RC 20670 (12/20/96)
Computer Science

# IBM Research Report

## Multimedia Caching Strategies for Heterogeneous Application and Server Environments

Asit Dan and Dinkar Sitaram

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

# Multimedia Caching Strategies for Heterogeneous Application and Server Environments[1]

Asit Dan Dinkar Sitaram

IBM Research Division
T. J. Watson Research Center
Hawthorne, NY 10532
{asit, sitaram}@watson.ibm.com

**Abstract:** In a multimedia system, *storage* and *bandwidth* are critical resources since any presentation requires a large volume of data to be delivered in real-time. Caching of multimedia documents in local storage can alleviate large retrieval bandwidth requirements. An important requirement for a multimedia caching policy is to guarantee continuous delivery even when a stream is served from cache. It should also cope with dynamic changes in workload and heterogeneity arising from large and small multimedia files. The proposed *Generalized Interval Caching* policy, that caches intervals between successive streams of a large file as well as entire small files, satisfies all the above criteria. A caching policy needs to cope with additional challenges in a large scale distributed multimedia environment consisting of many heterogeneous servers. The issues include a) routing of requests to ensure good cache hits in each server, and b) balancing of loads across servers. For routing of requests, we introduce the notion of an *asset group* and propose an affinity routing policy based on this concept. Finally, we adapt the GIC policy for load balancing across servers.

## 1 Introduction

The rapid development and deployment of large scale multimedia applications have been hindered by the lack of available server and network delivery bandwidth (BW). The multimedia files are large in comparison with the requirements of most traditional applications, and hence, require a large space for storage and a large BW for transport [4, 2, 9, 19]. While new networking infrastructures are being deployed at a slow pace to tackle this problem, the delivery BW and storage space will continue to remain critical resources for such applications. The presentation of multimedia objects require a guarantee on the available BW [1, 33, 7, 6]. This makes proper usage of such critical resources even more challenging. Caching of multimedia documents in a local storage can reduce the requirement of retrieval BW. For example, storage of frequently used documents in the client nodes can reduce the requirement of expensive network BW for retrieval from remote servers. Similarly, storage of documents shared by many users in the server memory can reduce disk retrieval BW.

Traditional caching policies employed for non-multimedia applications are not appropriate for multimedia applications. Block-level caching policies (e.g., LRU) that cache unrelated sets of blocks instead of complete multimedia objects can not guarantee continuous delivery of streams. Hence, in a multimedia environment, large multimedia objects (e.g. movie) need to be cached in their entirety. Also, the traditional policies such as LRU and CLOCK [15] are based on the concept of a *hot set* of data which is much smaller in

[1]To appear in Vol. 4, No. 3, 1997 of **Multimedia Tools and Applications** for the special issue on *The State of the Art in Multimedia*.

size than the total set of data. This is also true for the database buffer management policies that exploit the semantics of queries in identifying a hot-set [35, 3, 11]. Generalizing the concept of a hot set in a multimedia environment (i.e. storing the most frequently accessed videos in the cache) may not be very useful since multimedia objects are large. Additionally, detecting changes in the frequency of access to various multimedia objects is difficult; bringing newly hot objects into the cache may require a large amount of time [8].

A number of caching policies have been proposed specifically for multimedia environments [30, 9, 8, 17, 34, 21]. These policies exploit the unique access characteristics to multimedia documents: multiple clients access the same (popular) documents sequentially separated by small time intervals.Hence, the blocks retrieved by one client can be reused by other closely following clients.[2] The proposed policy in [8, 10], referred to as the *Generalized Interval Caching* (GIC) policy, additionally analyzes the stream dependencies explicitly. By associating placement and replacement of data blocks with retrieval streams, the GIC policy ensures continuous delivery of data. It opportunistically caches the shortest intervals between successive streams to maximize the number of streams served from the cache. The GIC policy outperforms the policy of caching the hottest videos in terms of cache hit ratio. More importantly, it adapts to the dynamic changes in workload.

Multimedia applications can be quite diverse in terms of the sizes of the accessed documents. In a movie-on-demand application a single long video may be watched for a long duration (say two hours). In contrast, in a multimedia database system, say for previewing movie trailers, many small video clips may be retrieved in response to client commands. In a general environment, the mix of applications accessing various types of document may vary greatly with time. Additionally, for all applications, access patterns may change rapidly with time (e.g., with the time of the day, or in response to the availability of new videos in the above mentioned applications). The caching policies need to handle the very different characteristics of different applications. Hence, it is necessary for multimedia caching policies to perform well under all workloads containing both large as well as small video objects. The proposed GIC policy in [10] caches effectively both the small intervals of large files as well as entire small files and performs well under all workloads.

Another important requirement of multimedia environment is support of VCR control operations (e.g., PAUSE, RESUME) by clients. VCR control interrupts the sequential reading of multimedia files, and hence, optimizations exploited in caching policies. Note that the problem of dealing with VCR control arises as well in other aspects of multimedia operations (e.g., batching, load balancing). Hence, a general policy is required in a multimedia environment for dealing with VCR control operations and the resulting unpredictable BW requirement. In [7, 12, 6], a general policy of setting aside a portion of the server capacity for dealing with VCR operations is proposed. This can be integrated with the GIC policy as shown later. The access patterns generated by VCR control operations (e.g., skip) can also be exploited by the caching policy [29].

A related aspect of caching is prefetching of blocks to be accessed by a single stream,

[2] An alternative ways to exploit closely following streams in a long video is to group or batch related requests and serve using a single stream [6, 7, 21, 20]. Batching is integrated with the caching policy in [21].

rather than sharing of retrieved blocks across streams. Prefetching of blocks (also referred to as *buffering* in [9]) can mask the variance in response time (e.g., disk response time) and hence, can avoid jitter in presentation. Prefetching is also used to smooth the burstiness introduced by data compression in a single long data stream [36, 18] or composite multimedia documents [25, 40]. The policies in [36, 18] address optimal delivery schedules consisting of piecewise constant rate such that neither the prefetching of data overflows the client buffer, nor does the chosen delivery rate cause buffer underflow. Prefetching for smoothing burstiness in the end-to-end delivery path for retrieving composite documents is addressed in [25, 40]. Finally, prefetching can be employed to avoid jitter in retrieving composite documents stored across multiple devices [38, 40].

The remainder of the paper is organized as follows. In Section 2, we describe in detail the GIC policy and demonstrate its effectiveness under various multimedia workloads. The additional challenges of caching in a distributed server environment are addressed in the subsequent sections. Section 3 introduces the concept of an asset group which is used in conjuction with the affinity routing of requests for improving cache hit ratio in distributed server nodes. If the the cache can be shared by multiple storage devices or servers, the appropriate caching of contents can reduce the load imbalance across devices or servers. In Section 4, we propose extensions to the basic GIC policy for load balancing across storage devices. Section 5 contains a summary of results and conclusions.

## 2 Generalized Interval Caching Policy

The Generalized Interval Caching policy integrates the concept of stream aware cache management for caching intervals in large video objects as well as temporal locality in retaining short video objects. The main idea behind the GIC policy is illustrated in Figure 1 [10]. The small arrow marked $S_{ij}$ shows the playback positions of stream $j$ on video object $i$. The example shows various streams reading large movies 1 and 2, and video clips 3 and 4. In the interval caching policy [8], the data blocks between a pair of consecutive streams $S_{ij}$ and $S_{i(j+1)}$ accessing the same video object is referred to as an *interval*. The two streams associated with an interval are referred to as the *preceding* and the *following* streams. By caching a running interval, the following stream can be served from the cache using the blocks brought in by the preceding stream. The size of an interval is estimated as the time difference between the two streams in reading the same block. The number of blocks needed to store a running interval (i.e., interval size time the compression rate) is defined to be the cache requirement of an interval. To maximize the number of streams served from cache, the interval caching policy orders the intervals in terms of interval size and caches the shortest intervals.

For small video objects that are not being accessed concurrently, the definition of an interval is extended as follows. In Figure 1 arrow $S_{32}$ represents a stream presently reading small video object 3 and arrow $S_{31}$ represents the position that would have been reached by the previous stream accessing object 3 if the object had been larger. $(S_{31}, S_{32})$ is defined to form an interval for objects 3 even though stream $S_{31}$ has already terminated. The interval

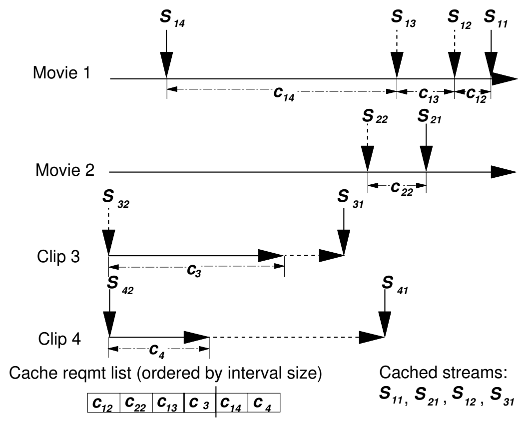



Figure 1: Illustration of GIC policy

size in this case is defined to be the time-interval between two successive accesses on the same object. However, the cache requirement is equal to the size of the object and is smaller than the interval size times the data rate. Therefore, if the interval is selected for caching, the entire video object will be cached.The anticipated interval size for an object can be estimated from the sizes of the previous intervals. In the GIC policy, the size of the last interval is assumed to be the anticipated interval size. The GIC policy thus caches intervals which may be either video segments or entire video objects. The policy maximizes the number of streams served from the cache by ordering all intervals (current or anticipated) in terms of increasing interval size (see, Figure 1) and allocating cache to as many of the intervals as possible. The ability to choose a small set of intervals from a large set of samples allows the policy to exploit the variation in interval size due to the variation in inter-arrival times. The GIC policy also incurs low implementation overhead since changes in the interval list may occur only due to the arrival of a new stream or termination of a current stream. Therefore, algorithms for reordering the interval list and allocation and deallocation of cache to intervals are executed only at the times of arrival and ending of a playback stream.

