A Performance Study of Dynamic Replication Techniques in Continuous Media Servers*

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

Abstract

Multimedia applications are emerging in education, information dissemination, entertainment, as well as many other applications. The stringent requirements of such applications make design of cost-effective and scalable systems difficult, and therefore efficient *adaptive* and *dynamic* resource management techniques can be of great help in improving resource utilization and consequently improving performance and scalability of such systems. In this paper, we focus on threshold-based policies, for dynamic resource management, and specifically, in the context of continuous media (CM) servers. Furthermore, we propose a mathematical model of user behavior and show, through a performance study, that not only does the use of this model in conjunction with dynamic resource management policies improves the system's performance but that it also facilitates significantly *reduced sensitivity* to changes in: (a) system architecture, (b) workload characteristics, (c) skewness of data access patterns, (d) frequency of changes in data access patterns, and (e) choice of threshold values. 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*.

*The work of ChengFu Chou and Leana Golubchik and was supported in part by the NSF CAREER grant CCR-96-25013. The work of John C.S. Lui was supported in part by the UGC Research Grant.

1 Introduction

Multimedia applications are emerging in education, information dissemination, entertainment, as well as many other environments. In general, multimedia applications, such as video stream delivery, place high demands for quality-of-service (QoS), performance, and reliability on storage servers and communication networks. These stringent requirements make design of cost-effective and scalable systems difficult, and therefore efficient *adaptive* and *dynamic* resource management techniques can be of great help in improving resource utilization and consequently improving performance and scalability of such systems. It is worthwhile to point out that work on *mechanisms* for adaptive and dynamic resource management is becoming a more active area of research, e.g., the Harmony project [8] focuses on proper mechanisms for dynamic resource management in large-scale distributed system. In this paper, we focus on *policies*, rather than mechanisms, for dynamic resource management, and specifically, in the context of continuous media (CM) servers.

In a CM storage server, the choice of data placement techniques can have a significant effect on the ability of the system to utilize resources efficiently. Existing data placement techniques in conjunction with scheduling algorithms address two basic inefficiencies that can arise in such systems: (1) various overheads in reading data from storage devices, e.g., due to disk arm movement as in [17] and (2) load imbalance, e.g., due to skews in data access patterns as in [19]. In this work, we focus on the latter issue[1].

Due to the enormous storage and I/O bandwidth requirements of multimedia data, a continuous media server is expected to have a very large disk farm. Thus, in terms of scalability and fault-tolerance, it would be unrealistic to consider a centralized design of a continuous media server (e.g., using a single disk cluster and/or a single processing node). One approach to a more scalable CM server design would be to consider a shared-nothing type architecture [16], composed of a collection of nodes, each with its own processing unit(s), memory sub-system, and I/O sub-system (refer to Figure 1). As already mentioned, an important consideration then is the placement of objects on the nodes of the CM server. One approach to data placement that addresses the load imbalance issue is to stripe each object across all the nodes in the system, e.g., as in [1]. However, this approach suffers from the following shortcomings. Firstly, some form of synchronization in delivery of a single object from multiple nodes must be addressed. In addition, it is not practical to assume

[1] We should point out that, unlike in more "traditional" processor-related load balancing problems, part of the difficulty here is that each resource can only service a subset of possible requests arriving to the system (based on the data placed on it).

that a system can be constructed from homogeneous disks, i.e., as the system grows or experiences faults (and thus disk replacement) we are be forced to use disks with different transfer and storage capacity characteristics — having to stripe objects across *heterogeneous disks* would lead to further complications.

Instead of striping each object across all the nodes, we can constrain the striping to a single node and replicate the popular objects on several nodes in order to provide sufficient bandwidth capacity to service the demand for these objects, where 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 [19], in the context of workloads with relatively infrequent changes in object access patterns (e.g., on a daily basis).

More frequent changes data in 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 this paper, we 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).

We begin with a brief survey of several related works on replication of objects in CM servers which address skewness in data access patterns. (There is a multitude of papers on design of CM servers, in general; we refer the interested reader to [7] for a more extensive coverage of this topic and the corresponding literature.) In [18] the authors consider skews in data access patterns but in the context of a static environment. As already stated above, in [19], 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). We believe that the policies suggested in this work can be complementary to the techniques developed in [19]. In [5, 4], the authors also consider dynamic replication as an approach to load imbalance, and in our previous work [6], we study a taxonomy of dynamic replication schemes. All of the above works, however, either (a) assume some knowledge of frequencies of data access to various objects in the system, and/or (b) do no 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 this work is largely due to considerations of applicability of dynamic replication techniques in more general settings and to a wider range of applications of continuous media servers.

Thus, the main contributions of this work are as follows. We present dynamic object replication schemes for CM servers which do *not* rely on knowledge of frequencies of object accesses but rather make the adjustments, in a threshold-based manner, simply based on the amount of resources currently available for servicing user requests for those objects. We believe that this is an important consideration, as determination (and use) of these frequencies in a highly dynamic (and perhaps distributed) environment may not be a simple matter (see Section 6 for details). The system's performance in such an environment depends largely on its ability to make the adjustments in number of replicas fairly *rapidly*. To this end we propose a mathematical model of user behavior[2] which facilitates reduction in (at least the "virtual") replication time (see Section 5 for details), or put another way, allows admission of users to a *partially* complete replica while still satisfying *quality-of-service* requirements of continuous media applications. We show that the model is not very sensitive to the accuracy of its parameters and thus is of reasonably practical use.

Lastly, and perhaps more importantly, we study the affects of various architectural and workload characteristics on dynamic replication policies. We show 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) system architecture, (b) workload characteristics, (c) skewness of data access patterns, (d) frequency of changes in data access patterns, and (e) choice of threshold values. 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*, including movies-on-demand as well as more interactive applications, such as educational video clips, training applications, news-on-demand, and so on.

The remainder of this paper is organized as follows. In Section 2, we present the details of the system under consideration. In Section 3 we describe the basic approach to dynamic resource management in a CM server, and in Section 4 we discuss the basic tradeoff involved in dynamic object replication. In Section 5 we propose a mathematical model of user behavior which aids in improving system performance, in conjunction with dynamic replication policies. In Sections 6 and 7 we describe the dynamic replication policies considered in this paper, and in Section 8 we present a performance study of these policies. Finally in Section 9 we give our concluding remarks.

[2]Here we refer to modeling the level of *interactivity* of user behavior.

