replacement. The segment is unlocked once it is sent to the client. However, there can be more than one lock on a segment, the other locks being from other movie streams playing the same movie.

Once a movie has been started, the movie manager also manages the flow of movie segments through the system. At a fixed rate (at which movie segments are displayed, one GOP per 0.4 second), the movie manager issues a prefetch request to the RAM to make sure the next segment is there. All the operations in the movie manager are assumed to take negligible time.

4.4 RAM

---

1 "Beginning" is defined as the first 10 GOPs in this simulation.

The RAM server manages a pool of buffers in RAM that is attached to the system bus. The RAM serves as the uppermost level cache/buffer for the video-on-demand system since it has the fastest access time. However, it is impractical to keep everything in RAM. At the time of writing, RAM prices hover around 40 dollars a megabyte. Since 1/2-hour long 1.5 Mbps movie stream takes about 337.5 megabytes, it is evident that it is impractical to keep 10 or 20 regular length (1.5 to 2 hour long) movies in RAM, let alone all the movies carried by the video-on-demand system.

There is also a small amount of RAM set aside to serve as I/O buffers. When movie segments come out from the tape jukebox, they will be written into the I/O buffer through the bus. Then the disk and the RAM will proceed to scoop the data from the I/O buffer.

Modern RAM access speed is typically 20 megabytes per second (MBps). Given a GOP size of 76.8 kbytes, each access to a GOP should be about 3.7 milliseconds. The access time for RAM is assumed to be 4 milliseconds per GOP since the unit time in the simulation is 1 millisecond.

If the RAM becomes full and the RAM server is unable to find a segment slot for an incoming segment, it will throw out the incoming segment and schedule a request for it 50 milliseconds later with the hope that it will then be able to keep the segment. Often, when this occurs, the RAM will not be able to get the segment in on-time, which results in glitches. The system is considered overloaded when such glitches become frequent.

The cache replacement policies used in the simulator include least-recently-used (LRU), first-in-first-out (FIFO), least-frequently-used (LFU), and random (RND) (see [10] for detailed descriptions of such strategies). As long as the cache pool is not filled up, the device will simply use the next available slot. However, when the cache is full, depending on which caching policy is in effect, a segment will be chosen and the data in it thrown out to accommodate incoming data. If no segment can be thrown out, the device will have to throw out the incoming movie segment. In all cases, locked segments are prevented from being replaced.

The RAM is implemented to search for a suitable replacement segment for LRU, FIFO and LFU policies in the cache linearly, which is not very efficient. However, the cache-

replacement calculation time is not simulated; therefore the inefficient implementation only contributes to slower simulation run time, but does not affect the result.

Random replacement, however, tries for 15 random locations in the cache to find a segment that can be replaced. If it fails after 15 trials, the device will have to throw out the incoming segment.

4.5 Disk

The disk actually consists of a disk array made up of six HP97560 disks. Each disk is autonomous, though managed by a single disk server that has a segment allocation table of the disks. Therefore, the server knows exactly what is in each disk and which physical segment a given logical segment is in. The allocation unit on disk is one movie segment, or a GOP. The access size, a GOP, is 76.8 kilobytes. I obtained disk read and write access times by issuing a series of 10,000 uniformly distributed GOP requests to a HP 97560 disk simulator [6] and then generated a cumulative distribution function (CDF) based on the access time [Figures 5 & 6].

Whenever there is a request for a movie segment from the disk, the disk will first check in the segment-allocation table to find the segment's physical location. The disk server will then obtain an access time by mapping a random number [0..1] into the read or write CDF mentioned above. With the access time, the disk server will record when the disk will be free next, write out new information into the physical segment, and update the information concerning the segment, e.g., the last accessed time and access frequency.

Whenever a new segment arrives, a segment slot is chosen on the next disk to be idle, and the segment written there. Therefore, no single disk will be tied up for an extended period of time while the other disks are sitting idle.

The cache-replacement policy of the disk is set to be the same as that of the RAM. The working of the policy is also similar, except for the fact that the policy will only choose from the segments on one disk, instead of the whole cache pool like in the RAM.

4.6 Jukebox

The jukebox is a tape library with 4 tape drives and a robot arm. The tape drive parameters used in this simulation, shown in Table 1, are taken from a paper describing striping in large tape libraries by Drapeau and Katz [1]. The authors did an in-depth study of a slow tape library, the Exabyte EXB-120. However, the tape-switch time and search time of EXB-120 system is too slow for video-on-demand use. An average tape switch takes up to 284 seconds. In an environment where tape switches could be frequent, the EXB-120 system is not practical. Therefore I have adopted Drapeau and Katz's estimated parameters for a large, high-performance tape library (DST800).