In the remainder of the section, we discuss further details of implementation of the GIC policy. We next describe a general workload that captures the earlier described complexities (e.g., interactive sessions, movie-on-demand, dynamic load changes) which need to be addressed by a multimedia caching policy. Subsequently, we demonstrate the superior performance of the GIC policy under various workload scenarios.

### 2.1 Implementation Details of the GIC Policy

Recall that under the GIC policy, intervals are created and destroyed only during OPEN and CLOSE operations on files. When an OPEN request is received, it is necessary to decide whether or not to cache the newly formed interval. Upon a CLOSE request, an interval will be destroyed, possibly allowing caching of another interval. The interval may also be destroyed or recreated upon pause and resume (i.e., VCR control operations). The OPEN, CLOSE and VCR control operations are detailed in the following subsections.

#### 2.1.1 Starting of a New Client Request

Figure 2 details the procedure to decide whether or not a new interval (formed by an OPEN request) is to be cached.

**OPEN:**

```
Form new interval with previous stream;
Compute interval size
   and cache requirement;
Reorder interval list;
If not already cached
   If space available,
      Cache this new interval;
   else if this interval is smaller
       than existing cached intervals
     and sufficient cache space
           can be released
      Release cache space from
            larger intervals;
      Cache this new interval;
```

Figure 2: Details of OPEN in generalized interval caching policy

As shown in Figure 2, the GIC policy first computes the interval size of the new interval and its cache requirement. If the request is for a frequently requested small object, all the blocks of the object may already be in the cache, allowing the request to be served from the cache. If the interval is not already cached, it is necessary to determine if it is desirable to cache the interval. The GIC policy attempts to retain the blocks with the lowest interval size, since these are the most frequently referenced object segments. Hence, the policy computes the total cache space available in (less desirable) intervals with a larger interval size and the free pool. If the cache requirement of the new interval is smaller than this space, it can be cached by allocating blocks from the free pool and by deleting the less desirable larger intervals (if needed). Note that deleting an existing cached interval requires switching the stream reading this cached interval to disk.

Caching a real interval implies that the blocks brought in by the current stream will be retained only until they are read and discarded by the concurrent following stream. In contrast, allocating cache to a predicted interval causes the corresponding multimedia object to be retained in the cache for reuse by an anticipated following stream. A flag is associated with an interval to determine whether or not it is a predicted interval.

#### 2.1.2 Ending of a Client Request

**CLOSE:**

```
If following stream of a real interval
  Delete the interval;
  Free allocated cache;
  If next largest interval can
           be cached
     Cache next largest interval;
```

Figure 3: Details of CLOSE in generalized interval caching policy

When a CLOSE request is received, indicating that the client is no longer viewing the current stream, the interval formed by the current stream and its preceding stream no longer exists. Figure 3 describes the subsequent steps executed by the GIC policy. First, the policy deletes the interval in which the client was the following stream and the GIC policy releases the space allocated to the interval. The newly released space may make it possible to cache another interval. Hence, the GIC policy considers the smallest uncached interval to determine if it should be allocated cache. The algorithm used is the same as that during the arrival of a new stream.

#### 2.1.3 Switching of Streams between Disk and Cache

In the previous steps, a stream may need to be switched from disk to cache or vice versa. Switching streams is not a high-overhead operation and can be accomplished using two state flags associated with each stream. The `source` state flag can take on two values – `DISK` or `CACHE` – that indicates whether the stream is being served from disk or from cache, respectively. During block retrieval, the `source` flag is consulted to see whether the next block for the stream should be retrieved from the disk or from the cache. The `block_disposal` state flag can take on two values - `DELETE` or `RETAIN`. Initially, the `block_disposal` flag is set to `DELETE`. This indicates that blocks retrieved for the stream should be deleted after transmission to the client. If it is determined that an interval should be cached, the `block_disposal` flag of the preceding stream of the interval is set to `RETAIN`. This causes the blocks retrieved for the preceding stream to be retained so that they can be used by the following stream. Switching streams between disks and cache can be accomplished by appropriately setting the values of the flags.

#### 2.1.4 VCR control

Video-on-demand systems may allow clients to pause and restart viewing at arbitrary times. This results in change in stream dependencies, which affect the caching decisions taken by the GIC policy. The challenges posed by the VCR control operations and the resulting unpredictable BW requirements are general problems in any multimedia environment, even in the absence of caching. If resources are released during a pause, they may not be available in general at the time of a resume request. On the other hand, not releasing resources leads to inefficient usage of resources. In [6] it is proposed that pause and resume requests be handled by setting aside a small pool of channels (i.e., resources needed for delivering a stream) called *contingency channels*. Under the contingency channel method, upon a pause, resources are released immediately either to the contingency pool or to an waiting request. The resume requests are served using resources from this contingency channel pool. The method provides a *statistical guarantee* that with high probability (e.g. 99%) a resume request can be serviced within a small pre-specified delay. It is shown in [7] that this method is more efficient than reserving resources for all paused streams.

The contingency channel method can easily be integrated with the GIC policy. The pausing of a sole following stream reading from the cache poses no problem other than deleting the cached interval. However, pausing of a preceding stream results in further actions. Its following stream may need to be switched to disk. If both a paused stream and its following stream have been reading from the cache the two intervals can be merged into a single interval, i.e., the blocks previously read by the paused stream continue to be cached. If the GIC policy favors another interval over this merged interval the total cache allocated to the merged interval will be freed and its following stream will be moved to disk. The caching algorithm then reallocates this cache to other interval(s) to be cached. Upon resume, a new stream is needed to serve this request. Sometimes, the resumed stream may lie in the middle of a cached interval, and it will immediately be served from the cache. Otherwise the contingency capacity is used to serve it from disk and the GIC policy is re-run as before to determine if the newly created interval should be cached.

### 2.2 Workload

We now describe the workload used to study the effectiveness of the GIC policy [10]. In a general multimedia environment, two types of applications may be present. In *interactive* applications, short video objects may be displayed in response to frequent client queries. Example of such applications are news-on-demand applications where users may view short video news clips, and shopping applications where users browse short product advertisements before making a purchase. On the other hand, in a *non-interactive* application such as movie-on-demand, client interactions are infrequent and the length of the displayed video objects are typically very long. A general multimedia workload is generated by combining an interactive workload (representing interactive applications) and a non-interactive workload. The arrival process of new client requests is modeled as a Poisson process with a mean inter-arrival time of $T$ seconds. Upon arrival, client sessions are randomly classified as interactive or non-interactive in accordance with an *interactive probability*.

#### 2.2.1 Interactive workload

A general multimedia environment may contain various different interactive applications (e.g. catalog shopping, news-on-demand, educational training). For an interactive application, clients establish a session during which it selects various short clips interactively. It is assumed that the video objects are selected according to an access distribution. The interactive workload is thus defined by the following parameters:

*Video object access skew*: Access to small video objects is assumed to be skewed so that 80% of the accesses are to 20% of the video objects.

*Video object length* distribution: It is assumed that the length of the short video objects is distributed uniformly between $L_{min}$ and $L_{max}$.

*Viewing time* per individual clip: In many interactive applications *video banners* consisting of a short clip are repeatedly displayed (e.g. a logo that is shown until the client makes a selection). This kind of behavior is modeled by assuming that each clip is viewed for a random viewing time $T_v$. Viewing time is assumed to be independent of the length of a clip, and can be smaller than the playback duration of an entire clip, with a minimum of 5 seconds.

*Client session duration*: Client sessions are assumed to be of length $T_s$, with clients repeatedly selecting clips until the length of the session exceeds $T_s$.

#### 2.2.2 Long video workload

Clients executing a movie-on-demand application are assumed to select long video objects according to a Zipf distribution with parameter 0.271 that fits empirical data on video rental frequencies well [43]. It is assumed that the server contains $N_l$ long video objects of length $L_l$.

### 2.3 Evaluation of the GIC Policy

The analysis of the interval caching policy can be related to the general problem of obtaining order statistics, and obtaining explicit expressions may be quite a difficult task [8]. Hence, simulation is used to study the effectiveness of the GIC policy. The server considered is a single server with a single set of striped disks with a pre-specified bandwidth. The change in disk load due to arrival and ending of client requests, or switching between cache and disk are modeled explicitly. However, it is not necessary to model individual block I/Os. Any request for a video that would cause the disk bandwidth to exceed the disk capacity is rejected. The cache is modeled as a collection of 1MB blocks. Hence the cache space required for cachable segments is always rounded up to the nearest megabyte. The required length of the simulation is estimated through trial runs made using the method of batch means. It is found that with a simulation duration of 8 hours, after ignoring the initial transient (1.5 hours), 99% confidence intervals of less than 10% are obtained for important quantities such as the average number of streams reading from the cache. Thus the simulations are run, in general, for 8 hours.

#### 2.3.1 Workload Parameters

| Parameter | Default value |
|---|---:|
| Number of clips | 500 |
| Length of clips | 1 to 30 sec. |
| Average viewing time | 30 sec. |
| Interactive sess. length | 30 min. |
| Number of long videos | 92 |
| Length of long videos | 90 min. |
| Request inter-arrival time | 6.25 sec. |
| Interactive prob. | 0.8 |

Figure 4: Default workload parameters for analysis of GIC policy

Figure 4 shows the default workload parameters chosen for evaluation of the GIC policy. The request inter-arrival time, $T$, is chosen so as to correspond to a server simultaneously serving on an average 400 streams. This value is typical of the range of active clients expected for future video-on-demand servers [16]. Various mixtures of interactive and long video workloads are modeled by varying the *interactive probability*. From the duration of interactive sessions and the length of videos, it can be seen that with the default value, $50\%$ of all I/O requests are made by the interactive sessions. All videos are assumed to be MPEG-1 with a delivery rate of 192 KB/s. Hence, the total amount of storage required per movie is one GB. For interactive applications, the total size of (all video clips in) the video catalog is 125 minutes. The number of video clips was varied in the simulation to study its impact on the server capacity. Experiments varying other parameters are reported in [10].