2 System

In this paper we consider a distributed continuous media (CM) server which has a set S of N nodes connected through a communication network, in a shared-nothing manner [16], as illustrated in Figure 1. Each node $x \in S$ has a finite storage capacity, C_x (*in units of CM objects*), as well as

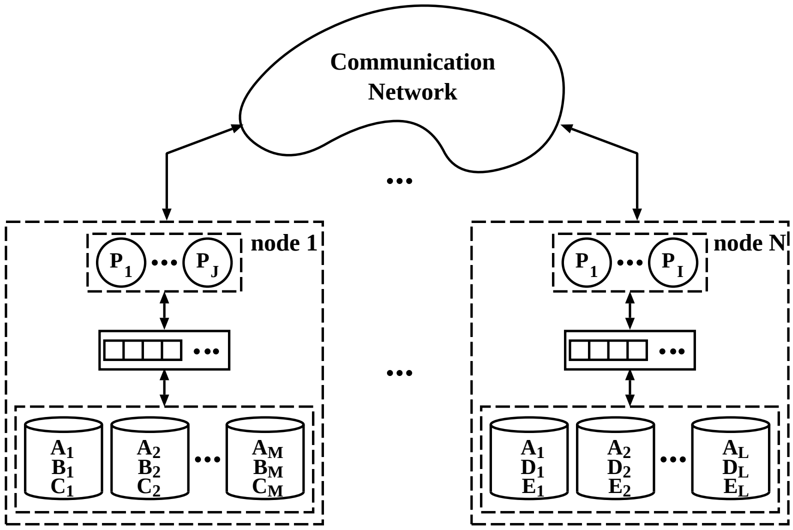


Figure 1: Multimedia System.

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 mins 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 $C_x = 12$ MPEG-2 videos. In general, different nodes in the system may differ in their storage and/or service capacities. This 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. The objects may be striped on the *intra-node* basis but *not* on the *inter-node* basis (see Section 1 for motivation for using this type of data layout). Objects that require more than a single node's service capacity (to support the corresponding requests) are *replicated* on multiple nodes. The number of replicas needed to support requests for a continuous object is a function of the demand, and therefore, this number should change when the the demand for that object changes. Let $R_i(t) \subseteq S$ denote the set of nodes containing replicas of object i at time t. Thus $R_i(t)$ varies with time as the popularity of object i

changes.

Customers arrive to the CM server with an average arrival rate of λ. Upon a customer arrival at time t, there is a probability $p_i(t)$ that the corresponding request is for object i. Let $L_x(t)$ be the load on node x at time t. The system examines the load on each node in $R_i(t)$, and if there is sufficient capacity to service the newly arrived request, the system assigns this request to the least-loaded node in $R_i(t)$. Otherwise, the customer is rejected. Our goal here is mainly to examine the CM server's responsiveness to changes in data access patterns, when using dynamic data management schemes. Thus, we do not consider queueing of customers that can not be admitted immediately as this would also entail consideration of scheduling policies for requests in the queue and consequently would make it more difficult to isolate the performance characteristics due to dynamic data replication techniques. The appropriateness of various queueing techniques and the customer's willingness to wait for service are, in general, largely a function of the particular application supported by the CM server.

Lastly, full VCR functionality (i.e., fast-forward, rewind, and pause/resume) is available to all admitted customers, with fast-forward and rewind provided at $n_{speed} > 1$ times the rate of normal playback. Let T_{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_{ff}, T_{rw}, and T_{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 [2].

Table 1 summarizes the main notation used in this paper. We will define this notation throughout the paper, as it is needed. Finally, note that, we focus on the management of the *disk storage* and *bandwidth resources*, i.e., we do not consider buffer space, computational, tertiary storage, etc. resources in our performance study; similarly we do not consider communication network issues.

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; $A_i(x) = \sum_{x \in R_i(t)}(B_x - L_x(t))$
$ReTh$	Replication threshold, i.e., the threshold for adding another copy of an object
$DeTh$	Dereplication threshold, i.e., the threshold for removing a copy of an object
D	difference between the replication and the dereplication thresholds, i.e., $D = ReTh_i - DeTh_i$
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
n_{speed}	ratio between the speed of fast forward (or rewind) and the speed of normal playback
T_{np}	mean amount of time spent in the normal playback mode each time
T_{ff}	mean amount of time spent in the fast forward mode each time
T_{rw}	mean amount of time spent in the rewind mode each time
T_{pause}	mean amount of time spent in the pause mode each time

Table 1: Summary of notation.

3 General Approach

As already stated, we consider a *dynamic* approach to reacting to changes in user data access patterns. Since the number of copies of object i *partly* determines[3] 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*.

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,
2. to *which* node should a (new) replicas 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

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

one).

In the remainder of this paper, we discuss techniques that address these issues in an *efficient* manner. In Sections 6 and 7 we present *policies* for triggering and performing the replication. In Section 8 we present results of a performance study of these policies. To assess the usefulness of these techniques we use the system's *acceptance rate* as our performance metric, which is defined as the percentage of all arriving customer requests that are accepted by the system (with zero waiting time[4]).

4 Main Tradeoff

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)[5]. 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 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. 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.

[4]Refer to Section 2 for discussion on waiting time.

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

Thus, we essentially have conflicting goals of (a) using as 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.

5 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 process continues). That is, once the system completes replication of the first T_{ea} time units[6] of a new replica of a CM object i, it will treat it as a "virtually" complete copy[7] and begin using it in servicing customer requests for object i. Note that, 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 system 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 quality-of-service (with a high probability). Below, we propose both, a deterministic and a stochastic approach to this problem.

[6] 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, e.g., bytes.

[7] Of course, we are motivated here by the continuous nature of the object, and essentially it is this property that we exploit in the remainder of this section.

5.1 Deterministic Model

Given that the replication process constructs a new copy of an object, from the beginning of the object to the end (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 2. Specifically, if a newly arrived customer is allowed to use

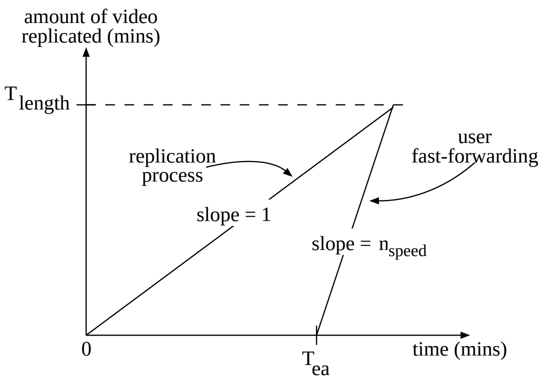


Figure 2: Deterministic model.

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) the "virtual replication completion time" of an object becomes T_{ea} as compared to T_{length}.

