Design of Scalable Continuous Media Servers with Dynamic Replication

ChengFu Chou
Department of Computer Science
University of Maryland at College Park

Leana Golubchik
University of Maryland Institute for Advanced Computer Studies
and Department of Computer Science
University of Maryland at College Park

John C.S. Lui
Department of Computer Science and Engineering
The Chinese University of Hong Kong

I-Hsin Chung
Department of Computer Science
University of Maryland at College Park

Abstract

Multimedia applications place high demands for quality-of-service (QoS), performance, and reliability on systems. These stringent requirements make design of cost-effective and scalable systems difficult. Therefore efficient *adaptive* and *dynamic* resource management techniques in conjunction with data placement techniques can be of great help in improving performance, scalability and reliability of such systems. In this paper, we first focus on data placement.

In the recent past, a great deal of work has focused on "wide" data striping as a way of dealing with load imbalance problems caused by skews in data access patterns. Another approach to dealing with load imbalance problems is replication. The appropriate compromise between the degree of striping and the degree of replication is key to the design of scalable continuous media (CM) servers. In this work we focus on evaluation of this compromise in the context of a hybrid CM server design.

Changes in data access patterns lead to other questions: (1) when should the system alter the number of copies of a CM object, and (2) how to accomplish this change. We address (1) through an adaptive threshold-based approach, and we use dynamic replication policies in conjunction with a mathematical model of user behavior to address (2). We do this without any knowledge of data access patterns and with provisions for full use of VCR functionality. Through a performance study, we show that not only does the use of this mathematical model in conjunction with dynamic resource management policies improves the system's performance but that it also facilitates *reduced sensitivity* to changes in: (a) workload characteristics, (b) skewness of data access patterns, and (c) frequency of changes in data access patterns. We believe that not only is this a desirable property for a CM server, in general, but that furthermore, it suggests the usefulness of these techniques across a *wide range of continuous media applications*.

1 Introduction

With the rapid growth of multimedia applications, there is a growing need for large-scale continuous media (CM) servers that can meet user demand. Multimedia applications (such as video stream delivery, digital libraries, and distance learning systems) place high demands for quality-of-service (QoS), performance, and reliability on storage servers and communication networks. These, often stringent, requirements make end-to-end design of cost-effective and *scalable* CM servers difficult. The scalability of a CM server's architecture depends on its ability to:

- expand as user demand and data sizes grow;
- maintain performance characteristics under degradation of system resources, which can be caused by losses in network and storage capacities;
- maintain performance characteristics under growth or re-configuration.

In particular, the choice of data placement techniques can have a significant effect on the scalability of a CM server and its ability to utilize resources efficiently. Existing data placement techniques in conjunction with scheduling algorithms address two major inefficiencies in such systems: (1) various overheads in reading data from storage devices and (2) load imbalance, e.g., due to *skews* in data access patterns. In this work, we focus on the latter issue and specifically on its bearing on the *scalability* characteristics of a *distributed* (read-only) CM server.

Due to the enormous storage and I/O bandwidth requirements of multimedia data, a CM server is expected to have a large disk farm. Thus, we must necessarily consider designs which contain multiple disk clusters and processing nodes, i.e., we must consider *distributed* designs. An important issue then is the placement of objects on the nodes of the CM server which directly affects its load balancing characteristics.

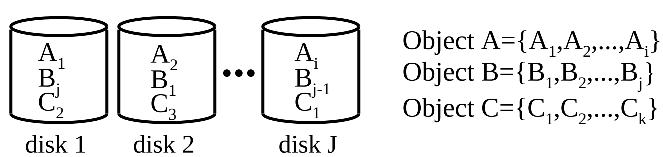


Figure 1: Objects striped across disks.

In the recent past, a great deal of CM server designs, e.g., as in [2, 10, 14], have focused on "wide" data striping techniques, where each object is striped across all the disks of the system, i.e., disk 1 through disk J, as illustrated in Figure 1. The potential load imbalance is largely due to the *skews* in data access patterns which, without data striping, could result in high loads on some disks containing the more popular objects, while the disks containing less popular objects may remain idle. Moreover, this problem is exacerbated by the fact that access patterns change over time. Thus, an advantage of wide data striping is that it "implicitly" achieves load balance by decoupling an object's storage from its bandwidth requirements. However, wide data striping also suffers from several shortcomings:

1. It is not practical to assume that a system can be constructed from homogeneous disks, i.e., as the system grows or experiences faults (and thus disk replacement) we are forced to use disks with different transfer and storage capacity characteristics; having to stripe objects across *heterogeneous disks* would lead to further complications [1].

2. An appropriate choice of a striping unit, the object size, and the communications network infrastructure dictate an upper bound on the number of disks over which that object can be striped, beyond which replication of objects is needed to increase the number of simultaneous users [10], e.g., to the best of our knowledge, in implementations described in [10, 14] striping is performed over (at most) a few tens of *homogeneous* disks only[1]. Note that, delivery of relatively short CM objects is of use to many applications, including digital libraries and news-on-demand systems.

3. Due to the continuity constraints, some form of synchronization in delivery of a single object from multiple nodes[2] must be considered. The need for some form of "synchronization" arises from the fact that different fractions of an object are being delivered from different nodes at different times during the object's display, and hence some form of coordination between these nodes (and perhaps the client) is required in order to present a "coherent" display of the object.

4. As user demand and data sizes grow and hence the system requires more storage and disk bandwidth capacity, the needed expansion results in re-striping of *all* the objects.

5. Due to the potential need for communication of CM data between the nodes over which the data is striped, the capacity of the communication network limits the performance of the distributed CM server. This limitation directly affects the scalability of the CM server and is one of the main issues we investigate in this work.

Another approach to dealing with the load imbalance problem arising from skews in data access patterns is replication, i.e., creating a sufficient number of copies of a (popular) object so as to meet the demand for that object. Note that, disadvantages of replication are: (1) a need for additional storage space, and (2) a need for techniques that adjust the number of replicas as the access patterns change. Some of these issues are addressed in [26], and in our previous work [19, 7]. In addition, here we also improve on the dynamic replication schemes in [19, 7].

[1] In fact, we are not aware of, in the current literature, any *large-scale* implementation that utilizes *heterogeneous* disks.

[2] A more detailed definition of a node is given in Section 2.

In this paper, we consider a *hybrid* approach (refer to Section 2), which addresses the above listed shortcomings. Specifically, the main focus here is on the *tradeoffs* between striping and replication, which are as follows. In a small-scale CM server, where all disks are assumed to be connected to a single node, data striping can provide better performance characteristics than replication because of its ability to deal with load imbalance problems without the need for additional storage space and without significant networking constraints. However, in a large-scale CM server, data striping can result in a need for significant communication network capacities which can lead to poor scalability characteristics and high costs. Essentially, striping is a good approach to load balancing while replication is a good approach to "isolating" nodes from being dependent on other ("non-local") system resources. That is, the wider we stripe in a distributed CM system, the more we are dependent on the availability of network capacity as well as resources not local to a node. Furthermore, replication has the benefit of increased reliability (see Section 4.1) in terms of: (a) longer mean time to loss of data from the disk subsystem, and (b) dealing with lack of network resources, including network partitioning. The downside of replication is that it increases storage space requirements and hence cost of storage. However, as storage costs decrease (fairly rapidly) and the need for scalability grows, replication becomes a more attractive technique. In summary, the appropriate compromise between the degree of striping and the degree of replication is key to the design of a *scalable* distributed CM server. This is the first major focus of our paper.

Instead of striping each objects across all the nodes of the system, we take a hybrid approach, as in [26], and constrain the striping to a single node while replicating popular objects on several nodes in order to provide sufficient bandwidth capacity to service the demand for these objects. Here important questions include: (a) how many copies of each CM object should the system maintain, (b) on which nodes should these copies be placed, and (c) (possibly) how to migrate users from one node to another, in mid-stream, in order to admit more users through adjustments to current load allocations. For instance, an interesting approach to addressing these questions is given in [26], in the context of workloads with relatively *infrequent* changes in object access patterns (e.g., on a daily basis).

More frequent changes in data access patterns lead to the following additional important questions: (1) *when* should the system alter the number of copies of a CM object, and (2) *how* to accomplish this change. Thus, in the paper we also address load imbalance problems arising from relatively frequent *changes* in data access patterns — we address (1) through a *threshold-based* approach, and we use dynamic replication policies in conjunction with a mathematical model of user behavior to address (2).

Related Work and Our Contributions

Many of the issues raised above in the context of CM servers have been considered in the context of either distributed database systems or web-servers, such as the dynamic replication issues considered in [11, 12], to name a few. Briefly, works on these topics differ from ours in one or more of the following main aspects: (1) our system (the CM server), is a *read-only* system (from the user's point of view), thus we do not consider consistency/coherency type issues; (2) our objects (e.g., movies) are extremely large (e.g., on the order of several gigabytes of data *per* object), consequently in our system replication times are much longer, which leads to different considerations for dynamic replication policies; and (3) due to the real-time constraints in the delivery of media such as video, we must insure that there is no jitter in the delivery of objects, which also reflects on the dynamic replication policies (real-time constraints are not a consideration in the above mentioned works).

Much research has also been done on design of CM storage servers, e.g., as in [2, 10, 14], which mostly falls into several broad categories: (1) small-scale servers, where in most cases all disks are connected to a single node; (2) medium-scale LAN-based servers, and (3) medium-scale (either distributed or not) servers, which employ high speed interconnects, such as ATM-based technology. To the best of our knowledge, most of these designs employ wide data striping techniques and the corresponding existing successful implementations employ only tens of disks. In contrast, the use of replication for the purpose of addressing changes in data access patterns has been less explored. In [25] the authors consider skews in data access patterns but in the context of a static environment. In [26], the authors address various questions arising in the context of load imbalance problems due to skews in data access patterns, but in a less dynamic environment (than we investigate here). In [9, 8], the authors also consider dynamic replication as an approach to load imbalance, and in our previous work [19, 7], we study a taxonomy of dynamic replication schemes. However, all of these works, except our work in [6, 7], either (a) assume some *knowledge of frequencies of data access* to various objects in the system, and/or (b) do *not* provide users with *full use of VCR functionality*, and/or (c) consider less dynamic environments than the one considered here. Our motivation in doing away with such assumptions in our work is largely due to considerations of applicability of dynamic replication techniques in more general settings and to a wider range of applications of CM servers.

Lastly, to the best of our knowledge, previous works do not consider *alternative design characteristics* that affect the *scalability* of CM servers in an *end-to-end* setting under changes in access patterns (i.e., taking into consideration both the network and the storage resource constraints). The *quantitative* study of such issues and the cost/performance and reliability characteristics that distributed designs exhibit under growth, reconfiguration, degradation of resources, and changes in workloads are essential to assessing the *scalability* of proposed architectures and to the development of *large-scale* CM servers, in general.

Thus, the main contributions of our work are as follows. (Initial results of this study are given in [6, 7]).

- A mathematical model of user behavior is given, which facilitates reduction in (at least the "virtual") replication time (see Section 3.2 for details). We show that the model is not very sensitive to the accuracy of its parameters and thus is of reasonably practical use.
- Quantitative evaluation of *performance* and *resource demand* characteristics of wide data striping vs. hybrid techniques in large-scale distributed CM servers; such an evaluation is crucial to achieving a *scalable* design of CM servers.
- Quantitative evaluation of *performance* characteristics of dynamic replication policies. We show that the use of the mathematical model of user behavior in conjunction with dynamic threshold adjustment method (1) improves the performance of the more "conservative" (in terms of resource usage) dynamic replication policies as well as (2) facilitates *reduced sensitivity* to changes in workload characteristics, skewness of data access patterns, and frequency of changes in data access patterns.
- Improved, as compared to [7], dynamic replication techniques for distributed hybrid CM servers, needed to achieve better performance, which adjust the number of replicas in the system based on changes in data access patterns and user demand.
- Quantitative evaluation of reliability characteristics of wide data striping vs. hybrid systems.
- Quantitative evaluation of the tradeoffs between various resources (such as local switch capacity, local storage space capacity and global switch capacity) of the CM server. Based on this evaluation, current costs and technology trends, the designer can make *system sizing* decisions.
- Quantitative evaluation of performance characteristics of hybrid CM servers with heterogeneous resources. In practice, we expect CM servers to have heterogeneous resources as a result of system growth (due to expansion) and disk failures (and subsequent replacement). We show that the hybrid design naturally extends to systems with heterogeneous resources with little or no loss in performance as compared to the corresponding homogeneous systems. We also note that we are not aware of wide data striping techniques for systems with heterogeneous resources which do not suffer from loss of either bandwidth or storage resources. Hybrid designs do not suffer from such a loss.

Based on this end-to-end cost/performance and reliability study we argue that *hybrid* designs in conjunction with dynamic replication schemes result in large *scalable*, reliable, high performance, and cost-effective CM systems.

2 Hybrid CM System Architecture

A hybrid system architecture is illustrated in Figure 2(a). It consists of a set of nodes connected by,

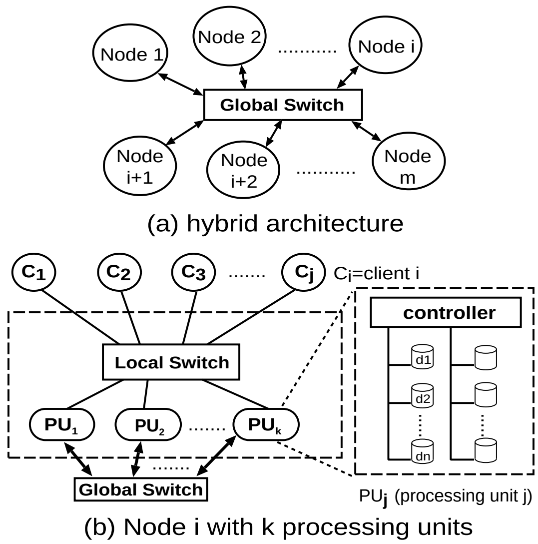


Figure 2: Hybrid System Architecture.

what we termed, a high speed *global* switch, which is a high bandwidth resource that can, for instance, correspond to a high capacity WAN or an ATM-type infrastructure. Each node i, as depicted in Figure 2(b), contains one or more processing units (PUs) and one, what we termed, *local* switch (e.g., switched Ethernet) which is used to connect all local PUs as well as local clients. Each client connects to its local switch. Requests from this client which are serviced by a PU from node i are termed "local". When a request from a client cannot be serviced by its local node i, it is forwarded to a remote node j, which contains a replica of the requested object. We term this request "global", as its service requires some capacity of the global switch, i.e., to deliver data from the remote node, through the global switch to the local node and subsequently to the client.

Each PU has one or more CPUs, memory, and an I/O sub-system (e.g., a cluster/array of disks), and it is also connected to the global switch. Each node $x \in S$, where S is the set of nodes in the system, has a finite storage capacity, D_x (*in units of CM objects*), as well as a finite service capacity, B_x (*in units of CM access streams*). For instance, consider a server that supports delivery of MPEG-2 video streams where each stream has a bandwidth requirement of 4 Mbits/s and each corresponding video file is 100 minutes long. If each node in such a server has 20 MBytes/s of bandwidth capacity and 36 GB of storage space, then each such node can support $B_x = 40$ simultaneous MPEG-2 video streams and store $D_x = 12$ MPEG-2 video objects[3]. Likewise, we measure the global and local switch capacities *in units of access streams*. In general, different nodes in such a *hybrid* system may differ in their storage,

[3] The constant bit rate description is used for simplification of this example only.