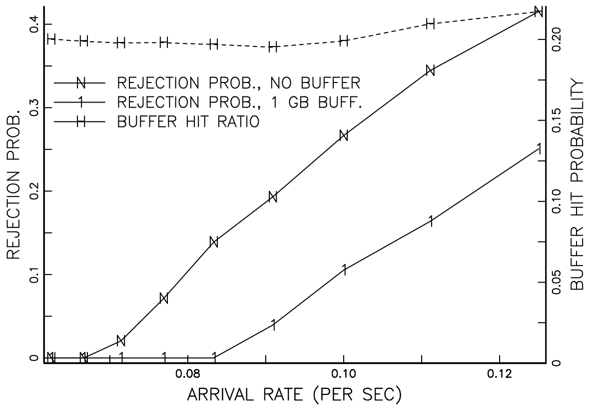
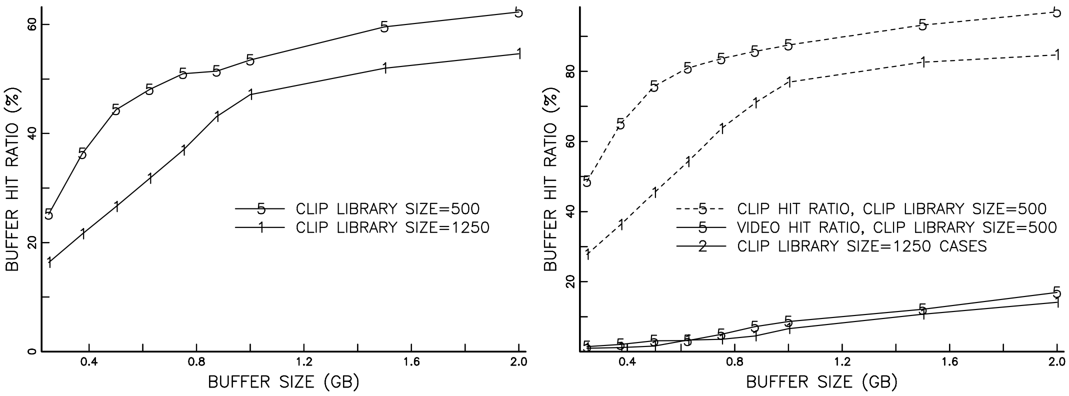



Figure 5: Rejection probability; 400 stream disk capacity

#### 2.3.2 Admission Control

Caching increases the number of streams that can be served by a video server. The server capacity can be defined as the request rate that can be supported with no or a small rejection probability. Figure 5 shows the rejection probability as a function of the arrival rate for a system with a disk capacity of 400 streams. The curve plotted with ‘N’ is for a system without any cache while the curve plotted with ‘1’ is for a system with 1 GB cache. In both cases, the rejection probability is 0 in the region where the system capacity is not exceeded. After this point the rejection rate rises approximately linearly. It can be seen that the 1GB cache has increased the system capacity by 25%, considering the arrival rates where the rejection probabilities become non-zero (.067/sec and .085/sec). The curve plotted with ‘H’ shows the corresponding overall cache hit ratios. The cache hit ratio is defined as the number of reads served from the cache as a fraction of the total number of reads. It can be seen from the graph that the hit ratio (which is approximately constant) is 0.22. Estimation of server capacity as a function of hit ratio translates to a 25% increase, which matches with the earlier estimate considering the non-zero rejection probability points. Hence, only the cache hit ratio is studied in the subsequent experiments.

#### 2.3.3 Cache hit ratios

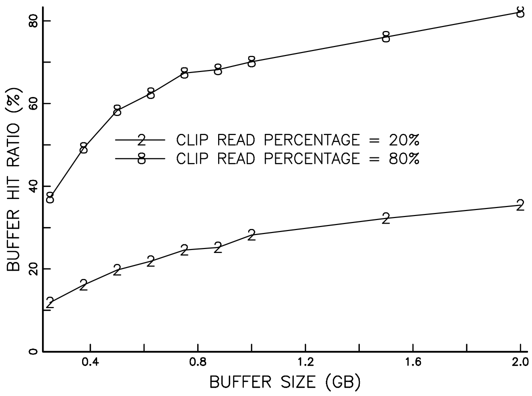



(a) Overall cache hit ratio

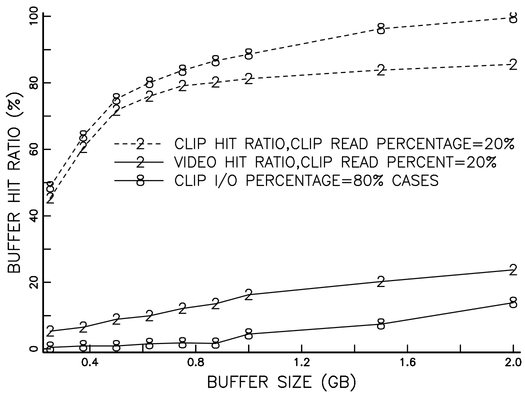
(b) Interactive and movie cache hit ratios

Figure 6: GIC policy under base mixed workload

The GIC policy performs well under a wide range of workload parameters. We first consider the base mixed workload in Figures 6(a) and 6(b). Figure 6(a) shows the variation in the overall cache hit ratio as a function of the cache size. The two curves are for interactive library sizes of 500 clips and 1250 clips. For a given cache size, as expected, the overall hit ratio with 500 clips is higher than that with 1250 clips since with a smaller number of clips, a larger fraction of the accesses falls on the clips retained in the cache. The hit ratios of the individual components of the workload are shown in Figure 6(b). The clip hit ratio shown in the figure is defined as the fraction of clip reads satisfied from the cache. Similarly, the movie hit ratio is the fraction of movie reads serviced from the cache. The clip hit ratio is higher, since access to the short videos are more likely to be found in the cache, and only the shortest intervals of long videos are satisfied from the cache. However, both clip and movie hit ratios are smaller for the workload with 1250 clips.




(a) Overall cache hit ratio




(b) Interactive and movie cache hit ratios

Figure 7: Effect of varying interactive fraction

We next consider the impact of varying the workload mix on the GIC policy. Figure 7(a) shows the variation of the overall cache hit ratio with the cache size for different fractions of short video clips. The interactive library size contains 500 clips as in the base workload. However, the workloads differ in the fraction of accesses to the short clips, 20% and 80%, respectively. The workload with a higher fraction of accesses (80%) to interactive clips has a much larger cache hit ratio, since by caching the hot clips in a small amount of cache, a larger fraction of the accesses can be served from the cache. Figure 7(b) shows the interactive clips hit ratios and the movie hit ratio separately for the above values of access to interactive clips. The dotted lines are for the interactive clips while the solid lines are for the movies. As before, in both cases the clip hit ratio is higher than the movie hit ratio. However, the clip hit ratio for the 80% read case is higher than that for the 20% read case while the relationship is reversed for the movie hit ratios. This is because with a lower clip access frequency a smaller number of hot clips are retained in the cache. This results in a lower clip cache hit ratio and a higher movie cache hit ratio.

Figure 8 plots the cache hit ratio against the cache size for two different values of average clip size (8 seconds and 30 seconds). The cache hit ratio with an average clip size of 8 seconds is higher than that for an average clip size of 30 seconds, since less cache is required for smaller clip size. The difference in cache hit ratios is most marked at smaller cache sizes. For larger cache sizes, hot clips can be retained in both cases.

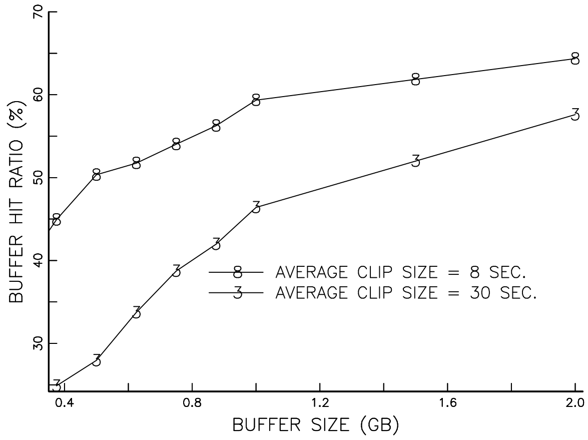



Figure 8: Effect of varying average clip size on overall cache hit ratio

### 3 Affinity Routing for Multimedia Applications

In the earlier section, we demonstrated that the GIC policy is effective in coping with heterogeneous workloads, however, in a single server environment. In a distributed server environment, a caching policy may need to address additional concerns, such as, routing of requests and load balancing across server nodes for efficient system operation. In a large scale single server system, caching policy exploits economies of scale [22], since cache reuse is more likely with a larger number of concurrent requests. However, such large scale systems may be built with a large number of nodes. A typical distributed server (see Figure 9) consists of a set of front-end nodes that retrieve video data from shared disks or storage servers [41, 31, 28]. The cache is partitioned across the server nodes, e.g., front-end nodes and possibly storage server nodes. If different requests are served by different nodes without any coordination with the caching policy, the cache hit ratio will be quite low. The principle of caching is reuse, and therefore, if related requests are not served by the same nodes, reuse will be low. This coordination in serving of related requests is referred to as *affinity routing*.

In this section, we will describe in detail the operations of a distributed multimedia server, and subsequently, how the GIC policy can be adapted to such an environment. The extension embraces two new concepts: i) *path based* routing, i.e., routing based on the availability of a complete path up to the front-end nodes which may include cache, and ii) *asset group* i.e., logical grouping of small related files that may be accessed from the same client session. We subsequently demonstrate the effectiveness of this policy via simulations.