The jukebox takes requests that comes from the movie manager, and initiates a large read on the tape where the movie is located. If the tape is not currently loaded, an idle tape will have to be rewound, ejected and unloaded, before the new tape can be mounted. The jukebox manager asks the tape drive to read in 500 GOPs for every request. Any incoming request for segments that falls into the range of movie segments pending service from the jukebox are dropped to avoid redundant overlapping reads. As the segments are read, they are sent to the disk via the system bus. The first 300 of the 500 GOPs are also sent to the RAM.

The cutoff point of 260 is calculated from the worst-case tape-switch time, where the mounted tape has to be rewound, and the new tape has to be searched for the requested segment. An estimate of 260 video segments would have been requested during the 104-second worst-case time. The tape drive is asked to read a whole movie, which is 4500 segments, and send them to the disk for buffering; the first 260 segments in a jukebox read are also sent to the RAM. It would certainly be interesting to study what different cutoffs would do to the system performance.

4.7 Error Reporting

The RAM, disk and jukebox each check incoming requests against the system clock. If the request is late (i.e., the system time is past the event's deadline), the facility will record the late event in a log file. The RAM drops a segment when it fails to send out the segment when the movie manager requests it. Every time a segment is dropped by the RAM, the client experiences a "glitch" (a blank moment lasting ~0.4 second).

Although late arrival of requests are recorded by every facility, only failures in RAM are responsible of causing the glitches.

The error log file essentially records the late and dropped requests. The information in each record includes the time the drop occurred, the deadline of the request, the process id (there is a process id for each movie successfully requested), the ID of the facility that dropped the request, the movie name (represented with integer ID), the segment ID of the request, and the number of active movie streams at the time the error occurred.

4.8 Software-Engineering Issues

In this section, I shall examine some decisions that I made while writing the simulator. These decisions and issues range from the queuing-event model provided by Simpack to the efficiency in my code.

Simpack does not support multiple event queues. In a system where there are multiple facilities talking to each other, kludges had to be made in order to distinguish events for one facility to events for another. In the case of an event containing an error, it becomes very hard to track the error down, since it will be mixed into tens or even hundreds of other events in a single event queue. The system becomes prone to bugs as a result.

I also made some decisions that caused inefficiencies in the execution of the simulation. The caches were implemented to use linear search for cache replacement. In my system, where cache replacements were extremely frequent, the decisions led to slow execution. Slow execution made it difficult to debug and to run experiments.

As my first venture into the world of C++, I also realize that I did not exploit the advantages of C++ fully, especially in the inheritance of class attributes. The version of Simpack I use is only half converted from C to C++. It certainly would have helped if I had access to a fully C++ version. In retrospect, the coding process of this project took too much time and energy, both as a result of a bug-prone system and the inefficient implementation of an important data structure. I should have paid more attention to developing an early design of data structures.

5 Metrics

The primary metric for this simulation project is the number of glitches (blank moments) the client experiences during the play time of a movie. As the number of concurrent movie streams increases, the number of glitches increases. The system is considered overloaded when any movie stream blanks out for more than 1 second (~2 GOPs) during its play time.

However, it is very important to note that the maximum number of movies that can be supported depends very strongly on the distribution of the requested movies. If most of the movie requests cluster around a few popular ones, the cache will serve its purpose well and help increase the capacity of the server.

6 Results

The simulator was configured with 512 MBytes of RAM, four HP97560 disks of 1 Gigabyte capacity each, and a video library with 4 tape drives and a robot arm. Each tape has a capacity of 25 Gigabytes, which is room for seventy-five 1/2-hour long movies[1]. Thus, with a movie bank of 375 movies, the library will have 5 tapes. The decision was made to choose a small disk cache so that the disks will fill up fast and to reduce the simulation run time.

The purpose of this simulation was to observe the effect of different caching strategies on a loaded video-on-demand system. While varying the maximum number of movie streams played concurrently, I fed the system with a set of requests of normal and another set of uniform distribution [Figure 3 & 4]. The number of glitches was recorded for all four of the cache-replacement strategies [Table 2 & 3].

6.0.1 Normal Distribution

The tests using requests from a normal distribution showed a gradual decrease in performance in all cache replacement policies [Table 2]. The random cache-replacement policy performed much worse than LFU, LRU and FIFO, when the cache slots were full and a large percentage of slots were locked. However, all four policies performed comparable at low loads. Random replacement also experienced the most dramatic increase in the number of glitches when the system load was increased. The replacement policy did not scale well because it did not exploit the regular access pattern that exists in continuous-media applications like the VoD. As the system load increased, more segments in the cache were locked; as a result, there was a greater chance that the policy would fail to find a free slot[2].

