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

Cheng-Fu Chou Leana Golubchik
University of Maryland
{chengfu,leana}@cs.umd.edu

John C.S. Lui
The Chinese University of Hong Kong
cslui@cse.cuhk.edu.hk

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. Another approach to dealing with load imbalance problems is replication. The appropriate compromise between the degree of striping and the degree of replication is key to the design of scalable CM servers. Thus, the main focus of this paper is a study of scalability characteristics of CM servers as a function of tradeoffs between striping and replication.*

1. Introduction

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

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

*This research was supported in part by the NSF CAREER grant CCR-98-96232 and the CUHK Mainline Research Grant and RGC Grant.

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

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

In the recent past, a great deal of CM server designs, e.g., as in [2, 3, 9], have focused on "wide" data striping, where each object is striped across all the disks of the system. The potential load imbalance is largely due to the *skews* in data access patterns which, without data striping, could result in high loads on some disks containing the more popular objects, while the disks containing less popular objects may be idling. Moreover, the problem is exacerbated by the fact that access patterns change over time. Thus, an advantage of wide data striping is that it "implicitly" achieves load balance by decoupling an object's storage from its bandwidth requirements. However, wide data striping also suffers from several shortcomings:

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

disks would lead to further complications [1].

2. An appropriate choice of a striping unit, the object size, and the communications network infrastructure dictate an upper bound on the number of disks over which that object can be striped, beyond which replication of objects is needed to increase the number of simultaneous users [3], e.g., to the best of our knowledge, in implementations described in [3, 9] striping is performed over (at most) a few tens of *homogeneous* disks only[1]. Note that, delivery of relatively short CM objects is of use to many applications, including digital libraries and news-on-demand systems.
3. Due to the continuity constraints, some form of synchronization in delivery of a single object from multiple nodes must be considered. The need for some form of "synchronization" arises from the fact that different fractions of an object are being delivered from different nodes at different times during the object's display, and hence some form of coordination between these nodes (and perhaps the client) is required in order to present a "coherent" display of the object.
4. As user demand and data sizes grow and hence the system requires more storage and disk bandwidth capacity, the needed expansion results in re-striping of *all* the objects.
5. Due to the potential need for communication of CM data between the nodes over which the data is striped, the capacity of the communication network limits the performance of the distributed CM server. This limitation directly affects the scalability of the CM server and is one of the main issues we investigate in this work.

Another approach to dealing with the load imbalance problem arising from skews in data access patterns is replication, i.e., creating a sufficient number of copies of a (popular) object so as to meet the demand for that object. Note that, disadvantages of replication are: (1) a need for additional storage space, and (2) a need for techniques that adjust the number of replicas as the access patterns change; some of these issues are addressed in [13], and in our previous work [11, 5]. In addition, here we also improve on the dynamic replication schemes in [11, 5] (refer to Section 3).

In this paper, we consider a *hybrid* approach (refer to Section 2), which addresses the above listed shortcomings. Specifically, the main focus here is on the tradeoffs between striping and replication, which are as follows. In a small-scale CM server, where all disks are assumed to be connected to a single node, data striping can provide better performance characteristics than replication because of its ability to deal with load imbalance problems without the need for additional storage space and without significant networking constraints. However, in a large-scale CM server, data striping can result in a need for significant communication network capacities which can lead to poor scalability characteristics and high costs. Essentially, striping is a good approach to load balancing while replication is a good approach to "isolating" nodes from being dependent on other ("non-local") system resources. That is the wider we stripe in a distributed CM system, the more we are dependent on the availability of network capacity. Furthermore, replication has the benefit of increased reliability (see Section 4) in terms of: (a) longer mean time to loss of data from the disk sub-system; and (b) dealing with lack of network resources, including network partitioning. The downside of replication is that it increases storage space requirements and hence cost. 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.

