Deterministic Load-Balancing Schemes for Disk-Based Video-On-Demand Storage Servers[1]

Yitzhak Birk[2]
Technion—Israel Institute of Technology
Haifa, Israel

Abstract

A video-on-demand (VOD) storage server is a parallel, storage-centric system used for playing a large number of relatively slow streams of compressed digitized video and audio concurrently. Data is read from disks in relatively large chunks, and is then "streamed" out onto a distribution network. The primary design goal is to maximize the ratio of the number of concurrent streams to system cost while guaranteeing glitch-free operation. This paper focuses on load-balancing for the purpose of providing throughput that is independent of viewing choices. At the interdisk level, data striping is the obvious solution, but may lead to a quadratic growth of RAM buffer requirements with system size. At the intradisk level, multizone recording results in variable disk throughput. Deterministic schemes for solving each problem are discussed, as well as their joint operation. Finally, efficient staging of data from tertiary storage devices to disk is shown to be possible.

Introduction

Data layout

Video-on-demand (VOD) is used here to refer to a system and service that enable a very large number of end users to concurrently access large repositories of stored data (often of a stream nature such as video and audio), navigate through the material, choose items for viewing, and view them immediately. It is furthermore expected that the "feel" of the service would be one of a private repository. Important applications include movie libraries, educational and training material, video clips for various applications, home shopping, personalized television programming, and probably many applications that have yet to be conceived.

Combining the required resources per active user with the expected number of concurrent users is perhaps one of the greatest challenges to designers of computer and communication systems, as well as to potential service providers. Presently, virtually all major computer and communications system vendors and service providers are engaged in research, development, and initial deployment of VOD systems.

A system capable of providing VOD services comprises three major components: a video server, which is the subject of this paper, user-premise equipment (sometimes referred to as a "set-top box"), and a distribution network. In large-scale, multivendor environments, various gateways are required as well. Each component comprises hardware as well as software. A discussion of such heterogeneous environments appears in references [1], [2], and [3].

VOD servers

A VOD server comprises a storage subsystem (typically using magnetic disk drives as the primary storage device), a large RAM buffer, a streaming and network interface unit, an internal communication subsystem, and a control unit. Data is read from disk into the RAM buffer in relatively large chunks (in order to reduce disk-access overhead), and data for multiple video streams is then streamed out onto the distribution network in small units, such as ATM cells. While in the server, data may also be operated upon for purposes such as error correction, encryption, and content-customization. Interesting papers on various issues pertaining to real VOD servers include [4] and [5].

VOD applications call for large systems and, unlike in many other applications, the storage subsystem plays a central role, not merely occupying a low level in the memory hierarchy. Also, most of a VOD server's cost lies in its storage subsystem. This warrants a careful look at the design of the storage subsystem for VOD. We next characterize the requirements placed on the server's storage subsystem. We should mention that the design of the data paths within a server and the implementation of the streaming function

[1]The work was supported in part by the Franz Ollendorff Foundation, and some of it was carried out at Hewlett-Packard. The author holds the Milton and Lilian Edwards Academic Lectureship.
[2]Email: birk@ee.technion.ac.il

are also challenging, and that the nature of the application permits unique solutions [6], but these issues are beyond the scope of this paper.

Storage subsystem requirements. A VOD storage server must provide a large number of concurrent streams of data. Each stream is typically read from disk, and the rate of each stream is several times lower than the sustained transfer rate of a single magnetic disk drive. (1.5–6.0Mb/s/stream vs. 25–50 Mb/s/drive.) Once its viewing begins, a stream must not overrun or starve the available RAM buffers. High availability is also important. The server must respond promptly to user requests, but the response time to subsequent requests for data may be masked at the cost of extra buffer memory.

VOD applications differ quite significantly from other prominent applications of large-scale storage subsystems: in on-line transaction processing (OLTP), for example, performance is measured in accesses per second, and jitter is hardly an issue; in scientific computing, one often wishes to maximize the transfer rate for a single stream; in file servers, there is usually no notion of streams, and the exact performance measures depend on file size and type of access. Much work has been devoted to optimization of storage performance in various applications. For example, the organization of data in a disk array used for OLTP is discussed in [7]. For general-purpose workstations, schemes such as placing the most latency-critical data in centrally located tracks and placing different types of data in different disk drives have been proposed ([8],[9]). VOD servers have been receiving much attention lately, but much has yet to be done.

Storage subsystem cost. The cost of disk drives is the largest component of a VOD server's cost, and servers are expected to be bandwidth- rather than storage-limited, so disks must be used efficiently. However, one must also keep in check the cost of RAM buffers, which are required for masking disk response time and for storing the chunks of data received from disk. (Data caching, and even reading ahead, are largely useless and even harmful in VOD servers.)

