Load Management in Distributed Video Servers

Nalini Venkatasubramanian and Srinivas Ramanathan
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94304
(nalini,srinivas)@hpl.hp.com

Abstract

In this paper, we define and formulate various policies for load management in distributed video servers. We propose a predictive placement policy that determines the degree of replication necessary for popular videos using a cost-based optimization procedure based on a priori predictions of expected subscriber requests. For scheduling requests, we propose an adaptive scheduling policy that compares the relative utilization of resources in a video server to determine an assignment of requests to replicas. To optimize storage utilization, we also devise methods for dereplication of videos based on changes in their popularities and in server usage patterns. Performance evaluations indicate that a load management procedure which uses a judicious combination of the different policies performs best for most server configurations.

Advances in storage technologies are making high-performance video servers a reality. These video servers are being deployed over emerging broadband networks to deliver a variety of interactive, digital video services to thousands of residential subscribers. To meet the scalability requirements in such large deployments, distributed video server architectures are being considered [3]. In this paper, we propose various methods for load management that are targeted at improving the cost-effectiveness of distributed video servers. The paper is organized as follows: Section 1 motivates the need for load management and discusses its implications on a video server's cost-performance. Section 2 presents the different policies for load management while Section 3 discusses more optimizations for load management. Section 4 presents a comparative analysis of the different policies. We conclude in Section 5 and outline areas for future research.

1 Load Management in Video Servers

Figure 1 illustrates the architecture of a typical video server, the main components of which are: a number of independent and distributed data sources, each of which includes high capacity storage devices,

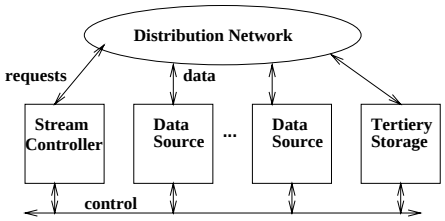

Figure 1: Architectural view of a networked multimedia system

processor, buffer memory, and high-speed network interfaces for real-time video retrieval and transmission. Since the number of video objects requested for access by subscribers far exceeds the available disk capacity, the video server also includes tertiary storage. A central distribution controller (DC) governs the operation of the data sources. The DC receives all subscriber requests directed to the video server, considers the concurrent commitment of the data sources, and evaluates whether a new request can be admitted for service without affecting service for the requests being currently serviced and the data source to which the new request should be assigned. All the above components are interconnected via an external distribution network that also transports video information to computers and set-top devices at the subscriber end. A lower speed back-channel conveys subscriber commands back to the data sources via the DC. The scalability of the above architecture is attributable to the ability to add additional data sources to the server in order to provide increased storage capacity and transfer bandwidth.

1.1 Load Management Mechanisms

There are three fundamental mechanisms for load management in video servers:

Replication: When the existing copies of a video object are not capable of supporting additional subscriber requests, replication of the video object becomes necessary. At the time of replication, it is first necessary to choose a data source on which a new

replica should be created and then to determine the rate of replication. Replication at a rate that is faster than real-time minimizes the time for replication but consumes greater server resources.

Request migration: Sometimes, instead of replicating a video object to service a new request, it may be simpler and more efficient to migrate an existing request to another data source. However, since it requires explicit tear-down and reestablishment of network connections as well as exchange of state information between data sources, both of which can cause significant playback jitter, migration is not suitable for distributed video servers.

Dereplication: Since storage space is a premium resource in a data source, dereplication, based on observations of video object popularities, is fundamental for effective load management. The copy to be dereplicated must be carefully chosen based on the expected future demand for the video objects and the current load on the different data sources. Like replication, dereplication may also not occur instantly: a copy chosen for dereplication can be removed only after all requests that are currently being serviced by it have completed. Considering the delay between the initiation and completion of (de)replication , it is most likely that both of these load management operations have to be initiated predictively, based on expected future subscriber request patterns.