5.2 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), $\mathcal{M}$, with the following state space $\mathcal{S}$:

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

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. There are 4 types

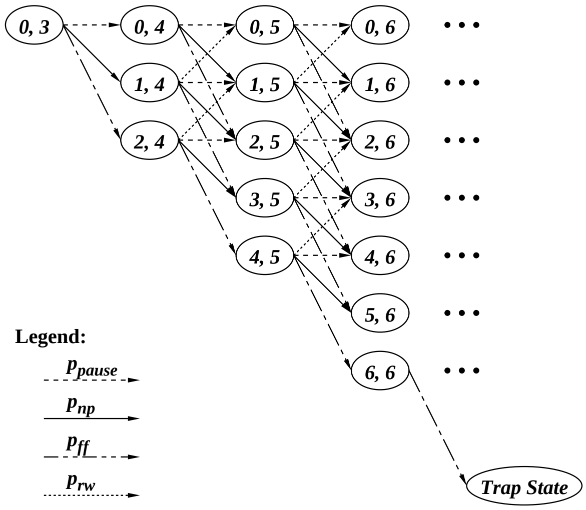


Figure 3: State Transition Diagram for $\mathcal{M}$ with $n_{speed} = 2$, $T_{ea} = 3$.

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

$$
\begin{array}{lcll}
(V,R) & \longrightarrow & (min(V + n_{speed} * t_u, T_{length}), R + t_u) & \\
 & & \multicolumn{2}{l}{\text{Prob} = p_{ff}\mathbf{1}\{\ ((V + n_{speed} * t_u) \leq (R + t_u)) \wedge ((R - V) < T_{length}\frac{n_{speed}-1}{n_{speed}}) \wedge (R < T_{length})\ \}} \\
(V,R) & \longrightarrow & (max(V - n_{speed} * t_u, 0), R + t_u) & \\
 & & \multicolumn{2}{l}{\text{Prob} = p_{rw}\mathbf{1}\{\ (R < T_{length}) \wedge ((R - V) < T_{length}\frac{n_{speed}-1}{n_{speed}})\ \}} \\
(V,R) & \longrightarrow & (V, R + t_u) & \\
 & & \multicolumn{2}{l}{\text{Prob} = p_{pause}\mathbf{1}\{\ (R < T_{length}) \wedge ((R - V) < T_{length}\frac{n_{speed}-1}{n_{speed}})\ \}} \\
(V,R) & \longrightarrow & (V + t_u, R + t_u) & \text{Prob} = p_{np}\mathbf{1}\{\ (R < T_{length})\ \} \\
(V,R) & \longrightarrow & (\text{Trap State}) & \text{Prob} = p_{ff}\mathbf{1}\{\ ((V + n_{speed} * t_u) > (R + t_u)) \wedge (R < T_{length})\ \} \\
(V,R) & \longrightarrow & (V,R) & \text{Prob} = 1\mathbf{1}\{\ ((R - V) = T_{length}\frac{n_{speed}-1}{n_{speed}}) \vee (R = T_{length})\ \} \\
(\text{Trap State}) & \longrightarrow & (\text{Trap State}) & \text{Prob} = 1
\end{array}
$$

where $\mathbf{1}\{x\}$ is an indicator function and $\mathbf{1}\{x\} = 1$ 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[8] in the DTMC $\mathcal{M}$. Finally,

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

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 $\mathcal{M}$ [15], i.e., by solving the following set of equations[9]:

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

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 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 complexity of 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 complexity 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

[8]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.

[9]More sophisticated methods for computing transient results exist [15], but are outside the scope of this paper.

value for $\pi_{(\text{Trap State})}(t_n)$ is obtained, which, corresponds to the desired quality-of-service (QoS) to be provided by the system. We will illustrate in Section 8 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 once, on a per application basis[10]. We will show in Section 8 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 only occur after significant changes in the interactive nature of an application.

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 complexity results in more *conservative* estimates of T_{ea}, and thus we would be trading off system's performance for complexity of the solution (for reasons similar to the ones stated above). We elaborate on this approach next.

Before proceeding, 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 [10]; however, these have been used for a somewhat different purpose and to the best of our knowledge, with interest in *steady state* characteristics only.

5.3 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[11]. More formally,

[10] That is, a set of statistics or measurements corresponding to 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 $\mathcal{M}$.

[11] This is still "safe" since pause and rewind can not cause the viewer to access an unreplicated portion of the data. 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.

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

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

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. There are 2 types

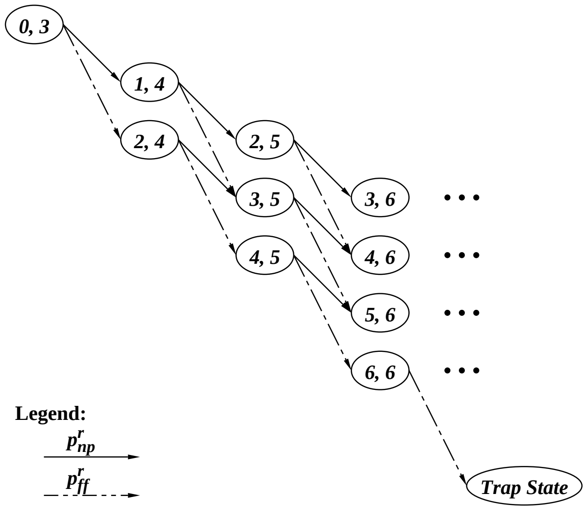


Figure 4: State Transition Diagram for $\mathcal{M}^r$ with $n_{speed} = 2$, $T_{ea} = 3$.

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

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

where

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

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 complexity 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 ≈ 3000 states in $\mathcal{M}$). As before, we can trade off computational speed 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) using MATLAB numerical solutions package requires less than one minute of computation[12]).

6 Threshold-based Resource Activation

We use a *threshold-based* approach to triggering continuous media object replication and dereplication. Both, replication and de-replication of an object i 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 [11]. 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[13] a copy, and thus it should be done "sparingly".

An important question here, of course, is how to choose "good" thresholds. Intuitively, we would like the amount of service capacity available to each object i to be proportional to its demand, which is changing with time. Thus, we could attempt to maintain a number of copies of each object proportional to $p_i(t)$'s — the question of *when* to trigger creation (or deletion) of a copy would still remain, though. In general, although we could try to collect statistics on access

[12]This computation time is based on running MATLAB on a Sun Ultra-1 machine. As we mentioned before, better than brute force transient analysis techniques can be used to further improve on this time [15].

[13]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.

demands for the various objects, many questions would remain open: over which period to collect the statics, when to make the decision that the probabilities have changed sufficiently to reflect this change in the system's configuration (likely, we do not want to do this "continuously"), how much confidence to have in the collected statistics and thus how aggressively or cautiously to "evolve" the system from an old state (i.e., with old access probabilities) to a new state (i.e., with new access probabilities).

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 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. $A_i(t) < ReTh$, where $ReTh$ is the replication threshold value and $A_i(t)$ is the available service capacity for object i at time t, i.e., $A_i(x) = \sum_{x \in R_i(t)}(B_x - L_x(t))$.
2. Object i is not currently under replication.