Load-balancing versus load-matching

Load-balancing is recognized as an important issue in many parallel systems. Typically, it is taken to mean an even distribution of work among equal computation (or other) resources. The situation in VOD servers is somewhat different in that the (playback) load is tied to the location of the data, which cannot be altered without penalty. Moreover, the dependence of a disk's transfer rate on track location adds another dimension to the problem, whereby the same work output (data rate) requires variable amounts of effort (fraction of disk time). This last problem gives rise to two approaches:

- *Load-matching*, whereby an attempt is made to match the track location of data to the frequency with which it is accessed: the most frequently accessed data is placed in the "best" (outermost) track locations, and the least frequently accessed data in the innermost ones. When successful, this approach maximizes the expected value of the server's streaming capacity.
- *Load-balancing*, whereby an attempt is made to achieve a streaming capacity that is independent of viewing choices and of the location of the desired data. Ideally, this also maximizes the guaranteed streaming capacity.

Fault tolerance

Disk drives are very reliable devices, with a calculated MTBF of nearly one million hours. Consequently, even in a system with hundreds of disk drives, the failure rate may be acceptable. Moreover, the data is mostly prerecorded, so one can keep a spare copy on tape and need not fear losing data. However, since data for any given movie is striped across many, possibly all, disk drives, any failure constitutes a common event to numerous users. Consequently, a very high degree of availability is desirable.

One could try to provide high availability by keeping a spare, empty disk drive, and loading it with the data that used to be on the drive that failed. Doing so (by keeping each movie on a tape and rebuilding the disk from tapes) would be prohibitively time-consuming, since numerous tapes would have to be loaded and unloaded, and entire tapes would have to be scanned, because the appropriate data is not contiguous. Alternatively, one could keep an image of every disk on tape; the usefulness of such a scheme would depend on the frequency of changes to the disks' contents. Another option is to use redundancy that permits reconstruction of the faulty disk's data from the data on other disks, that is, some kind of RAID [10]. In this paper, our discussion of fault tolerance assumes this approach.

The conventional use of RAIDs entails reading, either at all times or only when a disk has failed, reading an entire stripe into memory, and reconstructing the data of the faulty disk if there is one. In applications such as on-line transaction processing (OLTP), the amount of data that needs to be read per transaction is very small, and striping such data across multiple disk drives would severely limit the number of accesses per second. Consequently, when operating in degraded mode (when a disk has failed), most of the data in the parity group is required only for reconstruction

and is discarded immediately. The severe degradation in system performance is unavoidable. In applications that require very large amounts of data as fast as possible, proper data layout causes all the data in the parity group to be useful, and it is furthermore needed immediately. There is thus hardly any degradation in performance when a single disk fails.

VOD is different from the above extreme cases: all the data in the parity group is useful, but not immediately. There is consequently a dilemma: discarding the data following reconstruction and reading it again when needed would save buffer space but reduce streaming capacity, whereas storing the data until needed would minimize performance degradation but would require larger RAM buffers. With many disk arrangements, discarding the data would result in the halving of streaming capacity; we will therefore restrict the discussion in this paper to schemes that keep the data.

In the remainder of this paper, we explore deterministic schemes for load-balancing in VOD servers, at both the inter- and intradisk levels. In sections 2 and 3, these two issues are discussed in isolation, and section 4 discusses their joint operation. Section 5 examines the ramifications of also using the disks for staging of data from high-speed tertiary storage devices, and section 6 offers concluding remarks.

Interdisk load-balancing

Background

In the case of partitioning data among disks, unlike that of placing the data within a disk, both load-balancing and load-matching can be achieved by striping the data of every movie across all disk drives. The granularity of the striping is determined by weighing disk utilization, which is maximized by coarse striping, against RAM buffer size, which is minimized with fine striping. Since there is no sense in making the granularity of the data placement coarser than one chunk (a chunk is the amount of data read for a single stream in a single time slice), a reasonable size with current disk drives would be on the order of 128–256KB.

The RAM buffer explosion

Consider an array of M disk drives, with data striped across all drives. Suppose initially that the disk drives are operated in synchrony and that the chunks of an entire stripe are read simultaneously into RAM buffers. The individual chunks are held in RAM until needed for streaming, at which time they are discarded. In the event of a faulty disk drive, reconstruction can take place using parity information.

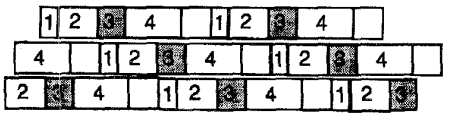

Figure 1. Staggered access to three disks, jointly capable of playing four concurrent streams. Time slices for stream #3 are highlighted.

