Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with 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

Abstract

Multimedia applications place high demands for QoS, performance, and reliability on storage servers and communication networks. These, often stringent, requirements make design of cost-effective and *scalable* continuous media (CM) servers difficult. In particular, the choice of data placement techniques can have a significant effect on the scalability of the CM server and its ability to utilize resources efficiently. In the recent past, a great deal of work has focused on "wide" data striping as a technique which "implicitly" solves load balancing problems; although, it does suffer from multiple shortcomings. Another approach to dealing with load imbalance problems is replication. The main focus of this paper is a study of scalability characteristics of CM servers as a function of tradeoffs between striping and replication. More specifically, striping is a good approach to load balancing while replication is a good approach to "isolating" nodes from being dependent on other system resources. The appropriate compromise between the degree of striping and the degree of replication is key to the design of a scalable CM server. This is the topic of our work.

Keywords: continuous media servers, distributed systems, performance evaluation, scalability, end-to-end design.

Technical areas: Multimedia and Digital Libraries; Distributed Systems Architecture.

*The work of ChengFu Chou and Leana Golubchik and was supported in part by the NSF CAREER grant CCR-98-96232.

1 Introduction

With the rapid growth of multimedia applications, there is a growing need for large-scale continuous media (CM) servers that can meet the user demand. Multimedia applications (such as video stream delivery, digital libraries, distance learning systems, and so on) 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* continuous media 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 growth or re-configuration;
- maintain performance characteristics under degradation of system resources, which can be caused by losses in network and storage capacities.

In particular, in a continuous media server, the choice of data placement techniques[1] can have a significant effect on the scalability of the system 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, e.g., due to disk arm movement 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* CM server.

Due to the enormous storage and I/O bandwidth requirements of multimedia data, a CM server is expected to have a very 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 consideration then is the placement of objects on the nodes of the CM server. As in traditional database systems, data placement on a distributed storage sub-system directly affects the load balancing characteristics of that system.

In the recent past, a great deal of research work, e.g., as in [2, 20, 3, 8, 10], has focused on "wide" data striping as a data placement technique for designing continuous media servers. By wide data

[1] Here by data placement we mean decision of which object or fraction of an object to place on which disk or disk cluster, i.e., this does *not* refer to data placement issues *within a single disk*.

striping we mean that each object is striped across all the disks of the system. Recall that the potential load imbalance is largely due to the *skews* in data access patterns [4] 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 be idling. Moreover, the problem is exacerbated by the fact that access patterns change over time, i.e., the popularity of a particular object is a function of 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 be 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 (as described in [8]), e.g., to the best of our knowledge, in implementations described in [8, 9] striping is performed over (at most) a few tens of *homogeneous* disks only[2]. Note that, delivery of relatively short continuous media objects is of use to many important applications, including digital libraries, news-on-demand systems, and so on.

3. Due to the continuity constraints, some form of synchronization in delivery of a single object from multiple nodes 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 the user demand and data sizes grow and hence the system requires more storage and disk bandwidth capacities, the resulting expansion of the disk sub-system results in re-striping of *all* the objects.

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

5. Due to the need for communication of data between the nodes over which an object 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. Specifically, we consider a *hybrid* approach where instead of striping each objects across all the nodes of the system, we constrain the striping to a single node and replicate popular objects on several nodes in order to provide sufficient bandwidth capacity to service the demand for these objects.

Of course, a disadvantage of this approach is a need for additional storage space. Furthermore, techniques are needed for adjusting the number of replicas as the access patterns change. Some of these issues are addressed in [21], in the context of workloads with relatively infrequent changes in object access patterns as well as in our previous work [13, 5], where we propose dynamic replication techniques in the context of more frequent changes in data access patterns. In this paper, we improve on our previous work on dynamic replication techniques, as described in Section 4.

However, the main focus of this paper 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 results 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. Furthermore, replication has the benefit of increased reliability in terms of: (a) longer mean time to loss of data from the disk sub-system (see Section 5); and (b) dealing with lack of network resources (see Section 5), including network partitioning. The downside of replication is that it increases storage space requirements and hence cost. 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 topic of our paper.

1.1 Related Work

Recently much research has been done on design of continuous media, and specifically video-on-demand, storage servers, e.g., as in [2, 20, 3, 8, 10], to a name a few. Much of this work falls into several broad categories[3], which include: (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 workload demand problems has been less explored. In [19] the authors consider skews in data access patterns but in the context of a static environment. In [21], 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 used in this paper can be complementary to the techniques developed in [21]. In [7, 6], the authors also consider dynamic replication as an approach to load imbalance, and in our previous work [13, 5], we study a taxonomy of dynamic replication schemes. However, all of these works (except our work in [5] 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 continuous media 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 (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.

[3] We divide these into broad categories as it would be nearly impossible to list all papers on CM server designs.

1.2 Our Contributions

The main contributions of this work are as follows.

- Quantitative evaluation of *performance* and *resource demand* characteristics of data striping vs. data replication techniques in large-scale distributed CM servers. Such evaluation is crucial to achieving a *scalable* design of continuous media servers.
- Improved dynamic replication techniques for distributed hybrid CM servers, needed to achieve better performance by adjusting the number of replicas in the system based on changed in data access patterns and user demand.
- Quantitative evaluation of reliability characteristics of data striping- vs. data replication-based systems.
- Illustration of ease of designing *heterogeneous* hybrid CM systems *without* loss in performance characteristics.

Based on this end-to-end cost/performance and reliability study we argue that *hybrid* designs result in large *scalable* continuous media systems.

2 Hybrid CM System Architectures

A hybrid system architecture is illustrated in Figure 1. It consists of a set of nodes connected by a high speed switch, which we term a *global* switch. The global switch is a high bandwidth resource which can, for instance, correspond to a high capacity WAN or an ATM-type infrastructure. Each node i contains one or more processing units (PUs) and one *local* switch which is used to connect all local PUs as well as local clients. Each client connects to the nearest local switch (depending on their geographical location) which is also connected to some node i. Requests from this client which are serviced by a PU from node i are termed “local” clients or “local” requests. When a request from a client cannot be serviced by its local node i, the request is forwarded to a remote node j, which contains a replica of the object being requested. We term this request a “global” client or a “global” request, as it requires some capacity of the global switch in order to receive