#### 3.1 Issues in Affinity Routing

The *server controller* in Figure 9 receives client requests for setting up of an application session and selects the server to which the request is to be routed [13, 31, 28, 22]. To guarantee continuous delivery, the server controller reserves bandwidth on all the components needed to deliver a video (referred to as a *logical channel* or *path*). Hence, the server controller is also responsible for admission control. It keeps track of available resources (e.g. disk bandwidth, CPU) on each server. Application specific client requests and additional processing (e.g., catalog search) are handled by the application server [14]. After establishment of an application session, all subsequent client requests are routed directly to the selected application server. For an interactive application, it may be necessary to switch the session to another node with available bandwidth if it is not possible to serve a future requests in the current node. For example, resource requirements may change as future video clips may not be present in the cache, and no additional resources may be available in that node. Also, during a long running session, the capacity of a node may change due to failure of one or more components. The switching is done in coordination with the server controller.

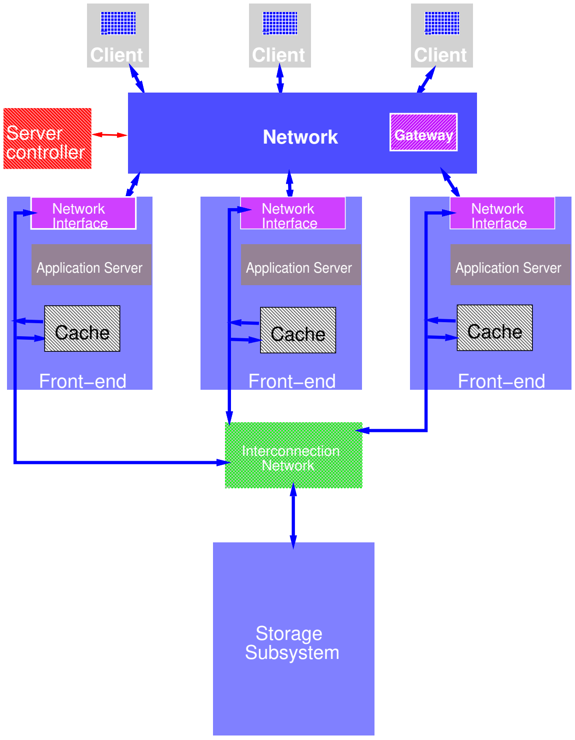



Figure 9: Distributed Server Environment

**Path based routing:** The bottleneck in such distributed systems may be in the storage subsystem, the interconnection network or the front-end nodes. Front-end node caching can improve the performance of the system by reducing the needed retrieval bandwidth from the storage subsystem. However, if the server controller routes requests at random to front-ends, front-end caching will not be effective since the resulting cache hit ratio will be very low. Hence, it is necessary for the server controller to route application requests taking into account the access affinity of front-end nodes. Affinity routing has been proposed in various environments (e.g., file system [39], database [42]) for improving the cache hit ratio at the front-end nodes of a distributed server. As mentioned earlier, in a multimedia environment, setting up a logical channel requires reserving bandwidth on the entire path to the client (e.g., front-end, network interface). Therefore, unlike traditional applications, it is not enough to consider only the access affinity for improving cache hit or balancing of front-end node utilization across nodes. The server controller additionally has to take into account the availability of all resources needed to form a logical channel. In this paper, we propose a *path based affinity* routing policy that takes into account the bottleneck resources when routing requests.

**Asset group:** For access to long videos (e.g., movies) affinity can be based on the identity of videos. However, affinity routing based on the identity of the clip during interactive sessions will lead to frequent switching of front-end nodes. This is undesirable since setting up a logical channel across network and server nodes may introduce a large latency, and incurs processing overheads. Also, as in transaction processing, the application server builds up state of the client (i.e., items purchased so far in browsing through a shopping mall), and switching front-end nodes would require distributed application processing. In many applications, only a subset of the video clips on the server are accessed during a session (e.g., shopping clips of a particular store). Related short clips which are most likely to be accessed together during a single session can be logically grouped into an *asset group*. Routing of application sessions based on the affinity to an asset group will avoid undesirable switching. The incoming application specifies the likely asset group(s) it would access.

The incoming application will be routed to a node as follows:

1. First consider only the nodes that have one or more applications accessing the same asset group(s). Order the nodes in terms of affinity, i.e., the number of application sessions accessing the same asset group(s). Select a node in order that has enough available resources to set up a logical channel, where the source could be the disks or the cache.[3]

2. Otherwise, consider only those nodes that retain in their cache some of the clips of the asset groups(s) associated with the current application. Note that no active application sessions accessing the requested asset group(s) are being served by these nodes. Select any node with sufficient resources for setting up a logical channel.

3. Otherwise use any node for setting up a new channel that will reduce imbalance in available capacity.

4. If no bandwidth is available in any of the node reject this request.

For an interactive application, if a future request can not be satisfied by the current node, the above algorithm is reexecuted to select a new node for the application. An interactive application may have to wait if no bandwidth is currently available in any of the node. Alternatively, contingency capacity (as described in the earlier section for coping with VCR control operations) may be used. In our simulation, we will count only the sessions that are served without any such delay to estimate the effective server capacity.

[3]It is assumed in our experiments that the identity of the video to be accessed is known only for long videos. The identity of the clips to be accessed are not known in advance for an interactive session.

### 3.2 Simulation Parameters

| Parameter | Default value |
|---|---|
| Number of clips | 560 |
| Length of clips | 1 to 60 sec. |
| Interactive sess. length | 30 min. |
| Number of asset groups | 8 |
| Number of long videos | 100 |
| Length of long videos | 30 min. |
| Request inter-arrival time | 1.4 sec. |

(a) Simulation parameters

| Resource | Total capacity |
|---|---|
| Disk B/W | 1024 streams |
| Cache Size | 288 MB |
| Network capacity | 1280 streams |

(b) Server capacities

Figure 10: Default values

We next study via simulation the effectiveness of the proposed affinity routing policy. Figure 10(a) shows the default values of the workload parameters as described in Section 2. The clips are assumed to be distributed evenly over the asset groups, i.e., each asset group contains both hot and cold clips. The effective server capacity and the cache hit ratio are used to measure system performance. These estimates are derived from sample statistics gathered every 5 minutes of the number of active streams, the number of streams being served from the cache, and the number of sessions completed successfully. The cache hit ratio is measured as the ratio of the number of I/Os being served from the cache to to the total number of I/Os. Before gathering performance estimates, the experiment is run for 40 minutes to allow the simulated system to stabilize. Subsequently, the statistics are collected for 30 minutes.

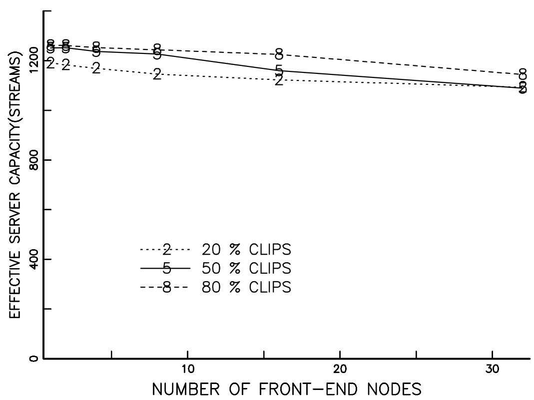
### 3.3 System Configurations

The performance of the affinity routing policy is studied by comparing the cache hit ratio and throughput of a distributed server using the policy against that of an idealized large server. It is assumed that in the idealized server the resources needed to set up a logical channel: network interface, front-end node, cache, disk B/W are well matched. The capacities of the idealized server are shown in Figure 10(b). The cache size of 288 MB is sufficient to store a 30 minute 1.2 Mb/s video. The network capacity of 1280 streams can match a cache hit of at least 20 %.

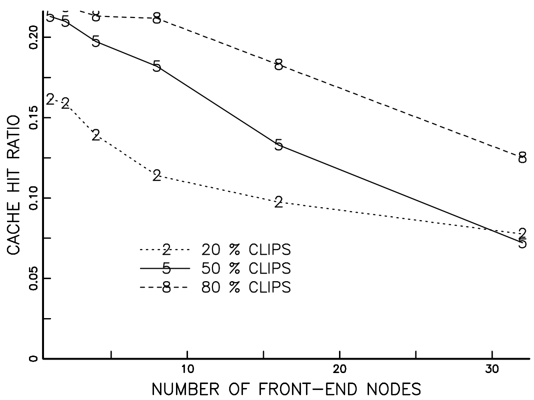
Two different types of distributed server configurations are studied to demonstrate the robustness of the policy. Under a *symmetric configuration*, the total system capacity is divided amongst the nodes equally. Note however, that due to the unpredictable cache hit ratio, the required front-end node capacity and network interface capacity cannot be predicted in advance. Hence it is impossible to design a distributed server configuration where all the resource capacities are matched, even if all the resources are divided equally. The experimental results assume that only the network capacity is equally divided amongst all the nodes, with other parameters being varied in order to study their impact.

The distributed server may also be built using many heterogeneous nodes. In our experiments, by dividing the resources unequally amongst the nodes, a large number of *asymmetric configurations* can be generated. In the following, to limit the scope of the experiments, only a specific type of asymmetric configuration is considered. It is assumed that the system consists of two types of nodes, and the capacity of the larger nodes are three times those of the smaller nodes. The results can easily be generalized for other configurations.