As was explained earlier, the granularity of the striping is dictated by disk-overhead considerations, and is thus independent of the array's size. Consequently, the amount of data read per stream in a single time slice increases linearly with the size of the array across which it is striped, and so does the amount of RAM buffer required for each stream. Since the maximum number of streams also increases linearly with the number of disks, total RAM size would increase quadratically with system size. In fact, this can be observed in the graphs of [11], though the small system size hides the significance.

One way of overcoming this problem is to stagger the access schedules to the different disks so that each stream is served by at most one disk at any instant. This results in a constant buffer size per stream. Figure 1 depicts the schedules of three disk drives that are jointly playing four concurrent streams. The streams receive time slices with equal frequencies, but not necessarily of the same size. Nonetheless, note that the time slices for any given stream are equispaced in time. If all streams are awarded time slices with the same frequency, this can also be described as M disks with identical schedules, which are delayed in time relative to one another by $1/M$ of a schedule round. This also indicates that glitch-free operation should be possible in such cases with moderate buffer requirements.

Figure 2 depicts the RAM buffer occupancy for a single stream as a function of time with simultaneous access and with staggered access.

Unfortunately, the benefits of staggered access disappear when there is a faulty disk drive, since the parity group is unavailable for reconstruction, and making it available would again require the large amount of memory. It is possible to somewhat reduce buffer sizes by reading a chunk only when needed, that is, at the earlier streaming time and that of the chunk on the faulty disk, and releasing it upon its streaming. (The reconstruction can be carried out incrementally, so at most one partially reconstructed chunk per stream would need to be kept in RAM until the streaming time of the reconstructed chunk.) Such a refinement is important, but does not solve the scalability problem. It may also complicate the scheduling of the disks, leading to problems in other streams. We will assume the basic scheme for facility of exposition.

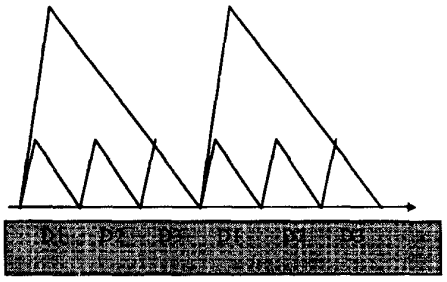

Figure 2. Occupied buffer space vs. time, shown for a single stream in a three-disk array. The "fine teeth" represent staggered access and the "coarse" ones represent simultaneous access to all disks.

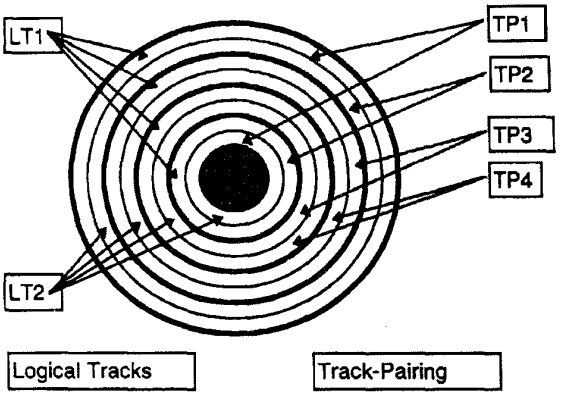

Figure 3. A representative surface of a disk drive with four recording zones, each with two tracks. An arrangement into two logical tracks is shown on the left, and into four track pairs on the right.

A possible solution: partitioned RAID

The buffer-size scalability problem can be solved by partitioning the set of disks into arrays of fixed size, for example, k+1. (This increases the storage overhead from $1/M$ to $1/k$.) Each array would be operated as a RAID with simultaneous access, but the access schedules to the different arrays would be staggered. For a system with M disks, the total RAM buffer size would increase as $k \cdot M$, which is linear in M. Other solutions are possible, but are beyond the scope of this paper.

Intradisk load balancing

Multizone recording

Modern magnetic disk drives employ multizone recording, which is a close approximation of fixed linear recording density: the disk is partitioned into sets of contiguous tracks, called zones, and track capacity within each zone is equal to the maximum permissible capacity of the zone's innermost track. Such disks rotate at fixed RPM, so transfer rate depends on track location. A typical dynamic range can be as high as 1.8:1 ([12],[13]).

If a movie occupied a large fraction of a disk, the permissible number of concurrent viewers of any given movie would change with time. If, as is often the case, each movie is striped across several disks and occupies only (the same) small fraction of each of them, the permissible number of viewers changes only when they make their selections, but does depend on those. Since the number of concurrent video streams is the most important measure of a VOD server's performance, this issue must be addressed.