1.2 A Load Management Example

Consider the example in Figure 2. Suppose that at some time instant, a video server that has two data sources S_1 and S_2 has sufficient storage to accommodate eight video objects on S_1 and two video objects on S_2. Further, suppose S_1 and S_2 have sufficient bandwidth to support three and eight additional requests, respectively. Also, suppose that ten additional requests, eight for a video object V_1 and one each for video objects V_2 and V_3 are expected to arrive in the near future. Considering the disk space and bandwidth availabilities of the data sources, it is prudent to allocate V_1 to S_2 and V_2 and V_3 to S_1. On the other hand, if V_2 and V_3 were allocated to S_2, the video server would only be able to handle five of the ten requests - three for V_1 and one each for V_2 and V_3. The simple example above illustrates a case in which localization (i.e., minimizing the number of copies) of a video object, V_1, actually leads to improved server throughput. However, if additional requests are received for V_1, replication becomes essential. In general, the exact load management mechanism has to be decided depending upon the current utilization of resources in the data sources of the video server and the expected subscriber requests including buffer memory and CPU processing power.

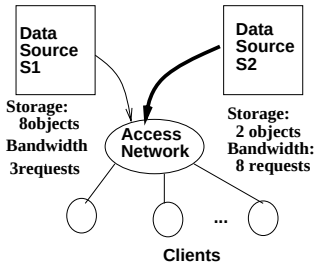

Figure 2: A load placement scenario

2 Policies for Load Management

Consider a video server with s data sources, S_1,..., S_s. Suppose that there are v video objects V_1,..., V_v that are to be stored in the video server. Let R_i denote a request for video object V_i. Let DB_i, M_i, CPU_i, and X_i denote the disk bandwidth, memory buffer space, CPU cycles, and network transfer bandwidth, respectively, that are necessary for supporting request R_i. Likewise, let DB_j, M_j, CPU_j, and X_j denote the corresponding resource availabilities of data source S_j. Let the function $Map(R_i, S_j)$ denote whether or not the video object V_i corresponding to request R_i is available at the data source S_j.

2.1 Characterization of Server Load

We first characterize how loaded the video server's data sources are at any time as the number of additional requests that the data source can support. The ability of a data source to support additional requests is dependent not only on the resources that it has available, but also upon the video object requested and the characteristics of the request (e.g., playback rate, resolution, etc.). We characterize the degree of loading of a data source S_j with respect to request R_i in terms of its load factor, $LF(R_i, S_j)$, considering the various resources, as:

$$LF(R_i, S_j) = max(\frac{DB_i}{DB_j}, \frac{M_i}{M_j}, \frac{CPU_i}{CPU_j}, \frac{X_i}{X_j})$$

The difference between the different ratios in the above equation gives an idea of the nature and extent of the performance bottleneck at the data source. Since different requests have different resource requirements, the load factor may vary from one request to another. Lower the load factor, greater is the capacity of the data source to service additional requests; $\lfloor 1/(LF(R_i, S_j)) \rfloor$ is the maximum number of requests similar to R_i that S_j can support.

2.2 Adaptive Scheduling of Requests

Since subscriber requests may not conform to predicted patterns, assignment of requests to data sources is done adaptively using the load factor as a basis. In the adaptive scheduling procedure we propose, when a DC (distribution controller) receives a request R_i for video object V_i, considering only data sources that hold a copy of V_i and that have sufficient resources to support R_i (i.e., $LF(R_i, S_j) \leq 1$), the DC chooses the data source to serve R_i to be the one for which $LF(R_i, S_j)$ is minimum. If no such data source can be found, the DC can decide to create a new replica of V_i. The data source S_j on which the new replica is to be created can be chosen to be the one for which $LF(R_i, S_j)$ is the minimum among the data sources that do not have a replica of V_i. Such an assignment maximizes the possibility of servicing future requests to V_i. A key drawback with this approach is that in order to be feasible and attractive, the replication must proceed at a very high rate, thereby consuming vital server resources. Alternatively, when the DC is unable to find a data source with a pre-existing replica of V_i that can service R_i, it can decide to reject R_i. By analyzing the rate of rejections, over a longer time frame, the DC can decide whether a change in placement of video objects is necessary. In the next section, we discuss a predictive placement module that uses a priori estimates of request patterns to derive an allocation of video objects to data sources.