Related Work and Our Contributions

Much research has been done on design of CM storage servers, e.g., as in [2, 3, 9], which mostly falls into several broad categories: (1) small-scale servers, where in most cases all disks are connected to a single node; (2) medium-scale LAN-based servers, and (3) medium-scale (either distributed or not) servers, which employ high speed interconnects, such as ATM-based technology. To the best of our knowledge, most of these designs employ wide data striping techniques and the corresponding existing successful implementations employ only tens of disks. In contrast, the use of replication for the purpose of addressing workload demand problems has been less explored. In [12] the authors consider skews in data access patterns but in the context of a static environment. In [13], the authors address various questions arising in the context of load imbalance problems due to skews in data access patterns, but in a less dynamic environment (than we investigate here). In [8, 7], the authors also consider dynamic replication as an approach to load imbalance, and in our previous work [11, 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

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

environments than the one considered here. Our motivation in doing away with such assumptions in our work is largely due to considerations of applicability of dynamic replication techniques in more general settings and to a wider range of applications of CM servers.

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

Thus, the main contributions of this work are as follows.

- Quantitative evaluation of *performance* and *resource demand* characteristics of data striping vs. hybrid techniques in large-scale distributed CM servers; such an evaluation is crucial to achieving a *scalable* design of CM 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 changes in data access patterns and user demand.
- Quantitative evaluation of reliability characteristics of data striping vs. hybrid 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* CM systems.

2. Hybrid CM System Architecture

A hybrid system architecture is illustrated in Figure 1(a). It consists of a set of nodes connected by a high speed *global* switch, which is a high bandwidth resource that can, for instance, correspond to a high capacity WAN or an ATM-type infrastructure. Each node i, as depicted in Figure 1(b), 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. Requests from this client which are serviced by a PU from node i are termed "local". When a request from a client cannot be serviced by its local node i, it is forwarded to a remote node j, which contains a replica of the requested object. We term this request "global", as

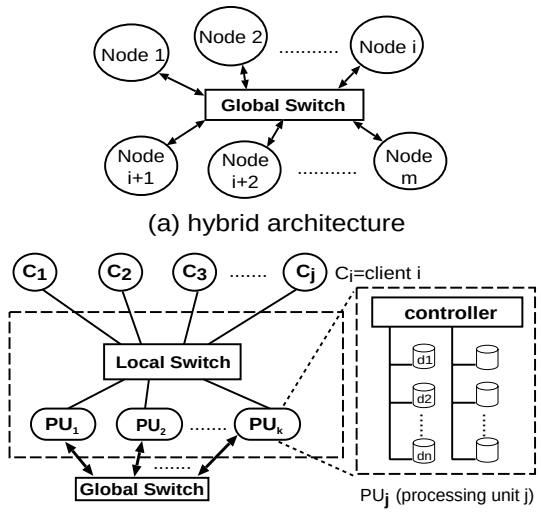

Figure 1. Hybrid System Architecture.

its service requires some capacity of the global switch, i.e., to deliver data from the remote node, through the global switch to the local node and subsequently to the client.

Each PU has one or more CPUs, memory, and an I/O sub-system (e.g., a cluster/array of disks), and it is also connected to the global switch. Each node $x \in S$, where S is the set of nodes in the system, has a finite storage capacity, D_x (*in units of CM objects*), as well as a finite service capacity, B_x (*in units of CM access streams*). 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 local switch service capacity. This flexibility of the hybrid architecture should result in a scalable system which can grow on a node by node basis.

Each CM object resides on one or more nodes of the system depending on its current popularity. An object is striped *only* across local disks which belong to the same node. Objects that require more than a single node's service capacity (to support the corresponding demand) are replicated on multiple nodes. The number of replicas needed to support requests for a CM object is a function of demand, and therefore this number should change as the demand changes. Let $R_i(t)$ denote the set of nodes containing replicas of object i at time t.

Upon a customer's arrival at time t, there is a probability $p_i(t)$ that the corresponding request is for object i and a probability $q_j^i(t)$ that this request is generated by a client local to node j. The admission of this customer into the system proceeds as follows. If at time t object i resides on node j and there is service capacity available at node j, then the system admits and serves

this new request at node j, i.e., locally. Let $L_x(t)$ be the load on node x at time t. If at time t object i does not reside on node j or there is no service capacity available at node j, then the system examines the load information on each node in $R_i(t)$, and if there is sufficient capacity (on at least one of these node and in the global switch), to service the newly arrived request, the system assigns this request to the least-loaded node in $R_i(t)$. Otherwise, the customer is rejected.