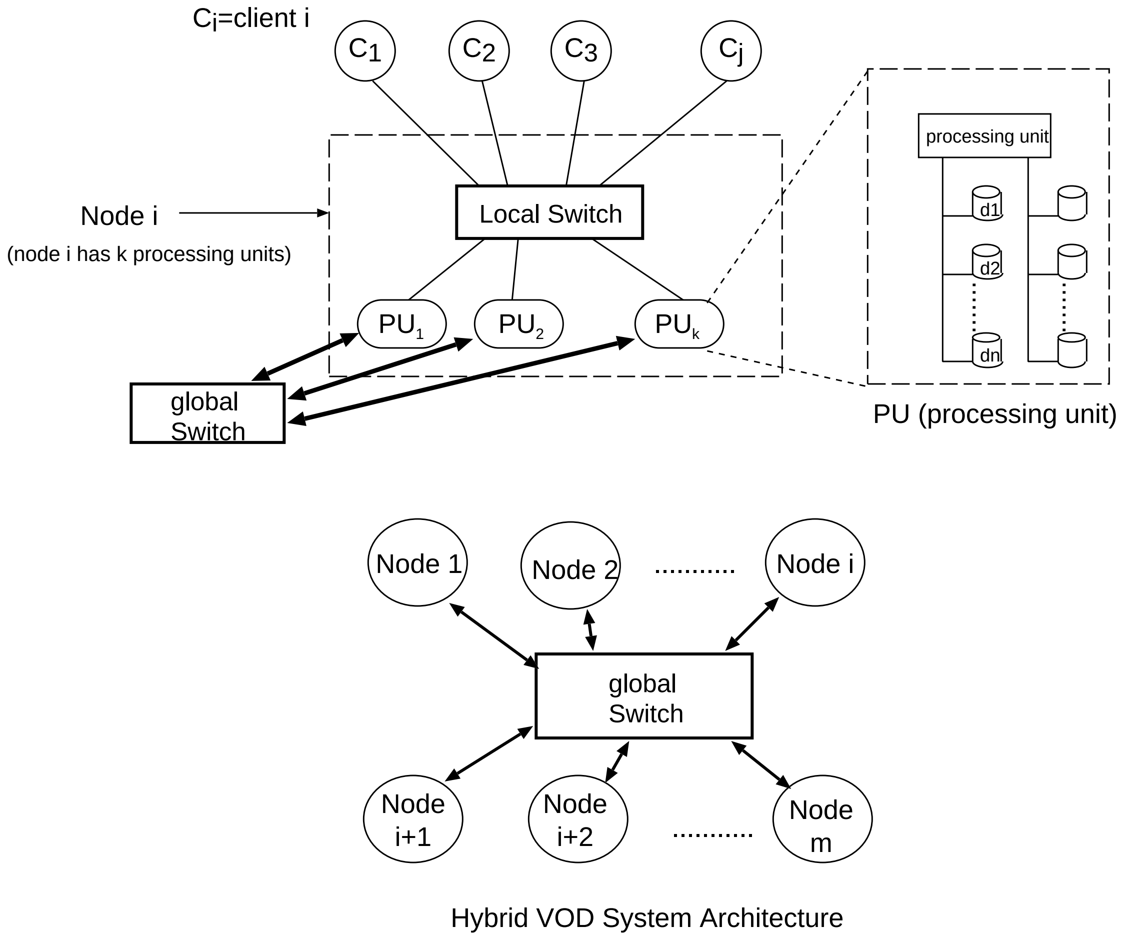

Figure 1: Multimedia System.

service. That is, when a remote node services a client, the continuous media data is delivered 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). Furthermore, each PU (of each node) 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 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 $D_x = 12$ MPEG-2 video objects. 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, I/O bandwidth, and networking (i.e., 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 on the intra-node basis but not on the inter-node basis. That is, 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 continuous object is a function of demand, and therefore this number should change when the demand for that object changes. Let $R_i(t)$ 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. (The precise details of how $R_i(t)$ changes over time are given in Section 3.)

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 interconnection network, i.e., 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 is rejected.

Note that, in the hybrid system we need to maintain load information on remote nodes as well as other bookkeeping information (including recomputation of replication/dereplication thresholds, as described in Section 4). This will require some communication capacity, although this required capacity is significantly smaller than the capacity needed for transmitting CM object data from a remote node to a local node. Furthermore, communication of such bookkeeping information is needed in the wide striping case as well, since coordination between nodes is needed for scheduling of *each* request in the system. Since the exact amount of bookkeeping information depends on a particular implementation, we do not consider this any further here. However, we would like to point out that in the case of the wide data striping architecture, the bookkeeping information must be exchanged between nodes to schedule a newly arrived request. On the other hand, in the hybrid architectures, we can tradeoff the frequency of collecting such information (or the "up-to-dateness" of the information) for performance, i.e., we can tradeoff relying more on local information (rather than remote information) for some loss in performance.

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*, which is defined as the percentage of all arriving customer requests that are accepted by the system (with zero waiting time);
2. the capacity (in units of access streams) of the global switch required to support a particular architecture and corresponding acceptance rate;
3. the capacities (in units of access streams) as well as the number of local switches required to support a particular architecture and corresponding acceptance rate;
4. the amount of disk storage (in units of continuous media objects) required to support a particular architecture and corresponding acceptance rate;
5. the mean time to failure (MTTF) of a particular architecture.

Note that, in the performance evaluation done in this work, we do not consider queueing of customers that can not be admitted immediately, since that would entail consideration of scheduling policies for requests in the queue. The appropriateness of various queueing disciplines and the customer's willingness to wait for service are, in general, largely a function of the particular application supported by the CM server. Thus, although in practice a CM server can have some finite queueing capacity, here we do not consider queueing of customer requests that can not be admitted at request time since the queueing disciplines appropriate for these requests are largely a function of specific applications using the CM server and we do not limit this study to a particular application.

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; $N = \|S\|$
N_d	total number of disks in both wide data striping and hybrid systems
K	number of distinct objects in the system
B_x	maximum service capacity of node x (in streams)
$\bar{B}$	average service capacity of nodes in the system (in streams)
$\bar{D}$	average storage capacity of nodes in the hybrid system
$\bar{D}^w$	average storage space capacity of nodes in the wide data striping system
D_x	total storage capacity on node x (in units of objects) in the hybrid system
$D_x(t)$	available storage space on node x at time t in the hybrid system
D_x^w	total storage capacity on node x in the wide data striping system
D^w	total storage space in the wide data striping system
$L_x(t)$	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
$DeTh_i$	dereplication threshold, i.e., the threshold for removing a copy of object i
H_i	difference between the replication and the dereplication thresholds, i.e., $H_i = ReTh_i - DeTh_i$
T^i_{length}	length of object i (in units of normal playback time)
T^i_{ea}	early acceptance time for object i (in units of normal playback time)
λ	average arrival rate to the system
at_i	the latest access time for object i in the system
$R_i(t)$	set of nodes containing a replica of 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 an arriving request for object i being for node j at time t
D_S	storage space threshold for activating the dereplication process

Table 1: Summary of notation.