The least frequently used (LFU) policy also did not perform very well. The main reason was because of the aging process (the access counts were set to age by one for every

---

1 1/2-hour long movies were chosen so that simulation time did not need to be too long to observe the effect of movie switching.

2 The random replacement policy tries 15 times to find a replaceable slot at random locations. If could not find a slot in 15 trials, it considers the cache full and fails.

thousand accesses). As the system load increased, aging occurred more often. It would eventually wiped out the access counts the policy needs to optimize its performance[1].

FIFO did not perform as well as LRU mainly because the normal distribution exhibits temporal locality in the requests. If the system had enough memory of what had occurred in the past, LRU would be able to retain the more popular movies in cache. Since FIFO ignored the access frequency and access time of the segments, it might end up throwing out segments that were popular, thus forcing a future reload. More glitches resulted when the jukebox could not keep up with the requests.

For lower system loads, every cache-replacement has the same number of glitches. The glitches were not the result of ineffective cache-replacement, but was due to the inability of the jukebox to service requests fast enough, so that the data could get to the disk and RAM on time from the jukebox.

6.0.2 Uniform Distribution

When the requests were uniformly distributed, there were essentially no cache hits in the tests I ran. Therefore the cache filled up quickly, with a substantial number of segments in cache locked. As a result, random replacement policy performed poorly throughout. LFU performance degraded quickly due to the large number of calls for replacement segments (as mentioned in 6.0.1, aging wiped out access counts).

At high load, the LFU performance appeared unstable because of the thrashing in the tape jukebox. When needed tapes (working set) are swapped out, the robot arm became a bottleneck. That was especially the case when the movies requested were uniformly distributed. Since the tape jukebox uses a simple-minded replacement policy which swaps out the tape that will be idle first, regardless of whether it would be needed immediately after, the performance of the system became hinged on the combination of active movies. If the combination causes constant tape-swapping, the performance of the system would degrade dramatically.

---

[1] In my implementation, aging occurs after every 1000 replacement calls. Alternatively, aging can be implemented to occur after every 1000 hits of cache slots, which may actually yield better results for the policy.

LRU did not perform very steadily with a uniform request distribution. From examining the error log file, I believe that the reason was because some of the prefetched segments in the cache was thrown out in favor of recently used ones. In this extreme case of no cache hits, such action did not bring about any advantage. On the other hand, more reload requests had to be issued. When the jukebox could not keep up with the requests, a series of failures appeared.

FIFO, however, performed rather well in this test because it does not assume any temporal locality in the requests. It happens to do the best thing for the uniform distribution that I fed in (it just happened that no movie was requested more than once in the duration of the test, so there really was no point in keeping any used segments; FIFO did the right thing by throwing the oldest segment old to make room for the incoming segment).

6.1 Limitations

There are a few limitations to remember while reading the results from this paper. One should not take the maximum number of concurrent movies that system can support at its literal value. Real hardware will certainly have different parameters, so the results from a real system will be different. However, the observation that the system works better when the requests exhibits a normal distribution should hold true.

The physical layout of segments on the disk array is also highly abstracted in this study. A more in-depth study of the disk layout and more optimization work in the disk scheduling algorithms is necessary to design high-performance systems.

The length of the movies simulated in this project were limited to on 1/2-hour long. Ideally, they should be of the order of one to two hours, which would also require large amount of cache and longer play time. The selection of movies (375 of them) was also quite limited compared to a real system that serves thousands of users. The bottleneck at the tape jukebox could only get worse if the number of tapes the movies reside on increases.

7 Conclusions

The simulation showed that when the requests are clustered around a few popular movies, the system performed much better than when the requests were uniformly distributed among all movies. We also observed that as the system became overloaded, the performance plummeted, instead of degrading gracefully.

Unless one has advance preloading directives, and a huge pool of cache/buffer, or a highly concentrated pool of requests, caching does not seem to help much. Since the popular movies probably vary according to the time of the day, as well as days in the week, a combination of the above is necessary to have a viable VoD. If, however, the pool of requests were highly concentrated, LRU is certainly the choice among the four that I tested. Random replacement should be avoided at all costs in a VoD application.

8 Future Work

A video-on-demand system consists of a number of devices interacting in a complex manner. This project makes many simplifying assumptions on many complex issues. The network issues, for example, needs to be addressed in a future project, where network delay and errors are taken into account. It would also be beneficial to use benchmarked parameters for the jukebox, instead of numbers compiled from some product literature.

The prefetching strategy used in this simulation is very straightforward and does not provide room for optimization. I believe that the performance of the system can be enhanced through a more intelligent prefetching strategy.