I/O bandwidth, and local switch service capacity. This flexibility of the hybrid architecture should result in a scalable system which can grow on a node by node basis.

Each CM object resides on one or more nodes of the system depending on its current popularity. An object is striped *only* across local disks which belong to the same node. Objects that require more than a single node's service capacity (to support the corresponding demand) are replicated on multiple nodes. The number of replicas needed to support requests for a CM object is a function of demand, and therefore this number should change as the demand changes.

Let $R_i(t)$ denote the set of nodes containing replicas of object i at time t. Upon a customer's arrival at time t, there is a probability $p_i(t)$ that the corresponding request is for object i and a probability $q_j^i(t)$ that this request is generated by a client local to node j. The admission of this customer into the system proceeds as follows. If at time t object i resides on node j and there is service capacity available at node j, then the system admits and serves this new request at node j, i.e., locally. Let $L_x(t)$ be the load on node x at time t. If at time t object i does not reside on node j or there is no service capacity available at node j, then the system examines the load information on each node in $R_i(t)$, and if there is sufficient capacity (on at least one of these node and in the global switch), to service the newly arrived request, the system assigns this request to the least-loaded node in $R_i(t)$. Otherwise, the customer cannot be served immediately. In this case we consider two different cases: (1) the customer is immediately rejected, or (2) the customer joins a FIFO queue and awaits service (there is no restriction on the size of the queue). We refer to case (1) as the no queueing case and to case (2) as the queueing case. These are two extreme cases (i.e., allowing no queueing at all or allowing infinite queueing). And, although these are not necessarily representative of how a real system should operate[4], they are useful in our performance evaluation study, i.e., these cases provide the necessary insight for the design of CM server (as we will illustrate in the remainder of the paper).

Note that a "stream migration" approach to dealing with more "short-term" fluctuations in access patterns is given in [26]; this is an orthogonal approach and can be combined with techniques presented in this paper. However, in the interest of isolating the performance effects of dynamic replication we do not consider this here further. We also note that the problem of determining whether or not there is "sufficient capacity" (under either CBR or VBR stream models) is orthogonal to the problems studied in this work; much literature exists on this topic (refer to [13]), and such solutions can be used in conjunction with policies developed in this paper.

Full VCR functionality (i.e., fast-forward, rewind, and pause/resume) is available to all admitted

[4]In a real system one is likely to use a reasonably sized finite queue, whose size depends on the required performance characteristics. Hence, the actual size of the queue should be a function of the performance characteristics required by an application.

customers, with fast-forward and rewind provided at $n_{speed} > 1$ times the rate of normal playback. We assume that the user *views* the data as he/she is searching (e.g., fast-forwarding or rewinding) through it, and thus n_{speed} is *finite*. Let T^i_{np} be the mean amount of time that a customer spends in the normal playback mode, before entering some VCR function mode. And, let T^i_{ff}, T^i_{rw}, and T^i_{pause} be the the mean amount of time a customer spends in fast-forward, rewind, and pause modes, respectively, before returning to the normal playback mode. We also assume that the use of VCR functionality (such as fast-forward and rewind) does not require additional service capacity on the part of the CM server. This can be accomplished, for instance, by using techniques proposed in [4].

Note that, in a hybrid system we need to maintain load information on remote nodes and other bookkeeping information, which will require (a relatively small amount) of communication capacity; the exact amount depends on a particular implementation, and we leave these considerations to future work. Note also that in the case of wide data striping, the bookkeeping information must be exchanged between nodes to schedule *each* newly arrived request, whereas in hybrid architectures, we can tradeoff relying more on local (rather than remote) information for some loss in performance. Quantitative assessment of this tradeoff is left for future work.

To assess the *scalability* characteristics of the potential designs in an environment where data access patterns change over time, we consider the following cost/performance and reliability metrics:

1. the system's *acceptance rate*, in the no queueing case, which is defined as the percentage of all arriving customer requests that are accepted by the system;
2. the mean and variance of the waiting time as well as the mean and variance of the queue length in the queueing case. The mean waiting time is the mean amount of time a customer spends in the queue before he/she is served. The mean queue length is the mean number of customers in the queue awaiting service. The variances are defined accordingly and are used as a measure of QoS (as described later in the paper).
3. the capacity of the global switch required to support a particular architecture and corresponding acceptance rate;
4. the capacities as well as the number of local switches required to support a particular architecture and corresponding acceptance rate;
5. the amount of disk storage required to support a particular architecture and corresponding acceptance rate;
6. the mean time to failure (MTTF) of a particular architecture.

Table 1 summarizes the main notation used in this paper. We will define this notation throughout the paper, as it is needed.

S	set of all nodes in the system
N	number of nodes in the system; $N = \|S\|$
K	number of distinct objects in the system
B_x	maximum service capacity of node x (in streams)
$\bar{B}$	average service capacity of the nodes in the system (in streams)
C_x	maximum storage capacity of node x (in streams)
$L_x(t)$	load on node x at time t (in streams)
$A_i(t)$	available service capacity for object i at time t; $A_i(x) = \sum_{x \in R_i(t)}(B_x - L_x(t))$
$ReTh_i$	replication threshold, i.e., the threshold for adding another copy of object i
$DeTh_i$	de-replication threshold, i.e., the threshold for removing a copy of object i
D_i	difference between the replication and the de-replication thresholds, i.e., $D_i = ReTh_i - DeTh_i$
T^i_{length}	length of object i
λ	average arrival rate to the system
$R_i(t)$	set of replica nodes for object i at time t
$p_i(t)$	probability of an arriving request being for object i at time t
$q^i_j(t)$	probability of a request for object i being made at time t by a client local to node j
n_{speed}	ratio between the speed of fast forward (or rewind) and the speed of normal playback
T^i_{np}	mean amount of time a customer spends in normal playback mode each time it enters that mode, for object i
T^i_{ff}	mean amount of time a customer spends in fast forward mode each time it enters that mode, for object i
T^i_{rw}	mean amount of time a customer spends in rewind mode each time it enters that mode, for object i
T^i_{pause}	mean amount of time a customer spends in pause mode each time it enters that mode, for object i

Table 1: Summary of notation.

3 Dynamic Replication

We consider a *dynamic* approach to reacting to changes in user data access patterns. Since the number of copies of object i *partly* determines[5] the amount of resources available for servicing requests for that object, we adjust the number of replicas maintained by the system *dynamically*. Of course, the system's performance depends on its ability to make such adjustments *rapidly* and *accurately*.

3.1 General Approach and Main Tradeoff

Given the distributed server described in Section 2, such a dynamic replication approach gives rise to several interesting design issues, including:

1. *when* is the right time for the system to reconfigure the number of replicas, i.e., when to create an additional copy of an object and when to remove a copy;

[5] Other factors include requests for other objects being made at the same time.

2. to *which* node should a (new) replica be added or from *which* node should a no longer (deemed) useful replica be removed; and

3. what are *proper policies* for actually creating a new replica (or removing a no longer useful one).

In the remainder of this paper, we discuss and evaluate techniques that address these issues in an *efficient* manner. As already stated, the system's *acceptance rate* is our performance metric in the no queueing case, which is defined as the percentage of all arriving customer requests that are accepted by the system (with zero waiting time; refer to Section 2 for the relevant discussion); the system mean queue length and mean waiting time (as well as the respective variances) are used as performance metrics in the queueing case. We note that unless otherwise stated, the following discussion is given in the context of the no queueing case[6].

In general, a replication process of a CM object has a *source* node (which currently contains a copy of object i) and a *target* node (on which we are placing a new copy of object i)[7]. One simple approach to performing the replication is to "inject" a single replication stream into each of the source and target nodes, for reading and writing of the replica, respectively. We refer to this strategy as *sequential* replication. The sequential replication policy results in a relatively small increase in load on the source and target nodes, i.e., equal to the bandwidth requirements of a single user stream. However, such a policy results in a relatively long replication time (i.e., the replication time is equal to the playout time, at the normal display rate, of the object being replicated), and consequently many customers may be rejected/queued during the replication period due to lack of resources for *that* object, i.e., a lack of *other nodes* in the system, that can service requests for *that* object.

Clearly, one approach to reducing the replication time would be to increase the rate at which the replication is performed, i.e., to read (write) the CM object from (to) the source (target) node at M times the rate of a single stream. This, of course, requires M times the bandwidth of a single user stream on both the source and the target nodes. We refer to this strategy as *parallel* replication, i.e., conceptually this is equivalent to using M single streams in parallel to do the replication. Although this approach reduces the replication time, it also creates an additional load on both, the source and the target nodes, which could result in rejection of customer requests, possibly for CM objects *other* than the one being replicated, due to lack of resources on the *source* and/or *target nodes*, which are being used by the replication process. Thus, we essentially have conflicting goals of (a) using as

[6] At first, we do not consider queueing of customers that cannot be admitted immediately, as it would entail consideration of scheduling policies for queued requests; these are a function of customers' willingness to wait and the corresponding application. That is, in the interest of isolating performance characteristics of data layout policies and dynamic replication schemes, we first consider the no queueing case. Later, we also consider a simple queueing policy (i.e., FIFO) in order to further explore characteristics of the data layout and dynamic replication policies.

[7] We will discuss source and target node selection in Section 3.4.

few resources as possible to perform the replication (in order not to interfere with "normal" system operation) while (b) trying to complete the replication process as soon as possible.

3.2 Early Acceptance

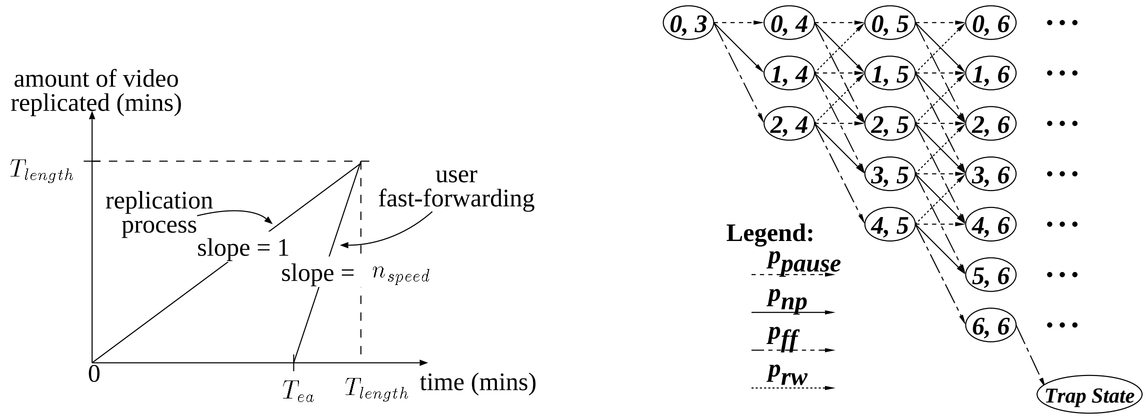
In an attempt to reach a compromise between the conflicting goals stated in the previous section, we consider "early acceptance" of customers, where admitted customers are allowed to use *incomplete* replicas (while the replication is in process). That is, once the system completes replication of the first T_{ea}^{i} time units of a new replica of a object i, it will treat it as a "virtually" complete copy and begin using it in servicing customer requests for object i. We are motivated by the continuous nature of objects, and essentially it is this property that we exploit here. Note that, for ease of presentation, in the remainder of the paper, we measure the amount of replication completed in time units of *normal playback time of that object*, from the beginning of the object, rather than in storage size, e.g., bytes. Furthermore, for simplicity and clarity of exposition of ideas, in the remainder of this section, much of the discussion is in terms of a specific object being replicated, and thus we drop the superscript i from our notation (the meaning should still be clear from the context of the discussion).

The issue that we need to consider is that a user might attempt to access a portion of an incomplete copy which has not been replicated yet, e.g., by fast-forwarding past the replication point. To allow customers to have full use of VCR functionality, we need to determine a "safe" value for T_{ea}. Clearly, one safe value is $T_{ea} = T_{length}$ (full length of the CM object). However, the intuition is that a smaller value of T_{ea} should result in a higher (at least in the "short term") acceptance rate of customer requests.

In order to lower T_{ea} (and improve performance) we construct a model of user behavior which allows us to compute a "safe" (but lower than T_{length}) value of T_{ea} while still providing the desired QoS (with a high probability). Below, we give a deterministic and a stochastic approach to this problem.

3.2.1 Deterministic Model

Given that the replication process constructs a new copy of an object, from the beginning to the end of the object (i.e., in a "linear" fashion), and using only knowledge of the ratio between normal playback and fast-forward, i.e., n_{speed}, we can construct a very simple model which will allow us to compute T_{ea}, as illustrated in Figure 3(a). Specifically, if a newly arrived customer is allowed to use an incomplete replica *after* $T_{ea} = T_{length}\frac{(n_{speed}-1)}{n_{speed}}$ time units of the object have been replicated, then he/she will *not* access data beyond the replication point with probability 1. Thus, if we use this *deterministic* model for admission of customers to the new replica, then (under a *sequential* policy, refer to Section 3.1) the "virtual replication completion time" of an object becomes T_{ea} as compared to T_{length}.



(a) Deterministic Model

(b) Stochastic Model
State Transition Diagram for $\mathcal{M}$ with $n_{speed} = 2$, $T_{ea} = 3$

Figure 3: Mathematical Models of User Behavior.

3.2.2 Stochastic Model

To further lower the value of T_{ea}, we employ a stochastic model of user behavior, at the cost of lowering the probability that the user will not access data beyond the replication point (of course, this probability still has to be high, but less than 1). Specifically, we model the combination of the behavior of a user watching a display of a partially replicated object and the corresponding replication process using a Discrete Time Markov Chain (DTMC)[8], $\mathcal{M}$, with the following state space $\mathcal{S}$:

$$\begin{aligned}\mathcal{S} \;=\; \{(V,R) \;\mid\; & (0 \le V \le T_{length}) \;\wedge\; (T_{ea} \le R \le T_{length}) \;\wedge \\ & (V \le R) \;\wedge\; ((R-V) \le T_{length}\frac{n_{speed}-1}{n_{speed}})\} \cup \{(\text{Trap State})\}\end{aligned}$$

where V is the current viewing position of the customer and R is the current replication position of the *partial* copy being viewed by that customer.

An example state space for $\mathcal{M}$ with $n_{speed} = 2$ is illustrated in Figure 3(b). There are 4 types of state transitions between adjoining states, which are attributed to: (1) normal playback, (2) fast-forward, (3) rewind, and (4) pause, which occur with probabilities p_{np}, p_{ff}, p_{rw}, and p_{pause}, respectively. A more formal specification of the state transitions in $\mathcal{M}$ with a corresponding one step transition matrix,

[8] Although by using a DTMC as our *model* we make the assumption that the amount of time a user spends in a particular playback mode has a memoryless distribution, we show in Section 4 that the performance of the *system* is fairly insensitive to either the *parameters* of the model or to the *distributional* assumptions.