3 Dynamic Replication

In a hybrid CM server, we use a *dynamic* approach to reacting to changes in user data access patterns. Since the number of copies of object i partly determines 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* (which can require a non-negligible amount of resources) and *accurately*. Thus, when adjusting the number of replicas in the system, we essentially have conflicting goals of (a) using as few resources as possible to performance the replication (in order not to interfere with "normal" system operation) while (b) trying to complete the replication process as soon as possible.

In an attempt to reach a compromise between these conflicting goals, we use "early acceptance" of customers, as proposed in our previous work [5], 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^i_{ea} time units [4] of a new replica of a CM object i, it will treat it as a "virtually" complete copy and begin using it in serving customer requests for object i.

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 when viewing CM objects, we need to determine a "safe" value for T^i_{ea}. Clearly, one safe value is $T^i_{ea} = T^i_{length}$ (full length of the CM object). However, the intuition is that a smaller value of T^i_{ea} should result in a higher (at least in the "short term") acceptance rate of customer requests.

In order to lower T^i_{ea} (and improve system performance) in [5], we employ a stochastic model of user behavior, at the cost of lowering the probability that a user will not access data beyond the replication point (of course, this probability still has to be high, but less than 1). Specially, we model the combination of the behavior of: (1) a user watching a display of a partially replicated object (including his/her possible use of VCR-type functionality), where that user is allowed to begin the display after the first T^i_{ea} time units of the object have been replicated; and (2) the corresponding replication process, using a Discrete Time Markov Chain (DTMC) [18]. We then perform transient analysis [18] on this Markov chain to compute the probability that the user attempts to access a part of the object that has not been replicated yet. If this probability results in a reasonable quality of service (QoS), then the corresponding value of T^i_{ea} is acceptable. Otherwise, we recompute with a new value. Although the general approach is applicable to a variety of multimedia applications, the particular values of parameters for the DTMC model as well as what constitutes acceptable QoS depend on the application(s) using the CM server.

Results given in [5] indicate that relatively small values of T^i_{ea} result in good QoS. For instance, in [5] we show that for a system with 90 minute continuous media objects and $T^i_{ea} = 12$ minutes $\forall i$, the corresponding probability, as computed from the DTMC model, that a user will attempt to access an unfinished portion of a replica is below 0.05. Simulation results of the same system, indicate that this probability is nearly 0. (This is due to the conservative nature of our model, which is a good characteristic when QoS is of importance.)

In [5], we also show that this model is not very sensitive to the accuracy of its parameters (e.g., such as the exact probability of a user requesting VCR-type functionality) and thus is of

[4] For ease of presentation, 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, e.g., in bytes.

reasonably practical use. Based on these results, in this work we use the following policies (as described below) for: (a) *replication*, i.e., the process of adding another copy of an object to our system, (b) *dereplication*, i.e., the process of removing a copy of an object from our system, and (c) *triggering* of addition or removal of a copy of an object (i.e., determining *when* a copy should be added or removed).

Replication policy:
We use the SREA (sequential replication + early acceptance) policy, where the replication is performed "sequentially". That is, a single stream is injected into both, the source node (*from* which the replication is performed), and the target node (*to* which the replication is performed). Then, given the value of T^i_{ea} which is determined through the use of the mathematical model of user behavior (as described above), newly arrived users are admitted to the new (incomplete) replica as soon as T^i_{ea} time units of that object have been replicated on the target node.

De-replication policy:
We use the DM (delay migration) policy, which removes a replica of object i only when there is no customer currently viewing object i. This is motivated by the (possible) implementation complexity of migrating customers from one node (where the removal occurs) to another node (which contains a copy of the object being removed).

Replication/dereplication triggering:
We use a *threshold-based* approach to triggering continuous media object replication and dereplication in order to react to an environment where data access patterns change over time. That is, when a request for object i arrives, the system checks if the available service capacity for that object is above the replication threshold. If not, the system triggers a replication process for object i (using the SREA policy described above). On the other hand, if at the time of finishing service of a request for object i: (1) the total available storage *space* capacity has fallen below the space threshold **and** (2) the excess remaining capacity for serving requests for that object goes above the dereplication threshold, the system triggers the dereplication process (using the DM policy described above). The computation of all threshold values is described below.

Threshold-based techniques for reacting to changes in workload are employed often for improving the cost/performance characteristics of systems, 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 and resources to replicate an object or remove a copy, and thus it should be done "sparingly".

Now, we state more formally how the hybrid CM server triggers replication and de-replication processes. 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 $A_i(t)$ 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 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** sufficient available service capacity on the target node.
5. There is sufficient available service capacity in the global switch. (i.e., interconnection network).

Once the replication is triggered, we must select a source and a target node for the replication process. 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 and has available storage capacity. More formally, we choose the node x such that

$$x \notin R_i(t), L_x(t) = \min_{y \in (S - R_i(t))} \left\{ \frac{B_y - L_y(t)}{1 + \gamma_y(t)} \right\}, \tag{1}$$

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).

As already stated, dereplication is invoked at the customer departure instances when an object is determined to have excess service capacity **and** when the total available storage space capacity in the entire system falls below D_S (which is measured in units of CM objects). Since the replication (and in general the dereplication) processes are not instantaneous, "pro-active" dereplication is needed to reduce the probability of the system going into a "bad" state (i.e., a state where we have to reject requests over a long period of time). 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. $A_i(t) = \max_{j \in S}\{A_j(t) > ReTh_i\}$. 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 dereplication 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 $(B_x - L_x(t))$. In general (i.e., for a general dereplication policy), 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 $C_{ix}(t)$. In the case of the dereplication policy we use in this paper, i.e., the Delayed Migration (DM) dereplication policy, there is an additional constraint, namely that $C_{ix}(t)$ must be equal to 0.

3. $\sum_{x \in S} D_x(t) < D_S$

Lastly, to prevent the system from oscillating between replication and dereplication, a difference of H_i is introduced between $ReTh_i$ and $DeTh_i$, i.e., $DeTh_i = ReTh_i + H_i$. That is, we introduce *hysteresis* into the system.

One of the more important issues in designing threshold-based resource activation schemes is the choice of replication and dereplication threshold values. In the following section, we give the details of how we use "dynamic threshold value adjustment" to improve the system's performance. Note that, we do this *without* collecting and maintaining statistics of user data access patterns.

4 Dynamic Threshold Adjustment

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 $p_i(t)$. However, in practice, $p_i(t)$ is unknown and varies over time. Although we could try to collect statistics on access demands for the various objects, many questions would remain open: over which period to collect the statistics, 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 our work we use dynamic data replication techniques which do *not* assume knowledge of access probabilities. More specifically, we use the last interarrival time for object i to (very coarsely) "approximate" $p_i(t)$ and compute threshold values as follows:

1. For each object i, we keep its last request access time at_i. At the time of arrival, t, of a request for object i, we compute the latest interarrival time, i.e., $(t - at_i)$, for that object. We use this latest interarrival time as a very coarse approximation of popularity of object i at time t. Whenever a new request for object i arrives, we update the replication and dereplication threshold values[5] for all objects in the system accordingly, as described next, as well as record the last access time at_i at time t.