The actual choice of value for $ReTh$ depends on the replication policy used. Intuitively, the more "efficient" the replication policy, the longer we can delay the decision about replication (i.e., in general, shorter term decisions can be more accurate, but system performance would still depend on being able to act on them quickly). Thus, we set $ReTh = C \times \bar{B}$, where $0 < C < 1$ and $\bar{B}$ is the average service capacity of the nodes in the system. We explore the affects of C on system performance in Section 8. The rational for setting $ReTh$ no larger than $\bar{B}$ is that it is usually wasteful to keep more than one copy of a "cold" object.

In the case of dereplication, 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:

1. $i = \max_j \{A_j(t) > ReTh\}$. 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.,

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

   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. in the case of the Delayed Migration (DM) de-replication policy *only* (see Section 7), there is an additional condition, namely that $C_{ix}(t)$ must be equal to 0. (We include this condition here, before actually defining the DM policy, for completeness sake.)

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

7 Policies

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

7.1 Replication 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 capacity. More formally, we choose the node x from the set S_t such that:

$$S_t = \left\{ x | x \notin R_i(t) \text{ and } L_x(t) = \min_{y \in S_t} \left\{ \frac{B_y - L_y(t)}{1 + \gamma_y(t)} \right\} \right\} \tag{3}$$

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). We now describe the replication policies.

Sequential Replication (SR): The replication is performed "sequentially" (as described in Section 4), i.e., the system replicates at the rate of a normal display 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 above. Except, that given the value of T_{ea}, determined through the use of the DTMC model described in Section 5, newly arrived users can be admitted to the new (incomplete) replica as soon as T_{ea} time units of that object have been replicated on the target node. Furthermore, this "virtual" replication completion time (i.e., T_{ea}) is used in checking the satisfaction of condition (2) in the decisions of when to create a new replica (see Section 6).

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 policy PR above, except that users are admitted to the new (incomplete) replica after the first T_{ea} time units of the replica are completed. Furthermore, as in the case of the SREA policy, this "virtual" replication completion time (i.e., T_{ea}) is used in checking the satisfaction of condition (2) in the decisions of when to create a new replica.

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

7.2 De-replication Policies

The decision process of *which* replica to remove, i.e., which object i, was described in Section 6. 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). This is motivated by the (possible) implementation complexity of migrating customers from one node to another[14].

Immediately 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 y where $y = \min_x\{C_{ix}(t)\}$; the C_{ix} customers are distributed evenly among the *remaining* nodes in $R_i(t)$.

Immediately 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 y where $y = \max_x\{C_{ix}(t)+$

[14] Although, migration has been suggested as a reasonable approach, but in the context of adjusting to load imbalances in [19].

$(B_x - L_x(t))\}$; the C_{ix} customers are distributed evenly among the *remaining* nodes in $R_i(t)$.

8 Discussion of Results

In this section we present results of our study of dynamic replication policies in conjunction with early acceptance of customers, which is accomplished through the use of a Markov chain model of user behavior. We first summarize the main issues and tradeoffs that effect the performance of the system and which we illustrate quantitatively in the remainder of this section.

1. **Use of resources for replication vs. use of resources for servicing customers**: recall that the main tradeoff is between (a) using as few resources as possible to perform the replication (in order not to interfere with "normal" system operation) and (b) trying to complete the replication process as soon as possible (see Section 4). We achieve a compromise between these essentially conflicting goals through the use of *early acceptance*.

2. **Sensitivity to the above tradeoff as a function of the "architecture" used as well as the skewness in data access patterns**: how much a system's performance is affected by the compromise between servicing the normal workload vs. performing replication depends on (a) the system's architectures (e.g., how large the service capacity of each node is — refer to the two architectures used in this study, as given in Table 2) and (b) how skewed the data access patterns are. The choice of architecture depends partly on the storage and network technologies available, the intended applications, etc. How to choose the "best" architecture is *not* the focus of this work; rather, we explore the sensitivity of the dynamic replication policies to the choice of the architecture. Similarly, skewness in data access patterns is largely a function of the application(s) using the CM server. Our goal here is to show that *early acceptance* aids in reducing the sensitivity to both, choice of architecture and skewness in data access patterns.

3. **Choice of threshold values:** in general, the performance of threshold-based policies is often sensitive to the choice of the actual threshold values and can lead to fairly significant changes in system behavior. Determining optimal threshold values is, in general, a difficult problem and is *not* the focus of this work; rather, our goal is to show that the *sensitivity* to the choice of threshold values can be reduced partly through the use of *early acceptance*.

This study is performed via simulation, with the following simulation parameters. The arrival process (of requests for objects) is Poisson[15] with a mean arrival rate of $\lambda = \frac{a\bar{B}N}{T^i_{length}}$, where $0 \le a \le 1$ is the "relative arrival rate". For ease of presentation, in the remainder of this section we discuss the results in terms of the *relative* arrival rate, 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).

There is a multitude of parameters that can be varied in studying performance of dynamic replication policies. Table 2 lists the parameters considered in this study along with their default values and alternatives as used in the remainder of this section[16]. (Refer to Table 1 for the definition of the notation used in Table 2.) Several of the entries in this table require a few words of

Parameter	Default	Alternatives
Arrival rate	constant, $a = 1.0$	(1) constant, $a = 0.8$ (2) "time of day" based $a = 0.9$ for 7 hrs, $a = 0.5$ for 17 hrs
User Behavior Model (used in computing T_{ea})	Stochastic with no reduction in state space (DTMC $\mathcal{M}$ with $\pi_{\text{Trap State}}(t_n) = 0.1$)	
$ReTh$	$\frac{1}{2}\bar{B}$	$(\bar{B} - 1)$, $\frac{3}{4}\bar{B}$, $\frac{1}{4}\bar{B}$
$DeTh$	$(\bar{B} + 1)$	
T^i_{length}	$90\ \forall i$	$10\ \forall i$
Playback Mode distribution (NP,FF,RW,PAUSE)	Uniform [$0.95\times$ mean, $1.05\times$ mean]	
Interactivity Parameters	NP:FF:RW:PAUSE = $19:1:1:1$ $T^i_{length} = 90$, n_{speed}=4 $T_{np} = 9.5$, $T_{ff} = 0.5$, $T_{rw} = 0.5$, $T_{pause} = 0.5$ $T_{ea} = 12$	(1) NP:FF:RW:PAUSE=$19:1:1:1$ $T^i_{length} = 10$, n_{speed}=4 $T_{np} = 1.9$, $T_{ff} = 0.1$, $T_{rw} = 0.1$, $T_{pause} = 0.1$ $T_{ea} = 3$ (2) NP:FF:RW:PAUSE=$4:1:1:1$ $T^i_{length} = 90$, n_{speed}=4 $T_{np} = 2$, $T_{ff} = 0.5$, $T_{rw} = 0.5$, $T_{pause} = 0.5$ $T_{ea} = 18$ (3) NP:FF:RW:PAUSE=$4:1:1:1$ $T^i_{length} = 10$, n_{speed}=4 $T_{np} = 2$, $T_{ff} = 0.5$, $T_{rw} = 0.5$, $T_{pause} = 0.5$ $T_{ea} = 3$
De-replication policy	IMMO	DM
Access Probability change	"gradual"	
Skewness distribution	Zipf, $\theta = 0.0$	Geometric, $\chi = 0.618$
Architecture	(1) $B_x = 20\ \forall x$ $C_x = 7\ \forall x$ $N = 80$	(2) $B_x = 80\ \forall x$ $C_x = 28\ \forall x$ $N = 20$
K	400	

Table 2: Parameters.

clarification, which are as follows.