One possible approach is load-matching, which was described in the introduction. However, the viewing pattern may deviate significantly from the "typical" one as it changes with the time of day or in response to unexpected events. In this case, the permissible number of streams may drop by tens of percents, and the problem may persist for an hour or so. One could dynamically rearrange the material on disk, but this takes up time and bandwidth, especially in fault-tolerant systems.

An alternative "static" approach, referred to as load-balancing, is to maximize the guaranteed (over the range of viewing choices) permissible number of concurrent video streams. This can be done using randomized layouts, but it is then difficult to guarantee the short-term behavior. There are, however, two deterministic schemes for achieving this goal.

Logical tracks [14]

For a disk that has like numbers of tracks in all zones, one constructs fixed-size logical tracks comprising an equal number of same-numbered physical tracks from every zone. (See Figure 3.) While the original purpose of this scheme was apparently to adapt multizone disks for use with operating systems that can only handle fixed-size tracks, recording each movie in logical-track order would guarantee a sustained transfer rate at playback time that is independent of viewing choices.

Track-pairing [15]

This scheme is based on the observation that with fixed linear recording density, track capacities form an arithmetic sequence. By conceptually pairing the innermost track with the outermost one, the second innermost with the second outermost, and so on, both the capacity and the net reading time are the same for all pairs, and consequently so is the transfer rate. (See Figure 3.) By recording a movie

alternately on a range of contiguous "outer" tracks and their "inner" counterparts, the disk's throughput becomes independent of viewing choices with essentially no penalty in terms of disk overhead. The method has been implemented on an HP C2247 1GB disk drive under the Microsoft Windows NT operating system [16], and implementation on a pair of IBM disk drives (pairing track i on one disk with N-i on the other) is nearing completion. Track-pairing can be combined with load-matching by excluding a band of tracks from the pairing and reserving them for "hot" movies.

Scheduling with track-pairing is quite simple: if the disk arm encounters the reading point of each stream in a certain order during its inward sweep and reads a single chunk per stream (either in an outer track or in an inner one), it will encounter the next reading point of each stream in the same order during its outward sweep. This is because both the relative radial order of the next reading points of the different streams and the direction of the sweep are reversed [15]. Thus, a bidirectional elevator schedule can be used instead of a circular scan.

Track-pairing requires more bookkeeping than logical tracks, but offers important advantages in terms of the buffers required to mask the short-term variability in transfer rate, since the fixed rate is already achieved after two chunks.

Proposition 1: Streaming capacity with both logical tracks and track-pairing is equal to the expected streaming capacity if blocks were read at random and disk-overhead were ignored, and this is the highest streaming capacity that one can guarantee without knowing the viewing choices in advance.

Proof: Since streaming capacity with both LT and TP is independent of viewing choices, let us consider (without loss of generality) choices that call for the reading of every block on the disk exactly once. Clearly, this is the same "mix" as when randomly accessing blocks with equal probability, so the streaming capacity will be the same if overhead is ignored. The second claim is proven by contradiction. Suppose it were possible to guarantee a higher streaming rate regardless of viewing choices. We would then again choose to view every block once, leading to a conclusion that it is possible to read the entire disk faster than when reading every block once, a clear contradiction.

Comparison

The main difference between LT and TP in the context of VOD is the required buffer space. Considering a single disk drive, the buffering is required to bridge over the time between consecutive time slices allocated to a given stream, and to mask the short-term variability in disk transfer rate for that stream.

With track-pairing, the maximum buffer size for any stream is equal to the size of a data chunk from the outermost zone. (This ignores required spares on one hand, and the fact that data is streamed out during the reading, albeit at a much slower rate than the disk rate.)

With logical tracks, the situation is more complicated. To avoid problems with initial conditions, we assume that the logical track begins in the outermost zone. The amount of data read from the outermost half of the zones exceeds the amount needed for streaming, whereas the amount read from the remaining zones is below that, with the total being equal. The buffer occupancy therefore reaches its maximum after reading from the outer half of the zones. The amount of data in the buffer at that time is equal to the sum of the chunk sizes of the outer zones, less the amount streamed out. (Unlike TP, the time window considered here spans multiple time slices for the stream. Since the playing of a stream takes place all the time, whereas reading its data from disk occurs only in its time slices, it is no longer possible to neglect the amount of data that is streamed in the process.) With N_Z zones, the amount of data read is equal to $N_Z/2$ times the chunk size of zone number $N_Z/4$, and the amount of data streamed during the same time is equal to $(N_Z/2 - 1)$ times the chunk size of zone $N_Z/2$. (The peak is reached before the last chunk is streamed out.) A more precise derivation appears in [15] (along with numerical results for a specific disk drive), indicating that the required buffer size with LT is substantially higher, and the difference becomes more pronounced as the number of zones increases.