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

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]);
2. the capacity of the global switch required to support a particular architecture and corresponding acceptance rate;
3. the capacities as well as the number of local switches required to support a particular architecture and corresponding acceptance rate;
4. the amount of disk storage required to support a particular architecture and corresponding acceptance rate;
5. the mean time to failure (MTTF) of a particular architecture.

3. Dynamic Replication

Since the number of copies of object i partly determines the amount of resources available for serving requests for i, we adjust the number of replicas maintained by the system *dynamically*, as access patterns change. 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, we essentially have conflicting goals of (a) using as few resources as possible for replication 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 (as the replication process continues), as long as the replication of the first T^i_{ea} time units of a new copy of i is complete. To allow customers to have full use of VCR functionality we need to determine a "safe" value for T^i_{ea} which results in a reasonable QoS; we do this in [5] by employing a stochastic model of user behavior. Due to lack of space we do not present the details; however based on the results in [5], we use the following policies in this paper.

Replication policy: We use the SREA (sequential replication + early acceptance) policy, where 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); 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, which is motivated by the (possible) implementation complexity of migrating customers "in mid-stream".

Replication/dereplication triggering: We use a *threshold-based* approach to triggering replication and dereplication. When a customer request for object i arrives at time t, replication of object i (using SREA) is initiated **iff**: (1) $A_i(t) < ReTh_i$, where $ReTh_i$ is the replication threshold and $A_i(t) = \sum_{x \in R_i(t)}(B_x - L_x(t))$; (2) object i in not currently under replication; (3) there is sufficient available service capacity on the source node; (4) there is sufficient available storage space capacity **and** service capacity on the target node; (5) there is sufficient available service capacity in the global switch. Once the replication is triggered, we select the source node, by choosing the least-loaded node in the set $R_i(t)$, and the target node, by choosing node x, where (1) $x \notin R_i(t)$, (2) $L_x(t) = \min_{y \in (S - R_i(t))} \left\{ \frac{B_y - L_y(t)}{1+\gamma_y(t)} \right\}$, where $\gamma_y(t)$ corresponds to the number of replication processes already in progress on node y at time t, **and** (3) the remaining storage capacity on x is sufficient for the new replica.

Dereplication is invoked at the customer departure instances, where a replica of object i at node x is removed at time t **iff**: (1) $A_i(t) = \max_{j \in S}\{A_j(t) > ReTh_i\}$; (2) i has "crossed" the dereplication threshold, $DeTh_i$, i.e., $A_i(t) - (B_x - L_x(t)) - C_{ix}(t) > DeTh_i$ where $C_{ix}(t)$ denotes the number of customers viewing

[2] We do not consider queueing of customers that can not be admitted immediately, as it would entail consideration of scheduling policies for queued requests; these are a function of customers' willingness to wait and the corresponding application.

object i at node x at time t — the above equation is for a general dereplication policy, but in the case of DM, $C_{ix}(t) = 0$; (3) $\sum_{x \in S} D_x(t) < D_S$, where D_S is the storage space threshold for activating dereplication.

Dynamic Threshold Adjustment: Intuitively, we would like the available service capacity for object i to be 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 object access demands, many questions would remain open (refer to [6]). Thus, we use dynamic replication techniques which do *not* assume knowledge of access probabilities. We use the last interarrival time for object i to (coarsely) "approximate" $p_i(t)$ and compute threshold values as follows:

1. For each object i, keep its last request access time at_i. At arrival time, t, of a request for i, compute its latest interarrival time, $(t - at_i)$, and use it as a coarse "approximation" of $p_i(t)$. Whenever a new request for i arrives, update the thresholds for all objects accordingly and record at_i.
2. $ReTh_i = \lceil \frac{\bar{D}}{\bar{D}^w} * \frac{T^i_{ea}}{t - at_i} \rceil$, where $\bar{D}$ and $\bar{D}^w$ are the average node storage capacities in the hybrid and wide data striping systems, respectively.
3. $DeTh_i = ReTh_i + H_i$, where $H_i = \lceil \frac{\bar{D}}{\bar{D}^w} * \frac{T^i_{length}}{t - at_i} \rceil$, i.e., we introduce a hysteresis.