2.3 Predictive Placement of Video Objects

Since reallocation of video objects to data sources is a lengthy and expensive process, it should ideally be performed in advance, to satisfy expected future subscriber requests. Hence, a predictive placement policy is used by the DC to determine when, where, for how long, and how many replicas of each video object should be placed in a video server so as to maximize the number of requests serviced by it. Whereas adaptive scheduling handles short-term variations in request arrival times and patterns, predictive placement handles longer-term variations in request patterns. Predictive placement can be initiated periodically, in particular for popular video objects, or may be executed at other times when extraneous events occur (e.g., loading new objects, changes in access patterns, components failures, etc.).

During each prediction period, the DC must determine how many replicas of each video object are necessary, and which of the data sources these replicas should be allocated to. To answer these questions, the DC has to derive a tentative assignment of replicas to data sources. This implies that the adaptive scheduling module may abide by this assignment only if the actual subscriber requests match the expected pattern. The procedure that the DC uses to determine the allocation of video objects to data sources, referred to as the *pseudo-allocation procedure* only determines when and to where replication must be initiated. The DC then has to figure out the rate at which replication should proceed. Replication may be initiated at the maximum feasible replication rate and this rate may even be changed dynamically, depending upon the instantaneous load on the data sources and the number of replications scheduled to be performed from the same data source.

	S_1	S_2
V_1	$\min(N_1, \lfloor(\frac{1}{LF(R_1,S_1)})\rfloor)*r_1$	$\min(N_1, \lfloor(\frac{1}{LF(R_1,S_2)})\rfloor)*r_1$
V_2	$\min(N_2, \lfloor(\frac{1}{LF(R_2,S_1)})\rfloor)*r_2$	$\min(N_2, \lfloor(\frac{1}{LF(R_2,S_2)})\rfloor)*r_2$
V_3	$\min(N_3, \lfloor(\frac{1}{LF(R_3,S_1)})\rfloor)*r_3$	$\min(N_3, \lfloor(\frac{1}{LF(R_3,S_2)})\rfloor)*r_3$
	$\max(PM(V_i, S_1))$	$\max(PM(V_i, S_2))$

Table 1: Placement matrix for request R_i

In this paper, we restrict the discussion of the pseudo-allocation procedure to the case in which new video objects are to be placed on appropriate data sources. A simple extension of this approach for placement of existing video objects in the presence of time-varying request patterns is discussed elsewhere [6]. In the ensuing discussion, we assume that the requests expected for different video objects during the next prediction period are available. We are currently investigating methods for predicting expected subscriber requests to video objects depending upon past history, classification of video objects, etc.

We formulate pseudo-allocation as an optimization problem, in which the metric to be optimized is the total revenue that the video server generates. In this procedure, each request R_i is associated with a revenue r_i. r_i collectively characterizes the revenue that can be generated by servicing a request R_i relative to other requests, and is dependent on the resources required to satisfy R_i, the characteristics of the video object V_i that is requested by R_i, and the popularity of V_i. By using a revenue-based method for pseudo-allocation, the DC can tailor the placement of video objects so as to best utilize the video servers' resources. Hence, by taking up more resources, a request may in fact be preventing other requests from being serviced; so the net profitability of the server needs to be considered.

Suppose N_i is the expected number of requests in a prediction period for each video object V_i (for now, we make a simplifying assumption that all requests

to a video object require the same quality of playback and the same type of service). In order to derive a mapping of video objects to data sources, the DC constructs a placement cost matrix that represents the relative costs of servicing subscriber requests from each of the data sources. Columns in the placement matrix represent data sources and rows represent video objects (see Table 1). Each entry in the matrix, $PM(V_i, S_j)$ represents the maximum revenue that can accrue from allocating V_i to S_j. This in turn is the product of r_i and the maximum number of requests for V_i that can be serviced by S_j (the minimum of N_i and $\lfloor 1/(LF(R_i, S_j)) \rfloor$). In the above computation, the load factor value for a data source S_j that does not have a copy of video object V_i should be corrected, to account for the extra bandwidth, transfer rate, and memory resources necessary for loading a new replica of V_i on S_j.