Combining interdisk and intradisk load balancing

In the two previous sections, we discussed inter- and intradisk issues in isolation. In this section, we explore the combinations. Clearly, all combinations will retain the property of maximizing the guaranteed streaming capacity and, in so doing, making streaming capacity independent of viewing choices. The real challenge here is to minimize the required size of RAM buffers.

In a fault-tolerant storage system, RAM buffers are required for three purposes: 1) conversion between the coarse granularity of disk accesses and the fine granularity of sharing distribution-network resources; 2) smoothing over short-term variability in transfer rate of MZR disks when employing LT or TP; and 3) keeping data that was read "prematurely" for reconstruction purposes, in order to obviate the need to read it again when needed. (Reading again would introduce substantial storage-bandwidth overhead, which in many organizations is unacceptable.)

The combination of intra- and interdisk schemes gives rise to yet another purpose of buffering—storing chunks

that were read prematurely and out of sequence. With TP, for example, if we read a parity group comprising chunks from zone i, the streaming of any two consecutive chunks from this group is separated by the streaming of a chunk from zone $N_Z - i$, thereby doubling the time during which chunks must be buffered.

The design space

There are several ways of applying logical tracks and track-pairing to an array of disks. The degrees of freedom are data placement and the construction of parity groups. The considerations in choosing among the options are buffer requirements, bandwidth overhead, and storage overhead. We next list the options and examine them in the contexts of LT and TP. We discuss both fault-free and degraded mode operation. Throughout the discussion, we assume that chunks are not discarded if needed later. Also, we assume synchronous access to the entire parity group, which is consistent with a partitioned RAID.

Data placement. Recording an entire logical track (or pair of chunks with TP) on one disk drive before moving on to the next would create scheduling problems, and would result in a coarse granularity of the load-balancing among disks. We therefore consider only schemes in which consecutive chunks of any movie are recorded on consecutive disks in cyclical order.

Having decided to record a single chunk on a disk before moving on to the next one, let us define the stride of a striped logical-track (or track-paired) layout as the number of disks containing consecutive data chunks of the stream in the same zone. A stride of one entails changing zones at each disk change. A stride of M entails placing consecutive chunks in the same zone of all disks before moving to the next zone. Of particular interest is a stride of size k, whereby k consecutive chunks are recorded, one each, in the same location on each of the k disks holding the data for a parity group, and the parity chunk is recorded in the same location on the remaining disk of that group. The next k (appropriately sized) chunks are then recorded in the next zone on each of the next k disks, and so on. The process wraps around from the last disk in the array to the first one. We refer to this as "layout in parity-group order." The stride size represents a trade-off between storage overhead, storage bandwidth, and buffer size, as will be clarified shortly. Strides larger than M offer no benefits and are not considered.

Parity groups. These must be constructed from same-zone chunks. Two choices that must be made in constructing parity groups are whether the chunks constituting a parity group must all belong to the same movie and whether a parity group must include all disks or may include only a subset.

Logical tracks on multiple disks

The most natural combination of LT and staggered access is to choose the sequence of physical tracks constituting a single logical track from successive disk drives (unit stride). A logical track could thus comprise a track in the outermost zone of some disk, a track in the second outermost zone of the next disk, and so on, with disk numbers wrapping around when the last disk is reached. This scheme is quite appealing so long as there are no faulty disks, since the required buffering per stream remains the same as with a single disk drive.

Unfortunately, if chunks are stored in this logical-track order, consecutive chunks of the same (movie,zone) pair are separated in the streaming order by $N_Z - 1$ chunks of the same stream. Consequently, when operating in degraded mode, regardless of any further details, the duration of the buffering of a chunk is N_Z times longer than with a single disk. With N_Z often exceeding 10, and compounded by the need to read the entire parity group, this is clearly unacceptable even with mitigating schemes.

The use of the same layout in conjunction with partitioned RAIDs would reduce the number of chunks that need to be read prematurely and buffered from M to k, but would not reduce the buffering time. This is therefore unacceptable as well.

With layout stride equal to M, the minimum time window required to achieve the average transfer rate is maximized. With a partitioned RAID this furthermore offers no advantage over a stride size of k. We therefore drop this option as well.