2. Then the replication threshold for object i is $ReTh_i = \lceil \frac{\bar{D}}{\bar{D}^w} * \frac{T^i_{ea}}{t - at_i} \rceil$, where $\bar{D}$ is the average storage capacity of nodes in a hybrid system and $\bar{D}^w$ is the average storage capacity of nodes in the wide data striping system. That is, the replication threshold, $ReTh_i$, represents the amount of workload, corresponding to requests for object i, that we expect to receive in the next T^i_{ea} time units, which is the amount of time needed to create a new (virtual[6]) replica of object i (should we deem it necessary). Thus, the motivation for this setting of the replication threshold value is that $\frac{T^i_{ea}}{t - at_i}$ corresponds to the expected number of requests for object i that can arrive in the next T^i_{ea} time units. Note that, depending on the amount of "excess" storage space (i.e., in excess of minimum needed to hold at least one copy of each object) we can afford to be more or less aggressive about triggering replication of objects. Thus, we also normalize $\frac{T^i_{ea}}{t - at_i}$ by the factor of excess storage capacity at the target node x, i.e., by $\frac{\bar{D}}{\bar{D}^w}$.

[5] We discussed the issue of overheads related to recomputation of thresholds in Section 2.

[6] Recall that we only need to replicate a fraction of the object before allowing users to view it, as described in Section 3.

That is, the more excess storage capacity we have, the more aggressive we can afford to be in triggering the replication process.

3. The constraint relating the replication and the dereplication threshold values is as follows: $DeTh_i = ReTh_i + H_i$, where H_i represents the introduction of hysteresis into the system.

4. The hysteresis for object i is computed as follows: $H_i = \lceil \frac{\bar{D}}{\bar{D}^w} * \frac{T^i_{length}}{t - at_i} \rceil$. 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 can 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 resource, which can be used to service requests for object i, are released.

5 Performance/Scalability/Reliability Evaluation

In this section we present results of our study of scalable designs for continuous media servers, using the following cost/performance and reliability metrics (as given in Section 2):

1. the system's *acceptance rate*, which is defined as the percentage of all arriving customer requests that are accepted by the system (with zero waiting time);

2. the capacity (in units of access streams) of the global switch required to support a particular architecture and corresponding acceptance rate;

3. the capacities (in units of access streams) as well as the number of local switches required to support a particular architecture and corresponding acceptance rate;

4. the amount of disk storage (in units of continuous media objects) required to support a particular architecture and corresponding acceptance rate;

5. the mean time to failure (MTTF) of a particular architecture.

This study is performed via simulation, with the following simulation parameters. 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, 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).

Parameter	Default Value
Arrival rate	$a = 1.0$
$ReTh_i$	$\lceil \frac{\bar{D}}{\bar{D}^w} * \frac{T^i_{ea}}{t - at_i} \rceil$
H_i	$\lceil \frac{\bar{D}}{\bar{D}^w} * \frac{T^i_{length}}{t - at_i} \rceil$
$DeTh_i$	$ReTh_i + H_i$
T^i_{length}	90 mins
K	400
System capacity	1600 streams
Replication policy	SREA
De-replication policy	DM
Access Probability change	(1)"gradual" and (2)"abrupt"
Skewness distribution	Zipf, $\theta = 0.0$
$q^i_j(t)$	uniformly distributed between 1 and N, for each object i, $\forall t \geq 0$
Architecture	(1)arch 1 (2)arch 2 group (3)arch 3 group (4)arch 4 group (5) arch 5 group (for details see Table3)

Table 2: Parameters for our simulation study.

In the results presented in this section, we consider the design of a CM server with the following capacity requirements (also given in Table 2): (1) a total service capacity of $N * \bar{B} = 1600$ streams; (2) a total storage capacity of $K = 400$ distinct objects; with (3) each object's length $T^i_{length} = 90$ minutes. Several of the entries in Table 2 require a few words of clarification, which are as follows.

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 the system as a function of such changes. That is, the workload will have the characteristic that every "rotation time period" of X mins $p_i(t)$'s change. The change in access probabilities is described by Equation (3), which is intended to emulate a relatively "gradual" increase/decrease in popularities[7].

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

where t and t' refer to two consecutive rotation time periods and for ease of presentation we assume that K is even. To test our policies further, we also consider a more abrupt change in

[7]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 changes in access patterns for many CM applications.

access probabilities, as described by Equation (4), which is used to emulate an "abrupt" increase in popularity of currently unpopular objects as well as a "gradual" decrease in popularity of the currently more popular objects.

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

Furthermore (at any fixed value of t), we use the Zipf distributions [12] to describe the skewness of the access probabilities, where Prob[request for object i] $= \frac{c}{i^{(1-\theta)}}$ $\forall i = 1, \ldots, K$ and $0 \leq \theta \leq 1$ where $c = \frac{1}{M_K^{(1-\theta)}}$ and $M_K^{(1-\theta)} = \sum_{j=1}^{K} \frac{1}{j^{(1-\theta)}}$. In our experiments we set $\theta = 0.0$, which corresponds to the *measurements* performed in [4] (for a movies-on-demand application). Lastly, we use the following interactivity settings: NP:FF:RW:PAUSE =19 : 1 : 1 : 1 — this is the ratio between the mean amount of time spent in various user playback modes, where NP refers to normal playback, FF refers to fast-forward (with preview), RW refers to reward (with preview), and PAUSE refers to pause. Based to these setting and our mathematical model of user behavior (as summarized in Section 3), we obtain $T^i_{ea} = 12$ mins, given that $T^i_{length} = 90$ mins.

The architectural settings considered in this study are summarized in Table 3, where the possible architectures differ in the number of nodes, service capacity of each node, and available storage capacity of each node[8]. (Recall that we need to meet the overall capacity requirements given at the beginning of this section.) Architecture 1 corresponds to wide data striping. Architecture groups 2–5 correspond to various configurations of a *hybrid* CM server (where the groups differ in the size of the node capacities and subsequently the total number of nodes). For each hybrid CM group architecture below 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, in the experiments that follow, we consider the affect on the overall system performance of limitations of communication network resources. Let nc represent the ratio of the global switch (or interconnection network) capacity to storage system service capacity, i.e., $nc = 1.0$ represents equal service capacities in the storage and communication sub-systems. Then we vary the service capacity of the global switch, i.e., $0.1 \leq nc \leq 1$, and compute the subsequent degredation in performance experienced by the various architectures. The motivation for doing these experiments

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

Arch. type	No. of nodes	Serv. capacity/node Local switch capacity (in streams)	Stor. space/node (in objects)
arch 1	20	80	20
arch 2 group	20	80	22 or 24 or 26 or 28 or 30
arch 3 group	10	160	44 or 48 or 52 or 56 or 60
arch 4 group	5	320	88 or 92 or 96 or 100 or 104
arch 5 group	2	800	205 or 210 or 215 or 220 or 225

Table 3: Parameters for different architecture groups.

is two fold: (1) to observe the performance degredation characteristics of the possible CM server designs (as this is an indication of their scalability characteristics) and (2) to assess whether reductions in overall required global switch capacity are possible without significant loss in the overall system performance, i.e., whether we can operate the system with a smaller network which should lead to lower a cost system.