To determine an allocation of video objects to data sources, the DC iteratively employs a greedy heuristic. As per this heuristic, the DC attempts to allocate the video object V_i to data source S_j such that the total revenue resulting from such an allocation is maximum, i.e., $Map(V_i, S_j) = 1$, if $PM(V_i, S_j) = \forall a \forall b max(PM(V_a, S_b))$. Having determined a candidate data source for V_i, the DC decides to either utilize the data source to its fullest capacity, or to allocate all of the expected results for V_i to S_j. In the first case, S_j is no longer eligible for servicing other requests, and hence, the column corresponding to S_j is removed from the placement matrix. In the second case, all the requests for V_i are satisfied by S_j and hence, the row of the placement matrix that corresponds to V_i is deleted. In either case, the DC then reconstructs the reduced prediction matrix and iteratively applies the greedy heuristic to the resulting matrix to derive the complete mapping of video objects to data sources.

In the above procedure, when the DC encounters a tie for the maximum revenue generating video object-data source combination, it uses one of several tie-breaking heuristics.

(a) When a tie exists between the pairs (V_i, S_j) and (V_m, S_n), the DC assigns requests to the video object that already exists on the corresponding data source, thereby avoiding the extra overhead of creating a new replica.
(b)To break a tie that results if both V_i and V_m exist on S_j and S_n, respectively, the DC allocates the video object with the maximum number of requests, so as to maximize the number of requests serviced by a single replica.
(c) If neither V_i exists on S_j or V_m exists on S_n, the DC must create a new copy of one of the video objects. To make a choice, for both pairs, (V_i, S_j) and (V_m, S_n), the DC computes the space factor, defined as the ratio of the disk space available at the data source to the disk space required to store a copy of the video object. The absolute difference between the space factor and the load factor, $Abs(SF(V_i, S_j) - LF(R_i, S_j))$ indicates the degree of mismatch in the commitment of the different resources in data source S_j. Smaller the difference, lesser the mismatch, and greater are the chances that S_j will be well utilized and provide higher throughput. Hence, to break the tie between (V_i, S_j) and (V_m, S_n), the DC chooses the pair with the smaller absolute difference between their space and load factors.

3 Load Management Optimizations

Eager Replication: In between the invocations of the adaptive and predictive phases, to cater for periods of future high demand, we propose an additional replication strategy called eager replication. Here, replication of video objects occurs in anticipation, even though the demand for these objects may not be apparent immediately (e.g., replication of movies during non peak-hours). Video objects slated for eager replication may be chosen as part of the predictive placement procedure, but may actually be replicated only if sufficient video server resources become available. Eager replication is assigned lowest priority relative to the other load management tasks, so as not to impact the video server's performance. Since replication is a time-consuming and resource-intensive process, eager replication can significantly reduce the overheads of replication during periods of great demand, especially for popular videos. By doing so, eager replication can not only enhance video server utilization and throughput, but can also lower the latency between the reception and servicing of requests.

Lazy Dereplication: In view of the critical nature of storage resources, video object copies should be dereplicated as and when possible, to make room for other objects that may be in demand at a later time. Just as replication can be performed eagerly, dereplication can be performed in a lazy fashion. As per this strategy, when a video object V_i is dereplicated, V_i's storage resources are released and marked as being available. However, the disk blocks that were being used for V_i are rewritten only if there is an immediate need to reuse these blocks for storage of some other object. In the interim period, between the time dereplication is initiated and the time when the disk blocks of V_i are overwritten, V_i exists on the data source and can be reclaimed if so desired.