One of the regrets my regrets in this project is that I had not tested the optimal cache-replacement policy, which utilizes knowledge about the future need in finding replacement slots. On most other systems, this is not practical but with a VoD system, one can predict what the system needs by looking at the current cache content. It would be very interesting to see how well the optimal strategy compared to LRU, which utilizes knowledge of the past.

In addition, the simulated VoD system also does not support pause, rewind, and fast-forward. To make the system fully interactive, these features are essential. MPEG-2 specification has certain provision for such actions. A simulation with MPEG-2

parameters would be more realistic. In addition, these actions will also have some interesting effects on the cache content. It makes a very challenging topic of research.

9 Acknowledgments

I want to thank Professor David Kotz, my thesis advisor, for pointing me to most of the parameters used in this simulation. He also led me through much of the difficulties in the implementation and working of the simulator. Without his patience, this troubled project probably would not finish, let alone result in this paper. My most sincere gratitude.

I would also like to thank Chen Yang for helping me to go through a number of stressful cycles of having results and then finding them invalid, with the deadlines coming and passing. Without her the stress would have been too much for me to bear.

References

[1] Ann Drapeau and Randy Katz, “Striping in Large Tape Libraries”, Proceedings of Supercomputing ‘93, pp. 378-387.

[2] Craig Federighi and Lawrence A. Rowe, “A Distributed Hierarchical Storage Manager for a Video-on-Demand System”, UCB Technical Report No. UCB/CSD 94/795.

[3] Craig Freedman, David J. DeWitt, “The SPIFFI Scalable Video-on-Demand System”, University of Wisconsin Technical Report, submitted to IEEE TPDS, 1994.

[4] Borko Furht, Deven Kalra, Frederick Kitson, Arturo Rodriguez and William Wall, “Design Issues for Interactive Television Systems”, IEEE Computer, May 1995, pp. 25-39.

[5] D. James Gemmell, Harrick Vin, Dilip Kandlur, P. Venkat Rangan and Lawrence A. Rowe, “Multimedia Storage Servers: A Tutorial”, IEEE Computer, May 1995, pp. 40-49.

[6] David Kotz, Song Bac Toh, Sriram Radhakrishnan, “A Detailed Simulation Model of the HP 97650 Disk Drive”, Dartmouth College Technical Report PCS-TR94-220.

[7] O. Rose, “Statistical Properties of MPEG video traffic and their impact on modeling in ATM systems”, University of Würzburg, Institute of Computer Science, Research Report Series, Report No. 101, February 1995.

[8] Yen-Jen Oyang, Meng-Huang Lee, Chun-Hung Wen, and Chih-Yuan Chen, “Design of Multimedia Storage Systems for On-Demand Playback”, Proceedings of the 11th International Conference on Data Engineering (ICDE '95).

[9] phade@cs.tu-berlin.de, “MPEG-FAQ/part 1”, comp.compression

[10] Andrew S. Tanenbaum, "Operating Systems: Design and Implementation", Prentice Hall, ISBN: 0-13-637331-3.

[11] Paul Fishwick, "SIMPACK: Getting Started with Simulation Programming in C and C++", http://www.cis.ufl.edu/~fishwick/simpack/simpack.html