4. Scalability Study

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

Parameter	Default Value
Arrival rate	$a = 1.0$
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 obj. i, $\forall t \geq 0$
D_S	5

Table 1. Parameters for the simulation study.

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

Since the main motivation for using *dynamic* replication is the need to react to changes in 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 as follows:

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

which is intended to emulate a relatively "gradual" increase/decrease in popularities and where t and t' refer to two consecutive rotation time periods; for ease of presentation we assume that K is even. To test our policies further, we also emulate an "abrupt" increase in popularity of currently unpopular objects as well as a "gradual" decrease in popularity of the currently more popular objects as follows:

$$p_i(t') = \begin{cases} p_1(t) & \text{if } i = \text{K} \\ p_{i+1}(t) & \text{if } 1 \leq i \leq K-1 \end{cases} \quad (2)$$

Furthermore (at any fixed value of t), we use the Zipf distribution [10] to describe the skewness of the access probabilities, where Prob[request for object i] $= \frac{c}{i^{(1-\theta)}}$, $1 \leq i \leq K$, where $c = \frac{1}{M_K^{(1-\theta)}}$ and $M_K^{(1-\theta)} = \sum_{j=1}^{K} \frac{1}{j^{(1-\theta)}}$, with $\theta = 0.0$, which corresponds to the *measurements* in [4]. 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[3]. Based on these setting and the results in [5] we use $T^i_{ea} = 12$ mins, given that $T^i_{length} = 90$ mins.

The architectural settings considered in this study are summarized in Table 2[4]. Here arch1 corresponds to wide data striping, where a single copy of each object is striped across all nodes of the system, and arch groups 2–5 correspond to various configurations of a *hybrid* CM server (as described in Section 2). For the hybrid architectures we experiment with different amounts of per node storage space capacity, in order to illustrate the tradeoff between storage space capacity local to a node and the corresponding required capacity of the global switch.

[3] NP refers to normal playback.

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

Arch type	No. of nodes	Srv cap/node Lcl switch cap (in streams)	Stor space/node (in objects)
arch1	20	80	20
arch2 group	20	80	22; 24; 26; 28; 30
arch3 group	10	160	44; 48; 52; 56; 60
arch4 group	5	320	88; 92; 96; 100; 104
arch5 group	2	800	205;210;215;220;225

Table 2. Parameters for architecture groups.

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

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

4.1. Wide Data Striping System vs. Hybrid System

Figures 2 and 3(b) illustrate that under lower network capacities, a hybrid system has better overall performance as well as performance degradation characteristics than the wide data striping system. More importantly, the hybrid architecture allows us to tradeoff capacities of the various system resources in order to achieve a more cost-effective system overall. Specifically, we can tradeoff local storage space and local switch capacities with global switch capacity and achieve nearly the same performance characteristics. For instance, for a particular architectural setting, the larger the local storage space capacity is, the smaller the global switch capacity need be, in order to achieve the same overall system performance, e.g., consider the "arch 2 group" in Figure 2(a) — in the case of the 24 objects/node architecture, the corresponding required service capacity of the global switch[5] is 1280 streams,

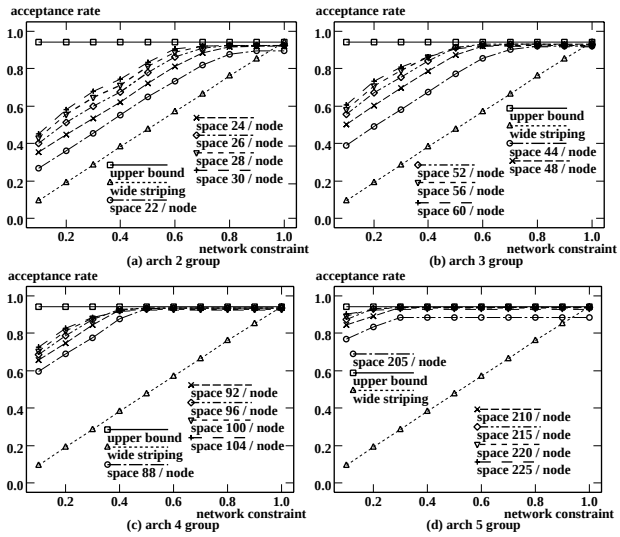

Figure 2. Different network constraints.