Lastly, in the experiments that follow, “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*.

5.1 Wide Data Striping System vs. Hybrid System

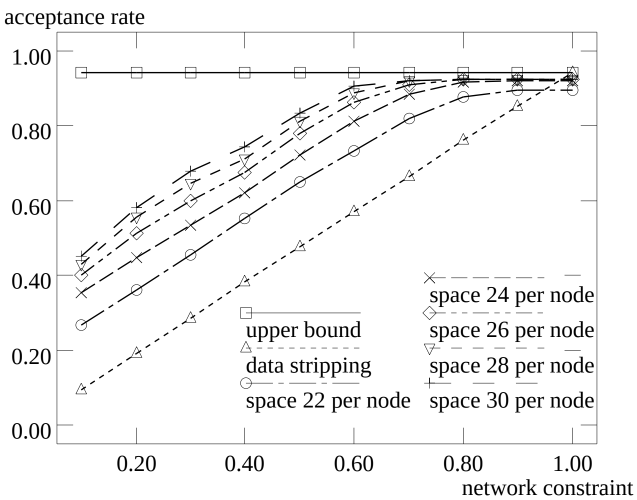

Figure 2: Architecture 2 group under different network constraints.

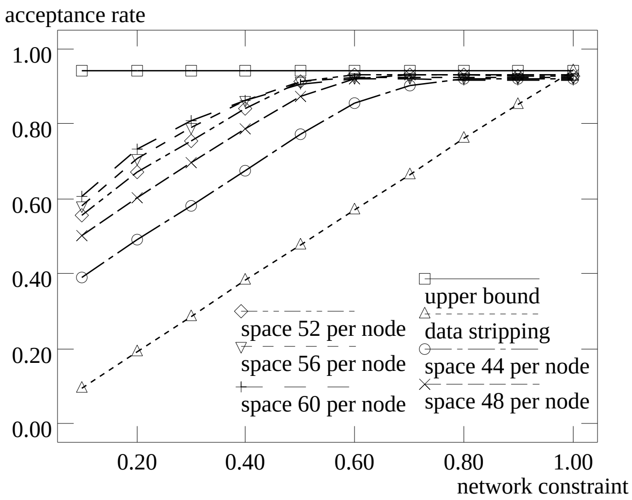

Figure 3: Architecture 3 group under different network constraints.

Figures 2 through 5 and Figure 6(b) illustrate that a hybrid system has better overall performance as well as performance degradation characteristics than the wide data striping system under lower network capacities, i.e., with $nc < 1.0$. 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 capacity and local switch capacity with global switch capacity and achieve nearly the same performance characteristics. For instance, for a particular architectural setting (i.e., with a fixed number of nodes and corresponding node service and local switch capacities) 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. As an example, consider the "arch 2 group", in Figure 2 — in the case of the architecture with a storage space capacity of 24 objects per node, the corresponding required service capacity of the global switch[9] is 1280 streams, whereas in the case of the architecture with a storage space capacity of 30 objects per node, the corresponding required service capacity of the global switch is only 960 streams.

Conversely, the larger the local switch is, the more we can reduce the storage space and global switch capacities. As an example consider the "arch 2 group" in Figure 2 — in this case with a

[9]The needed global switch capacity is determined from Figure 2 by first fixing the acceptance rate that we would like to achieve. In this example, we fix the required acceptance rate to be *at least* $0.95 *$ acceptance rate of "upper bound result". Given the acceptance rate, we can determine, using Figure 2, the smallest network constraint that satisfies that acceptance rate for the appropriate architecture curve. Let that constraint be c; then, the required global switch capacity is $c * 1600$ streams where 1600 streams corresponds to maximum required system capacity in our experiments, as described before.

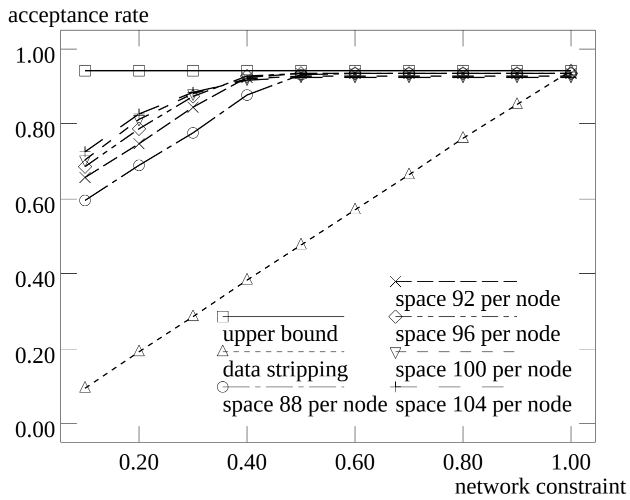

Figure 4: Architecture 4 group under different network constraints.

local switch capacity of 80 streams[10], the corresponding required total storage space capacity is 600 objects (i.e., 30 objects per node) and the corresponding required service capacity of the global switch[11] is 960 streams. Consider now the "arch 4 group" in Figure 4 — 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 (i.e., 92 objects per node). These results are due to the fact that with larger local switch capacities, we can service more customer requests locally (hence the reason for 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 data striping *within* a node (hence the reason for the smaller required storage space capacity per node).

5.2 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, i.e., one would like to size the system resources so as to achieve the best performance possible for the lowest

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

[11]The determination of the required global switch capacity is done as in the previous example.

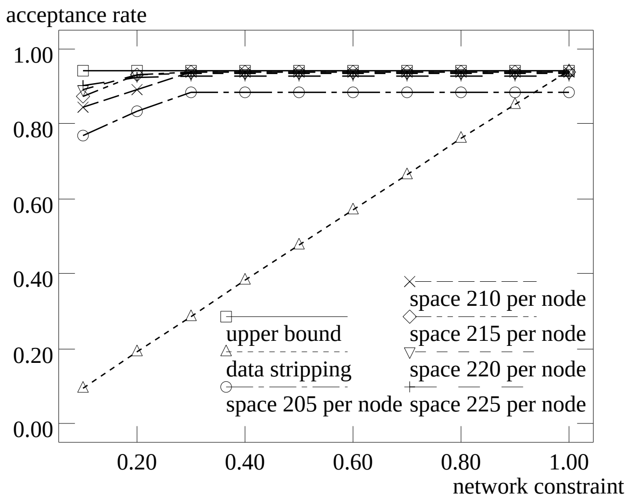

Figure 5: Architecture 5 group under different network constraints.

cost possible. However, cost considerations are a complex issue, given that costs depend on many factors, including the particular technology used for the various components of the system. Thus, next we instead evaluate the different hybrid designs based on the amount/capacity 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 picking a specific technology to use for each part of the system[12]. The purpose of these experiments is to illustrate how a CM server designer can deal with these (fairly complex) *system sizing* issues.