[15] 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 here; refer to [13].

[16] All values are given in units of minutes, unless otherwise specified.

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 min $p_i(t)$'s change. The change in access probabilities is described by Equation (4), which is intended to emulate a relatively "gradual" increase/decrease in popularities[17].

$$p_i(t') = \; = \begin{cases} p_{i+2}(t) & \text{if } i \text{ is odd and } 1 \leq i < K-1 \\ p_K(t) & \text{if } i \text{ is odd and } i = K-1 \\ p_{i-2}(t) & \text{if } i \text{ is even and } 2 < i \leq K \\ p_1(t) & \text{if } i \text{ is even and } i = 2 \end{cases} \tag{4}$$

where t and t' refer to two consecutive rotation periods and for ease of presentation we assume that K is even.

Furthermore, we consider two distributions for the skewness of the access probabilities. The default one is the Zipf distribution [12], as described in Equation (5) with $\theta = 0.0$ which corresponds to the measurements performed in [3] (for a movies-on-demand application).

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

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

In addition, we also consider a more skewed distribution, namely a finite geometric distribution [14], given in Equation (6), with $\chi = 0.618$.

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

The motivation for also studying the system performance under geometricly distributed access patterns, is that we believe 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 (i.e., ones analogous to measurements performed in [3]), 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, 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

[17] 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.

functions (FF, RW, PAUSE/RESUME) and T_{np}, T_{ff}, T_{rw}, T_{pause} are as defined in Table 1. These values are used as parameters of $\mathcal{M}$ in the computation of T_{ea} (refer to Section 5). The default values are in agreement with the range of values used in [10].

Unless otherwise stated, in the figures below, we use the default values given in Table 2. Recall also that we are using the *acceptance rate* as our performance metric.

Lastly, 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*. We assume that the 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 [19].) 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[18] — if this is not possible, due to storage *space* being the bottleneck, 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 may not be realistic assumptions (as is maybe 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[19] 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

[18] This is not the best solution to the so called "apportionment problem"; however, it suffices for our purposes of comparison. For better solutions see [9].

[19] There is no stability issue here, since there is no queueing in the system.

recommend that the system is *operated* at such high loads; e.g., clearly, under extremely frequent changes in access patterns[20] the acceptance rate will be low under very high loads and thus, under such conditions the real system should be operated at lower loads.

Static vs. Dynamic

We begin with a motivation for using *dynamic* replication policies, as opposed to static ones. To this end we compare the performance of dynamic policies (described in Section 7) to the static policy described above. This comparison is depicted in Figures 5 and 6, where the more important observations are as follows. Under fewer resources on a single node (e.g., Architecture (1) in Table 2) and not extremely skewed data access patterns (e.g., Zipf distribution), the static policy can perform better than *some* dynamic policies, and specifically those that require large amounts of resources to perform the replication (such as MPEA, PREA, PR). The dynamic policies that require few resources to perform the replication (such as SPEA) outperform the static policy (e.g., by $\approx 15\%$ in Figure 5(a)).

As the distribution becomes more skewed (e.g., geometric distribution) the static policy can not keep up with the dynamic ones; this is depicted in Figures 6(a) and 6(b), where the difference can be as high as $\approx 100\%$. This is due to the fact that under more skewed access, greater alterations are required to the number of copies mainted for each object, as the access patterns change, and thus the dynamic nature of the replication policies becomes more useful.

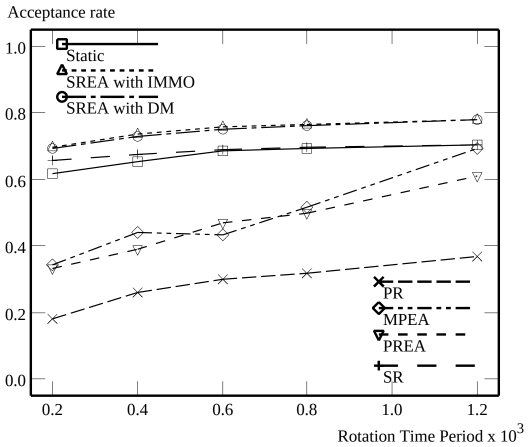
Furthermore, as nodes are configured with higher service capacities (e.g., Architecture (2) in Table 2), the static policy also can not keep up with the dynamic policies; this is depicted in Figures 5(b) and 6(b), where the difference is anywhere from $\approx 20\%$ to $\approx 100\%$. This is due to the fact that, as the capacity of a single node grows, using some fraction of this capacity to perform the replication has a less significant effect on the overall system performance. This also accounts for the fact that the dynamic policies employing some form of parallel replication (i.e., PR, MPEA, PREA) are also doing better under Architecture 2. In fact, for a similar reason, under less frequent changes in access probabilities there is little difference in performance, at least between the policies using *early acceptance*.

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, may result in complications in the implementation. To this end, we consider the DM de-replication policy (refer to Section 7), which

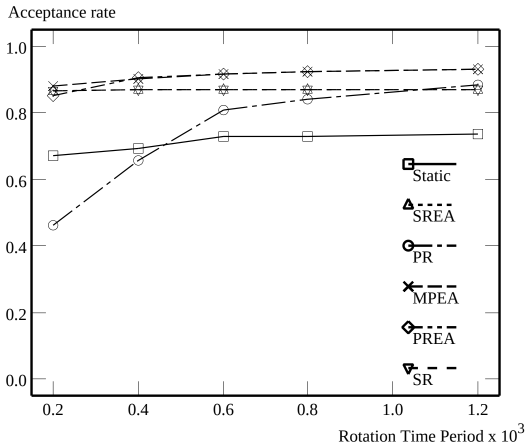
[20] We include these for the sake of completeness.

does not require movement of users between nodes. The results for the default settings are depicted in Figure 5(a), where the performance of the system under the DM policy is only $\approx 1.5\%$ worse than under the IMMO policy (with all other things being equal).

Note that, even some of the better policies perform relatively poorly under certain workloads, and specifically under very frequent changes in access patterns (which may not necessarily represent realistic workloads). As already mentioned, our concern here is with exploring the tradeoffs by considering the relative performance of the policies, and we include such workloads for completeness. In a real system, to improve the acceptance rate one would, in general: (a) allow queueing[21] and/or (b) operate the system at lower loads (which we consider later in this section).