With layout in parity-group order, a parity group contains sequential chunks of the same stream, thereby minimizing the buffering time for any given parity-group size. The price is an increase (relative to unit stride) in the minimum time window over which the constant streaming capacity can be guaranteed by a factor equal to the size of the parity group. This manifests itself in the form of larger buffers. (The increase in the time window is actually by a factor equal to the number of chunks in the parity group that belong to the stream of interest.) An increase in the number of zones and/or a reduction in the size of the parity group favor layout in parity-group order.

To complete the discussion of layout stride, it is important to distinguish between the effects of the two factors on buffer size: the increase in buffer sizes that results from the larger number of time slices (over which the transfer rate must be averaged to attain the fixed rate) occurs at all times; in contrast, buffering time is only affected during degraded mode and only when the parity group containing

the faulty disk is accessed. With partitioned RAIDs and a single faulty disk, only k/M of the disk accesses are to this group. If the buffers are shared among all disks, the effects should be weighted properly when picking a stride. Overall, it appears that using partitioned RAIDs in conjunction with strides equal to the number of disks in a partition is the preferred method when there are a large number of zones.

We have thus far assumed that all of the chunks forming a parity group belong to the same stream, but this need not be the case. The benefit of combining chunks from two or more streams in the same parity group would be a reduction in storage overhead without the penalties associated with larger parity groups; the cost would be an increase in storage bandwidth required to read useless chunks of data. We believe that this will not be cost-effective in most cases, since higher-capacity disk drives are more cost-effective and systems are thus likely to be bandwidth-limited.

In summary, then, the best combination of logical tracks and striping appears to entail the use of a partitioned RAID, with recording stride equal to the size of a partition.

Track-pairing on multiple disks

Regardless of the number of zones, the behavior of TP in this context is that of LT with $N_Z = 2$. In this case, the penalty of recording the data in an alternating-zone order is merely a doubling of the buffering time of each chunk, and only applies to chunks in the parity group that contains the faulty disk. In return, the fixed data rate per stream is already guaranteed after reading two of its chunks, and this applies to all streams all of the time. Consequently, it appears that a unit stride might be optimal for TP, though the difference depends on the values of the configuration parameters.

With unit stride, both data layout and arm motion depend on whether the number of disk drives is even or odd. With an even number, any given movie resides either on outer tracks or on inner tracks of any given disk, and the arm of each disk moves in a circular scan. (Alternate drives read only during an inward sweep or only during an outward sweep.) It is therefore necessary to partition the movies among the two patterns or to artificially switch several times in the course of recording a movie. (This is only a storage-balancing issue, not a performance issue.) With an odd number of disks, each movie occupies both inner and outer tracks of every disk, and the arm scheduling uses a bidirectional elevator algorithm, as was the case for a single disk.

Staging data from tertiary storage

Background

The research, development, and implementation activity in video servers has concentrated on disk-based servers, with all data stored on disk and played from disk via minimal buffering. This is often the best approach, since many systems are expected to be bandwidth-limited, requiring a larger number of disk drives than would be required to merely store the most popular material. Yet, as more and more material is digitized, reflecting a desire to have "everything" on line, most of the material will be viewed infrequently. In such a case, the very low cost of tertiary storage media like magnetic or optical tape, along with their high volumetric density, make them attractive.

An additional potential role for tertiary storage is holding spare copies of all material. The temptation is high, because most of the server's data is prerecorded. However, since data is striped across many disks, and disks tend to fail individually, a simplistic approach (whereby a copy of each movie is stored sequentially on tape) is likely to result in very long restoration times or require many tertiary (for example, tape) drives and be very expensive. Disk-based solutions with sufficiently large parity groups may be more attractive. While this issue is worthy of further research, it is beyond the scope of this paper. In the remainder of this section, we focus on incorporation of tertiary storage into the storage hierarchy of the video server.

The use of tertiary storage in video-on-demand servers has recently been explored in [17]. In that paper, a distinction is made between tertiary storage devices, whose transfer rates are substantially higher than the data rate of a single (compressed) video stream, and devices whose rate is comparable to that of a video stream. High-speed magnetic tape drives and CD-ROM drives serve as examples for the respective cases. Also, two modes of using the tertiary devices are considered: direct playback from the tertiary device, and staging onto magnetic disks of a video server for subsequent playback. (Yet another option is to copy the data directly from the tertiary device to user-premise storage equipment, but this is outside the scope of both [17] and the current paper.)