In order to investigate system sizing issues, we further refine our set of architectural settings as described in Table 4. As before, these settings differ in the number of nodes, per node service capacity, per node storage space capacity, local switch capacity, and global switch capacity. We choose the per node storage space capacity and the corresponding global switch capacity of each architectural setting based on the results obtained in the previous section. More specifically, we choose those architectures that can achieve an acceptance rate of *at least* $0.95 *$ acceptance rate of the "upper bound result" with reasonably small per node storage space and global switch capacities[13]. Recall, that the upper bound is computed *without* considering any

[12]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.

[13]If costs of specific system components are known, then one would choose to investigate those architectural settings with lowest costs. Since we do not wish to make technology/cost-related assumptions, we use a heuristic, i.e., we choose architectures that have reasonably small storage and global switch capacities.

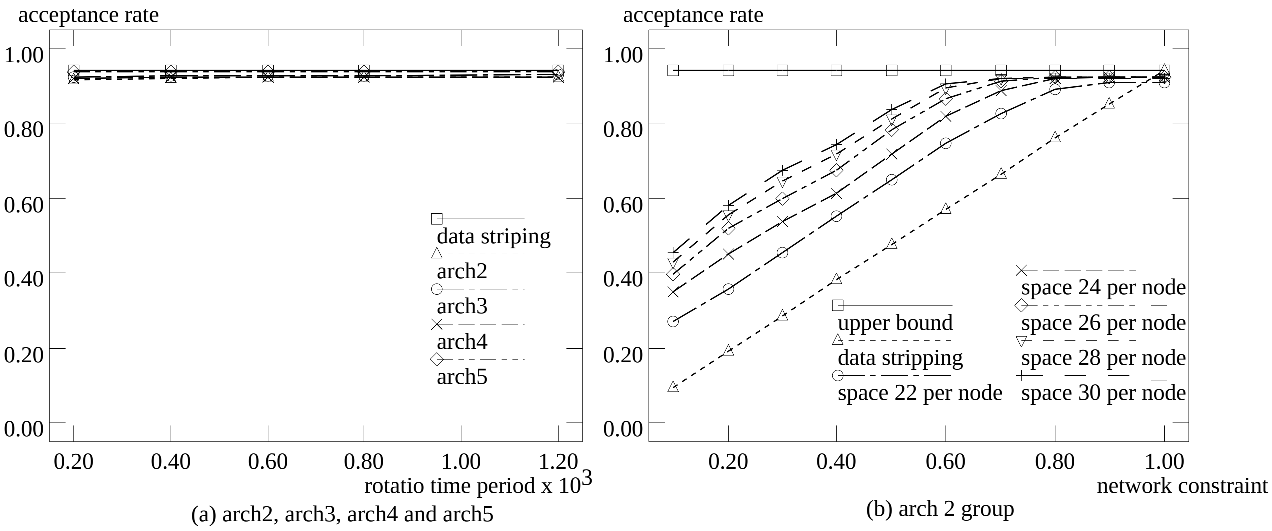

Figure 6: Abrupt increase and gradual decrease in access probabilities.

network capacity constraints. Since the "upper bound results" are not always achievable by architectures we are studying here, we choose a performance goal (i.e., acceptance rate) that is reasonably close to the "upper bound result".

Arch. type	No. of nodes	Serv. capacity/node Local switch capacity (in streams)	Stor. space/node (in objects)	Global switch capacity (in streams)
arch 1	20	80	20	1600
arch 2	20	80	26	1120
arch 2.1	20	80	30	960
arch 3	10	160	52	800
arch 4	5	320	92	640
arch 5	2	800	215	320
arch 5.1	2	800	225	160

Table 4: Parameters for different testing architectures.

Figure 7 depicts the comparison in resource requirements between the various hybrid architectures and the wide data striping architecture. We report each resource requirement as the *ratio* between the resource requirement of a particular hybrid architecture and the wide data striping architecture ("arch 1" in in Table 4), i.e., each graph in Figure 7 represents

$$\frac{\text{resource requirement of arch } i}{\text{resource requirement of arch 1}} \quad \forall i \neq 1.$$

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 the wide data striping architecture (i.e., "arch 1" in Table 4).

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, i.e., local node storage space capacity, local node service capacity and local switch capacity, as well as global switch capacity. Based on these results and current costs, the designer of a large-scale CM server can make system sizing decisions in conjunction with decisions of choice of technologies to use for each of the system's components.