$\boldsymbol{P}$, is as follows:

$$
\begin{array}{lll}
(V,R) & \longrightarrow & (min(V+n_{speed}*t_u, T_{length}), R+t_u) \\
 & & \text{Prob} = p_{ff}\mathbf{1}\{\ ((V+n_{speed}*t_u) \leq (R+t_u)) \wedge ((R-V) < T_{length}\frac{n_{speed}-1}{n_{speed}}) \wedge (R < T_{length})\ \} \\
(V,R) & \longrightarrow & (max(V-n_{speed}*t_u, 0), R+t_u) \\
 & & \text{Prob} = p_{rw}\mathbf{1}\{\ (R < T_{length}) \wedge ((R-V) < T_{length}\frac{n_{speed}-1}{n_{speed}})\ \} \\
(V,R) & \longrightarrow & (V, R+t_u) \\
 & & \text{Prob} = p_{pause}\mathbf{1}\{\ (R < T_{length}) \wedge ((R-V) < T_{length}\frac{n_{speed}-1}{n_{speed}})\ \} \\
(V,R) & \longrightarrow & (V+t_u, R+t_u) \qquad \text{Prob} = p_{np}\mathbf{1}\{\ (R < T_{length})\ \} \\
(V,R) & \longrightarrow & (\text{Trap State}) \qquad \text{Prob} = p_{ff}\mathbf{1}\{\ ((V+n_{speed}*t_u) > (R+t_u)) \wedge (R < T_{length})\ \} \\
(V,R) & \longrightarrow & (V,R) \qquad \text{Prob} = 1\mathbf{1}\{\ ((R-V) = T_{length}\frac{n_{speed}-1}{n_{speed}}) \vee (R = T_{length})\ \} \\
(\text{Trap State}) & \longrightarrow & (\text{Trap State}) \qquad \text{Prob} = 1
\end{array}
$$

where $\mathbf{1}\{x\}$ is an indicator function (i.e., $\mathbf{1}\{x\} = 1$ if x is true and 0 if x is false), and t_u is the "granularity" of our model, i.e., the number of time units in an object's display (under normal playback) corresponding to a unit of time[9] in the DTMC $\mathcal{M}$. Finally,

$$
\begin{aligned}
p_{ff} &= \frac{T_{ff}}{(T_{np}+T_{rw}+T_{pause}+T_{ff})} \\
p_{rw} &= \frac{T_{rw}}{(T_{np}+T_{rw}+T_{pause}+T_{ff})} \\
p_{pause} &= \frac{T_{pause}}{(T_{np}+T_{rw}+T_{pause}+T_{ff})} \\
p_{np} &= 1 - p_{ff} - p_{rw} - p_{pause}
\end{aligned}
$$

where T_{ff}, T_{np}, T_{rw}, and T_{pause} are application-dependent model parameters which were defined in Section 2, and the "Trap State" in $\mathcal{M}$ is a state corresponding to $V > R$, which represents a user's attempt to access data which has not been replicated yet.

Our goal then is to determine a value of T_{ea} for which the probability of entering the "Trap State" *before the time the replication process completes* (i.e., before $R = T_{length}$) is sufficiently low. Or, conversely, given a value of T_{ea}, we need to compute the probability of entering the "Trap State" by time $t_n < T_{length} - T_{ea}$, which can be accomplished through a *transient* analysis of $\mathcal{M}$ [24], i.e., by solving the following set of equations[10]:

$$\boldsymbol{\pi}(t_n) = \boldsymbol{\pi}(0) * \boldsymbol{P}^{t_n} \quad \text{and} \quad \sum_{j \in \mathcal{S}} \pi_j(t_n) = 1 \tag{1}$$

where $\boldsymbol{\pi}(t_n)$ is the vector of transient state probabilities at time t_n, $\boldsymbol{\pi}(0) = \boldsymbol{e}_{(0,T_{ea})}$ is the initial state vector which is equal to a row vector of 0's in all components except for a 1 in the component

[9] For instance, if the object is a video clip, then a "natural" time unit in its display would be the amount of time corresponding to the normal playback time of a single frame (on the order of $(\frac{1}{30})^{th}$ of a sec). However, in order to maintain a reasonable size of the DTMC state space, we allow t_u to take on larger time scales, e.g., on the order of minutes — essentially, performing (in general, approximate) aggregation of states.

[10] More sophisticated methods for computing transient results exist [24], but are not the subject of this paper.

corresponding to state $(0, T_{ea})$. Our interest then is in $\pi_{(\text{Trap State})}(t_n)$, which is the probability that the user will attempt to access data which has not been replicated yet.

Clearly, the efficiency of solving the above solution depends on the size of $\mathcal{M}$, which is finite but can be quite large. For instance, with $T_{length} = 90$, $t_u = 1$ min, and $n_{speed} = 2$, the size of $\mathcal{M}$'s state space is on the order of 3000 states. We can trade off computational cost for the system's performance by using higher values of t_u, e.g., for $t_u = 2$ min, the state space can be reduced to approximately 750 states. This modification can result in higher values of T_{ea}, due to a "coarser granularity" of the model, and hence the (potential) loss in the system's performance.

In any case, a simple approach to determining the value of T_{ea} would be to solve the model (possibly) multiple times (e.g., using binary search), with different values of T_{ea}, until a desired value for $\pi_{(\text{Trap State})}(t_n)$ is obtained, which, corresponds to the desired QoS to be provided by the system. We will illustrate in Section 4 that it is not necessary to obtain extremely low values for $\pi_{(\text{Trap State})}(t_n)$ in order to provide a reasonable QoS — this is due to the fact that the model tends to be conservative, especially with higher values of t_u.

In general, this is an acceptable approach since it only needs to be performed rarely, on a per application basis. That is, a set of statistics or measurements corresponding to interactivity characteristics of an application intended to be run on the CM server can be used to compute the model parameters (i.e., T_{np}, T_{ff}, T_{rw}, and T_{pause}), needed to solve $\mathcal{M}$. We will show in Section 4 that the model is not very sensitive to the accuracy of the input parameters and thus is of reasonably practical use — this is partly due to its conservative nature. Therefore, we expect that the need for "re-solving" of the model with new parameters would be quite rare and occur only after significant changes in the *interactive nature* of an application have been detected.

However, if the state space of $\mathcal{M}$ is still unacceptable or a more "run-time" approach to computing T_{ea} is desirable, instead of increasing the value of t_u, we can reduce the size of the state space by decreasing the amount of information included in the model about the user's behavior. Again, this reduction in computational cost results in more *conservative* estimates of T_{ea}, and thus we would be trading off system's performance for cost of the solution (for reasons similar to the ones stated above). We elaborate on this approach next.

We should note briefly that simple Markov chain models of user behavior have been employed in previous works on video servers, e.g., the two state Markov chain in [16]; however, these have been used for a somewhat different purpose and to the best of our knowledge, with interest in *steady state* characteristics only.

3.2.3 Reduction of the Stochastic Model

We can reduce the size of the state space and the number of transitions by not including all the information about user behavior in the DTMC. For instance, the state space and the number of transitions can be reduced by not including explicitly pause and rewind functionalities in the DTMC but rather "grouping" rewind and pause with the normal playback mode. This is still "safe" since pause and rewind can not cause the viewer to access an unreplicated portion of the data. That is, the reduced DTMC, $\mathcal{M}^r$, would have 2 types of state transitions between adjoining states, which are attributed to: (1) normal playback (with rewind and pause "grouped" here) and (2) fast-forward, which occur with probabilities p^r_{np} and p^r_{ff}, respectively. Similarly, we could have created another DTMC with only one of, pause or rewind, not explicitly included. We omit these variations since they are very similar in form to the one presented in this section.

More formally, the reduced DTMC, $\mathcal{M}^r$, has the following state space $\mathcal{S}^r$:

$$\begin{aligned}\mathcal{S}^r \;=\; & \{(V,R) \;\mid\; (0 \le V \le T_{length}) \;\wedge\; (T_{ea} \le R \le T_{length}) \;\wedge \\ & (V \le R) \;\wedge\; ((R-V) \le T_{length}\frac{n_{speed}-1}{n_{speed}}) \wedge ((R-V) \le T_{ea})\} \cup \{(\text{Trap State})\}\end{aligned}$$

where as before V is the current viewing position of the customer and R is the current replication position of the partial copy being viewed by that customer.

An example state space for $\mathcal{M}^r$ with $n_{speed} = 2$ is illustrated in Figure 4.

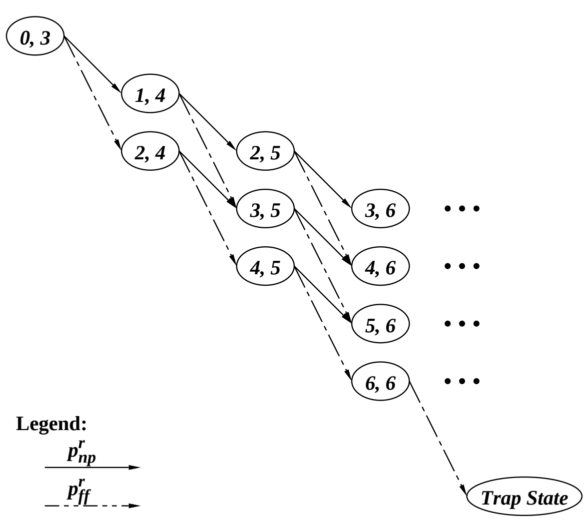


Figure 4: State Transition Diagram for $\mathcal{M}^r$ with $n_{speed} = 2$, $T_{ea} = 3$.

A more formal specification of the state transitions of $\mathcal{M}^r$ with a one step transition matrix, $\boldsymbol{P}^r$, is as follows:

$$
\begin{array}{llll}
(V, R) & \longrightarrow & (min(V + n_{speed} * t_u, T_{length}), R + t_u) & \\
 & & \multicolumn{2}{l}{\text{Prob} = p^r_{ff} \mathbf{1}\{ \ ((V + n_{speed} * t_u) \leq (R + t_u)) \wedge ((R - V) < T_{length} \frac{n_{speed}-1}{n_{speed}}) \wedge (R < T_{length}) \ \}} \\
(V, R) & \longrightarrow & (V + t_u, R + t_u) & \text{Prob} = p^r_{np} \mathbf{1}\{ \ (R < T_{length}) \ \}] \\
(V, R) & \longrightarrow & (\text{Trap State}) & \text{Prob} = p^r_{ff} \mathbf{1}\{ \ ((V + n_{speed} * t_u) > (R + t_u)) \wedge (R < T_{length}) \ \} \\
(V, R) & \longrightarrow & (V, R) & \text{Prob} = 1 \mathbf{1}\{ \ ((R - V) = T_{length} \frac{n_{speed}-1}{n_{speed}}) \vee (R = T_{length}) \ \} \\
(\text{Trap State}) & \longrightarrow & (\text{Trap State}) & \text{Prob} = 1
\end{array}
$$

where

$$
p^r_{ff} = \frac{T_{ff}}{(T_{np} + T_{rw} + T_{pause} + T_{ff})} \quad \text{and} \quad p^r_{np} = 1 - p^r_{ff}
$$

and T_{ff}, T_{np}, T_{rw}, and T_{pause} are as defined in Section 2, and again the "Trap State" is a state corresponding to an aggregate of all states where $V > R$, which represents a user's attempt to access data which has not been replicated yet.

As in the previous section, we can perform transient analysis on $\mathcal{M}^r$ to determine a "safe" value for T_{ea} with a sufficiently high probability of not entering the "Trap State" before the replication process completes.

Clearly, the cost of the solution will be reduced, as compared to $\mathcal{M}$, given the reduction in the state space and the number of transitions. For instance, with $T_{length} = 90$, $t_u = 1$ min, and $n_{speed} = 2$, the size of $\mathcal{M}^r$'s state space is on the order of 500 states (as compared to $\approx$ 3000 states in $\mathcal{M}$), where a brute force solution of Equation (1) takes several minutes to compute using MATLAB numerical solutions package on a Sun Ultra-1 machine.

As before, we can trade off computational cost for system's performance (and obtain a more conservative solution) by using higher values of t_u, e.g., for $t_u = 2$ min, the size of the state state is reduced to approximately 130 states. In this case, a brute force solution of Equation (1), requires less than one minute. As we mentioned before, better than brute force transient analysis techniques can be used to further improve on this time [24].

3.3 Threshold-based Activation Scheme

We use a *threshold-based* approach to trigger object replication and de-replication, both of which are only triggered at customer *arrival* and/or *departure* instances. Threshold-based techniques for reacting to changes in workload are employed often for improving the cost/performance ratio of a system, e.g., in communication protocols [17]. Here, as in other systems, the main motivation for using a threshold-based scheme is that there is a non-negligible cost for creating or removing a replica.

That is, it takes a non-negligible amount of time to replicate an object or remove[11] a copy, and thus it should be done "sparingly".

Furthermore, in such an environment, having the amount of service capacity proportional to the access probabilities (even if we knew them) would not necessarily insure acceptance of newly arrived customers. An important factor in the performance of the system is the mixture of requests that arrives and is ultimately serviced by the nodes of the CM server. That is, we may reject requests for object i on node j due to an influx of requests for other objects residing on node j, i.e., other than object i.

Thus, in this paper we study dynamic data replication and de-replication techniques which do *not* assume knowledge of access probabilities. Without such information, one simple approach is to increase (decrease) the amount of service capacity allocated to an object when the amount of available resources left in the system to service that object falls below (above) some threshold value.

More formally, when a customer request for object i arrives to the system at time t, replication of object i is initiated if and only if *all* of the following criteria are satisfied:

1. $A_i(t) < ReTh_i$, where $ReTh_i$ is the replication threshold and $A_i(t) = \sum_{x \in R_i(t)} (B_x - L_x(t))$;
2. object i in not currently under replication;
3. there is sufficient available service capacity on the source node;
4. there is sufficient available storage space capacity **and** service capacity on the target node;
5. there is sufficient available service capacity in local switches as well as the global switch (i.e., interconnection network).

In the case of de-replication, it should be performed before the system runs out of storage space. Basically, we do not want to leave this decision until the time the system actually *needs* the space for creating a new replica. This is due to the fact that there might be customers using the copy that we would like to delete, and either we will have to wait for them to complete their display, or we will have to relocate them. "Planning ahead" for removing copies of "cold" objects before the space is actually needed should improve the system's performance.

De-replication is invoked at both the customer request arrival and departure instances. More formally, a replica of object i at node x will be removed at time t if and only if the following conditions are satisfied:

[11] Even removal time can be significant, since the copy may be utilizes by users that, e.g., would have to be migrated to other nodes, at the time of removal, again due to real-time constraints on data delivery.

1. $A_i(t) = \max_{j \in S}\{A_j(t) > ReTh_i\}$. The motivation for this condition is that the number of replicas for object i at time t is more than its current workload demand and at this time it has the greatest excess of replicas among all relatively "cold" objects.

2. i has "crossed" the de-replication threshold, i.e.,

$$A_i(t) - (B_x - L_x(t)) - C_{ix}(t) > DeTh_i \tag{2}$$

   where $C_{ix}(t)$ denotes the number of customers viewing object i at node x at time t. With the deletion of object i at node x, $A_i(t)$ would be decreased by $(B_x - L_x(t))$. Since a customer viewing object i at node x will have to be migrated to other replica nodes in $R_i(t)$, $A_i(t)$ would be further decreased by $C_{ix}(t)$.