### 3.4 Affinity Routing without Constraint on Front-end Node Capacity




(a) Server capacity

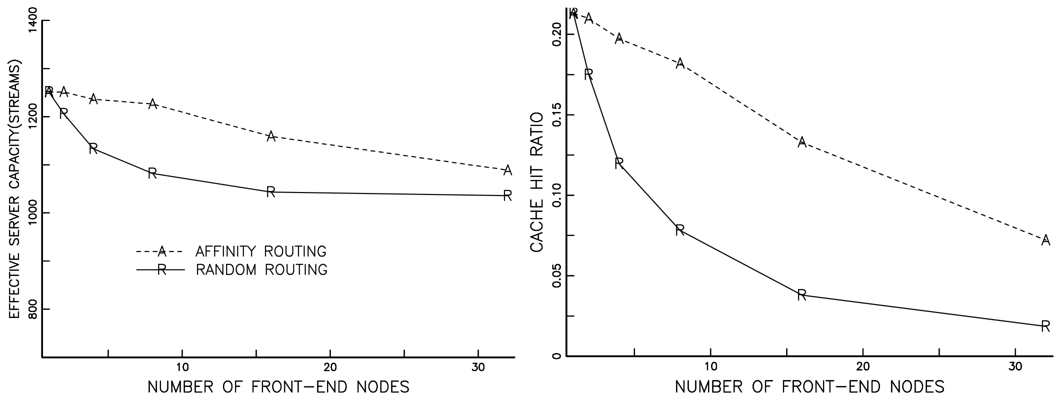



(b) Cache hit ratio

Figure 11: Affinity routing in a symmetric configuration; No front-end node bottleneck

We first assume that the capacity (e.g., CPU) of the front-end node is not a bottleneck resource. However, disk BW and the network interface (i.e., adapter) BW could be the bottleneck resources. Hence, the routing is based on the available network interface capacity, whether or not the stream can be served from the cache of its associated affinity nodes and the total available disk BW in the system. Figure 11(a) shows the performance of the affinity routing policy as the number of nodes in the server is varied. Three different workloads with different mix of interactive and long video applications are studied. The server capacity is measured as the number of applications sessions served without any delay. As can be seen from the graphs, under all three workloads affinity routing is quite effective for a small number of nodes, and even for a large number of nodes, (i.e., 32) a reasonable cache hit is obtained (see, Figure 11(b)). Affinity routing tries to avoid replication of cache entries (particularly of short clips) [39, 42]. Affinity routing also helps to create small intervals in long videos, and hence, improves the cache hit probabilities for long videos. If the workload is dominated by access to long videos (20% clip access case), the fragmentation of B/W implies hottest long videos are served by multiple nodes, and hence, cache hit probability degrades. However, since the network interface BW is the bottleneck resource, the change in cache hit has negligible effect on the effective server capacity (see, Figure 11(a)).

Figure 12(a) compares the server capacity under affinity and random routing of session[4] for an workload consisting of 50% clip access. For interactive workload, to avoid switching overhead, once the initial session is established, all subsequent clips are accessed through the same front-end node. Switching occurs only when the stream in a session was served from the cache, subsequent access to a clip can not be served from the same cache and no capacity is available in the current node. Note that a truly random routing (i.e., without any such concept of an asset group) will cause the cache hit to degrade further. In any case, the cache hit is significantly lower under a random routing policy (see Figure 12(b)). For a small number of nodes the affinity routing performs significantly better than the random routing policy. The performance improvement of an affinity routing policy over the random routing policy is shown in Figure 13. The improvement in server capacity is as high as 14 % in this example. For a large number of nodes it is difficult to concentrate the accesses to the same videos in the same nodes. As noted earlier, only after a substantial degradation in cache hit the disk BW becomes a bottleneck resource.




(a) System capacity

(b) Cache hit ratio

Figure 12: Affinity vs random routing in a symmetric configuration; No front-end node bottleneck

[4] Random routing is used only for initial session establishment. Subsequent requests within the same session still go to the same node, i.e., is based on asset group.

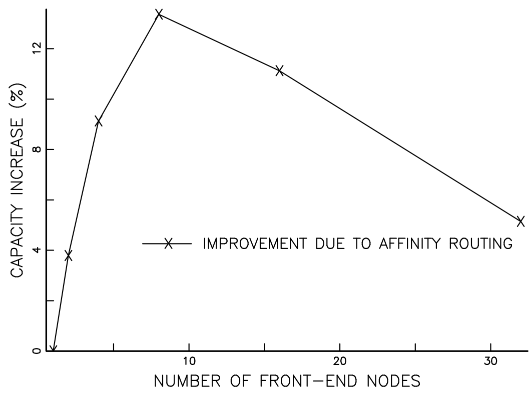



Figure 13: Improvement in system capacity: affinity vs random routing

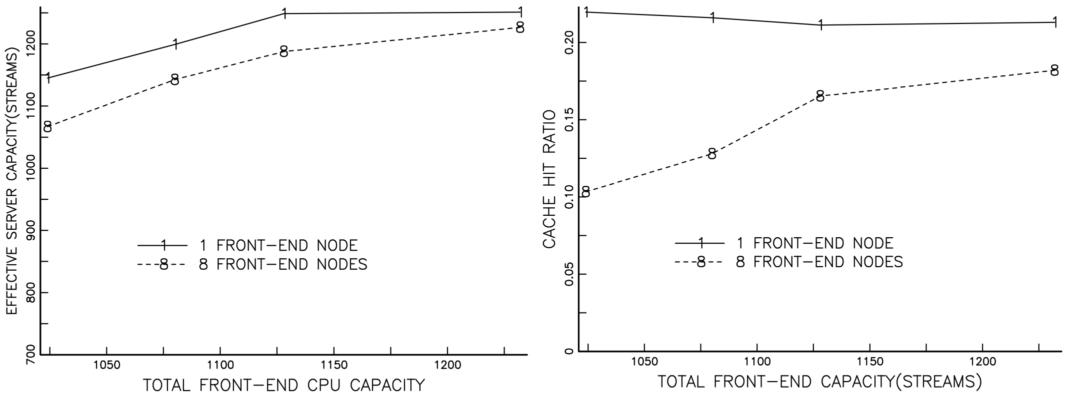
### 3.5 Mismatch in front-end node Capacity

Matching front-end node capacity to other resources is a difficult task in any configuration. First, cache hit is truly unpredictable. The demand on front-end node capacity comes from various sources. The application server has its own front-end node requirement (separate from the task of delivery of data). Serving a stream from disks requires front-end node mips for both I/O operations and delivery of data over the network. In contrast, serving of data from cache requires front-end node mips only for delivery of data over the network. We will assume that the front-end node mips requirement is for serving a stream from cache is half of that of serving it from a disk. In a video database environment, the admission control policy makes sure that all the resources are available before a session is accepted [33, 1, 6, 7]. The front-end node scheduling policy schedules front-end node among various tasks (e.g., stream data delivery, application) so as provide smooth delivery of data to the clients [32].

Depending on the cost of various system resources, front-end node can be over or under configured. In Figure 14(a) the front-end node capacity per front-end node is being varied. As can be seen the overall server capacity improves with the increase in server capacity. Increasing capacity per node (of all resources) allows more similar applications to be routed to the same nodes resulting in higher cache hit probability (see Figure 14(b). However, beyond a certain point front-end node is over configured and other resources (network interface capacity) becomes a bottleneck. This clearly demonstrates the challenge in matching various system resources. A routing policy has to be aware of this mismatch in resources.

### 3.6 Asymmetric Configurations

Asymmetric configuration implies cache also need to be divided across nodes in asymmetric manner. However, the cache hit is rarely proportional to the size of the cache. This creates further challenges in matching various resources. However, asymmetric configuration provides an opportunity for concentrating a larger number of requests to its larger nodes




(a) System capacity

(b) Cache hit ratio

Figure 14: Mismatch in network interface and front-end node capacity; Symmetric configuration

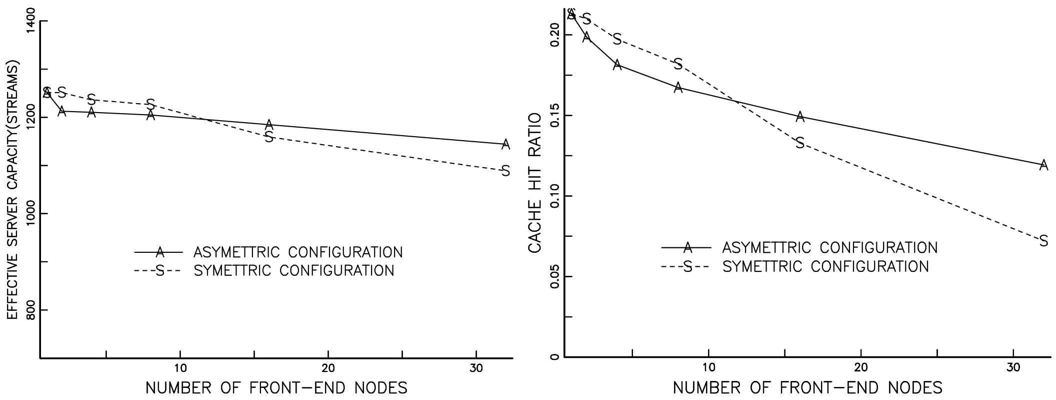
resulting in higher overall cache hit probability.

As in the earlier case, we will first explore system configurations where the front-end node is not a bottleneck resource. The network interface capacity is divided unequally, i.e., the capacity of a larger node is three times that of a smaller node. Figures 15(a) and 15(b) show the capacity and cache hit for such a configuration. As mentioned earlier, the larger node size provide a greater opportunity for assigning the hot videos to a smaller set of nodes, and hence, a greater cache hit and server capacity (see 32 node case). However, for a smaller number of nodes, this makes little difference in over all cache hit or server capacity.

We next consider a case where front-end node can be a bottleneck resource. Figure 16(a) compares the overall server capacity as the front-end node capacity is increased in both larger and smaller nodes. Since the network interface capacity and the cache hits are different in different nodes the increase in front-end node by the same percentage in all nodes may not be the best configuration. Therefore, we show two other cases marked by 'L' and 'S', where only the front-end node capacity of the larger and smaller nodes are varied, respectively. The front-end node capacity of the other type of nodes (smaller and larger, respectively) are kept constant to a value that approximately matches the cache hit ratio. Figure 16(b) shows the corresponding cache hit ratios.