Appendix

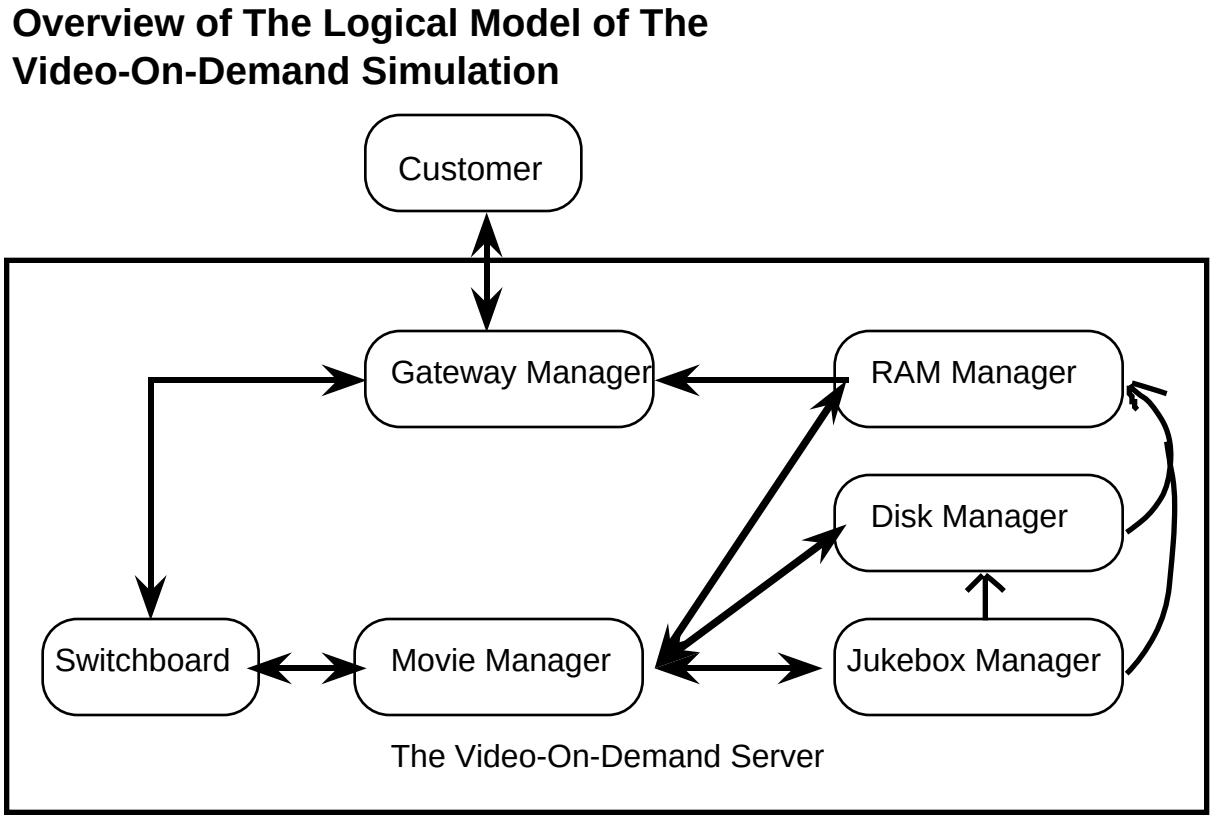
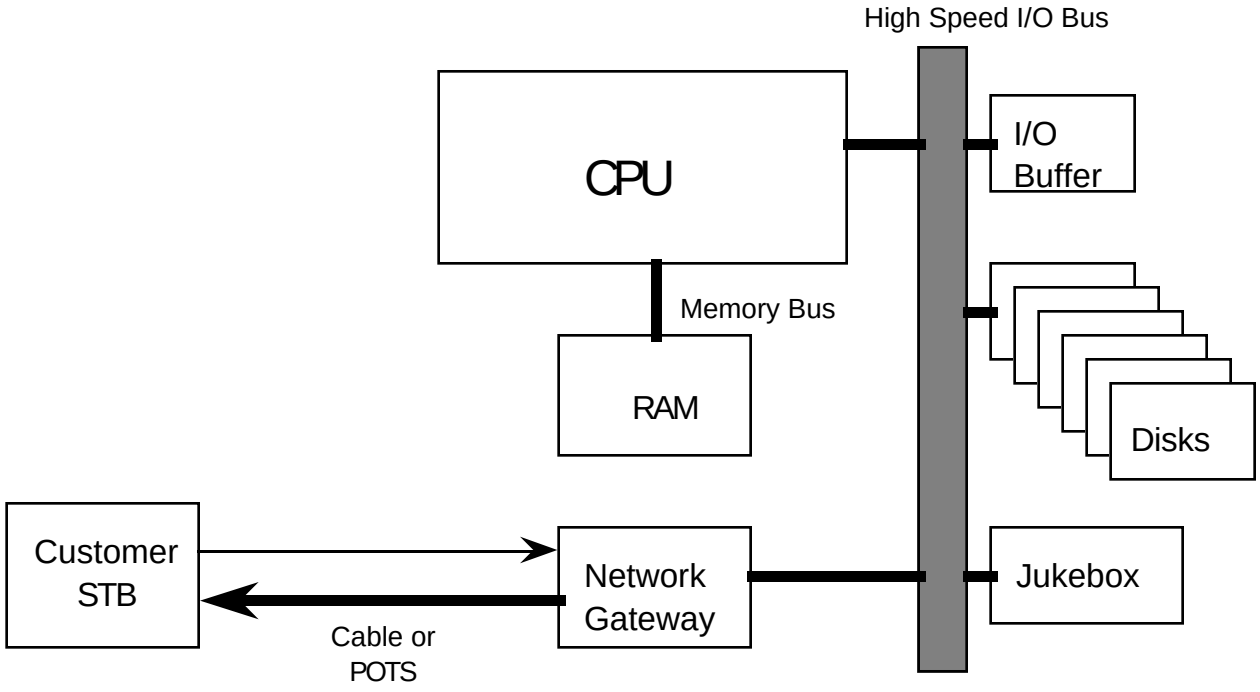


Figure 1: The overview of the logical model of the video-on-demand simulation.