3. $\sum_{x \in S} D_x(t) < D_S$, where D_S is the storage space threshold for activating de-replication.

4. In the case of the Delayed Migration (DM) de-replication policy *only* (see Section 3.4.2), there is an additional condition, namely that $C_{ix}(t)$ must be equal to 0.

To prevent the system from oscillating between replication and de-replication, a difference of D_i is introduced between $ReTh_i$ and $DeTh_i$, i.e., $DeTh_i = ReTh_i + D_i$. That is, we introduce *hysteresis* into the system.

Thus, we use dynamic replication and de-replication techniques which do *not* assume knowledge of access probabilities. To improve threshold estimation in the absence of access statistics, we use the last interarrival time for object i to (coarsely) "approximate" $p_i(t)$ and compute threshold values as follows:

1. For each object i, we record its last request access time, at_i. At arrival time, t, of a request for object i, we compute its latest interarrival time, $(t - at_i)$, and use it as a coarse "approximation" of $p_i(t)$. Whenever a new request for object i arrives, we record at_i and update the thresholds for the object i accordingly.

2. Then, $ReTh_i = \lceil \frac{T^i_{ea}}{t - at_i} \rceil$. That is, the replication threshold, $ReTh_i$, represents the amount of workload, corresponding to requests for object i, that are expected to arrive in the next T^i_{ea} time units, which is the amount of time needed to create a new (virtual) replica of object i (should we deem it necessary). Thus, the motivation for this setting of the replication threshold values is that if we have fewer resources than are estimated to be needed in a period representing the amount of time needed to create another replica (i.e., add resources), then we increase the number of replicas. That is, we attempt to match the available resources with anticipated workload.

3. Lastly, $DeTh_i = ReTh_i + H_i$, where $H_i = \lceil \frac{T^i_{length}}{t - at_i} \rceil$, i.e., we introduce a hysteresis. The motivation for this setting of the hysteresis value is similar to the motivation given above for

$ReTh_i$, except that $\frac{T^i_{length}}{t-at_i}$ corresponds to the expected number of requests for object i that might arrive in the next T^i_{length} time units, i.e., during the entire duration (in normal playback) of the display of object i. That is, T^i_{length} corresponds to an estimate of the amount of time that will elapse before some of the currently allocated resources, which can be used to service requests for object i, are released. That is, we can release resources that are in access of what we anticipate is needed in that time period.

Note that in this work we dynamically adjust threshold values[12] based on minimal amount of information, i.e., the last inter-arrival time of a request for object i. This is in contrast to the static threshold values we used in [7]. Extensive simulations showed that the system's performance (using the metrics described earlier) is significantly improved through the use of dynamic threshold values. The tradeoff is the need to adjust thresholds and the need to collect some information (i.e., the latest inter-arrival time of requests for each object in the system), which we note is fairly small. Hence, in the remainder of this paper, we only consider our replication policies under dynamic threshold values.

3.4 Policies

In this section we describe the node selection, replication, and de-replication policies of the CM server.

3.4.1 Selection Policies

Firstly, the choice of a source node for replication of object i is simple: we select the least-loaded node in the set $R_i(t)$. For the target node, we choose the node which has the highest estimated residual service capacity (in streams) and has available storage capacity. More formally, we choose the node x such that $x \notin R_i(t)$ **and** $L_x(t) = \max_{y\in(S-R_i(t))}\left\{\frac{B_y-L_y(t)}{1+\gamma_y(t)}\right\}$, **and** the remaining storage capacity on x is sufficient for the new replica, where $\gamma_y(t)$ corresponds to the number of replication processes already in progress on node y at time t. Intuitively, such a choice should avoid replication of multiple relatively popular objects on the same target node (which may later compete for that node's capacity).

3.4.2 Replication Policies

We now describe the replication policies. Recall that T^i_{ea}, as determined in Section 3.2, represents the initial amount of data that must be replicated before customers are allowed to use a *partial* replica, and it is measured in units of *normal playback time* of that object. Thus, the *value* of T^i_{ea} is *independent* of the replication policy used, but how *long* it takes to copy T^i_{ea} time units worth of data is a function

[12] This is similar in spirit to the approach we use in [6]. However, in that case we only applied this to the SREA policy (refer to Section 3.4) and using a different set of threshold equations.

of the replication policy. For example, if replication proceeds at the same rate as playback (as in the sequential policies below), then the replication time will be equal to T^i_{ea}, but if replication proceeds at a faster rate (as in the parallel policies below), then the replication time will be smaller than T^i_{ea}.

Sequential Replication (SR): The replication is performed "sequentially" (as described in Section 2), i.e., the system replicates at the rate of normal playback of a single stream by injecting a single read stream at the source node and a single write stream at the target node — each of these requires the same capacity as a single user stream. Thus replication of object i takes T^i_{length} time units, and users are not admitted to the new replica until the *entire* copy is complete. This policy is considered for comparison purposes *only*.

Sequential Replication + Early Acceptance (SREA): The replication is performed as in the SR policy, except that newly arrived users can be admitted to the new (incomplete) replica as soon as T^i_{ea} time units of that object have been replicated on the target node. Furthermore, this "virtual" replication completion time is used in checking the satisfaction of condition (2) in the decision of when to create a new replica (see Section 3.3).

Parallel Replication (PR): The system replicates at M times the rate of a normal display of a single user stream, where $M = \min((B_{source} - L_{source}(t)), (B_{target} - L_{target}(t)))$ at time t, when replication begins. Thus the "real" replication time of object i is reduced to $\frac{T^i_{length}}{M}$, and users are not admitted to the new replica until the *entire* copy is complete. This policy is considered in order to show a contrast in performance between policies that do and do not utilize the *early acceptance* technique.

Parallel Replication + Early Acceptance (PREA): The replication is performed in the same manner as in the PR policy, except that users are admitted to the new (incomplete) replica after the first T^i_{ea} time units of the replica are completed. Furthermore, as in the case of the SREA policy, this "virtual" replication completion time is used in checking the satisfaction of condition (2) in the decision of when to create a new replica.

Mixed Parallel, Early Acceptance + Sequential Replication (MPEA): The first T^i_{ea} time units of the object are replicated as in policy PREA and the remainder of the object is replicated as in policy SREA. Users are admitted to the new (incomplete) replica after the first T^i_{ea} time units of the replica are complete. And, as in the other policies using early acceptance, the "virtual" replication completion time is used in checking condition (2) in the decision of when to create a new replica.

3.4.3 De-replication Policies

The decision process of *which* replica to remove, i.e., which object i, will be described in Section 3.3. What remains to determine is the choice of the node from which to remove it. Part of the difficulty is in considering the customers that would have to be migrated from the node where the removal occurs. We consider the following de-replication policies.

Delayed Migration (DM): This policy removes a replica of object i only after the last customer finishes viewing the movie. That is, we only remove the replica of object i at node x at time t when $C_{ix}(t) = 0$ in Equation (2). No new users are admitted to this replica after the de-replication decision is made. This is motivated by the (possible) implementation complexity of migrating customers from one node to another.

Immediate Migration Minimum Overhead (IMMO): This policy chooses the node on which fewest customers are currently viewing object i. The motivation here is to reduce the (possible) system overheads associated with user migration. That is, at time t the replica of object i is removed from node x where $C_{ix} = \min_{y \in R_i(t)}\{C_{iy}(t)\}$ in Equation (2); The C_{ix} customers are distributed evenly among the *remaining* nodes in $R_i(t)$.

Immediate Migration Maximum Capacity (IMMC): This policy selects the node which could provide the greatest estimated (residual) service capacity after the replica of object i is removed. That is, at time t the replica of object i is removed from node x where $C_{ix} = \max_{y \in R_i(t)}\{C_{iy}(t) + (B_y - L_y(t))\}$ in Equation (2); the C_{ix} customers are distributed evenly among the *remaining* nodes in $R_i(t)$.

4 Discussion of Results

In this section, we first present a scalability study of CM server designs in the context of data placement techniques where the main concern is the system's load balancing characteristics and the subsequent system performance. Then, we focus on a performance study of dynamic replication schemes, used in conjunction with the data placement techniques, with emphasis on their sensitivity to user model parameters, workload characteristics, and skewness of data access patterns, as well as applicability to various CM applications.

Table 2 lists the parameters considered here along with their default values/distributions and alternatives as used in the remainder of this section. All values are given in units of minutes, unless otherwise specified. (Refer to Section 2 for the definition of the notation used in Table 2.)

4.1 Scalability Study

In this section we present results of our simulation and analytical study using the cost/performance and reliability metrics given in Section 2. The arrival process (of requests for objects) is Poisson with a mean arrival rate of $\lambda = \frac{a\bar{B}N}{T^i_{length}}$, where $0 \leq a \leq 1$ is the "relative arrival rate". For ease of presentation, we discuss the results in terms of a, i.e., relative to the *total* service capacity of the system (e.g., $a = 1.0$ corresponds to the maximum service capacity of the system).

We consider the design of a CM server with the following capacity requirements: (1) a total service capacity of $N * \bar{B} = 1600$ streams; (2) a total storage capacity of $K = 400$ distinct objects; and (3) each object is of length $T^i_{length} = 90$ minutes.

Since the main motivation for using *dynamic* replication policies is the need to react to changes in data access patterns, we consider the performance of these policies as a function of such changes. That is, the workload will have the characteristic that every "rotation time period" of X minutes, $p_i(t)$s change. One change in access probabilities is described by Equation (3), which is intended to emulate a relatively "gradual" increase/decrease in popularities:

$$p_i(t') = \begin{cases} p_{i+2}(t) & \text{if } i \text{ is odd } \& \ 1 \leq i < K-1 \\ p_K(t) & \text{if } i \text{ is odd } \& \ i = K-1 \\ p_{i-2}(t) & \text{if } i \text{ is even } \& \ 2 < i \leq K \\ p_1(t) & \text{if } i \text{ is even } \& \ i = 2 \end{cases} \tag{3}$$

where t and t' refer to two consecutive rotation periods and for ease of presentation we assume that K is even. This is to illustrate that even under a relatively gradual change, dynamic policies are still useful. Furthermore, we believe this is a reasonable "emulation" of change in access patterns for many CM applications.

To test our policies further, we also emulate an "abrupt" increase in popularity of currently unpopular objects as well as a "gradual" decrease in popularity of the currently more popular objects as follows:

$$p_i(t') = \begin{cases} p_1(t) & \text{if } i = \text{K} \\ p_{i+1}(t) & \text{if } 1 \leq i \leq K-1 \end{cases} \tag{4}$$

Here, we consider the Zipf distribution [18], given in Equation (5), for skewness of access probabilities.

$$\text{Prob[request for object } i] = \frac{c}{i^{(1-\theta)}} \qquad \forall\ i = 1, 2, \ldots, K \quad \text{and} \quad 0 \leq \theta \leq 1 \tag{5}$$

$$\text{where} \quad c = \frac{1}{H_K^{(1-\theta)}} \qquad \text{and} \qquad H_K^{(1-\theta)} = \sum_{j=1}^{K} \frac{1}{j^{(1-\theta)}}$$

We set $\theta = 0.0$, which corresponds to the *measurements* performed in [5] (for a movies-on-demand application). In Section 4.2 we also consider a finite geometric distribution, for skewness of access probabilities.

The architectural settings considered in this section are the default parameters of Table 2 together with Table 3[13]. Here arch2w corresponds to wide data striping, where a single copy of each object is striped across all nodes of the system, and arch2 through arch5 groups correspond to various configurations of a *hybrid* CM server (as described in Section 2). For the hybrid architectures we experiment with different amounts of per node storage space capacity, in order to illustrate the tradeoff between storage space capacity local to a node and the corresponding required capacity of the global switch.

Moreover, we consider the affect on the overall system performance of limitations of communication network resources. Let nc represent the ratio of the global switch and the storage system service capacities, i.e., $nc = 1.0$ represents equal service capacities in the storage and communication subsystems. Then we vary the service capacity of the global switch, $0.1 \leq nc \leq 1$, and compute the subsequent degradation in performance experienced by the various architectures. The motivation for these experiments is to: (1) observe the performance degradation characteristics of possible CM server designs (as this is an indication of their scalability), and (2) assess whether reduction in overall required global switch capacity (which should lead to lower costs) is possible without significant loss in the overall system performance.

Lastly, below "upper bound" on the acceptance rate refers to the acceptance rate that a wide data striping system can achieve *without considering network capacity constraints*; thus this is the only curve in the following figures that is *not* a function of nc.

4.1.1 Wide Data Striping System vs. Hybrid System

Figures 5 and 6(b) illustrate that under lower network capacities, a hybrid system has better overall performance as well as performance degradation characteristics than the wide data striping system. More importantly, the hybrid architecture allows us to tradeoff capacities of the various system resources in order to achieve a more cost-effective system overall. Specifically, we can tradeoff local storage space and local switch capacities with global switch capacity and achieve nearly the same performance characteristics. For instance, for a particular architectural setting, the larger the local storage space capacity is, the smaller the global switch capacity need be, in order to achieve the same overall system performance, e.g., consider the "arch2 group" in Figure 5(a) — in the case of the 24

[13] For a hybrid system that requires more storage space than the corresponding wide data striping system we only increase the storage *space* per disk, *not* the number of disks, as that would also increase the service capacity and hence would not make for a fair comparison.