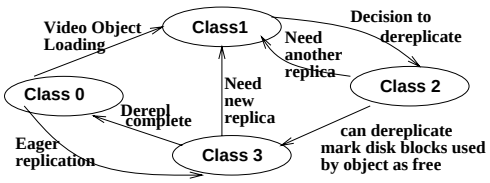

Figure 3: State transition diagram depicting the life of a video object

3.1 Classifying Video Objects for Eager Replication and Lazy Dereplication

To implement eager replication and lazy dereplication, a data source must support four types of video objects: (1) Class 0 video objects that do not exist on secondary storage in a data source. (2) Class 1 video objects that are currently on-line and in active use. New objects are assigned this class when loaded. (3) Class 2 video objects that are also still on-line but are marked for dereplication. After marking a replica for dereplication, the DC avoids further assignment of requests to this replica and releases storage after all servicing requests have terminated. (4) Class 3 video objects that are provided temporary storage on a data source, but the storage resources used by such objects are tagged as available for other objects, if required. Figure 3 depicts the transitions between the different video object classes.

3.2 Modifying the Pseudo-Allocation Procedure

Although eager replication and lazy dereplication can be initiated independently, these optimizations are most efficiently performed in conjunction with predictive placement. To support eager replication and lazy dereplication, besides maintaining the list of replicas and their locations for each video object, the DC must also maintain the class of each replica. Furthermore, for each data source, the DC must maintain the storage space used for the different classes of video objects. During each iteration of the pseudo-allocation procedure, the DC performs the following steps:

(1) **Downgrading Class 2 replicas:** Before each iteration of the pseudo-allocation procedure, the DC checks for existing Class 2 replicas that are no longer being used to service subscriber requests. These replicas are downgraded to Class 3, to potentially enable lazy dereplication during the next iteration and storage availability is recorded.

(2) **Determination of new Class 2 replicas:** After the pseudo-allocation procedure has assigned all the expected requests for a video object, V_i, the DC determines if any existing replicas of V_i have not been assigned new requests in the current prediction period and marks all such replicas as being Class 2 replicas.

(3) **Exploiting eager replication and lazy dereplication while creating new replicas:** Initially, pseudo-allocation proceeds as before, as if Class 2 and Class 3 replicas do not exist. If at any stage, the DC determines that it must create new replicas for a video object, V_i, it now attempts to determine whether Class 2 or Class 3 replicas of V_i exist and can be re-used. Since Class 2 replicas are already being used to service requests, they are assigned higher priority compared to Class 3 replicas. If at a later stage, additional replicas of V_i are necessary and no Class 1 or Class 2 replicas are available for request assignment, all Class 3 replicas of V_i are marked as Class 1 replicas and are made available for assignment of new requests. At the end of the pseudo-allocation procedure, as before, all of the unused replicas of V_i are downgraded back to Class 2 or Class 3, as appropriate.

4 Performance Evaluation of the Load Management Policies

In this section, using simulations, we present a comparative analysis of the performance of different load management policies under varying server resource constraints.

4.1 System Model

The basic video server configuration used in the simulations includes 6 data sources, each with a storage of 50 Gbits and a transfer bandwidth of 310 Mbits/second. A replication source (e.g. tertiary storage) that serves as permanent repository of video objects has a maximum replication bandwidth that is assumed to be 155 Mbits/second in the basic configuration. For simplicity, CPU and memory resources of the data sources are assumed not to be bottlenecks. The video server stores 50 MPEG-2 movies, each with an average bandwidth requirement of 3 Mbps. The average duration of the movies is assumed to vary uniformly between 30 and 120 minutes. Following the results reported in [2], we assume that the popularity of movies follows Zipf's law, with the request arrivals per day for each movie m_i being given by:

$$\Pr.(m_i \text{ is requested}) = \frac{K_M}{i} \text{ where } K_M = \left(\sum_1^M \frac{1}{i}\right)^{-1}$$

Assuming that the request arrivals per hour follow a Zipf-like distribution [5], we compute the probability of request arrival in hour j to be: $p_j = c/(j^{1-\phi})$, for $1 \leq j \leq 24$, where ϕ is the degree of skew and is assumed to be 0.8. The constant $c = 1/(\sum(1/j^{1-\phi}))$, $1 \leq j \leq 24$. From the request arrivals per day for each movie and the probability distribution of requests for