(a)Architecture 1



(b) Architecture 2

Figure 5: Default settings, but for both archictures.

Early Acceptance vs. No Early Acceptance

Next, we would like to 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 also depicted in Figures 5 and 6, where the more important observations are as follows.

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

[21] Refer to Section 2 for rational for not considering queueing here; note, however, that queueing would change some of the characteristics of the system performance.

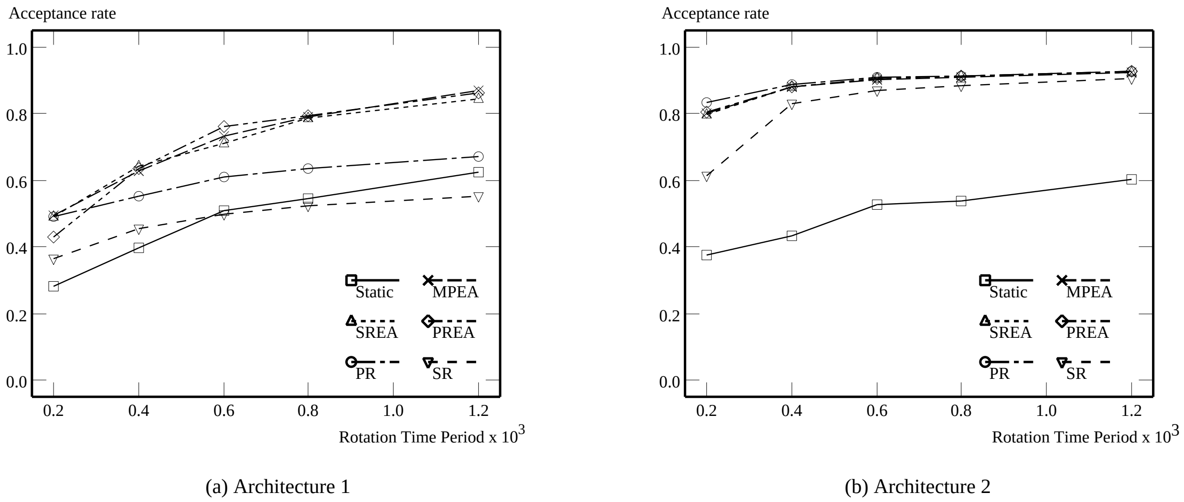


Figure 6: Default settings, but for geometric distribution and both architectures.

customer requests (as stated at the beginning of this section). This point is best illustrated by considering the SREA policy, which 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 5 and 6.

More specifically, we observe that SREA (for the above stated reasons) is less sensitive to (a) choice of architecture (compare Figure 5(a) vs. Figure 5(b) and similarly Figure 6(a) vs. Figure 6(b)), (b) skewness in the data access patterns (compare Figure 5(a) vs. Figure 6(a) and similarly Figure 5(b) vs. Figure 6(b)), as well as (c) choice of thresholds (as illustrated in Figure 7). This last point is worth elaborating on, as the choice of thresholds can have a significant effect on a system's performance (consider for instance PR's performance in Figure 7) and thus using a policy which is less sensitive to this choice is a plus. We conjecture that this lack of sensitivity is due to both, the sequential nature of the policy as well as its early acceptance capability. The ability to react to changes faster through early acceptance (i.e., shorter "virtual" replication time) can allow the system to delay making a decision about future demands (and thus perhaps make a better one), i.e., the system can use lower threshold values (this accounts for a small improvement in performance with lower threshold values for the early acceptance policies). The sequential nature of the policies forces the system to be more conservative about resource usage and thus less dependent on the choice of threshold values.

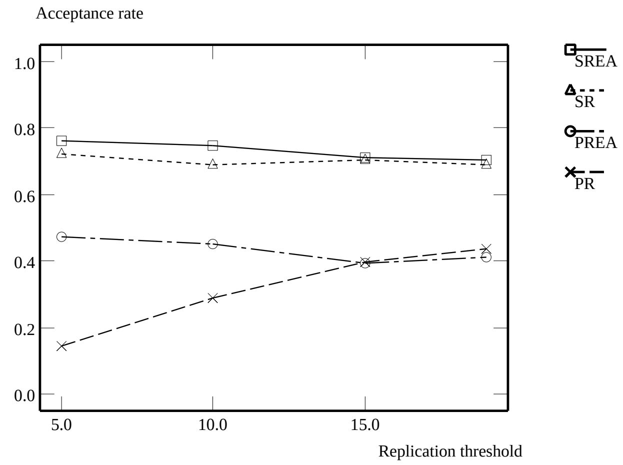


Figure 7: Default seetings, but with rotation time period of 600 min.

Sensitivity to User Model

Next we would like to show that the mathematical model of user behavior, presented in Section 5, 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 residense times in various user playback modes (NP, FF, RW, PAUSE) is uniform as compared to the exponential assumption made in the analytical model. For all cases where the interactivity model corresponds to NP:FF:RW:PAUSE = 19:1:1:1, the probability of entering the "Trap State", as computed by the *simulation*, is *zero* — recall that, in our computation of T_{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 (as explained in Section 5).

To further "stress test" our model we ran a set of simulations where T_{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 probability of entering the "Trap State", in the *simulation* results, i.e., it is still *zero*. This supports our conjecture (made in Section 5), 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 5).

These results are due to (1) the conservative nature of the analytical model and (2) the fact that the level of interactivity (19:1:1:1) is relatively low (although reasonable for a movies-on-demand application [10]). 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. Figures 8, 9(a), and 9(b) depict *simulation* results for the probability of a user entering the "Trap State", the mean amount of time a user spent in the "Trap State", given that he/she entered it, and the maximum amount of time a user spent in the "Trap State", given that he/she entered, respectively.

Even with such high interactivity levels, the probability of entering the "Trap State" is still reasonable (on the order of 10^{-5}). If however, the time spent in the trap state is not acceptable, then possible solutions include: (1) migration of customers entering the trap state to other nodes which contain a copy of the object they are viewing, or (2) increasing T_{ea} — recall, that in the cases presented in Figures 8 and 9, we used $\pi_{\text{Trap State}}(t_n) = 0.1$ to compute T_{ea}.