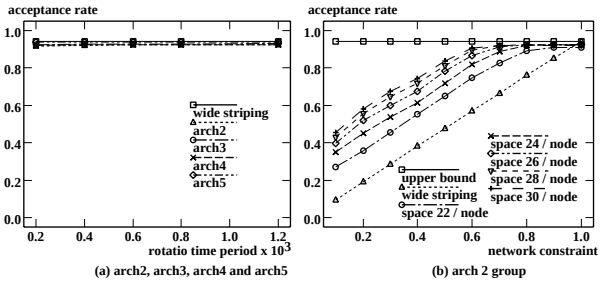

Figure 3. Abrupt increase/gradual decrease.

whereas in the case of the 30 objects/node architecture, it is only 960 streams.

Conversely, the larger the local switch is, the more we can reduce the storage space and global switch capacities, e.g., consider the "arch 2 group" in Figure 2(a) — in this case with a local switch capacity of 80 streams[6], the corresponding required total storage space capacity is 600 objects (i.e., 30 objects/node) and the corresponding required service capacity of the global switch is 960 streams. Consider now the "arch 4 group" in Figure 2(c) — although in this case the local switch capacity increases to 320 streams, the corresponding required service capacity of the global switch drops down to 640 streams and the corresponding required total storage space capacity drops down to 460 objects (i.e., 92 objects/node). These results are due to the fact that with larger local switch capacities, we can service more customer requests locally (hence the corresponding smaller required global switch capacity).

[5] The needed global switch capacity is determined from Figure 2(a) by first fixing the acceptance rate that we would like to achieve. Here, we fix the required acceptance rate to be *at least* 0.95 * acceptance rate of the "upper bound result", and then determine, using Figure 2(a), the smallest network constraint, c, that satisfies that acceptance rate for the appropriate architecture curve; then, the required global switch capacity is $c * 1600$ streams where 1600 streams corresponds to the maximum required system capacity.

[6] These values can be determined from Tables 2 and 3.

Arch type	No of nodes	Srv cap/node Lcl switch cap (in streams)	Stor space per node (in obj's)	Glbl switch cap (in streams)
arch1	20	80	20	1600
arch2	20	80	26	1120
arch2.1	20	80	30	960
arch3	10	160	52	800
arch4	5	320	92	640
arch5	2	800	215	320
arch5.1	2	800	225	160

Table 3. Parameters for testing architectures.

Furthermore, larger local switches and corresponding larger node service capacities also provide more opportunities to take advantage of the load balancing characteristics of striping *within* a node (hence the smaller required storage space capacity per node).

4.2. System Sizing Issues

Quantitatively evaluating the tradeoffs between local switch capacity, node storage space capacity, and global switch capacity is no easy task, as it is not immediately clear how to tradeoff one resource for another. Ideally, one would like to evaluate these tradeoffs based on cost. However, cost considerations are a complex issue, given that costs depend on many factors. Thus, next we instead evaluate the different hybrid designs based on the amount of each resource they require *relative* to the wide data striping system. Such an evaluation quantitatively illustrates to the designer the *relative* merits of the different architectures, without the need for choosing a specific technology[7]. The purpose of these experiments is to illustrate how a CM server designer can deal with these (fairly complex) *system sizing* issues.

We further refine our test cases in Table 3, and choose the per node storage space and corresponding global switch capacity of each architecture based on the results of the previous section, i.e., we choose those architectures that can achieve an acceptance rate of *at least* $0.95 *$ acceptance rate of the "upper bound result" with reasonably small per node storage space and global switch capacities[8]. Figure 4 depicts the results of this comparison for each resource as

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

Hence, the straight line at the value of 1.0 in each of the graphs of Figure 4 corresponds to the ("scaled") resource requirement of "arch 1".