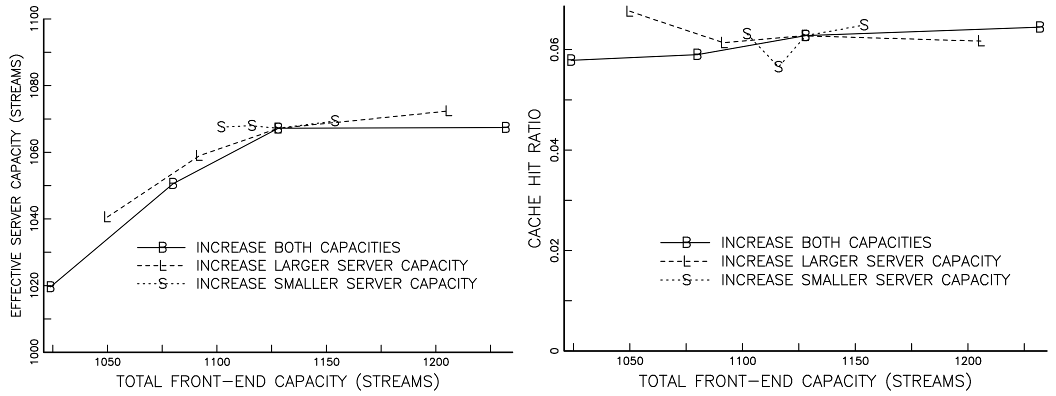
### 3.7 Variation in Workload

Cache hit depends very much on workload partitionability as well as the mix of applications. To show the robustness of the proposed affinity routing policy, we compare the server performance (effective capacity and cache hit, respectively) for three different workload in



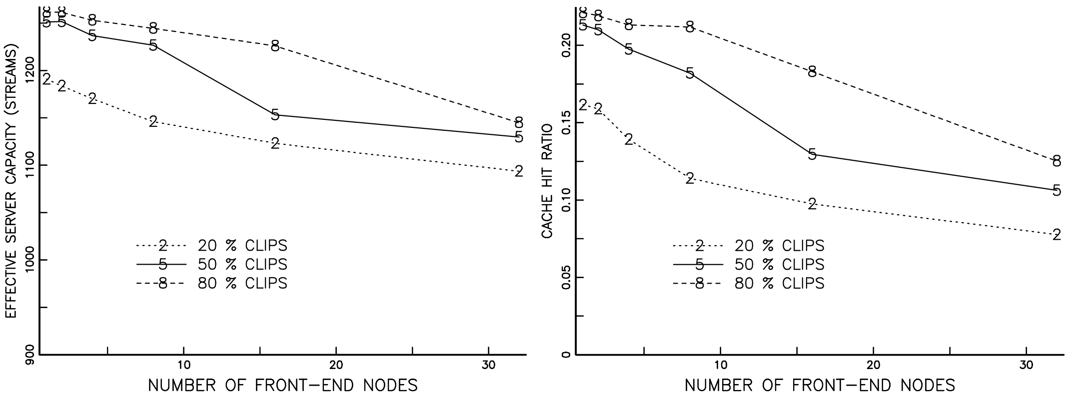

(a) System capacity (b) Cache hit ratio

Figure 15: Affinity routing in an asymmetric configuration; No front-end node bottleneck




(a) System capacity (b) Cache hit ratio

Figure 16: Mismatch in network interface and front-end node capacity; Asymmetric configuration




(a) System capacity

(b) Cache hit ratio

Figure 17: Interactive vs long video workload

Figures 17(a) and 17(b). The degradation is small with the increase in the number of nodes in all three cases.

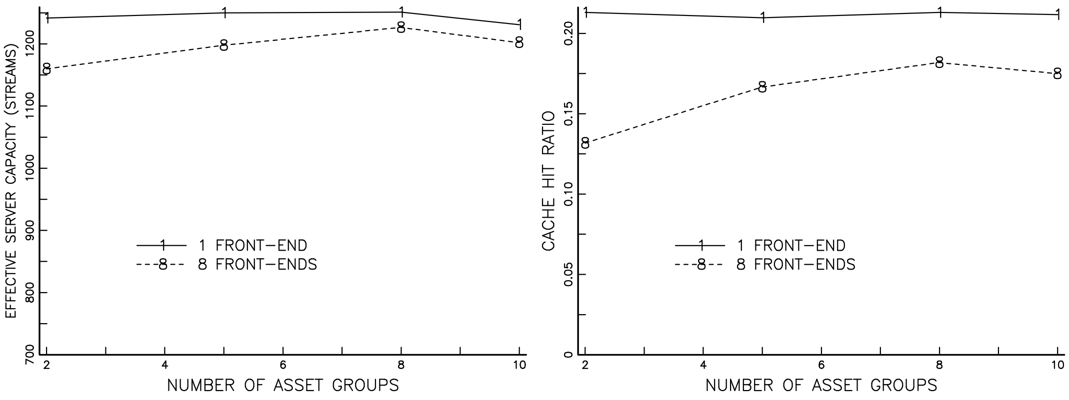
Figures 18(a) and 18(b) show the effective server capacity and cache hit as the clips are divided across varied number of asset groups. Increasing the number of asset groups, allows the applications to be routed evenly (or to better match the capacity of nodes in an asymmetric configuration) across all nodes, particularly for a large number of nodes. For a smaller number of asset groups, an asset group is cached in multiple nodes resulting replication of cached clips and hence, lower cache hit probability.

## 4 Caching for Load Balancing Across Servers

In the previous section, we proposed extensions to the GIC policy for dealing with routing of related requests to various front-end nodes. Another important issue a caching policy needs to address in a distributed environment is the fragmentation of BW of storage devices. Load imbalance across storage devices may occur depending upon the file placement and current popularity of various files. If multiple storage devices share a common cache, the load imbalance can be reduced by properly managing the content of the cache. In this section, we describe how the GIC policy can be adapted for this purpose.

### 4.1 Issues is Large Scale Storage Server

A distributed multimedia server is likely to contain multiple storage groups or subsystems [5]. Aggregating multiple physical devices into a single logical device, and striping multimedia files over the logical device, is a widely used technique for balancing the load over




(a) System capacity

(b) Cache hit ratio

Figure 18: Workload partionability into asset groups

the individual physical devices [27]. However, in practical environments, it may not be possible to combine all the physical devices in the system into a single storage group [5]. Real systems are likely to contain heterogeneous devices (e.g. disks of different types) and it is difficult to stripe over devices of different types due to mismatch in capacities (storage and BW). Hence, it is natural to have multiple striping groups, each containing devices of a single type. Some multimedia objects may be small, and thus be difficult to stripe over a very large number of devices. It has also been shown that in a system with multiple striping groups, high availability can be achieved by replicating a small number of popular videos [5].

The access rates to the videos stored in different striping groups may not be uniform. This will result in load imbalance across different storage devices and hence, in inefficient usage. Requests for videos on storage devices that are loaded to their capacity will be rejected even though sufficient bandwidth may be available on other storage devices. Caching can be used to balance the load on the storage devices, thereby increasing the effective throughput of the system. Load balancing for improving server throughput and response time has been studied in various environments. In the file server context, static replication has been proposed as a way to achieve both high availability and load balancing [24, 39]. In multimedia systems, such policies will incur a higher storage overhead since multimedia files are relatively large. However, replicating only frequently used files [37, 39] can reduce this storage overhead where advance knowledge of access patterns of the applications is available [26, 4]. For efficient usage of device capacities, the static placement of files should take into consideration both the BW and storage space utilizations [4]. Static placement alone cannot cope with sudden load surges, and a dynamic policy for dealing with sudden fluctuations in load is still needed. Placement and dynamic load balancing are complementary approaches since good initial placement makes the job of load balancing easier.

Load balancing by caching (i.e., dynamically replicating) of video segments in disk has been proposed in [5]. In this paper, we consider a complementary policy of caching in memory. First, we propose a modification to the GIC policy that balances the load among multiple storage devices. This is referred to as the *Caching for Load Balancing* (CLB) policy. Subsequently, the CLB policy is shown to be effective in reducing load imbalance across servers, and hence, to increase the effective throughput of the overall system.

### 4.2 Overview of the CLB Policy

As described earlier, the GIC policy attempts to maximize the number of streams served from the cache by retaining the smallest intervals. In a single storage server environment, the total system capacity expressed in the number of streams is increased by serving additional streams from the cache. In a distributed system with multiple storage devices simply maximizing the number of streams served from the cache is not adequate. Caching streams from the lightly loaded devices takes away valuable cache space that could be used for further reduction of load on the bottleneck devices. Therefore, the GIC policy needs to take into account the load on the different storage devices and caches only the streams from highly loaded devices. In the paper, we consider a disk to be heavily loaded if its load is greater than the average load over all the disks.

Under an overload condition where rejection of requests is unavoidable, the objective of caching policy needs to be changed. It may be preferable to minimize the rejection instead of balancing the load (and hence, balancing the rejection) across storage devices. Rejection is minimized by caching only the streams from overloaded storage devices using the basic GIC policy. Since the basic GIC policy maximizes the number of streams served from cache of the overloaded devices, the number of rejected requests is also minimized. The overall algorithm is bimodal. In the normal state, it balances the load, and under overload, it minimizes rejection. The overload condition is detected whenever a request is to be rejected. Alternatively, in the absence of any rejection, the algorithm switches to the normal state.

### 4.3 Simulation Parameters

The performance of the proposed caching load-balancing policy is studied via simulation. Default parameters for the workload are shown in Figure (b)(a). Each logical disk is assumed to contain one asset group with short clips. The access rates to the asset groups is non-uniform and is given by a Zipf distribution with the parameter 0.271. The access frequency to clips within each asset group is skewed (80-20). The request inter-arrival time is set to 3.6 seconds representing a server with approximately 500 users. The other default values are the same as those used in Section 3. The effective server capacity, average disk load, the number of cached streams of each disk and load imbalance between the disks are measured. The load imbalance between the disks is defined as follows. Let $d_M$ be the average load of the most highly loaded disk, and $d_m$ be the average load of the least highly

| Parameter | Default value |
|---|---|
| Number of clips | 560 |
| Length of clips | 1 to 60 sec. |
| Interactive sess. length | 30 min. |
| Number of asset groups | Number of disks |
| Number of long videos | 100 |
| Length of long videos | 30 min. |
| Request inter-arrival time | 3.6 sec. |

(a) Simulation parameters

| Resource | Total capacity |
|---|---|
| Disk B/W | 512 streams |
| Cache Size | 288 MB |

(b) Server capacities

Figure 19: Default parameters

loaded storage device. The load imbalance (expressed as a percentage) is then defined as $100(d_M - d_m)/d_m$. These measures are derived from 5-minute sample statistics as in the earlier sections.