Stated briefly, the conclusions of [17] are: direct playback is justified only when the data rate of the tertiary device is slightly higher than that of the video stream; with high-speed tape drives, only staging is sensible, because the tape drives are expensive and cannot be operated efficiently for direct playback (when the user lacks significant buffering capability); finally, the range of use-frequency of movies in which storage on disk is optimal is larger than one would expect, and the viewing-frequency threshold below which slow tertiary devices compare favorably with magnetic disks is much lower than the threshold below which magnetic tape libraries compare favorably with magnetic disks. The latter conclusion reflects the fact that only the media cost of tertiary storage is attractive; if a large number of drives is required, the advantage diminishes. As for the details of staging, the conclusion in [17] is that the staged

data must be striped across many disks and written concurrently to them, both for load-balancing and in order to efficiently utilize tape drives whose speed exceeds that of disk drives.

In the remainder of this section, we explore the issue of combining our load-balancing schemes for recording and playback with the concurrent staging of data from fast (relative to video rate) tertiary devices.

Concurrent staging and playback

From scheduling and performance perspectives, the staging of a movie from a high-speed tertiary storage system to the server's main disk system initially appears identical to the playing of an additional, high-speed stream. There are indeed some similarities, but also some very important differences, which will be explored shortly. Throughout the discussion, we assume that staged data is read from tertiary storage into RAM buffer, and is subsequently copied to disk. The tertiary-storage drive operates continuously because its efficient utilization is important. The main design goals are to avoid glitches while keeping all buffer sizes in check. To do so, we already know that a "regular" schedule of the playback disk-accesses is very important.

Consider a disk-based video server that employs track-pairing with staggered access and no failures. (In this case, partitioned RAIDs are equivalent to a smaller number of faster disks). Moreover, all streams are played at their nominal rates. In the event that different streams require different data rates for nominal-speed playback, we further assume that the chunk sizes and time slices were determined accordingly at recording time. This and the use of track-pairing result in simple, time-staggered round-robin disk-access schedules for all disks. When responding to a new playback request, the "phase," that is, place in the round-robin schedule, depends on the time of the request and the ongoing streams. Once it is picked, however, the simple round-robin schedule can be employed.

Next, consider the addition of a staging stream. The rate of this stream is that of the tertiary drive generating it, yet its chunks size and time slice on disk must be determined by its video rate in order to adhere to the playback scheme. Since the tertiary drive is to operate at capacity, the rate of disk-access requests cannot be equal to that of the playback streams, and it is no longer possible to avoid more complicated scheduling.

One can think of the request-load on any given disk as the sum of two periodic signals with different periods. This results in transient congestion at the beat frequency, bringing up the question of scheduling priority. Clearly, as long as the aggregate bandwidth of the disk server exceeds the sum of the requirements for staging and playback, glitches can be avoided through the use of sufficiently large buffers. The challenge is to minimize the required buffer size.

Both the staging streams and the playback streams require buffering in order to hide problems (from the tertiary drive and the viewers, respectively). However, there is a difference between the two requirements: the buffering of the staging stream is aimed at preventing overflow (or choking of the sender), whereas the buffering of the playback streams is aimed at avoiding underflow (or starvation of the recipient). This difference is fundamental, because buffer space for overflow-avoidance is required only when overflow is about to occur, whereas buffer space for starvation-avoidance must be allocated and filled in advance, because when the problem occurs it is too late. A further implication of this observation is that starvation-prevention buffers must be allocated to all playback streams, whereas overflow-prevention buffers must be allocated to staging streams only on an immediate-need basis. Moreover, occasional interruptions on the tertiary-drive side are not visible to the users and result in a negligible performance degradation.

A final important difference between the buffering of the staging streams and that of the playback streams is that the staging buffers are emptied into another huge buffer (disks), whereas data in the playback buffers is sent directly to the users whose buffering capability is assumed to be very limited. Accordingly, playback buffers must be emptied in stream order, whereas the staging buffers may be emptied into disks in any order, as soon as disk schedules permit.

It is also important to note that the data rate of a staging stream may be higher than that of a single disk, whereas that of a playback video stream is substantially lower. Any potential problem arising from this, however, is solved based on the observation that out-of-order reading from the staging buffer is permissible. Clearly, the concurrent reading from the buffer of data destined for different disks must also be permissible, and this capability should be provided in any implementation.

In view of the above, it appears that the efficient accommodation of staging streams should be possible without affecting playback performance and buffering requirements for playback streams. Moreover, the allocation of buffer space for staging only on an immediate-need basis and the flexibility in reading data from the staging buffer suggest that the additional buffer requirements for staging will be moderate despite the fact that disk-accesses on behalf of the staging streams receive lower priority.

Conclusion

Providing a large number of concurrent streams is an important goal for the designer of a storage system for VOD servers. In this paper, we characterized the requirements of

this class of applications and showed that they differ from those of other applications in several important ways.