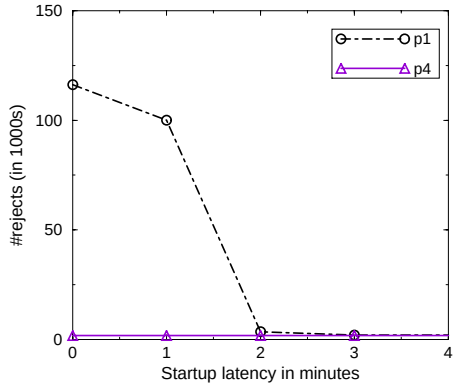

Figure 4: Effect of startup latency on request rejections

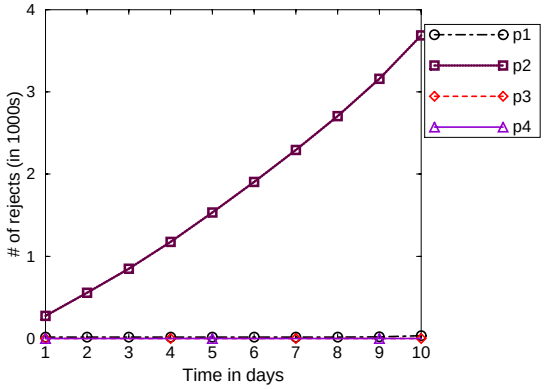

Figure 5: Comparison of the performance of the different load management policies for the basic configuration of the video server

each hour, the number of requests that arrive in each hour for each movie m_j is computed.

We study the performance of the different policies proposed in Section 3 in isolation as well as in different combinations. The combinations of policies we study are:
(1) *P1*, adaptive scheduling for request assignment to data sources and replication on-demand to create new replicas.
(2) *P2*, predictive placement for allocation of movies to data sources, and for request assignment. The tentative assignment of requests derived in the pseudo-allocation procedure is itself used to assign requests to replicas.
(3) *P3*, predictive placement for allocation of movies to data sources, and adaptive scheduling for request assignment.
(4) *P4*, predictive placement and adaptive scheduling with eager replication as a further optimization.
All of the above policies implement lazy dereplication. For policies that use predictive placement, we assume the availability of a perfect predictor.

4.2 Performance Comparisons

One of the primary distinctions between the different policies is illustrated in Figure 4. Because P1 relies on replication-on-demand, it introduces a finite and sometimes significant delay between the time when a subscriber request is received and when the request begins to be serviced. Figure 4 illustrates that if the maximum tolerable start-up latency is 2 mins, P1 forces a very large fraction of the requests received (as high as 85%) to be rejected. The other policies, which rely on predictive placement to pre-allocate movies before requests are received, do not entail any start-up latency. So as to not bias the performance of the different policies based on the choice of the start-up latency values, in the rest of the simulations, we have chosen the maximum start-up latency to be 5 minutes. We begin our comparative analysis by determining the performance of the four policies for the basic configuration of the video server. In subsequent experiments, we modify the basic configuration to highlight interesting distinctions between the load management policies.

(1) **Performance of the basic configuration:** Figure 5, highlights the inadequacy of P2. P2 relies on predictive placement not only for allocation of movies to data sources but also to schedule requests. Since predictive placement accounts only for the request times of movies, and does not take into account the exact completion times of requests, P2 is unable to allocate storage and bandwidth that are released when a request completes to other requests until the beginning of the next prediction period. This in turn translates into poor utilization of server resources, resulting in excessive rejection of subscriber requests. Figure 5 also illustrates that by using adaptive scheduling in conjunction with predictive placement, policies P3 and P4 avoid the drawback of P2. Next we attempt to vary the video server configuration to create bottlenecks in the data sources (bandwidth and storage) and in the replication source (bandwidth). We have chosen not to present the performance of P2 in the results, since P2 performs significantly worse than the other policies in all of the configurations.