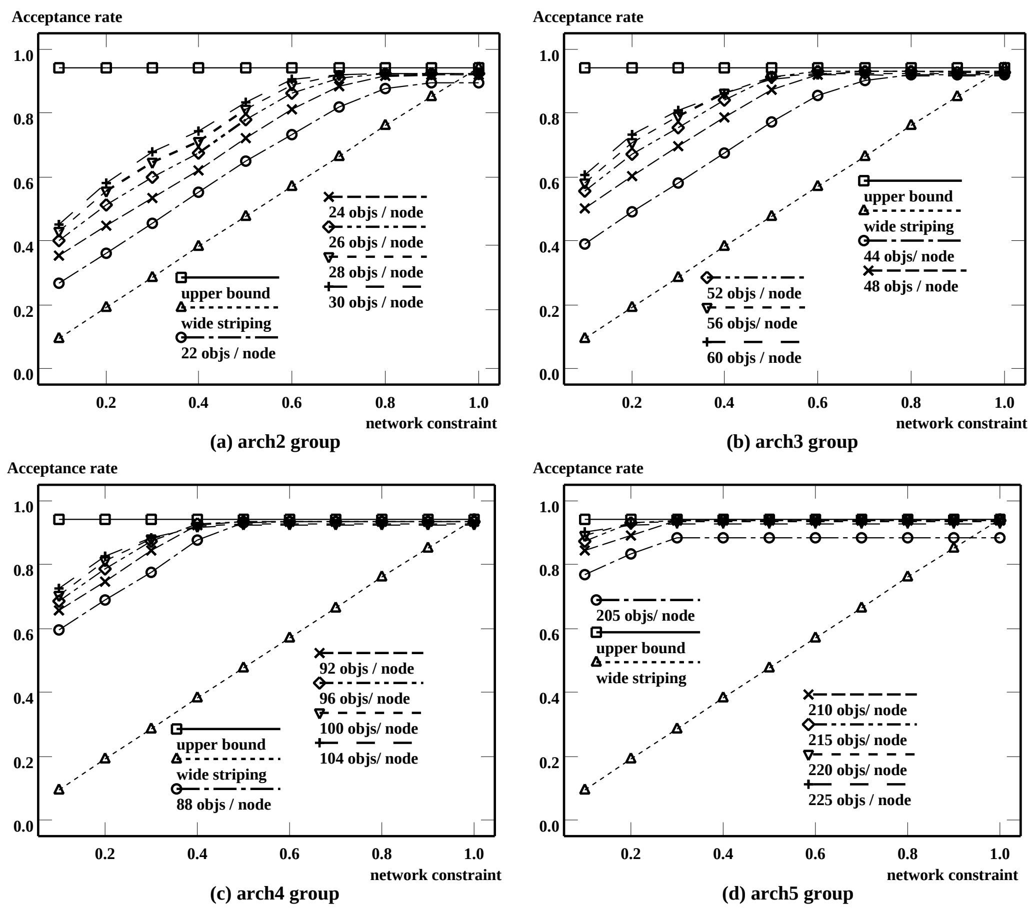


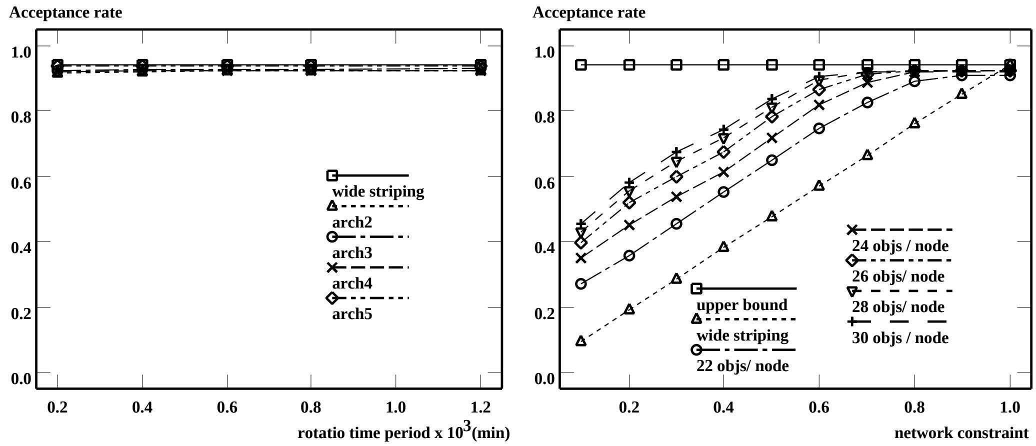
Figure 5: Different network constraints.

objects/node architecture, the corresponding required service capacity of the global switch[14] is 1280 streams, whereas in the case of the 30 objects/node architecture, it is only 960 streams.

Conversely, the larger the local switch is, the more we can reduce the storage space and global switch capacities, e.g., consider the "arch2 group" in Figure 5(a) — in this case with a local switch capacity of 80 streams[15], the corresponding required total storage space capacity is 600 objects (i.e., 30 objects/node) and the corresponding required service capacity of the global switch is 960 streams. Consider now the "arch4 group" in Figure 5(c) — although in this case the local switch capacity increases to 320 streams, the corresponding required service capacity of the global switch drops down to 640 streams and the corresponding required total storage space capacity drops down to 460 objects

[14] The needed global switch capacity is determined from Figure 5(a) by first fixing the acceptance rate that we would like to achieve. Here, we fix the required acceptance rate to be *at least* $0.95 *$ acceptance rate of the "upper bound result", and then determine, using Figure 5(a), the smallest network constraint, nc, that satisfies that acceptance rate for the appropriate architecture curve; then, the required global switch capacity is $nc * 1600$ streams where 1600 streams corresponds to the maximum required system capacity.

[15] These values can be determined from Tables 3 and 4.



(a) arch2, arch3, arch4 and arch5

(b) arch2 group

Figure 6: Abrupt increase/gradual decrease.

(i.e., 92 objects/node). These results are due to the fact that with larger local switch capacities, we can service more customer requests locally (hence the corresponding smaller required global switch capacity). Furthermore, larger local switches and corresponding larger node service capacities also provide more opportunities to take advantage of the load balancing characteristics of striping *within* a node (hence the smaller required storage space capacity per node).

4.1.2 System Sizing Issues

Quantitatively evaluating the tradeoffs between local switch capacity, node storage space capacity, and global switch capacity is no easy task, as it is not immediately clear how to tradeoff one resource for another. Ideally, one would like to evaluate these tradeoffs based on cost. However, cost considerations are a complex issue, given that costs depend on many factors. Thus, next we instead evaluate the different hybrid designs based on the amount of each resource they require *relative* to the wide data striping system. Such an evaluation quantitatively illustrates to the designer the *relative* merits of the different architectures, without the need for choosing a specific technology[16]. The purpose of these experiments is to illustrate how a CM server designer can deal with these (fairly complex) *system sizing* issues.

We further refine our test cases in Table 4, and choose the per node storage space and corresponding global switch capacity of each architecture based on the results of the previous section, i.e., we choose those architectures that can achieve an acceptance rate of *at least* 0.95∗ acceptance rate of the "upper

[16] Characterizing a resource using only its capacity may result in a simplification for certain types of resources; however, this is still a good abstraction for evaluating cost-effectiveness of designs, without having to choose a specific technology for each system component.

bound result" with reasonably small per node storage space and global switch capacities[17]. Figure 7 depicts the results of this comparison for each resource as

$$\frac{\text{resource requirement of arch } i}{\text{resource requirement of arch} 2w} \quad \forall i \neq 2w.$$

Hence, the straight line at the value of 1.0 in each of the graphs of Figure 7 corresponds to the ("scaled") resource requirement of "arch2w".

As already stated, these results illustrate to the designer the relative merits of the different architectures by quantifying the tradeoffs between the various resources of the CM server. Based on these results and current costs and technology trends, the designer can make system sizing decisions.

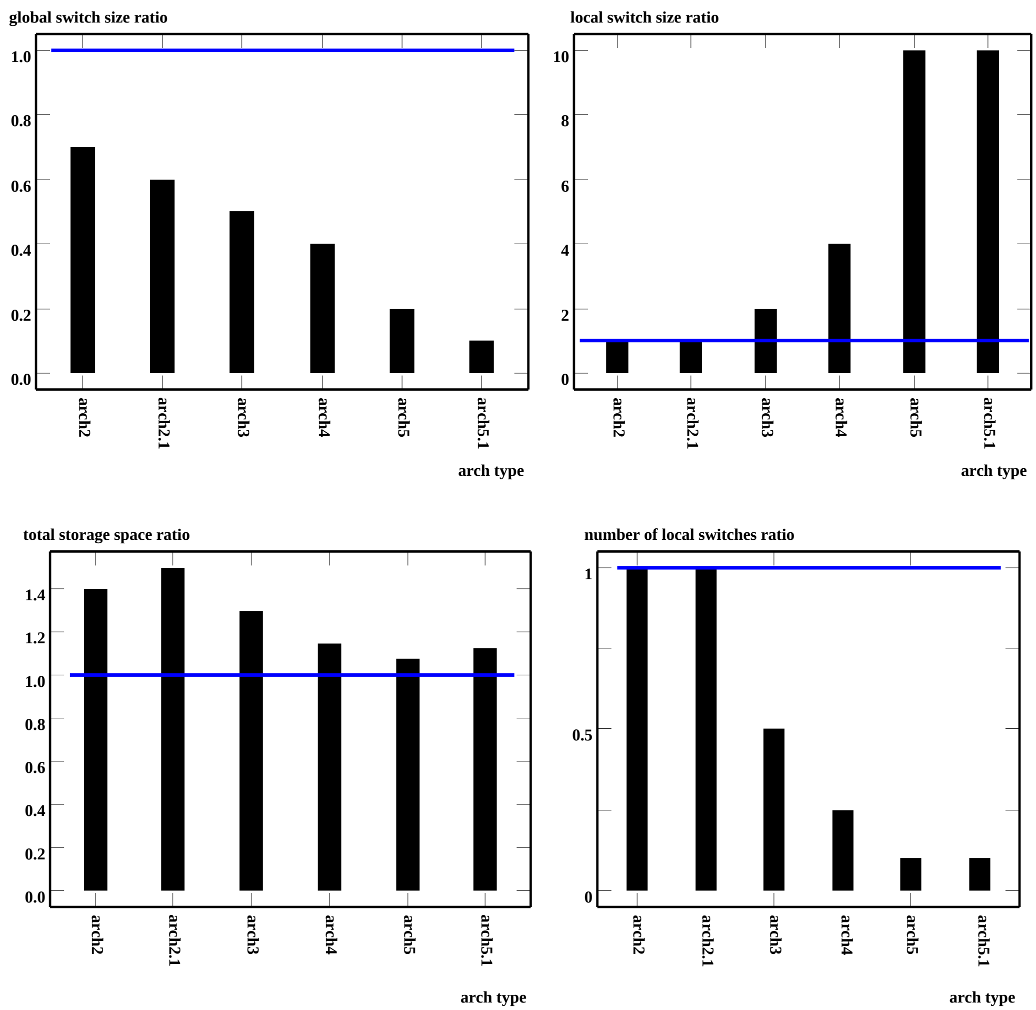
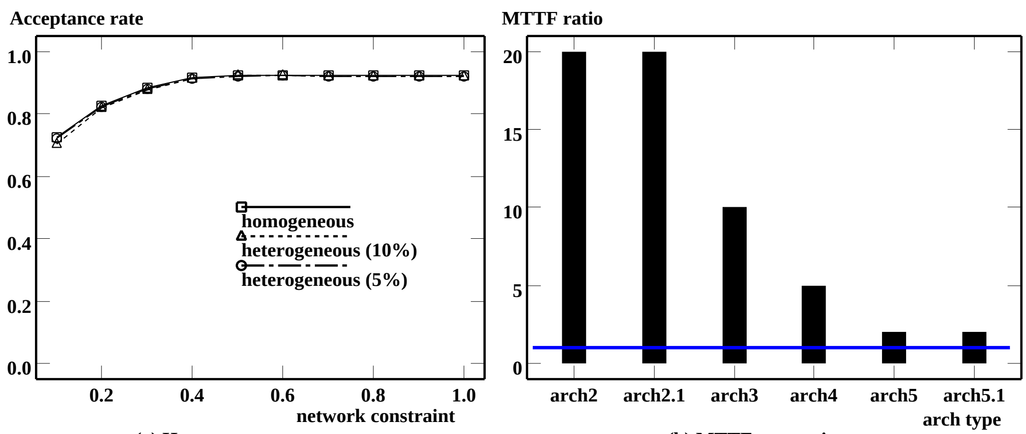


Figure 7: System sizing.

[17]The upper bound is computed without considering network capacity constraints; since it is not always achievable by an architecture, we choose a performance goal that is reasonably close.

4.1.3 Heterogeneous Systems

Next, we illustrate the ease of dealing with heterogeneous systems when using hybrid CM server designs *without* loss of performance as compared to an equivalent homogeneous case. For this purpose, we consider a hybrid CM architecture with 5 nodes and a *total* service capacity of 1600 streams. We use two test cases in the following experiments, both based on the homogeneous version of "arch4" with the storage space capacity of 104 objects per node (refer to Table 3). We introduce 5% and 10% differences in storage space and service capacities between the nodes of the system (as well as corresponding differences in local switch capacities), e.g., to emulate a system that gradually grows (as well as experiences replacements due to failures) and thus is forced to use heterogeneous resources[18]. The results, depicted in Figure 8(a), show that, using a hybrid design, we can achieve heterogeneous



(a) Heterogeneous systems (b) MTTF comparison

Figure 8: Heterogeneous systems and MTTF.

system performance that is comparable to homogeneous system performance.

4.1.4 Reliability

We use the mean time to failure (MTTF) as our reliability metric, which is defined as the mean time until some combination of disk failures results in loss of some data (i.e., losses that can no longer be recovered through the use of redundant information). We compare the architectures in Table 4, using a *conservative* estimate for the *hybrid* system based on an assumption of only a *single* copy per object. These results are depicted in Figure 8(b), as the following *ratio* between MTTF of a particular hybrid

[18] Hence, we have one test case of a 5 node system, with 84, 94, 104, 114, and 124 objects/node, respectively and service capacities of 256, 288, 320, 352, and 384 streams, respectively. And, we have another test case of a 5 node system, with 94, 99, 104, 109, and 114 objects/node, respectively and service capacities of 288, 304, 320, 336, and 352 streams, respectively.

architecture and the wide data striping architecture:

$$\frac{\text{MTTF of arch } i}{\text{MTTF of arch} 2w} \ \ \forall i \neq 2w.$$

where, the straight line at 1.0 corresponds to the ("scaled") MTTF of wide data striping. The derivations of the MTTF equations used to compute the results in Figure 8(b) are given in the Appendix.

These results clearly show that higher reliability can be achieved by hybrid systems, even for objects that only have a *single* copy, as compared to wide data striping. This increase in reliability is due to the "isolation" of fault affects, i.e., the wider we stripe an object, the more disk failures can affect it.

Of course, the reliability is even higher for objects with multiple copies, as is natural in a system which employs data replication. Thus, in a hybrid system, we can provide significantly higher reliability for the *popular* objects, as there will always be multiple replicas of such objects in a hybrid system. Lastly, under network failures or high workload conditions at remote nodes, local nodes can at least deliver *some* objects[19], which is not the case for wide data striping, as all nodes and network capacity must be available in order to serve a request for *any* object.

Summary

In summary, the main observations based on the experiments presented above are as follows:

- The use of hybrid designs allows us to tradeoff resources and specifically we can tradeoff a node's local storage space, its local switch capacity, and the system's global switch capacity given a particular performance requirement. This should allow a system designer to make better system sizing decisions by making appropriate tradeoffs, i.e., it should allow for cost-effective designs while satisfying the required system performance.
- Hybrid designs naturally extend to systems with heterogeneous resources with little or no loss in performance as compared to the corresponding homogeneous systems. Since in practice we expect CM servers to have heterogeneous resources as a result of system growth (due to expansion) and disk failures (and subsequent replacement), this is an important characteristic.
- Hybrid designs result in higher system reliability (as defined above) even under the conservative assumption of have a single copy per object (refer to the above discussion). Better reliability characteristics are of significant importance to CM servers since they service real-time workload and furthermore there is a need to recover from failure in real-time [3].

[19] For example, in a movies-on-demand application, even if a requested movie is not available, the user has the option to choose another movie that may be available.

4.2 Policy Performance Evaluation

In this section we conduct a number of experiments in order to understand the sensitivity of the dynamic replication policies to differences in architectures, user model parameters, workload characteristics, skewness in data access patterns, as well as applicability to different CM applications. This performance study is performed via simulation, with the simulation parameters[20] given in Table 2. We consider the following performance metrics (refer to Section 2 for details):

1. the system's acceptance rate (in the no queueing case), where the acceptance rate is defined as the percentage of all arriving customer requests that are accepted by the system;

2. the system mean queue length, mean waiting time, as well as the respective variances (in the queueing case).