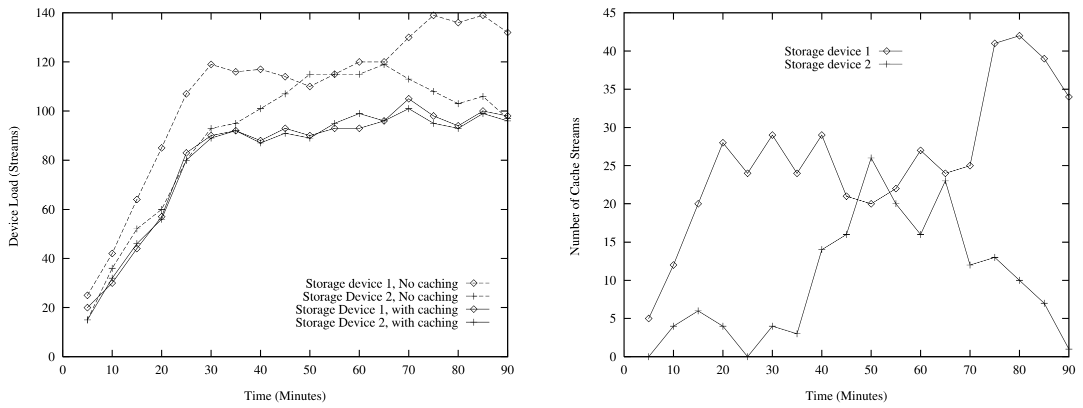
### 4.4 System Configurations

A distributed system with multiple storage devices with a single shared cache is assumed for studying the effectiveness of the CLB policy. The total bandwidth of 512 streams is divided equally amongst the storage devices. The number of storage devices (logical striping group) is varied from 2 to 6 in various experiments. Initial placement of files attempts to balance the expected load by allocating the long videos in a round-robin among the storage devices. The asset groups are allocated in a similar round-robin manner. However, to create a controlled load imbalance the most highly accessed asset groups and the most highly accessed long videos are placed on the same storage device. The results can be generalized to other placements.

In the following sections, we first study the impact of the CLB policy in terms of its ability to balance the load among the storage devices in the system. Subsequently, we consider the improvement in effective throughput produced by the policy.

### 4.5 Impact of cache load-balancing - base workload

We first consider the impact of cache based load-balancing in a system with two storage devices. Figure 20(a) shows the storage device load in terms of the average number of streams reading from storage device as a function of time. The dashed lines represent the




(a) Disk load vs time with and without caching

(b) Average number of cache streams vs time

Figure 20: Impact of caching

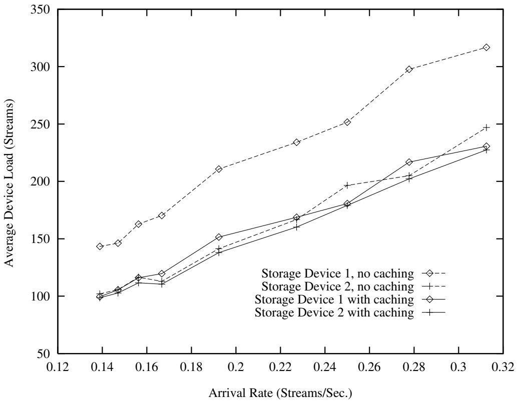
storage device load without caching; the solid lines show the storage device load when caching is used to balance the load. Examination of the dashed lines shows that without caching, there is a load imbalance among the storage devices with storage device 1 being more heavily loaded than storage device 2. Under the CLB policy, it can be seen that the load imbalance is much less, i.e., the solid lines are much closer together. The CLB policy caches a relatively larger number of streams from storage device 1 than from storage device 2. Figure 20(b) shows the average number of cached streams per storage device.

#### 4.6 Performance of the CLB Policy under Varying Workload

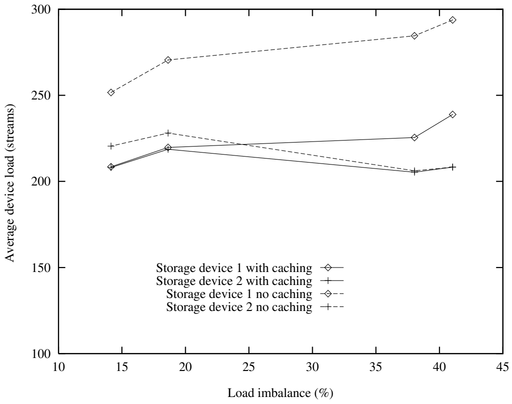
We now show that the proposed CLB policy is robust with respect to variation in workload parameters. Figure 21(a) shows that the CLB policy works well under a wide range of arrival rates. The x-axis represents the arrival rate while the y-axis represents the average load on each storage device. The arrival rate is varied from a value of low load to a value where rejections just begin to occur. As in the earlier case, the dotted and solid lines represent the average load on the storage devices expressed in streams with and without caching, respectively. The solid lines are closer to each other demonstrating good load balancing under the CLB policy.

Figure 21(b) shows how the CLB policy copes with varying amounts of load imbalance between the storage devices. The load imbalance is varied by changing the parameter of the Zipf distribution used for generating the access frequencies. Here, the x-axis represents the load imbalance between the storage devices while the y-axis represents the average load per storage device. The CLB policy reduces substantially the load imbalance between the storage devices for a wide range of workload imbalances; with the imbalance being eliminated completely at lower values.

The higher imbalance in load can be reduced significantly only with a larger amount of




(a) Average storage device load vs arrival rate with and without caching



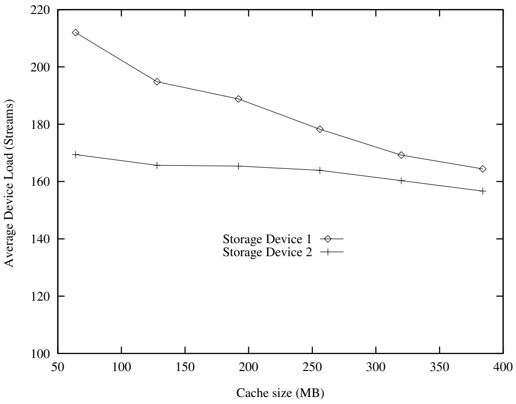

(b) Average storage device load with varying load imbalance under caching

Figure 21:

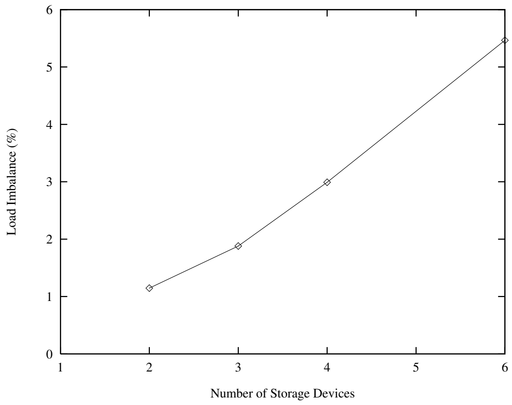
cache. Figure 22(a) shows the load imbalance as a function of cache size for a workload with 40% initial imbalance (the same as the extreme point in Figure 21(b)). As before, the closer solid lines imply less imbalance. The load imbalance tends to increase with a higher number of storage devices, as the bandwidth gets severely fragmented. Figure 22(b) plots the load imbalance against the number of storage devices in the system. For each configuration of the storage device, the Zipf parameter is chosen such that the initial load imbalance between the heavily and lightly loaded devices is 18% (See earlier definition of load imbalance). It can be seen that in all cases, the load imbalance under the CLB policy is quite low ($< 6\%$).

### 4.7 Increase in Effective Throughput

The load balancing results in an increase in effective throughput. Figure 23 compares the rejection ratios for systems with and without any cache. The rejection ratio is expressed as a percentage and plotted against the arrival rate. It can be seen that both with and without caching, the rejection rate is initially zero and beyond a certain point (referred to as break-point) begins to rise, indicating overload on the bottleneck device. The rejection ratio is lower with caching than without caching. The break-point occurs for a lower arrival rate without caching. The system capacity is typically defined as the arrival rate that can be supported with a pre-specified (say 1%) rejection ratio. It can be seen that caching has increased the capacity of the system by approximately 20% for zero rejection tolerance.




(a) Average storage device load vs cache size




(b) Load imbalance vs number of storage devices; 18% imbalance in workload

Figure 22: Effectiveness of CLB policy under varying workload and configuration

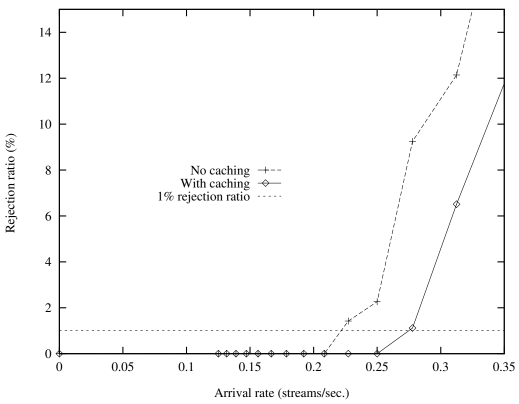



Figure 23: Rejection ratio vs arrival rate with and without caching

## 5 Conclusions

Storage and retrieval of large multimedia documents demand large storage space as well as large bandwidth for real-time delivery. As a result, storage and bandwidth are critical resources in a large scale multimedia system. The requirement in retrieval bandwidth can be reduced by caching multimedia documents in local storage (memory). Multimedia applications can also be quite diverse; examples include movie-on-demand applications where long videos are played back with little user interaction, as well as multimedia database applications where relatively short clips are displayed in response to frequent user requests. Hence, the workloads differ greatly in terms of the requirement in file sizes, number of videos and other parameters. Additionally, application requirements may vary greatly with time. VCR control operations (e.g. PAUSE and RESUME) that change the sequential access to the files poses further challenges to a caching policy.