(2) **Impact of varying storage at the data sources:** Figure 6, which compares the performance of the policies with changes in storage availability at the data sources (other parameters are maintained as in the basic configuration), illustrates the trade-offs between replication on-demand and predictive place-

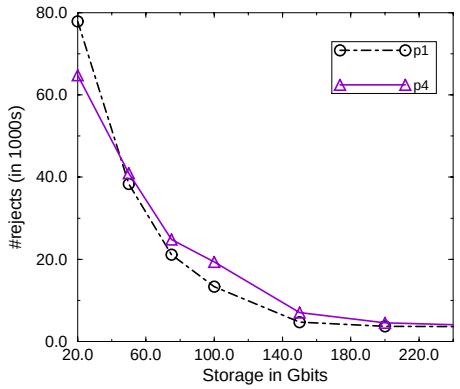

Figure 6: Effect of varying storage at the data sources.

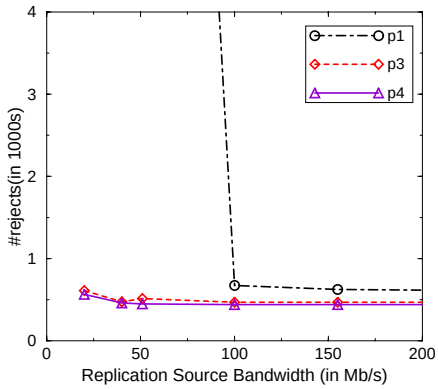

Figure 7: Effect of varying replication source bandwidth. P1 degrades in effectiveness at low replication bandwidths

ment policies. At low storage levels, P4 performs significantly better than P1. This is because predictive placement optimizes the utilization of storage by choosing to store the most popular movies that can be accommodated on the video server. P1 creates replicas of movies depending on relative arrival times of subscriber requests, and not based on movie popularity and performs worse than P4. At higher storage levels, P1 performs better than P4 because replication-on-demand permits P1 to dynamically allocate and reallocate storage. In contrast, P4 changes placement only at prediction period boundaries. When storage is the bottleneck resource, eager replication too is ineffective, and hence, P4 does not fare as well as P1. As storage increases, the difference in performance between P1 and P4 reduces. P3 behaves exactly like P4 under storage constraints, and is hence, not depicted in Figure 6.

(3)**Impact of replication bandwidth on performance:** Figure 7, which depicts the performance

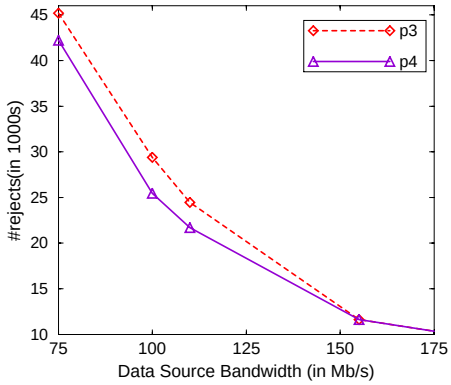

Figure 8: Effect of varying data source source bandwidth on performance of the video server. P4 outperforms P3.

of policies P1, P3, and P4 when the bandwidth available at the replication source is varied, illustrates the major drawback of replication-on-demand. Since P1 attempts to replicate movies only when subscriber requests for movies are received, this policy is highly dependent on the availability of sufficient bandwidth at the replication source and the data sources for replication. Therefore, at low replication bandwidths, P1 degrades dramatically in effectiveness. Since in practice, replication bandwidth is likely to be limited, we believe that a policy such as P1 that relies entirely on replication-on-demand for placement of movies has limited utility. We expect that the knee of the curve in Figure 7 that represents the minimum replication bandwidth necessary for P1 to be viable will occur at increasingly higher replication bandwidths as the start-up latency becomes more stringent. In sharp contrast to P1, policies P3 and P4 are less critically dependent on the availability of sufficient replication bandwidth. By anticipating requests in advance and by pre-scheduling replication, these policies can optimize their utilization of the available replication bandwidth. Even at high replication bandwidths, P3 and P4 outperform P1, offering a 25% reduction in the number of rejected requests. This result illustrates the advantages in utilizing predictive placement and adaptive scheduling in conjunction.