There is a multitude of parameters that can be varied in studying performance of dynamic replication policies. And, we performed a great number of experiments [7]; of course, we cannot present them all. We note that the architectures chosen in Table 2 reflect the state of disk technology at the time of experiments, with respect to storage vs. bandwidth capacities, and hence the choice of presenting these results here. We also note, that with larger storage capacities (given the trend of more rapid advances in disk space rather than bandwidth) our policies are expected to perform even better (i.e., the ability to maintain a greater number of replicas results in better performance). Several other entries in Table 2 require a few words of clarification, which are as follows.

The default arrival process (of requests for objects) is Poisson[21] with a mean arrival rate of $\lambda = \frac{a\bar{B}N}{T^i_{length}}$, where $0 \leq a \leq 1$ is the "relative arrival rate". As before, we discuss the results in terms of the *relative* arrival rate, a, i.e., relative to the *total* service capacity of the system. In order to further explore the benefits of early acceptance schemes, we consider another arrival process, specifically an on-off process [23] which, with proper parameter settings, exhibits more bursty characteristics [22]. An on-off process is a two state Markov chain, i.e., one state is an "on" state and the other state is an "off" state. Let β be the transition rate from "on" state to "off" state and α be the transition rate from "off" state to "on" state. When the process is in the "on" state, one customer arrives every (deterministic) time interval $\frac{1}{A}$. There are no arrivals when the process is in the "off" state. In order to make a meaningful comparison, the parameters of the on-off arrival process, i.e., the deterministic arrival rate

[20] Most results presented here are obtained with $95 \pm 5\%$ confidence intervals; all results are within $95 \pm 10\%$ confidence intervals.

[21] We believe it is reasonable for us to consider a Poisson arrival process for purposes of this study, since user requests are essentially considered on a "per session" basis; refer to [20] for details that support the use of Poisson arrivals in this case.

[22] In our case, the on-off process with $\frac{1}{\beta} = 0.5$ minutes and $\frac{1}{\alpha} = 6$ minutes has the same mean arrival rate, λ, as the corresponding Poisson process. However, its variance is 12λ times the variance of the corresponding Poisson process.

A and transition rates β and α are set such that the on-off process has the same mean arrival rate as the Poisson process, i.e., $A\frac{\alpha}{\alpha+\beta} = \lambda$, but a higher variance, i.e., $A^2\frac{\alpha\beta}{(\alpha+\beta)^2}$.

As already stated, the default is the no queueing case, i.e., a customer is rejected if its request cannot be serviced immediately. To further understand the characteristics of the dynamic replication policies in conjunction with data layout techniques, we also consider the queueing case. The (infinite) queue is FIFO, i.e., sorted by the arrival times of requests. At customer arrival and departure instances, as well as at completion of object replication instances, the system first adjusts the threshold values (as described in 3.3). Then, if excess service capacity is available, the system gives priority to replication of objects (based on current threshold values) where remaining capacity is given to currently "serve-able" requests in the queue, in a FIFO manner. A request for object i is "serve-able" at time t if $A_i(t) > 0$ and there is sufficient network capacity to serve this request.

In this performance study, we not only consider the Zipf distribution for the skewness of access probabilities (refer to Section 4.1), but also a finite geometric distribution [22], given in Equation (6), where we set $\chi = 0.618$.

$$\text{Prob[request for object } i] = \frac{(\chi)^{(i-1)}(1-\chi)}{1-\chi^K} \qquad \forall\ i = 1, 2, \ldots, K \tag{6}$$

The motivation being that some applications (other than movies-on-demand) may exhibit higher skewness in data access, e.g., news-on-demand. As we are not aware of measurements available for applications such as news-on-demand, we use a "generically" highly skewed distribution, i.e., the geometric. Furthermore, applications with relatively little skew in access patterns should not, in a sense, present a performance problem in this case, and thus we do not consider such access patterns here.

Moreover, the interactivity entry in Table 2 refers to how interactive the users are, with NP: FF: RW: PAUSE referring to the ratio between normal playback (NP) and the various VCR functions (FF, RW, PAUSE/RESUME), where T^i_{np}, T^i_{ff}, T^i_{rw}, T^i_{pause} are as defined in Section 2. These values are used as parameters of $\mathcal{M}$ in the computation of T_{ea} (refer to Section 3.2). The default values are in agreement with the range of values used in [16]. Unless otherwise stated, in all figures below we use the default values given in Table 2.

In order to also explore the benefits of *dynamic* replication, in general, we consider the following version of a *static* replication policy **for purposes of comparison only**. Note that, we do not suggest that it is realistic to implement such a policy, but we would like to make a *conservative* comparison and thus favor the static policy. (Furthermore, we are *not* aware of a better static policy in the existing literature.) We assume that a system using the static policy has *perfect* knowledge of the access probabilities, $p_i(t)$s, and that it alters the number of copies, based on this knowledge once per day.

(This is along the lines of suggestions, for adapting to data access pattern changes on a daily basis, made in [26].) That is, every 24 hours, the system alters the number of copies maintained for each object based on the current access probabilities. Specifically, for each object i, it attempts to provide $\lceil p_i(t) * N \rceil$ copies[23] — if this is not possible, due to storage *space* constraints then priority is given to "hotter" objects; if there is excess storage capacity, then the remaining storage space is filled with randomly chosen objects (of course, no more than one copy of an object per node). The only exceptions to these rules are that: (1) there is always a minimum of one copy per object, and (2) copies that are still being utilized by users at the time this alteration takes place are not removed. Note that, the change in the number of copies, in this *static* policy, is assumed to be performed *instantaneously* and *without the use of any additional resources*. These are not realistic assumptions (as is knowing the exact access probabilities), but they are made in order to *favor the static policy* in our comparison. As we are interested in benefits of dynamic replication, in general, we would like to make this comparison a conservative one.

Finally, we note that, although the evaluation of the replication policies presented in the remainder of this section is *quantitative*, the main focus of the following discussion is "trends" in the curves and relative performance of the policies, rather than absolute performance. This is due to the fact that our main motivation is to explore the above stated issues and tradeoffs, rather than to predict the (exact) performance of the system through simulation. To this end, we run the simulations at a very *high* load for the no queueing case [24] in order to illustrate our points (since it almost does not matter what resource management techniques are used at low loads). This is *not* to say that we recommend that the system is *operated* at such high loads; e.g., clearly, under extremely frequent changes in access patterns[25] the acceptance rate will be low under very high loads and thus, under such conditions the real system should be operated at lower loads. In the queueing case, we use somewhat lower loads to insure system stability (since we allow the queue to grow without bound as explained in Section 2).

4.2.1 Static vs. Dynamic

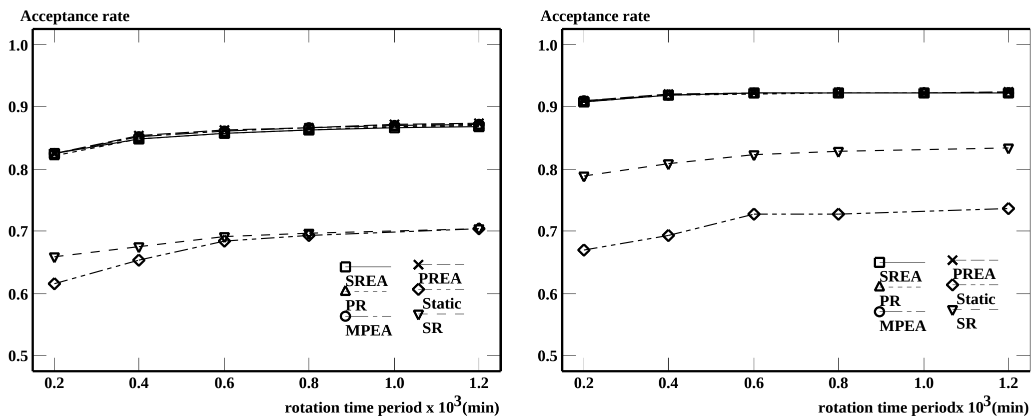
We first give the motivation for using *dynamic* replication policies, as opposed to static ones. This comparison is depicted in Figures 9 and 10, where the more important observations are as follows.

Based on extensive simulations, we conclude that dynamic replication with early acceptance does result in significantly better performance as compared to a static scheme. With a reasonably large amount of per node resources (e.g., arch2.0 as described in Table 2), dynamic policies perform better

[23] This is not the best solution to the so called "apportionment problem"; however, it suffices for our purposes of comparison. For better solutions see [15].

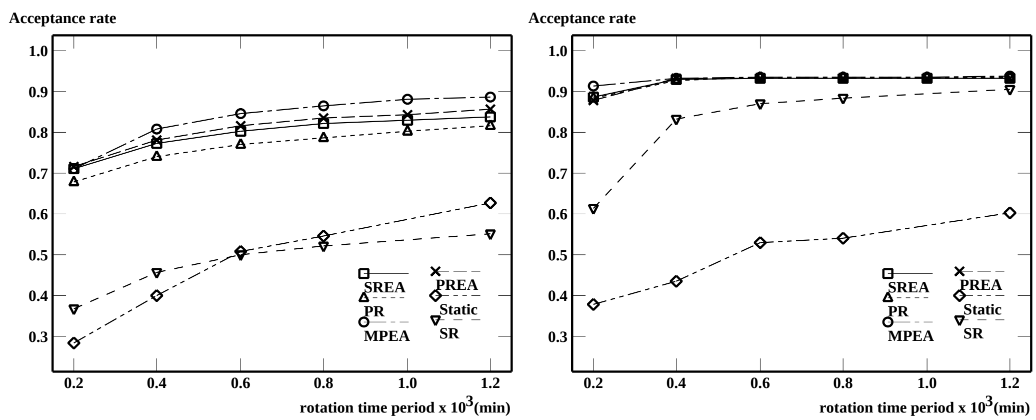
[24] There is no stability issue here.

[25] We include these for the sake of completeness.



(a) arch1.0 (b) arch2.0

Figure 9: Default settings.



(a) arch1.0 (b) arch2.0

Figure 10: Default settings, with geometric distribution of access patterns and both architectures.

than the static policy (refer to Figures 9(b) and 10(b)). As the per node resources become fewer, while keeping the same amount of overall system resources (e.g., arch1.0 as described in Table 2), the benefits of the more conservative dynamic policies (i.e., SR) are significantly diminished (refer to Figure 9(a) and 10(a)). However, addition of early acceptance (i.e., as in the case of SREA) mitigates this problem.

In general, these results are due to the fact that, dynamic replication schemes make good decisions about: (a) which CM objects are hot, and (b) when to replicate such objects. The problem with the more conservative policies such as SR is that the inability to make rapid adjustments in number of replicas is a more severe handicap when resources are few. As we saw above this problem is alleviated through the use of early acceptance. That is, our dynamic replication schemes, in conjunction with the early acceptance technique, allow the CM server to react to changes in data access patterns fairly accurately and rapidly. This results in the system's performance that appears to be fairly insensitive to the changes in data access patterns, i.e., the system maintains nearly the same performance regardless of how frequently the access patterns change.

One advantage of the static policy, of course, is that it is easier to implement. Specifically, the need to migrate users from one node to another (in mid-stream) during de-replication may result in complications in the implementation. In the results presented above as well as in the remainder of the paper, unless otherwise stated, we use the DM de-replication policy in an attempt to make a more fair comparison with the static policy.

4.2.2 Early Acceptance vs. No Early Acceptance

We now motivate the use of *early acceptance* techniques in conjunction with dynamic replication policies. To this end we compare performance of the dynamic policies with and without the use of early acceptance. This comparison is depicted in Figures 9, 10, 11, and 13, where the more important observations are as follows.

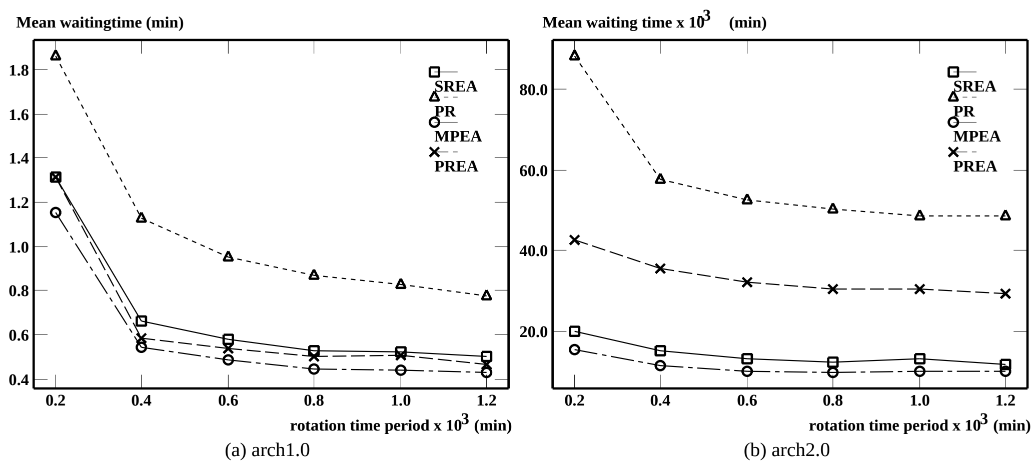


Figure 11: Default settings with a=0.8

Firstly, based on extensive simulations, we conclude that early acceptance does result in a nice compromise between using resources for performing replication and using resources for servicing customer requests (as stated earlier). This point is best illustrated by considering the more conservative policy with early acceptance, i.e., SREA, and comparing it to the least conservative policy we have without early acceptance, i.e., PR. SREA uses as few resources as possible for replication but still makes the new copy available to customers fairly quickly — this policy performs well consistently, i.e., it either results in the best or nearly the best performances in the test cases examined in Figures 9, 10, 11, and 13.