Overview of The Video-On-Demand Simulation Hardware Model



1. Thicker lines represents a connection with a higher bandwidth.

2. The network gateway is an I/O device that manages communication with outside clients. It is not simulated since this study does not cover the networking issue.

3. The customer set-to-box (STB) decodes downstream MPEG movies coming through cable or plain old telephone service (POTS); it sends customer commands upstream through a channel with a much smaller bandwidth.

4. The RAM is 512 Megabytes in size; the disk array consists of 6 disks, each of the size 1 Gigabyte. The jukebox consists of 4 tape drives and a robot arm that retreives and replaces tapes to the racks.

Figure 2: The overview of the video-on-demand simulation hardware model.

The *abnormal* normal distribution

The normal request distribution does not look like a bellshape curve because each of the number between 1 and 375 inclusive, has been mapped to a unique number also in the range of 1 and 375 inclusive, to destroy locality of requests on the tapes.

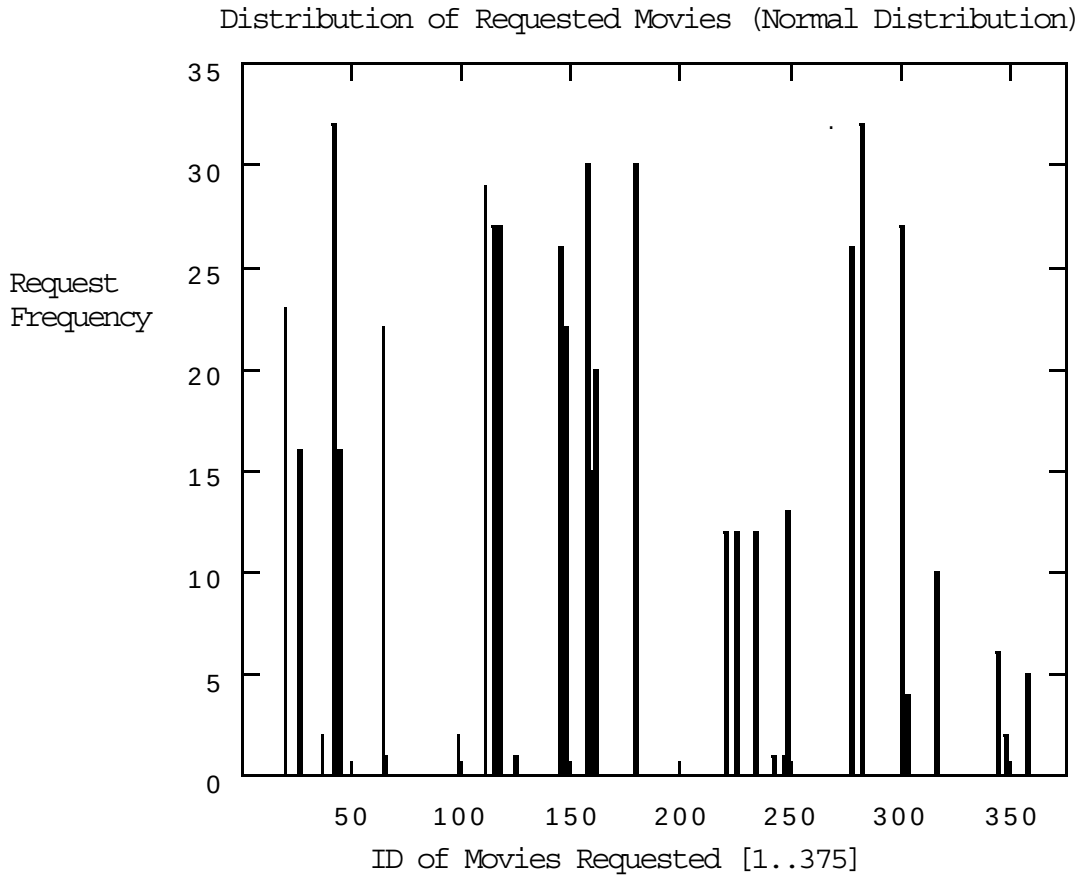


Figure 3: The normal distribution of Requested Movies

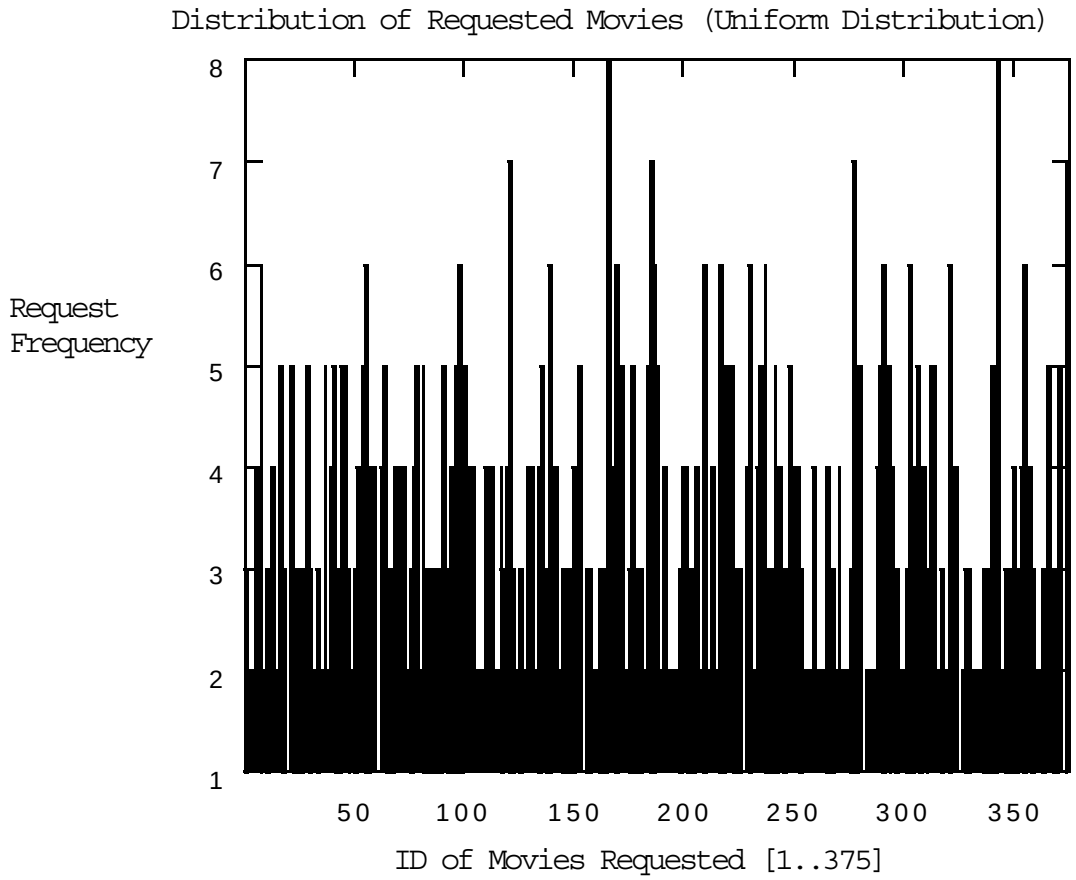


Figure 4: The uniform distribution of Requested Movies

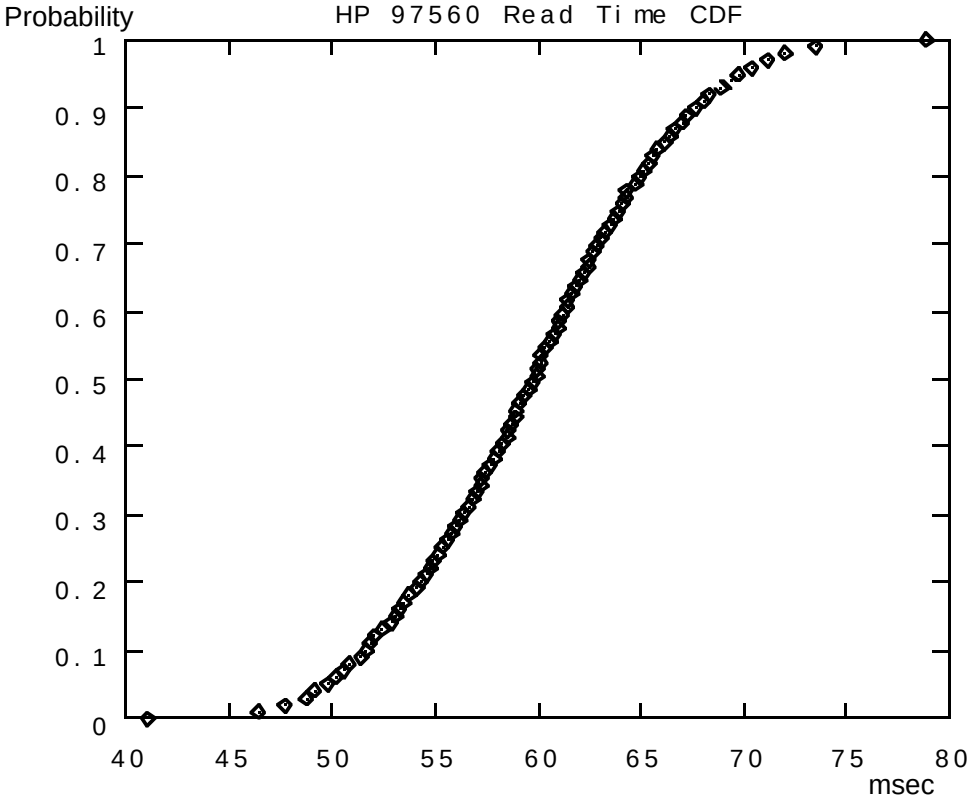


Figure 5: The CDF of Read Time for One GOP (76.8 kB) on an HP 97560 Disk

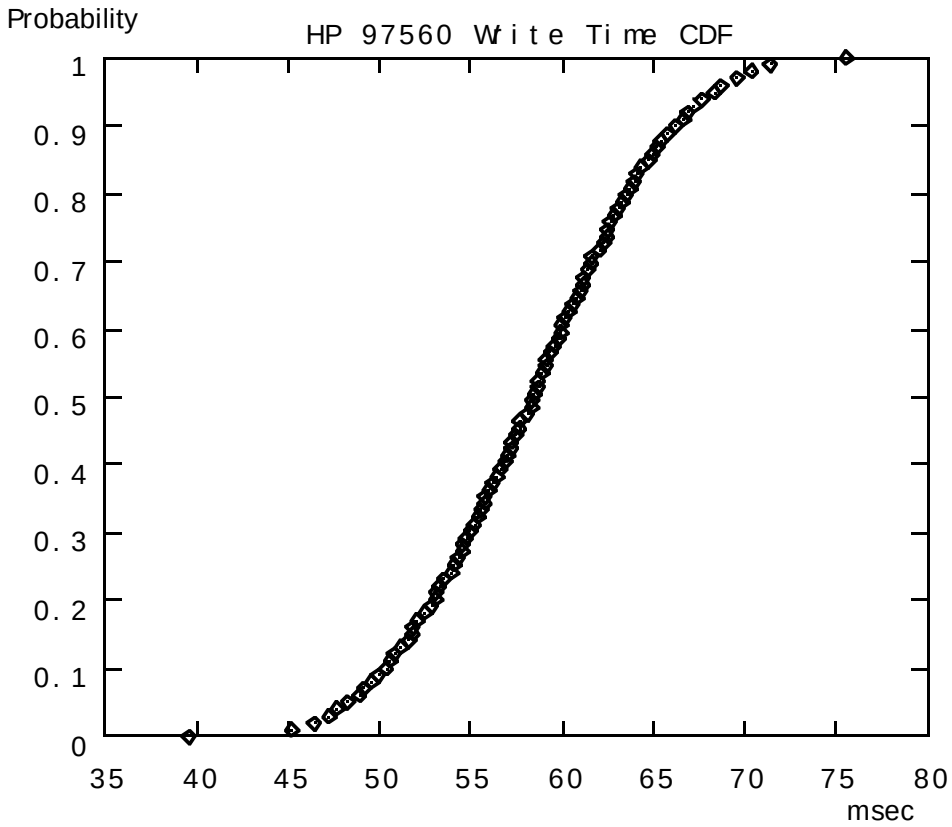


Figure 6: The CDF of Write Time for One GOP (76.8 kB) on an HP 97560 Disk

Operation	Time or Rate
Data Transfer Rate	15 MB/sec
Eject	5 sec
Load	5 sec
Search Startup Time	5 sec
Search rate after startup	750 MB/sec
Rewind Startup Time	5 sec
Rewind rate after startup	750 MB/sec
Robot move time	2 sec
Robot pick time	3 sec
Robot place time	3 sec

Tabel 1: Table of Tape Library Parameters

Simulation parameters for large, high-performance tape library, taken from [9], pp. 382. The authors, Drapeau and Katz claimed that they compiled these parameters from the product literature, so they are not exact performance measures.
Tape size for this library is 25 Gigabytes.

Number of Glitches for Each Policy

Max # Concurrent Movies	LRU	LFU	FIFO	Random
10	64	64	64	64
11	64	64	64	64
13	64	168	64	5217
14	257	1526	500	8041

Table 2: The System Performance with Normally Distributed Requests

Number of Glitches for Each Policy

Max # Concurrent Movies	LRU	LFU	FIFO	Random
7	0	31	0	11808
8	0	2953	0	15052
9	0	3688	0	19728
11	578	34531	8	40592
13	30	30121	32	58869

Table 3: The System Performance with Uniformly Distributed Requests