(4) **Impact of bandwidth availability at the data sources:** Figure 8, which compares the performance of P3 and P4 with changes in bandwidth available at the data sources, illustrates the advantages of eager replication. By utilizing the excess bandwidth available to create additional replicas of movies, eager replication offers more options for adaptive scheduling. This translates into a consistently better performance

for P4 as compared to P3. At very high bandwidths (above 150 Mbps in Figure 8) the advantages of replication in advance become minimal and hence, P3 begins to perform almost as well as P4.

5 Related Work

Many of the initial efforts in designing video servers have focussed on placement of media on disk to ensure real-time retrieval, admission control procedures, and buffer management policies [1, 7]. For load balancing across storage devices, [8] proposes a two-stage DASD dancing scheme. In the initial static stage, a greedy assignment of videos to disk groups is obtained using a graph-theoretic approach. The dynamic phase that follows uses the static assignment to perform real-time disk scheduling effectively. For effective load management, this phase incorporates a baton-passing scheme that enables request migration across storage devices.

Although similar in spirit to the DASD dancing work, the load management policies described in this paper are significantly different in their design, implementation, and utility. As alluded to in Section 1.1, migration schemes are not practical for load management in distributed video servers because of the significant jitter they can introduce during playback, as well as the network overheads they entail during teardown and reestablishment of network connections. The same drawbacks also apply to the partial segment replication, which has been proposed in [4] in the context of switched-storage video servers that comprise of processing unit(s) with shared access to a set of storage devices. A further difference is in the formulation of the predictive placement policy. Although not intended to handle individual subscriber requests, the predictive placement policy is designed to run in a real-time, on-line mode, and therefore, takes into account the overheads involved in replicating requests across data sources. In contrast, the static placement policy defined in [8] is intended to run infrequently, and does not take into account the overheads involved in reallocation of video objects. Another distinguishing feature of our work is the integration of replication and dereplication optimizations with the placement and scheduling policies. The design of the predictive placement policy draws upon the work of Dan et al. [5], who have studied the trade-offs between storage space and transfer bandwidth in video servers.

6 Conclusions

In this paper, we have explored various policies for load management in distributed video servers and compared their performance under varying storage and bandwidth constraints. Our analysis reveals that policies that utilize a combination of the predictive placement, adaptive scheduling, eager replication, and lazy dereplication policies perform well in most server configurations. We believe that the simplicity of implementation and flexibility offered makes these policies especially attractive for implementation in distributed video servers.

References

[1] D.P. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *ACM Transactions on Computer Systems*, 10(4):311–337, November 1992.

[2] C. Bisdikian and B. V. Patel. Issues on movie allocation in distributed video-on-demand systems. In *Proceedings of the IEEE International Conference on Communications*, volume 1, pages 250–255, June 1995.

[3] M. Buddhikot and G. Parulkar. Efficient data layout, scheduling and playout control in mars. In *Proceedings of NOSSDAV'95*, pages 339–351, April 1995.

[4] A. Dan, M. Kienzle, and D. Sitaram. Dynamic policy of segment replication for load-balancing in video-on-demand servers. *ACM Multimedia Systems*, 3(3):93–103, July 1995.

[5] A. Dan and D. Sitaram. An online video placement policy based on bandwidth to space ratio (bsr). In *SIGMOD '95*, pages 376–385, 1995.

[6] N. Venkatasubramanian and S. Ramanathan. Effective load management for scalable video servers. Technical report, Hewlett-Packard Research Laboratories, Feb 1996.

[7] H. M. Vin and P. V. Rangan. Designing a multiuser hdtv storage server. *IEEE Journal on Selected Areas in Communications*, 11(1):153–164, January 1993.

[8] J. L. Wolf, P. S. Yu, and H. Shachnai. Dasd dancing: A disk load balancing optimization scheme for video-on-demand computer systems. In *Proceedings of ACM SIGMETRICS '95, Performance Evaluation Review*, pages 157–166, May 1995.