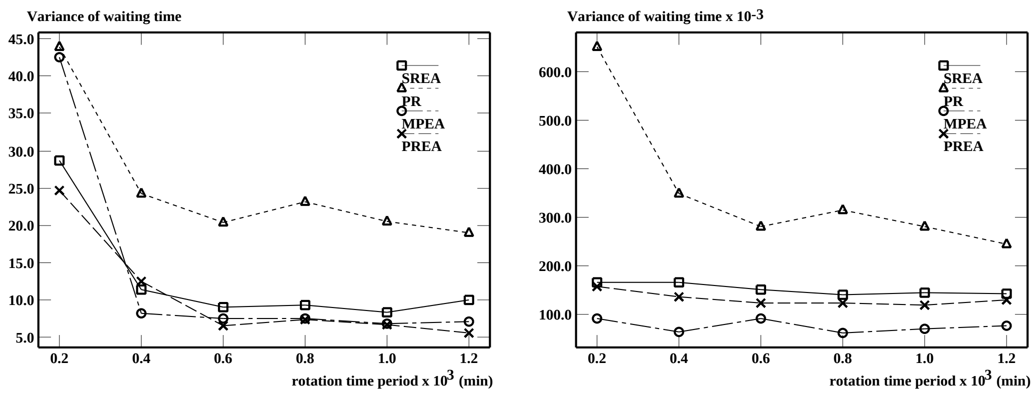
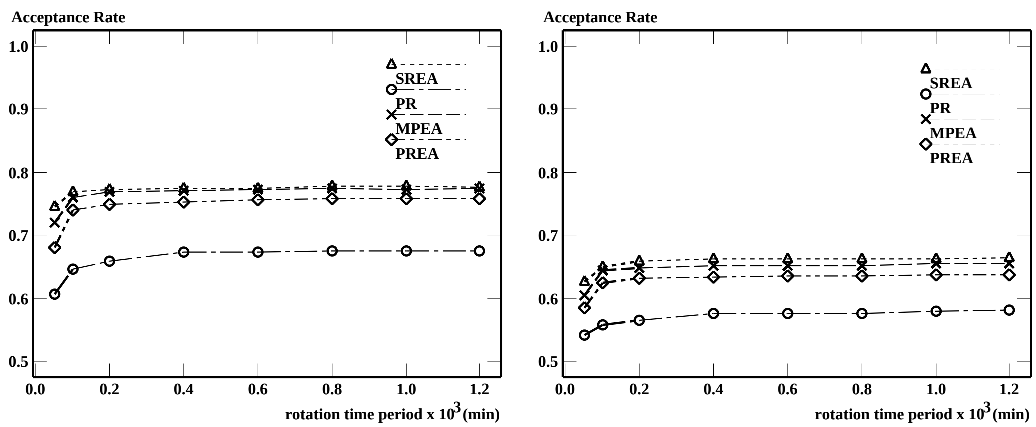


Figure 12: Default settings but with a=0.8



(a) a = 0.8 (b) a = 1.0

Figure 13: Default setting but with on-off process: $\frac{1}{\beta} = \frac{1}{2}$ (min), and $\frac{1}{\alpha} = 6$ (min)

To further explore the effects of early acceptance, we consider (1) more bursty arrival processes, i.e., the on-off arrival process (refer to Section 4.2) and (2) the queueing case (refer to Section 2). More specifically, we observe that all policies with early acceptance (such as SREA, MPEA, and PREA) as compared to the PR policy are: (a) less sensitive to the more bursty arrival process (compare Figure 9(b) with Figure 13(b)), and (b) result in better performance in the queueing case[26], i.e., smaller mean waiting time (see Figure 11) and better QoS, i.e., smaller waiting time variance (see Figure 12). Note that, in a system with a small mean waiting time but a large waiting time variance, it is still possible for a customer to suffer a fairly long waiting time period on occasion; this is clearly undesirable. Hence, we use the variance of the waiting time as our measure of quality of service.

[26] We already showed that these result in better performance in the no queueing case.

4.2.3 Sensitivity to User Model

Next we show that the mathematical model of user behavior is not very sensitive to the precision of the model parameters (which need to be computed based on statistics or measurements collected about user behavior), and thus it is of reasonably practical use. To validate this conjecture, in our *simulation* we *deviate* on several points from the analytical model. Firstly, in our simulations the distribution of residence times in various user playback modes (NP, FF, RW, PAUSE) is **uniform** as compared to the exponential assumption made in the analytical model. We experimented with other distributions as well, e.g., normal; the results were within a few percent of those given here (in the interests of brevity we do not present them). For all cases where the interactivity model corresponds to NP:FF:RW:PAUSE = 19:1:1:1, the fraction of admitted customers that enter "Trap State" in our simulations is zero — recall that, in our computation of T^i_{ea} we chose $\pi_{\text{Trap State}}(t_n) = 0.1$ (refer to Table 2). This is partly due to the fact that our analytical model tends to be conservative (see Section 3.2). We experimented with values of T^i_{ea} that were smaller than those predicted by the analytical model, to determine whether the model's conservative nature is resulting in some loss of performance — this turned out *not* to be the case in our experiments, that is, beyond a certain value of T^i_{ea} one only obtains diminishing returns in performance gains, but the fraction of accepted customers which enter the "Trap State" continues to grow. (We do not present those experiments here in the interests of brevity.)

To further "stress test" our model we ran a set of simulations where T^i_{ff} was increased by 20% in the *simulation*, as compared to the parameter used in the *analytical* model. The result is that there is no change in the fraction of admitted customers entering the "Trap State" in the simulation, i.e., it is still zero. This supports our conjecture (made in Section 3.2), that the parameters used in the analytical model do not have to be exact, with respect to the "real" user behavior — that is, fairly large inaccuracies in the collected statistics about the user behavior can be tolerated and consequently the model is reasonable, and "re-solving" of the model with new parameters only needs to be performed "occasionally" (and *not* necessarily in real-time as explained in Section 3.2).

These results are due to: (1) the conservative nature of the analytical model; (2) good dynamic threshold adjustment methods, i.e., the system is able to distinguish between "hot" and "cold" objects sufficiently well; and (3) the fact that the level of interactivity (19:1:1:1) is relatively low (although reasonable for a movies-on-demand application [16]). Thus, in order to further "stress test" the analytical model, we consider a workload with a significantly higher level of interactivity (i.e., alternative (2) for interactivity settings in Table 2) — this may not necessarily correspond to a realistic workload but is useful for purposes of illustration. Here we consider the following observations in simulations: the fraction of admitted customers entering the "Trap State", the mean amount of time a customer spent in the "Trap State", given that he/she entered it, and the maximum amount of time a customer spent in the "Trap State", given that he/she entered it.

Even with such high interactivity levels, the results show that the fraction of admitted customers which enter the "Trap State" in the simulation is quite small (on the order of 10^{-6}). That is, in our experiments, most admitted customers *do not* enter the "Trap State". Even in the rare case when an admitted customer did enter the"Trap State", the mean and the maximum time he/she spent in the "Trap State" in the simulation was less than 30 seconds. However, we would like to stress that this rarely occurs.

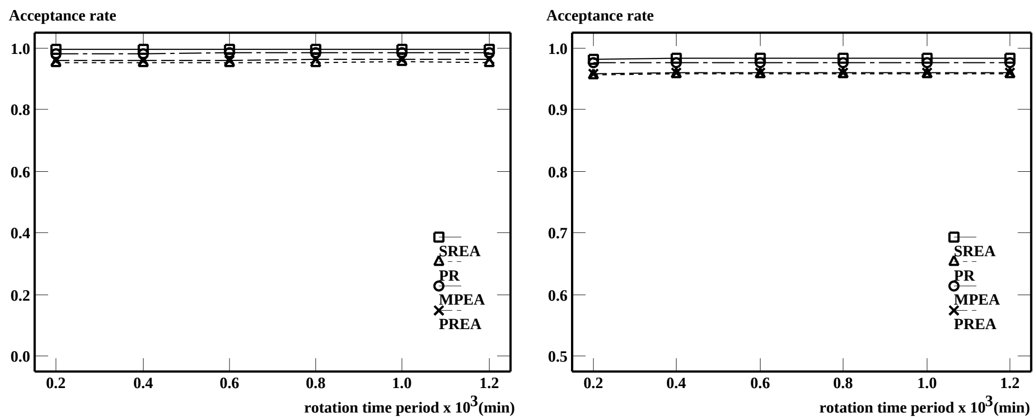
If we want to further reduce the time a customer may spend in the "Trap State", some possible solutions include: (1) migration of customers entering the "Trap State" to other nodes which contain a copy of the object they are viewing and have available service capacity, or (2) increasing T^i_{ea} (at the cost of some performance degradation).

4.2.4 Sensitivity to workload characteristics

Next, we show the lack of sensitivity to the workload characteristics, accomplished through the use of *early acceptance*. To this end we ran a set of simulations with three different modification to the workload characteristics (refer to Table 2), as compared to the default workload used thus far (i.e., as compared to a Poisson process with a constant rate and $a = 1.0$): (1) "time of day" based workload, which is still Poisson but with arrival rates based on time of day, i.e., with $a = 0.9$ over 7 hours of a day and $a = 0.5$ over the remaining 17 hours; (2) lower workloads, i.e., still Poisson with a constant arrival rate but with $a = 0.8$; and (3) more bursty workloads, i.e., an on-off arrival process with $a = 0.8$, $\frac{1}{\beta} = 0.5$ min and $\frac{1}{\alpha} = 6$ min. The results are depicted in Figures 13 and 14. More specifically, we observe that all policies with early acceptance (such as SREA, MPEA, and PREA) as compared to the PR policy are less sensitive to changes of workload characteristics (e.g., compare Figure 13(a) with Figure 14(a)).

4.2.5 Applicability to a variety of applications

Next, we consider applicability of dynamic replication with early acceptance to a wide range of applications of CM servers. To this end, we ran a set of simulations with: (a) smaller objects, i.e., shorter clips with $T^i_{length} = 10$ min $\forall i$, as well as *in addition* (to smaller clips) (b) higher levels of interactivity, NP:FF:RW:PAUSE=4:1:1:1; these cases correspond to interactivity alternatives (1) and (3) in Table 2. The results for cases (a) and (b) are illustrated in Figures 15 and 16, respectively. Qualitatively,



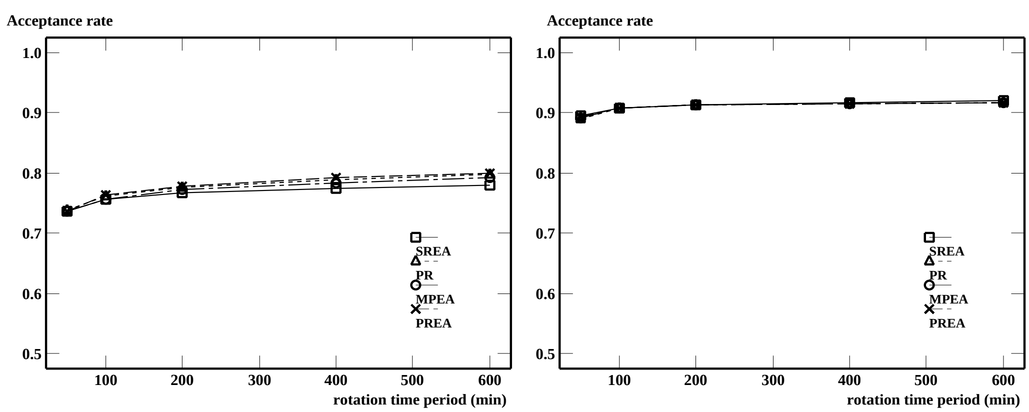
(a) a = 0.8

(b) a = 0.9 for 7 hours; a = 0.5 for 17 hours

Figure 14: Default settings, but with alternatives (1) and (2) for arrivals.

all the conclusions made above, in the context of $T^i_{length} = 90$, still hold[27].

We also consider the sensitivity of the mathematical model of user behavior to the new workload settings which are intended to represent different applications of CM servers. Specifically, in case (b), the fraction of admitted customers entering the "Trap State" in the simulation was on the order of 10^{-5}; the mean and maximum time in the "Trap State", given that an admitted customer entered it, in the simulation was smaller than 20 sec. Again, recall that this is a very rare occurrence. We obtained similar results in case (a).



(a) arch1.0

(b) arch2.0

Figure 15: Default settings with alternative (1) for interactivity settings.

Summary

[27]We observe that there is a significant reduction in the acceptance rate in Figure 16 as compared to Figure 15. This is due to the fact that with higher interactivity levels each customer stays in the system longer, on the average. That is, with higher interactivity levels each customer holds on to the resources allocated to him/her longer.

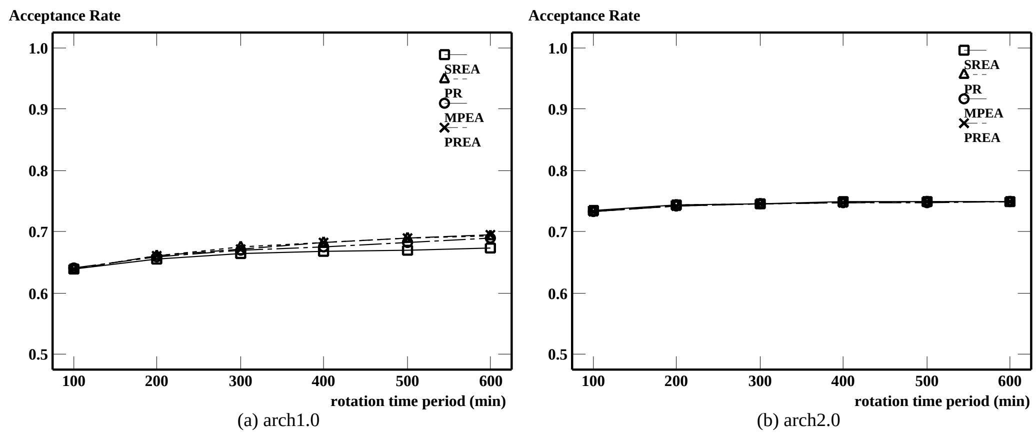


Figure 16: Default settings with alternative (3) for interactivity settings.

In summary, the main observations based on the experiments presented above are as follows: Dynamic replication schemes in conjunction with early acceptance:

- Dynamic replication schemes in conjunction with early acceptance provide a good compromise between using resources for performing replication and using resources for serving customers which in turn results in good performance characteristics across a wide range of workloads, skewness data access patterns and frequency of changes in data access patterns.
- The mathematical model of user behavior, which is used to allow early acceptance in a "safe" manner, is not very sensitive to the precision of the model parameters, as illustrated by testing over a wide range of workloads as well as by using different assumptions in the simulations as compared to the model. Thus, this approach is of reasonably practical use.
- Dynamic adjustment of threshold values, in conjunction with dynamic replication and early acceptance, results in systems that are able to make a reasonably good distinction between "hot" and "cold" objects. This in turn results in improved system performance.

Thus, we believe that these techniques are applicable to a wide range of applications of CM servers.

5 Conclusions

In this work, we studied the *scalability* of large CM end-to-end server designs as a function of their cost/performance and reliability characteristics under various workloads and system constraints. First, we focused on data placement issues and compared the scalability characteristics of hybrid vs. wide data striping architectures. We showed that hybrid designs, in conjunction with dynamic replication techniques, are less dependent on the availability of the interconnect network resources, provide higher reliability, and can be properly sized so as to result in cost-effective end-to-end systems. Thus, we believe that hybrid designs result in *scalable* large distributed CM servers.

Moreover, we have presented a performance study of dynamic replication techniques in conjunction with a mathematical model of user behavior, in the context of hybrid designs. These techniques were studied under relatively frequent changes in data access patterns but without making assumptions about knowledge of the statistics of such patterns and without having to collect such statistics. We have showed that not only does the use of the mathematical model of user behavior improve the performance of the more "conservative" (in terms of resource usage) dynamic replication policies but it also facilitates significantly *reduced sensitivity* to changes in: (a) workload characteristics, (b) skewness of data access patterns, and (c) frequency of changes in data access patterns. We believe that not only is this a desirable property for a CM server, in general, but that furthermore, it suggests the usefulness of these techniques across a *wide range of continuous media applications*.