We explored the issue of load balancing in such systems, and showed that the dependence of a disk's transfer rate on track location adds an interesting facet to this issue. Focusing on static, deterministic schemes, we looked into the problem at both the inter- and intradisk level, and then went on to examine the joint operation of schemes in the two domains. Among the approaches that were examined, the combination of track-pairing and a partitioned RAID with unit stride appears to be the best, followed by track-pairing with a stride equal to the size of a partition (and a parity group).

The most important measure, streaming capacity, was explored at the beginning, and discussion was restricted to schemes that maximize the guaranteed streaming capacity. The discussion was then focused on the remaining important cost and performance measures, namely storage overhead, storage bandwidth, and required buffer space.

Important temporal issues such as efficient disk scheduling were not discussed in great depth. However, the possibility of efficient scheduling with track-pairing has been established elsewhere [15], and the staggered reading schedules from the different RAID partitions can easily be shown to operate correctly and efficiently.

As the size of systems increases, staging of data from tertiary storage may become a necessity. We have shown that efficient staging can be combined with our load-balancing schemes for the disk-based system.

The issue of streams with different rates, quick response to user requests, and so on, and especially its effect on the "clockwork" operation of the coordinated schedules, require further study. To this end, we have been examining different approaches which appear to offer agile, robust, glitch-free operation at very high loads with moderate overhead. These will be reported elsewhere.

Acknowledgments

The donations of disk drives by IBM, software by Microsoft and a PC by Intel are gratefully acknowledged. The implementation of track-pairing was carried out in the Parallel Systems Lab of the EE department at the Technion.

References

[1] S.D. Dukes, "Next Generation Cable Network Architectures," *Proc. NCTA Conf.*, 1993, pp. 8–29.

[2] W.D. Sincoskie, "System Architecture for a Large Scale Video-On-demand Service," *Comp. Net. and ISDN Sys.*, Vol. 22, North-Holland, 1991, pp. 155–162.

[3] P.V. Rangan and S. Ramanathan, "Designing an On-demand Multimedia Service," *IEEE Comm.*, July 1992.

[4] F.A. Tobagi and J. Pang, "StarWorks—A Video Applications Server," *Proc. IEEE Spring Compcon 1993*, San Francisco, Calif., Feb. 1993, pp. 4–11.

[5] R.L. Haskin, "The Shark Continuous Media Server," *Proc. IEEE Spring Compcon 1993*, San Francisco, Calif., Feb. 1993, pp. 12–16.

[6] Y. Birk and R. Perets, "A Switch for Large-scale Video Servers," work in progress.

[7] J. Gray, R. Horst, and M. Walker, "Parity Striping of Disc Arrays: Low-cost Reliable Storage with Acceptable Throughput," *Proc. 16th Int'l Conf. Very Large Databases*, Brisbane, Australia, Aug. 1990, pp. 148–159.

[8] C. Ruemmler, J. Wilkes, "Disk Shuffling," Hewlett-Packard Technical Report *HPL-91-156*, Oct. 1991.

[9] K. Muller and J. Pasquale, "A High-Performance Multi-structured File System Design," *Proc. 13th ACM Symp. Operating System Principles (SOSP)*, Asilomar, Calif., Oct. 1991, pp. 56–67.

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

[11] F.A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID—A Disk Array Management System for Video Files," *Proc. 1st ACM Int'l Conf. Multimedia*, Aug. 1–6, 1993, Anaheim, Calif.

[12] Hewlett-Packard Company, C2240 SCSI-2 disk drive—Technical Reference Manual, 2nd ed., P/N 5960-8346, Apr. 1992.

[13] Hewlett-Packard Company, C2486A/88A/90A SCSI-2 Disk Drives—Technical Reference Manual, 1st ed., Sept. 1992.

[14] S.R. Heltzer, J.M. Menon, and M.F. Mitoma, "Logical Data Tracks Extending among a Plurality of Zones of Physical Tracks of One or More Disk Devices," US Patent No. 5,202,799, Apr. 1993.

[15] Y. Birk, "Track-Pairing: A Novel Data Layout for VOD Storage Servers," *Proc. Int'l Conf. Multimedia Computing and Sys.*, Washington, DC, May 15–18, 1995.

[16] E. Almog, N. Kogan, and I. Tadmor, "Video File Server Prototype," Project Report, Parallel Sys. Lab, EE Dept., Technion, Haifa, Israel, May 1994.

[17] M.G. Kienzle, A. Dan, D. Sitaram, and W. Tetzlaff, "Using Tertiary Storage in Video-on-Demand Servers," *Proc. IEEE CompCon Spr. '95*, Mar. 5–9, San Francisco, Calif., pp. 225–233.