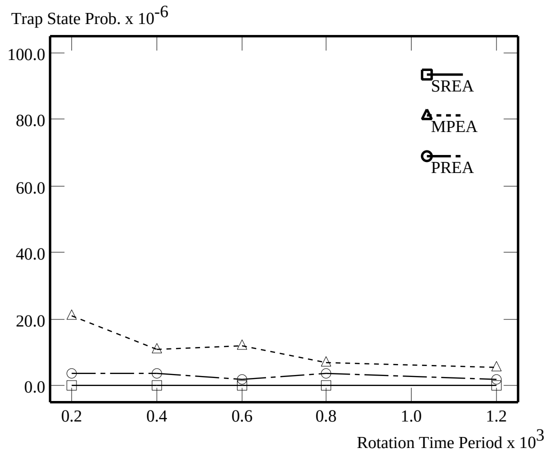
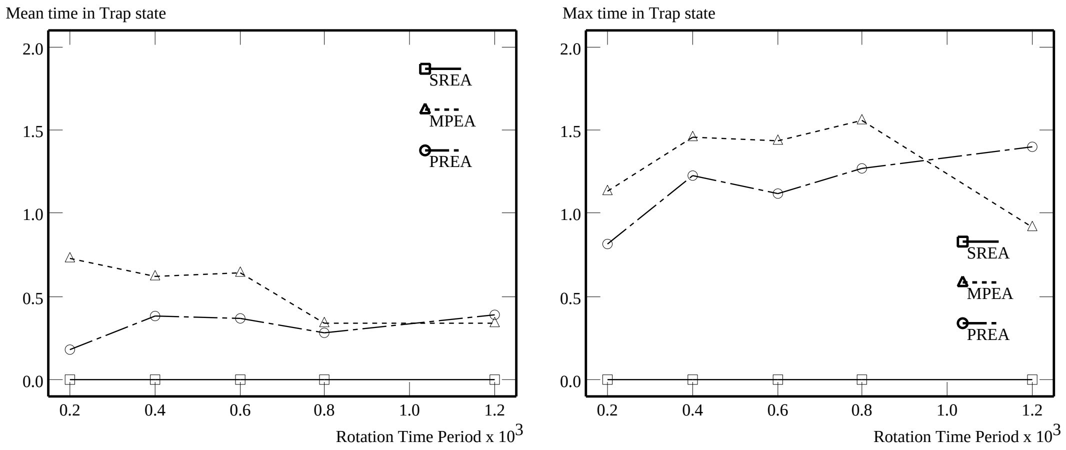


Figure 8: Deafult settings but with alternative (2) for interactivity settings.

Sensitivity to workload characteristics

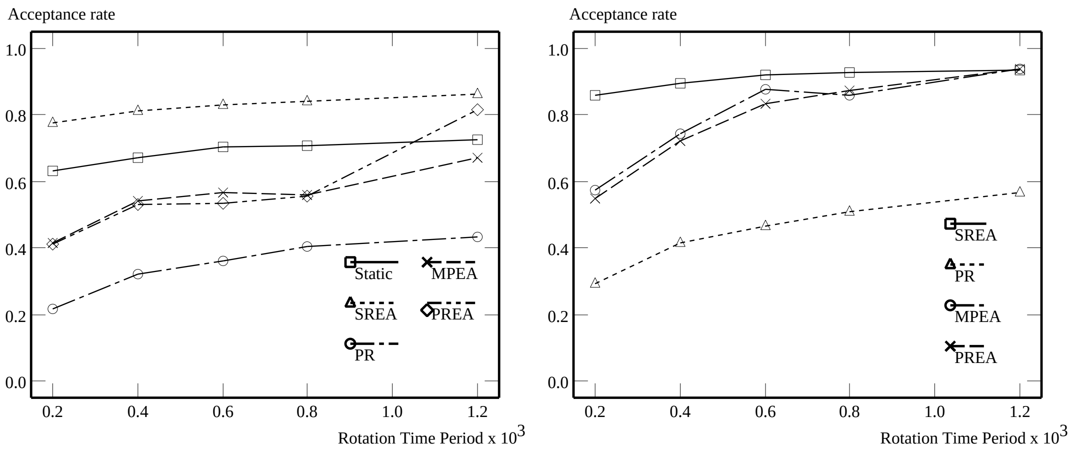
Next, we would like to show the lack of sensitivity to the workload characteristics, accomplished through the use of *early acceptance*. To this end we ran a set of simulations with two different modification to the workload characteristics (refer to Table 2), as compared to the default workload



(a) mean (b) max

Figure 9: Deafult settings but with alternative (2) for interactivity settings.

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 as well as (2) lower workloads, i.e., still Poission with a constant arrival rate but with $a = 0.8$. The results are depicted in Figures 10(b) and 10(a), for cases (1) and (2), respectively. Qualitatively, the conclusions made above (under the default workload) still hold.

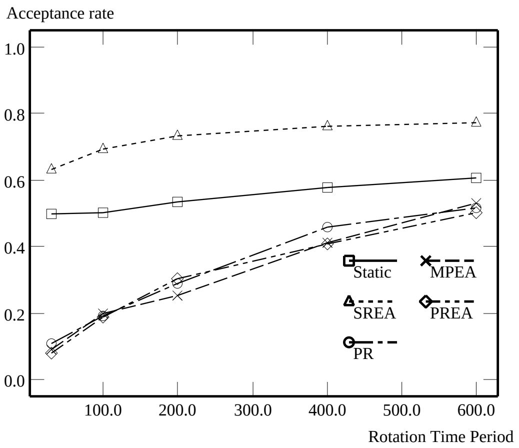


(a) a = 0.8 (b) a = 0.9 for 7 hours; a=0.5 for 17 hours

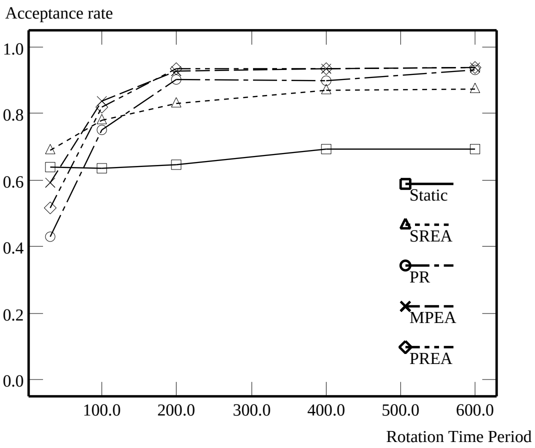
Figure 10: Default settings, but with alternatives (1) and (2) for arrival rates.

Applicability to a variety of applications

Finally, we would like to show applicability of dynamic replication with early acceptance to a wide range of applications of continuous media servers (as suggested in Section 1), i.e., not just movies-on-demand (e.g., these can be news-on-demand, education-on-demand, etc.). 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, i.e., NP:FF:RW:PAUSE=4:1:1:1 (i.e., interactivity alternatives (1) and (3) in Table 2). The performance results for case (a) are illustrated in Figures 11 and 12. The results of sensitivity to the methamatical model of user behavior for case (b) are illustrated in Figures 13 and 14. For case (a), qualitatively, all the conclusions made above, in the context of $T^i_{length} = 90$, still hold. For case (b), the probability of entering the "Trap State" goes up, but the mean and maximum amount of time spent there goes down; thus a reasonable QoS can still be provided in this case as well.