In this paper, we propose the *Generalized Interval Caching* policy for handling diverse workloads. The GIC policy caches small intervals in large videos as well as entire small objects so as to maximize the cache hit ratio. It automatically adapts to changes in workload. Additionally, the GIC policy can be integrated with the policy of setting aside contingency capacity for supporting VCR control operations. To study the effectiveness of the GIC policy, we proposed a multimedia workload that captures the requirements of diverse applications. The GIC policy is shown to be effective in such an environment using simulation.

In a distributed server environment containing many front-end nodes and storage devices, the cache is partitioned across nodes. The GIC policy can be used to manage the cache within each node. However, affinity routing of related requests to the same server is needed to achieve an effective cache hit ratio. We propose an affinity routing policy based on two new concepts i) *path based routing* which takes into account the availability of bandwidth on each server component needed to deliver a video and ii) grouping of related small files (e.g., clips needed by an application) into *asset groups*. The effectiveness of the proposed affinity routing policy is next demonstrated using simulation.

A distributed server may also contain multiple storage devices or servers. Depending on the placement of multimedia files across the storage devices and non-uniform access to them, load imbalance may develop among the storage devices. In such environments, caching can reduce load imbalance, thus increasing the effective capacity of the system. We further propose an extension of the GIC policy, called the *Caching for Load Balancing* policy, that preferentially caches the videos on the heavily loaded disks to balance the load. The CLB policy is also shown to be effective over a wide range of workloads using simulation.

## References

[1] Anderson, D. P., “Metascheduling for Continuous Media,” *ACM Transactions on Computer Systems*, Vol. 11, No. 3, August 1993, pp. 226–252.

[2] Chaudhuri, S., S. Ghandeharizadeh, and C. Shahabi, "Avoiding Retrieval Contention for Composite Multimedia Objects", *Research Report 95-618*, University of Southern California, 1995.

[3] Chou, H. T., and D. J. Dewitt, "An Evaluation of Buffer Management Strategies for Relational Database Systems," *Proc. 11th VLDB Conf.*, Stockholm, Sweden, 1985.

[4] Dan, A., and D. Sitaram, "An Online Video Placement Policy based on Bandwidth to Space Ratio (BSR)", *Proc. ACM SIGMOD*, San Jose, CA, May 1995, pp. 376–385.

[5] Dan, A., M. Kienzle and D. Sitaram, "Dynamic Policy of Segment Replication for Load-Balancing in Video-on-Demand Servers", *ACM Multimedia Systems*, Vol. 3, No. 3, July 1995, pp. 93–103.

[6] Dan, A., D. Sitaram, and P. Shahabuddin, "Dynamic Batching Policies for an On-Demand Video Server," *ACM Multimedia Systems*, Vol 4, No. 3, pp. 112–121, June 1996.

[7] Dan, A., P. Shahabuddin, D. Sitaram, and D. Towsley, "Channel Allocation under Batching and VCR Control in Video-On-Demand Servers", *Journal of Parallel and Distributed Computing*, Vol 30, No. 2, pp. 168–179, Nov 1995.

[8] Dan, A., and D. Sitaram, "Buffer Management Policy for an On-Demand Video Server", *IBM Research Report, RC 19347*, Yorktown Heights, NY, 1994.

[9] Dan, A. D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari, " Buffering and caching in large scale video servers," In *Proceedings of IEEE CompCon*, 1995, pp. 217–224.

[10] Dan, A., and D. Sitaram, "A Generalized Interval Caching Policy for Mixed Interactive and Long Video Environments" *Multimedia Computing and Networking*, San Jose, Jan 1996.

[11] Dan, A., P. S. Yu, and J. Y. Chung, "Characterization of Database Access Pattern for Analytic Prediction of Buffer Hit Probability", *The VLDB Journal*, Vol. 4, No. 1, Jan. 1995, pp. 127–154.

[12] Dey, J., J. Salehi, J. Kurose, and D. Towsley. "Providing VCR capabilities in large-scale video servers", *Proceedings of ACM Multimedia Conference*, pages 25–32, 1994.

[13] "Digital Equipment Corporation enters video-on-demand market", *Digital Video Server News*, DEC, October 19, 1993.

[14] DuLac, K., "Video Pump Interface Recommendation to DAVIC", *DAVIC/CFP/004*, Dec. 1994.

[15] Effelsberg, W. and T. Haerder, "Principles of Database Buffer Management", *ACM Trans. Database Systems,* Vol. 9, No. 4, Dec. 1984, pp.560-595.

[16] *Electronic Engineering Times*, March 15, 1993, pp 72.

[17] Escobar-Molano, M. L., S. Ghandeharizadeh and D. Ierardi, "An Optimal Memory Management Algorithm for Continuous Display of Structured Video Objects", *Research Report 95-602*, University of Southern California, 1995.

[18] Feng, W., F. Jahanian, and S. Sechrest. "An optimal bandwidth allocation for the delievery of compressed prerecorded video" *Technical Report CSE-TR-260-95*, University of Michigan, August 1995.

[19] Fox, E. A., "The Coming Revolution in Interactive Digital Video," *Communication of the ACM*, Vol. 7, July 1989, pp. 794–801.

[20] Golubchik, L., J. C.-S. Lui, and R. R. Muntz "Reducing I/O Demand in Video-On-Demand storage Servers", *Proceedings of ACM SIGMETRICS/ Performance '95*, Ottawa, Canada, May 1995.

[21] Kamath, M., K. Ramamritham, and D. Towsley, "Continuous Media Sharing in Multimedia Database Systems," *Proc. Fourth Intl. Conf. on Database Systems for Advanced Applications*, Singapore, April 10-13, 1995.

[22] Kienzle, M., A. Dan, D. Sitaram, and W. Tetzlaff, "The Effect of Video Server Topology on Contingency Capacity Requirements" *Multimedia Computing and Networking*, San Jose, Jan 1996.

[23] Laursen, A., J. Olkin, M. Porter, "Oracle Media Server: Providing Consumer Based Interactive Access to Multimedia Data" In *Proceedings of ACM SIGMOD*, pages 470–477, 1994.

[24] Liskov, B., et. al., "Replication in the Harp File System", *ACM Symp. on Operating Systems Principles,* 1991, pp. 226-238.

[25] T. D. C. Little and A. Ghafoor. "Multimedia synchronization protocols for broadband integrated services." *IEEE JSAC*, 9(9):1368–1382, December 1991.

[26] Little, T.D.C., and D. Venkatesh, "Popularity-based assignment of movies to storage devices in a video-on-demand system," *Multimedia Systems*, 2:280–287, 1995.

[27] Livny, M., S. Khoshafian, H. Boral, "Multi-Disk Management Algorithms", *Performance Evaluation Review*, Vol. 15, May 1987, pp. 69–77.

[28] "Microsoft's Interactive Television Strategy", *Microsoft Directions*, July, 1994.

[29] Moser, F., A. Krais, and W. Klas, "L/MRP, A Buffer Management Strategy for Interactive Continuous Data Flows in a Multimedia DBMS", *Proc. 21st VLDB Conference*, Zurich, Switzerland, 1995.

[30] Ng. R., J Yang, "Maximizing Buffer and Disk Utilizations for News-On-Demand," *Proc. VLDB*, 1994, pp. 451–462

[31] "nCube breaks new ground with Oracle Media Server", *Gartner Group Research Notes*, December 27, 1993.

[32] Ramakrishnan, K. K., L. Vaitzblit, C. Gray, U. Vahalia, D. Ting, P. Tzelnic, S. Glaser, W. Duso "Operating System Support for a Video-On-Demand File Service." *Multimedia Systems*, Vol. 3 No. 2, May 1995, pp.53–65.

[33] Rangan, P. V., H. M. Vin, and S. Ramanathan, "Designing an On-Demand Multimedia Service," *IEEE Communication Magazine*, Vol. 30, July 1992, pp. 56–65.

[34] Rotem, D., J. L. Zhao, "Buffer Management for Video Database Systems," *Proc. ICDE*, 1995, pp. 439–448.

[35] Sacco, G. M., and M. Schkolnick, "Buffer Management in Relational Database Systems", *ACM Trans. Database Systems,* Vol. 11, No. 4, Dec. 1986, pp.473-498.

[36] J. Salehi, J. F. Kurose Z. L. Zhang, and D. Towsley. "Supporting stored video: Reducing rate variablity and end-to-end resource reservation through optimal smoothing" *Technical Report Umass-TR-95-98*, University of Massachusetts, Amherst, November 1995.

[37] Satyanarayanan, M., et. al., "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Trans. on Computers, Special Issue on Fault-Tolerant Computing,* April 1990.

[38] Shahabi, C., and S. Ghandeharizadeh, " Continuous Display of Presentations Sharing Clips", *ACM Multimedia Systems* pages 76–90, 1995.

[39] Sitaram, D., A. Dan, and P. S. Yu, "Issues in the Design of Multi Server File Systems to Cope with Load Skew ", *IEEE PDIS Conference*, Jan. 1993.

[40] Song, J., A. Dan and D. Sitaram, "*JINSIL:* A System for Presentation of Composite Multimedia Objects in a Distributed Environment", *IBM Research Report, RC 20476.*

[41] Tetzlaff, W., and R. Flynn, "Elements of Scalable Video Servers", *Proc. CompCon95*, 1995, pp. 239–248.

[42] Yu, P., and A. Dan, "Performance Analysis of Affinity Clustering on Transaction Processing Coupling Architecture", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 5, Oct. 1994, pp. 764–785.

[43] *Video Store Magazine*, Dec. 13, 1992.