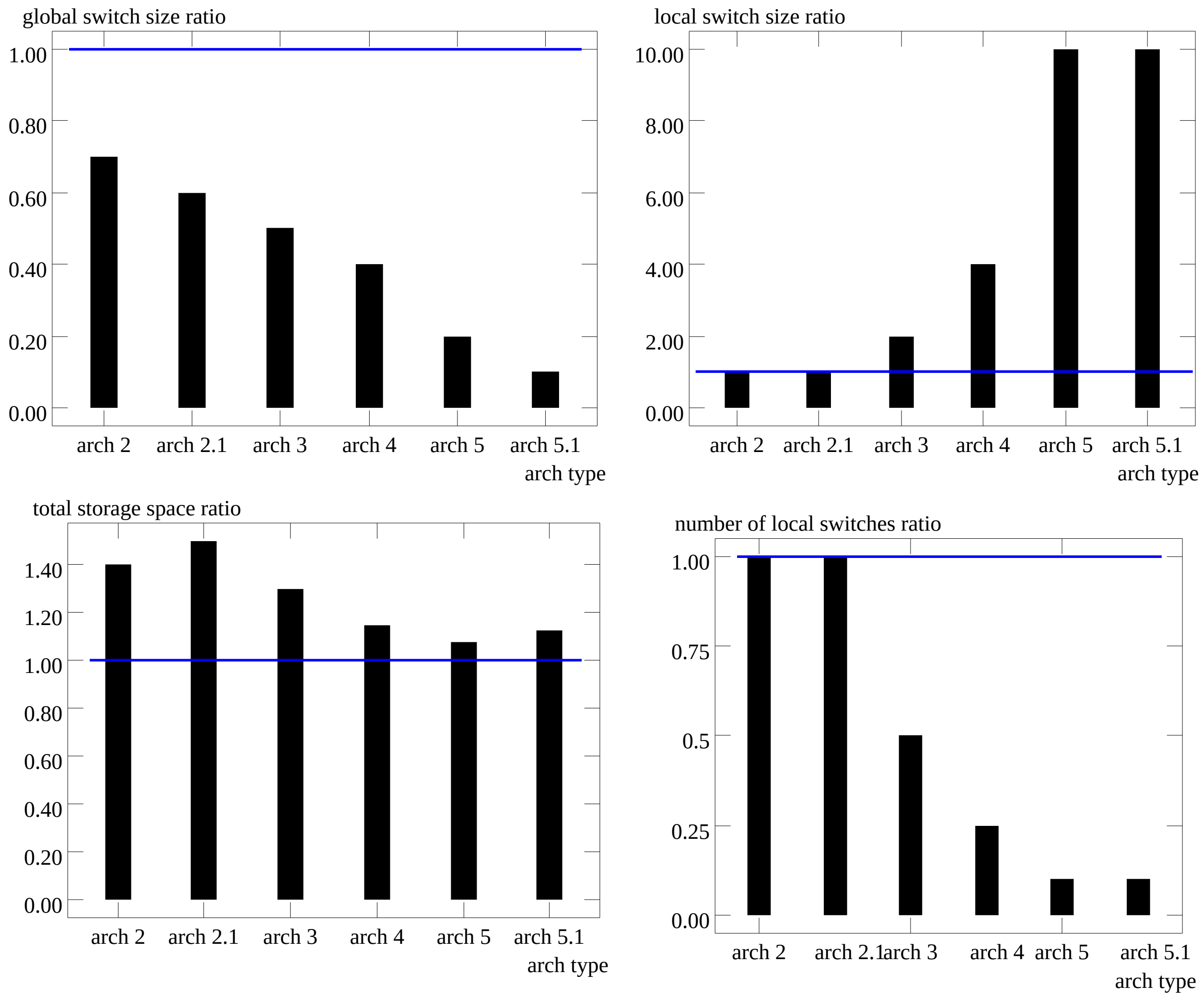

Figure 7: System sizing.

5.3 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 (i.e., same overall capacity) 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 "arch 4" with the storage space capacity of 104 objects per node (refer to Table 3). Specifically, 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. Hence, we have one test case of a 5 node system, with each node having storage space capacity of 84, 94, 104, 114, and 124 objects, respectively and service capacity of 256, 288, 320, 352, and 384 streams, respectively. And, we have another test case of a 5 node system, with each node having storage space capacity of 94, 99, 104, 109, and 114 objects, respectively and service capacity of 288, 304, 320, 336, and 352 streams, respectively.

The rusults, depicted in Figure 8, show that, using a hybrid CM server design, we can achieve heterogeneous system performance that is comparable to homogeneous system performance. We do not show results for heterogeneous wide data striping systems for the following reasons. Although, some research on adaptation of wide data striping over heterogeneous disks does exist, e.g., as in [17], to the best of our knowledge such schemes either give up some amount of disk storage space or some amount of disk bandwidth capacity in order to achieve this adapatation and consequently (usually) at best performance as well as the comparable homogeneous counterparts. Thus, in the case of heterogeneous systems, our evaluation of the goodness of hybrid designs (as compared to wide data striping) is conservative.

5.4 Dynamic Threshold Adjustment

Next, we would like to illustrate that our choice of dynamic threshold adjustment policy, in conjunction with replication and dereplication policies, results in a hybrid CM server that can react to changes in data access patterns accurately and rapidly. This results in the system's performance that appears to be *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. Moreover,

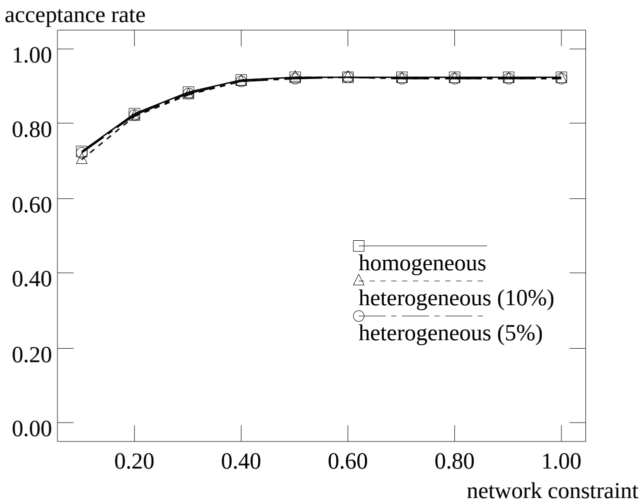

Figure 8: Heterogeneous system.

this performance is comparable to the performance of the wide data striping system, which is "naturally" insensitive to changes in data access patterns.

To illustrate this, we depict the system's performance (i.e., acceptance rate) in Figures 9 and 6(a) as a function of the "rotation time period", i.e., the amount of time that passes before the access probabilities of the various objects change. Recall, that the precise definition of changes in access probabilities for both gradual changes (as in Figure 9) and the more abrupt changes (as in Figure 6(a)) are given earlier in this section.

5.5 Reliability

Lastly, we discuss the fault tolerance characteristics of the wide data striping architecture as well as our hybrid system. We use the mean time to failure (MTTF) as our reliability metric, where the mean time to failure is defined as the mean time until some combination of disk failures results in loss of some data from the storage sub-system (i.e., losses that can no longer be recovered through the use of redundant information, such as parity computation or access of replicas, which in turn is due to losses of too many disks). The details of the derivations of MTTF equations used below are given in the Appedix.

If we employ the use of parity-based redundant information (as in disk arrays) and assume that

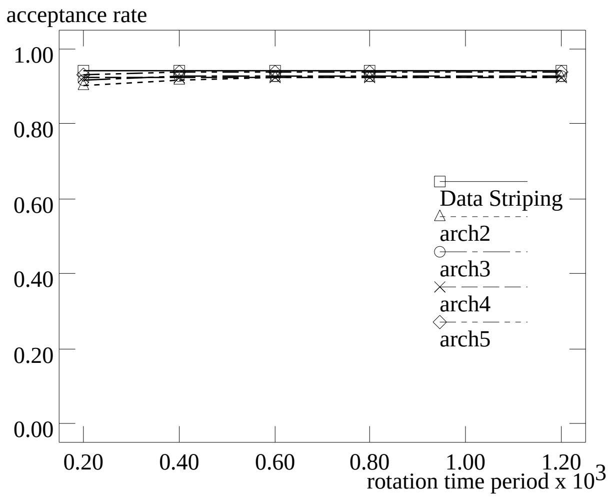

Figure 9: Dynamic threshold adjustment.

disks fail independently with exponential failure rates, as in [15], then the MTTF of an N_d disk wide data striping system with cluster (or disk array) sizes[14] of C disks each is approximately:

$$MTTF \approx \frac{(MTTF_{disk})^2}{N_d(C-1)MTTR_{disk}} \tag{5}$$

where $MTTF_{disk}$ is the mean time to failure of a single disk and $MTTR_{disk}$ is the mean time to repair of a single disk. Note that the $MTTF_{disk}$ depends on the architectural characteristics of the disk and the $MTTR_{disk}$ depends on the reconstruction algorithms used (e.g., as in [14]). Also note that the value of C determines the amount of redundant information stored in the system. For the sake of fairness (in computing MTTF's), we configure all architectures with the same amount of redundant information.