Parameter	Default	Alternatives
Arrival process	Poisson with $a = 1.0$	(1) Poisson with $a = 0.8$ (2) "time of day" based Poisson with $a = 0.9$ for 7 hrs, $a = 0.5$ for 17 hrs (3) on-off source with $a = 0.8$, $\frac{1}{\beta} = \frac{1}{2}$ (min), $\frac{1}{\alpha} = 6$ (min) (4) on-off source with $a = 1.0$, $\frac{1}{\beta} = \frac{1}{2}$ (min), $\frac{1}{\alpha} = 6$ (min)
User Behavior Model (used in computing T_{ea}^i)	Stochastic with no reduction in state space (DTMC $\mathcal{M}$ with $\pi_{\text{Trap State}}(t_n) = 0.1$)	
$ReTh_i$	$\lceil \frac{T_{ea}^i}{t - at_i} \rceil$	
H_i	$\lceil \frac{T_{length}^i}{t - at_i} \rceil$	
$DeTh_i$	$ReTh_i + H_i$	
T_{length}^i	$90\ \forall i$	$10\ \forall i$
Playback Mode distribution (NP,FF,RW,PAUSE)	Uniform [0.95× mean, 1.05× mean]	
Interactivity Parameters	NP:FF:RW:PAUSE = $19:1:1:1$ $T_{length}^i = 90$, n_{speed}=4 $T_{np}^i = 9.5$, $T_{ff}^i = 0.5$, $T_{rw}^i = 0.5$, $T_{pause}^i = 0.5$ $T_{ea}^i = 12$	(1) NP:FF:RW:PAUSE=$19:1:1:1$ $T_{length}^i = 10$, n_{speed}=4 $T_{np}^i = 1.9$, $T_{ff}^i = 0.1$, $T_{rw}^i = 0.1$, $T_{pause}^i = 0.1$ $T_{ea}^i = 3$ (2) NP:FF:RW:PAUSE=$4:1:1:1$ $T_{length}^i = 90$, n_{speed}=4 $T_{np}^i = 2$, $T_{ff}^i = 0.5$, $T_{rw}^i = 0.5$, $T_{pause}^i = 0.5$ $T_{ea}^i = 18$ (3) NP:FF:RW:PAUSE=$4:1:1:1$ $T_{length}^i = 10$, n_{speed}=4 $T_{np}^i = 2$, $T_{ff}^i = 0.5$, $T_{rw}^i = 0.5$, $T_{pause}^i = 0.5$ $T_{ea}^i = 3$
Replication policy	SREA	PR, MPEA, PREA
De-replication policy	DM	
Access Probability change	"gradual"	"abrupt"
Skewness distribution	Zipf, $\theta = 0.0$	Geometric, $\chi = 0.618$
$q_j^i(t)$	uniformly distributed between 1 and N, for each object i, $\forall t \geq 0$	
D_S	5	
Architecture	(1) arch2.0: $B_x = 80\ \forall x$ $C_x = 28\ \forall x$ $N = 20$	(2) arch1.0: $B_x = 20\ \forall x$ $C_x = 7\ \forall x$ $N = 80$ (3) arch2 group, (4) arch3 group, (5) arch4 group, (6) arch5 group, (7) arch2w (see Table 3) (8) heterogeneous arch (see Section 4.1.3)
K	400	
Request Queue Size	zero	infinite
Network constraint	$nc = 1.0$	$nc = 0.1, ..., 0.9$
Rotation time period	200 (min)	50,100,400,600,800,1000,1200 (min)

Table 2: Parameters.

Arch type	No. of nodes	Srv cap/node Lcl switch cap (in streams)	Stor space/node (in objects)
arch2w	20	80	20
arch2 group	20	80	22; 24; 26; 28; 30
arch3 group	10	160	44; 48; 52; 56; 60
arch4 group	5	320	88; 92; 96; 100; 104
arch5 group	2	800	205;210;215;220;225

Table 3: Parameters for architecture groups.

Arch type	No of nodes	Srv cap/node Lcl switch cap (in streams)	Stor space per node (in obj's)	Glbl switch cap (in streams)
arch2w	20	80	20	1600
arch2	20	80	26	1120
arch2.1	20	80	30	960
arch3	10	160	52	800
arch4	5	320	92	640
arch5	2	800	215	320
arch5.1	2	800	225	160

Table 4: Parameters for architectures used in Section 4.1.

References

[1] Personal Communication with Miscrosoft Research. 1999.

[2] S. Berson, S. Ghandeharizadeh, R. R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. *SIGMOD*, 1994.

[3] S. Berson, L. Golubchik, and R. R. Muntz. Fault Tolerant Design of Multimedia Servers. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 364–375, San Jose, CA, May 1995.

[4] M.-S. Chen, D. D. Kandlur, and P. S. Yu. Support for Fully Interactive Playout in a Disk-Array-Based Video Server. *Proceedings of the 2nd ACM Intl. Conf. on Multimedia*, pages 391–398, October 1994.

[5] A. L. Chervenak. Tertiary Storage: An Evaluation of New Applications. *Ph.D. Thesis, UC Berkeley*, 1994.

[6] C.-F. Chou, L. Golubchik, and J. C.S. Lui. Sriping doesn't scale: How to achieve scalability for continuous med ia servers with replication. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Taipei, Taiwan, April 2000.

[7] C.-F. Chou, L. Golubchik, and J. C.S. Lui. A performance study of dynamic replication techniques in continuous media servers. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco, CA, August 2000.

[8] A. Dan, M. Kienzle, and D. Sitaram. A Dynamic Policy of Segment Replication for Load-Balancing in Video-on-Demand Servers. *ACM Multimedia Systems*, 3:93–103, 1995.

[9] A. Dan and D. Sitaram. An Online Video Placement Policy Based on Bandwidth to Space Ratio (BSR). In *Proceedings of ACM SIGMOD'95*, 1995.

[10] Bolosky et al. The Tiger Video Fileserver. Technical Report MSR-TR-96-09, Michrosoft Research, 1996.

[11] J. Sidell et al. Data Replication in Mariposa. In *Proceedings of ICDE*, New Orleans, LA, 1996.

[12] M. Stonebraker et al. Mariposa: A Wide-Area Distributed Database System. *VLDB*, 1996.

[13] S. Ghandeharizadeh and R. R. Muntz. Design and Implementation of Scalable Continuous Media Servers. *Parallel Computing Journal*, pages 123–155, January 1998.

[14] R.L. Haskin. Tiger Shark : A Scalable File System for Multimedia. Technical report, IBM Research, 1996.

[15] T. Ibaraki and N. Katoh. *Resource Allocation Problems.* The MIT Press, 1988.

[16] K.D. Jayanta, J.D. Salehi, J.F. Kurose, and D. Towsley. Providing VCR Capacities in Large-Scale Video Servers. *In Proc. ACM Intl. Conf. on Multimedia*, pages 25–32, 1994.

[17] P.J.B. King. *Computer and Communication Systems Performance Modeling.* Prentice-Hall, 1990.

[18] D. E. Knuth. *The Art of Computer Programming, Volume 3.* Addison-Wesley, 1973.

[19] P. W. K. Lie, J. C.-S. Lui, and L. Golubchik. Threshold-Based Dynamic Replication in Large-Scale Video-on-Demand Systems. *Multimedia Tools and Applications*, 11(1):35–62, 2000.

[20] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3):266–244, June 1995.

[21] S. Ross. *A First Course in Probability*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1998.

[22] S. M. Ross. *Introduction to Probability Models*. Academic Press, Inc., 1989.

[23] M. Schwartz. *Boardband Integrated Networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1996.

[24] W. J. Stewart. *Introduction to Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[25] N. Venkatasubramanian and S. Ramanathan. Load Management in Distributed Video Servers. In *Proceedings of ICDCS*, pages 528–535, Baltimore, MD, May 1997.

[26] J. Wolf, H. Shachnai, and P. Yu. DASD Dancing: A Disk Load Balancing Optimization Scheme for Video-on-Demand Computer Systems. In *ACM SIGMETRICS/Performance Conf.*, 1995.

Appendix: Derivation of MTTF Equations

In this appendix, we give the derivation of the mean time to failure equations used in Section 4.1. (We use the terms "cluster" and "array" of disks interchangeably below.) We will use the following notation in this derivation:

$MTTF_{disk}$	mean time to failure of a disk
$MTTR_{disk}$	mean time to repair of a disk
$MTTF_{cluster}$	mean time to failure of a cluster or array of disks

Our goal is to compute the $MTTF_{cluster}$ under the following assumptions: (1) each disk has independent and exponential failure rate equal to $\frac{1}{MTTF_{disk}}$; and (2) the total number of disks in the cluster is C.

We first compute the mean time to failure of some disk in a cluster of C disks. Let X_i be the random variable corresponding to the mean time to failure of disk i with an exponential failure rate μ, where $1 \leq i \leq C$. Let $Y = min(X_1, X_2, \ldots, X_C)$, which corresponds to the mean time until *some* disk in a cluster of C disks fails (or the mean time between failures in a cluster of C disks). Then, given that the disk failures are *independent*, we have

$$\text{Prob}[Y \leq a] = 1 - \text{Prob}[X_1 \geq a \wedge X_2 \geq a \wedge \cdots \wedge X_C \geq a]$$
$$= 1 - \text{Prob}[X_1 \geq a] \times \text{Prob}[X_2 \geq a] \times \cdots \times \text{Prob}[X_C \geq a]$$
$$= 1 - (e^{-\mu a})^C = 1 - e^{-C\mu a}$$

Since the failure rate of each disk in our case is $\mu = \frac{1}{MTTF_{disk}}$, we have that the mean time until some disk fails, or the mean time between failures in a cluster of C disks, is $\frac{MTTF_{disk}}{C}$.

Now, a cluster of C disks is considered failed when 2 disks in that cluster have failed. Recall that a disk array is able to continue delivery of data under a single failure; once a second failure in an array of C disks occurs, some data is lost, and we term this failure of the array or cluster. Thus, to compute $MTTF_{cluster}$ we need to determine the probability that, given that one failure has already occurred in that cluster, a second disk failure will occur within $MTTR_{disk}$ time units. Hence, we have that

$$\text{Prob[a disk fails within } MTTR_{disk}] = 1 - e^{-\frac{MTTR_{disk}}{MTTF_{disk}}}$$

and

$$\text{Prob[a disk does not fail within } MTTR_{disk}]$$
$$= 1 - \text{Prob[a disk fails within } MTTR_{disk}] = e^{-\frac{MTTR_{disk}}{MTTF_{disk}}}$$

Then, given that one of the disks in a cluster of C has already failed, we have that

$$\begin{aligned}
&\text{Prob[at least one of the remaining disks fails within } MTTR_{disk}] \\
&= 1 - \text{Prob[none of the remaining disks fail within } MTTR_{disk}] \\
&= 1 - e^{-\left[MTTR_{disk}/\frac{MTTF_{disk}}{C-1}\right]} = 1 - e^{-\frac{MTTR_{disk}*(C-1)}{MTTF_{disk}}}
\end{aligned}$$

Given a well designed system, we can assume that $MTTR_{disk} << \frac{MTTF_{disk}}{C}$ where $\frac{MTTF_{disk}}{C}$ is the mean time to failure of some disk in a cluster of C. It is also well known that $(1 - e^{-x}) \approx x$ is a reasonable approximation when $0 < x << 1$. Then,

$$\text{Prob[at least one of the remaining disks fails within } MTTR_{disk}] \approx \frac{MTTR_{disk}}{MTTF_{disk}}(C-1)$$

Now we are ready to compute $MTTF_{cluster}$. Given that one disk in an array of C disks has failed, we can next observe an event which has one of two possible outcomes:

1. a second disk fails before the first is repaired and thus the entire array/cluster fails

2. the first disk is repaired before the second failure and thus the array/cluster continues to operate normally

we refer to the first outcome of this event as a "failure" and to the second outcome as a "success", i.e., this is a Bernoulli trial [21]. We also know that

$$\begin{aligned}
\text{Prob[“failure”]} &= \\
&\text{Prob[a second disk failure occurs before the first is repaired]} = \\
&\text{Prob[at least one of the remaining disks fails within } MTTR_{disk}] \\
\text{Prob[“success”]} &= \\
&\text{Prob[the first failure is repaired before a second failure occurs]} = \\
&1 - \text{Prob[“failure”]}
\end{aligned}$$

Thus, in determining the mean time to failure of a disk array we are interested in a sequence of event outcomes or trials, where all outcomes but the last one correspond to "success" and the last one corresponds to a "failure". It is well known that on the average it takes $\frac{1}{\text{Prob[“failure”]}}$ trials before we obtain a "failure" [21]. Now we have that

$$\begin{aligned}
&MTTF_{cluster} \\
&= \text{Expected[time between failures in a cluster]} * \text{Expected[number of trials before obtain a “failure”]} \\
&= \text{Expected[time between failures in a cluster]} \quad / \quad \text{Prob[“failure”]} \\
&= \frac{MTTF_{disk}}{C} \times \frac{1}{\frac{MTTR_{disk}}{MTTF_{disk}}(C-1)} \quad = \quad \frac{(MTTF_{disk})^2}{C(C-1)MTTR_{disk}}
\end{aligned}$$

Given the above derivation of $MTTF_{cluster}$, we can now compute the $MTTF$ equations used in Section 4.1 as follows. Let N_d be the total number of disks in the wide data striping as well as the hybrid architectures.

Wide data striping architecture:

Recall that this architecture only keeps a *single* copy of each object i in the entire system. Thus (using reasoning similar to the one used above to derive $MTTF_{cluster}$), we have that

$$MTTF(\text{wide data striping arch}) \approx \frac{MTTF_{cluster}}{number\ of\ clusters\ in\ the\ system} \approx \frac{(MTTF_{disk})^2}{N_d(C-1)MTTR_{disk}}$$

Hybrid architecture:

Here each node has $\frac{N_d}{N}$ disks, and as before we assume that the nodes are independent in terms of their failures. First, we consider the mean time to data loss for those objects which only have a *single* copy in the entire system, which is as follows (again, using reasoning similar to the one used above to derive $MTTF_{cluster}$):

$$MTTF(\text{hybrid arch/single copy}) \approx \frac{MTTF_{cluster}}{number\ of\ clusters\ in\ one\ node} \approx \frac{N(MTTF_{disk})^2}{N_d(C-1)MTTR_{disk}}$$

Next, we consider the mean time to data loss for object i which has k copies in the system, i.e., in k different nodes. Once the first disk failure occurs in (the first[28]) node containing a copy of object i, let Event A correspond to the event where: "a second disk fails in the cluster of the first node in $R_i(t)$ already operating under failure[29] **and** at least one cluster fails in each of the other nodes in $R_i(t)$. Then, as before, we are looking at a Bernoulli trial [21] where now Prob["failure"] = Prob[Event A occurs within $MTTR_{disk}$]. Consequently, we have

$$\begin{aligned}
&MTTF(\text{k copy object}) \\
&= \text{Expected[time between failures in a cluster]} * \text{Expected[number of trials before obtain a "failure"]} \\
&= \text{Expected[time between failures in a cluster]} \quad / \quad \text{Prob["failure"]} \\
&\geq \frac{\frac{MTTF_{disk}}{C}}{\frac{N_d}{CN}} \frac{1}{\frac{MTTR_{disk}(C-1)}{MTTF_{disk}}} \frac{1}{\frac{MTTR_{disk}N_d}{MTTF_{disk}N}} \frac{1}{\frac{MTTR_{disk}(C-1)}{MTTF_{disk}}} \cdots \frac{1}{\frac{MTTR_{disk}N_d}{MTTF_{disk}N}} \frac{1}{\frac{MTTR_{disk}(C-1)}{MTTF_{disk}}} \\
&\approx \frac{MTTF_{disk}^{2k}}{MTTR_{disk}^{2k-1}} \left(\frac{N}{N_d(C-1)}\right)^k
\end{aligned}$$

[28]This is a general derivation as the "numbering" of the nodes is logical.

[29]That is, this is the cluster where the first failure occurred.