(a) Architecture 1

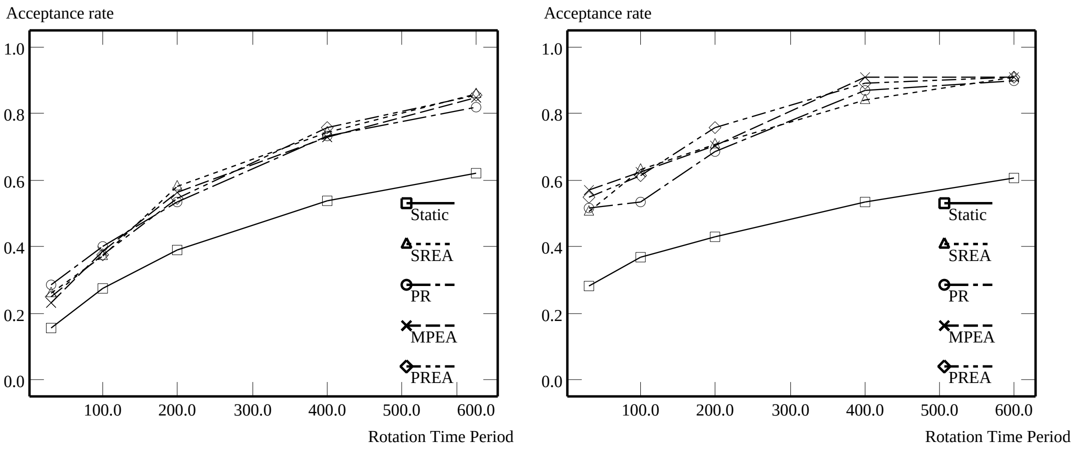


(b) Architecture 2

Figure 11: Default settings but with both architectures and alternative (1) for interactivity settings.

9 Conclusions

In summary, we have presented a performance study of use of dynamic replication techniques in conjunction with a mathematical model of user behavior, in continuous media servers. These techniques were proposed in the context of relatively frequent changes in data access patterns but without making assumptions about knowledge of the statistics of such patterns. We have showed that not only does the use of the mathematical model of user behavior improve the performance



(a) Architecture 1

(b) Architecture 2

Figure 12: Default settings but with both architectures, geometrics distribution, and alternative (1) for interactivity settings.

of the more "conservative" (in terms of resource usage) dynamic replication policies but it also facilitates significantly *reduced sensitivity* to changes in: (a) system architecture, (b) workload characteristics, (c) skewness of data access patterns, (d) frequency of changes in data access patterns, and (e) choice of threshold values. 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*.

References

[1] Steven Berson, Shahram Ghandeharizadeh, Richard R. Muntz, and Xiangyu Ju. Staggered Striping in Multimedia Information Systems. *SIGMOD*, 1994.

[2] 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 Second ACM International Conference on Multimedia*, pages 391–398, October 1994.

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

[4] 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.

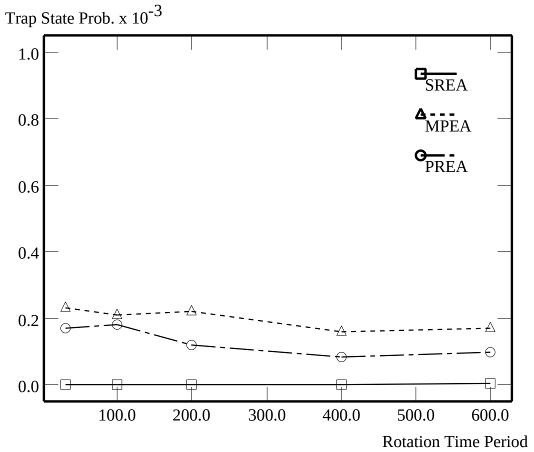


Figure 13: Default settings but with alternative (3) for interactivity settings.

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

[6] Anonymous for the purpose of double-blind refereeing.

[7] S. Ghandeharizadeh and R. R. Muntz. Design and implementation of scalable continuous media servers. *In the special issue of Parallel Computing Journal on Parallel Dat a Servers and Applications*, pages 123–155, January 1998.

[8] J. K. Hollingsworth and P. J. Keleher. Prediction and adaptation in active harmony. In *The 7th International Symposium on High Performance Distributed Computing*, pages 180–188, Chicago, July 1998.

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

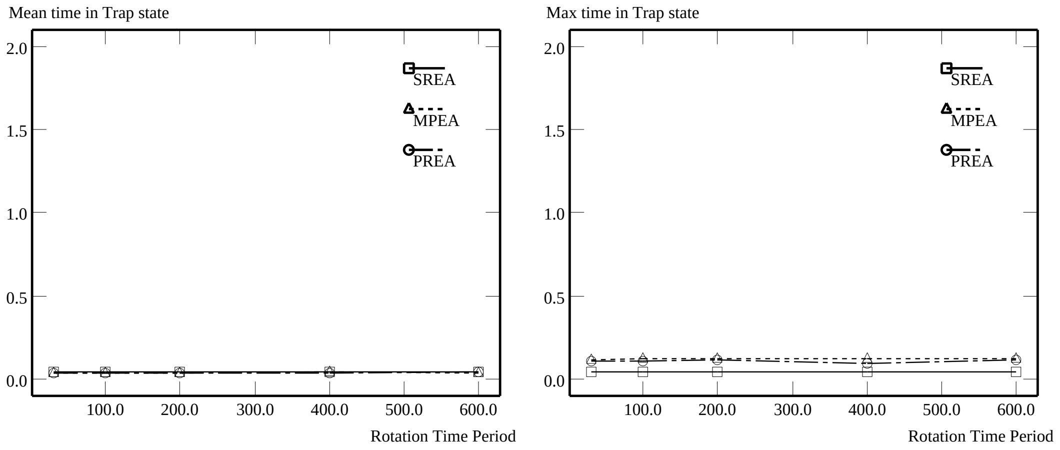
[10] K.D. Jayanta, J.D. Salehi, J.F. Kurose, and D. Towsley. Providing Vcr Capacities in Large-Scale Video Servers. *In Proc. ACM Intl. Conf. on Multimedia*, pages 25–32, 1994.

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

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

[13] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):266–244, June 1995.

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



(a) mean

(b) max

Figure 14: Default settings but with alternative (3) for interactivity settings.

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

[16] M. Stonebraker. A Case for Shared Nothing. *Database Engineering*, 9(1), 1986.

[17] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID - A Disk Array Management System For Video Files. *ACM Multimedia Conference*, pages 393–399, 1993.

[18] N. Venkatasubramanian and S. Ramanathan. Load management in distributed video servers. In *Proceedings of ICDCS*, pages 528–535, Baltimore, MD, May 1997.

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