Likewise, the MTTF of a hybrid system with N nodes, $\frac{N_d}{N}$ disks per node, and cluster sizes of C disks each is approximately:

$$MTTF \approx \frac{N(MTTF_{disk})^2}{N_d(C-1)MTTR_{disk}} \tag{6}$$

Note that the above MTTF for *hybrid* systems is conservative, in the sense that it computed based on the assumption of only a *single* copy per object in the entire system. For an object i that has k copies in a hybrid system, the mean time until its data is lost is approximately:

$$MTTF(k \text{ copy object}) \approx \frac{MTTF_{disk}^{2k}}{MTTR_{disk}^{2k-1}} * \left(\frac{N}{N_d(C-1)}\right)^k \tag{7}$$

[14]Here cluster size refers to the number of data disks plus a parity disk in a disk array, i.e., in a cluster of C disks, there are $C-1$ data disks and 1 parity disk [15].

Given these equations, we now show a MTTF comparison between the seven architectures given in Table 4, using the *conservative* estimate for the hybrid system of the MTTF equation based on a *single* copy per object only (i.e., using Equations (5) and (6)). These results are depicted in Figure 10, again as a *ratio* between MTTF of a particular hybrid architecture and the wide data striping architecture, i.e., the results in Figure 10 represent

$$\frac{\text{MTTF of arch } i}{\text{MTTF of arch 1}} \quad \forall i \neq 1.$$

Hence, the straight line at the value of 1.0 in Figure 10 corresponds to the ("scaled") MTTF of the wide data striping architecture.

These results clearly show that we can achieve higher reliability in a hybrid system, even for objects that only have a *single* copy, as compared to the wide data striping system. The increase in reliability, even for the single copy objects, is due to the "isolation" of fault affects, i.e., the wider we stripe an object, the more disk failures can affect the loss of data corresponding to that object. A quantitative expression of this intuition is given in the derivations of the Appendix.

Of course, the reliability is even higher for objects with multiple copies, as is natural in a system which employs data replication. An important point though, is that 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, this brings us to another interesting point. Another reliability-related advantage of hybrid systems is that under network failures (or network partitioning) and/or high workload conditions at remote nodes, local nodes (given that they still have some capacity remaining) can at least deliver *some* objects, e.g., given a movies-on-demand application, even if a movie being requested by the user is not available (due to above specified conditions), the user has the option to choose another movie that may be available. This is not the case for wide data striping architectures, as all nodes (and hence network capacity) must be available in order to service a request for any object.

6 Conclusions

In this work we studied the *scalability* of large continuous media end-to-end server designs as a function of their cost/performance and reliability characteristics under various workload and system constraints. We focused on data placement issues and compared the scalability characteristics of hybrid vs. wide data striping architectures. We have showed that hybrid designs, in conjunction

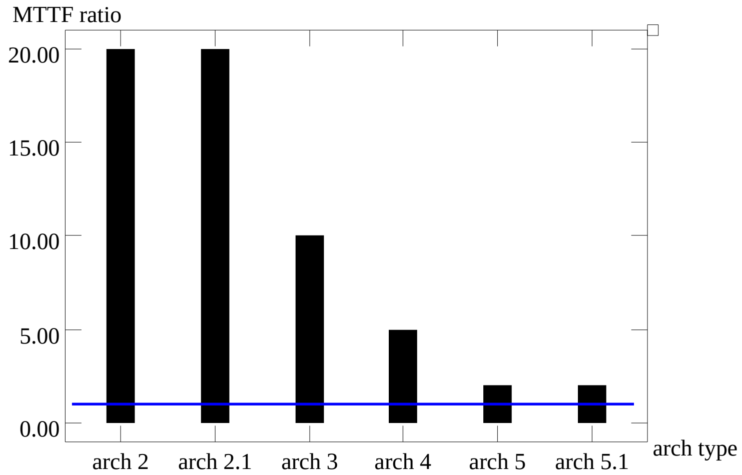

Figure 10: MTTF comparison for *single* copy objects.

with dynamic replication techniques, are less dependent on interconnection network constraints, 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 continuous media servers.

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] M.M. Buddhikot, G.M. Parulkar, and J.R. Cox. Design of a Large Scale Multimedia Stoarge Server. *Journal on Computer Networks and ISDN Systems*, 1995.

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

[5] C. F. Chou, L. Golubchik, and J. C.S. Lui. A performance study of dynamic replication techniques in continuous media servers. Technical Report CS-TR-3948, University of Maryland, October 1998.

[6] 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.

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

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

[9] R. Haskin and F. Schmuck. The Tiger Shark File System. In *COMPCON*, 1996.

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

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

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

[13] P. W. K. Lie, J. C.-S. Lui, and L. Golubchik. Threshold-Based Dynamic Replication in Large-Scale Video-on-Demand Systems. *RIDE '98*, February 1998.

[14] Richard R. Muntz and John C.S. Lui. Performance Analysis of Disk Arrays Under Failure. *VLDB Conference*, pages 162–173, 1990.

[15] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). *ACM SIGMOD Conference*, pages 109–116, 1988.

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

[17] J.R. Santos and R. Muntz. Performance Analysis of the Rio Multimedia Storage System with Heterogeneous Disk Configurations. *Proc. of The 6th ACM International Multimedia Conference*, September, 1998.

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

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

[20] M. Vernick, C. Venkatramani, and T. Chiueh. Adventures in Building the Stony Brook Video Server. *ACM Multimedia'96*, pages 287–295, 1996.

[21] 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 5. 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

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

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 occured in that cluster, a second disk failure will occur within $MTTR_{disk}$ time units. Hence, we have that

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

and

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

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

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

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,

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

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 [16]. 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" [16]. Now we have that

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

Given the above derivation of $MTTF_{cluster}$, we can now compute the $MTTF$ equations used in Section 5 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

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

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 $MTTF_{cluster}$):

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

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[15]) 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[16] **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 [16] where now Prob["failure"]

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

[16] That is, this is the cluster where the first failure occured.

= Prob[Event A occurs within $MTTR_{disk}$]. Consequently, we have

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