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

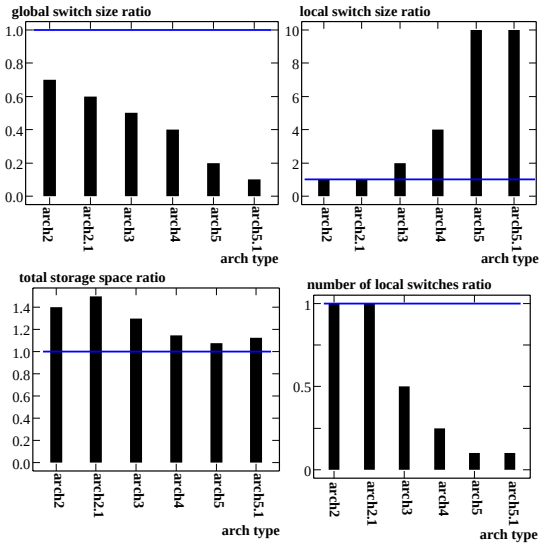

Figure 4. System sizing.

4.3. Heterogeneous Systems

Next, we illustrate the ease of dealing with heterogeneous systems when using hybrid CM server designs *without* loss of performance as compared to an equivalent homogeneous case. For this purpose, we consider a hybrid CM architecture with 5 nodes and a *total* service capacity of 1600 streams. We use two test cases in the following experiments, both based on the homogeneous version of "arch 4" with the storage space capacity of 104 objects per node (refer to Table 2). 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[9]. The results, depicted in Figure 5(a), show that, using a hybrid design, we can achieve heterogeneous system performance that is comparable to homogeneous system performance.

[7] 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.

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

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

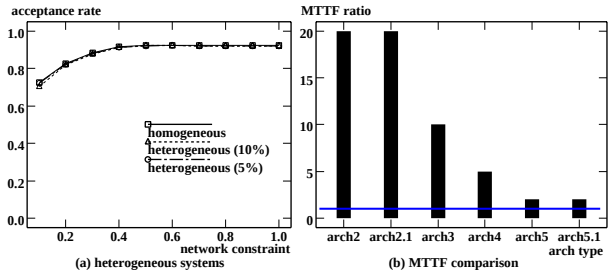

Figure 5. Heterogeneous systems and MTTF.

4.4. Reliability

We use the mean time to failure (MTTF) as our reliability metric, which is defined as the mean time until some combination of disk failures results in loss of some data (i.e., losses that can no longer be recovered through the use of redundant information). We compare the architectures in Table 3, using a *conservative* estimate for the *hybrid* system based on an assumption of only a *single* copy per object (refer to [6] for details). These results are depicted in Figure 5(b), as $\frac{\text{MTTF of arch } i}{\text{MTTF of arch 1}}$ $\forall i \neq 1$, where, the straight line at 1.0 corresponds to the ("scaled") MTTF of wide data striping. The derivations of the MTTF equations used to compute the results in Figure 5(b) are given in [6].

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

Of course, the reliability is even higher for objects with multiple copies, as is natural in a system which employs data replication. Thus, in a hybrid system, we can provide significantly higher reliability for the *popular* objects, as there will always be multiple replicas of such objects in a hybrid system. Lastly, under network failures or high workload conditions at remote nodes, local nodes can at least deliver *some* objects[10], which is not the case for wide data striping, as all nodes and network capacity must be available in order to serve a request for *any* object.

5. Conclusions

In this work we studied the *scalability* of large CM end-to-end server designs as a function of their cost/performance and reliability characteristics under various workloads and system constraints. We focused on data placement issues and compared the scalability characteristics of hybrid vs. wide data striping architectures. We showed that hybrid designs, in conjunction with dynamic replication techniques, are less dependent on 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 CM servers.

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

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

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

[5] C. Chou, L. Golubchik, and J. C. Lui. A Performance Study of Dynamic Replication Techniques in Continuous Media Servers. Technical Report CS-TR-3948, University of Maryland, Oct. 1998.

[6] C. Chou, L. Golubchik, and J. C. Lui. Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication. Technical Report CS-TR-4048, University of Maryland, Sep. 1999.

[7] 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.

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

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

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

[11] 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.

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

[13] 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.