An Implicitly Scalable Real-Time Multimedia Storage Server*

Frank Fabbrocino, Jose Renato Santos† and Richard Muntz
Multimedia Laboratory
UCLA Computer Science Department
{frank, santos, muntz}@cs.ucla.edu

Abstract

We are developing a next generation multimedia server that will provide the foundation for fully interactive access to tremendous amounts and varieties of both real-time and non real-time multimedia data by hundreds of simultaneous clients. Current multimedia servers are inadequate for this task given their limitations of supporting only basic multimedia data types, inherently non-interactive access semantics and intrinsic scaling limitations. Our solution abandons the common use of striping and object replication and implements a random data allocation scheme across a cluster of commodity computers. This scheme provides for load balancing both within and among storage nodes of the cluster while supporting virtually any multimedia data type. This paper presents the essential background, design and implementation, and simulation studies of our system. Our results show that we can guarantee with high probability that an arbitrary I/O requests can be satisfied within a small delay bound while obtaining high system utilization. Although our specific application is a real-time multimedia storage server, principles developed here can be applied to real-time distributed scheduling systems in general.

1 Introduction

Emerging real-time multimedia applications will provide an inherently visual and interactive interface in scientific visualization, hypermedia, multi-user collaboration and entertainment. One such application would allow clients to navigate a realistic model of an urban neighborhood combining relatively simple 3-dimensional models with aerial and street level photographs and video sequences [1]. Yet another application would allow doctors to explore the dynamics of "virtual aneurysms" using complex image processing and fluid flow analysis [2].

Common to all multimedia applications is that the retrieval and delivery of data is subject to real-time constraints, but the interactivity of next generation multimedia applications adds the element of unpredictability since clients, rather than the system, direct access. Therefore, multimedia servers which assume predictable, sequential client access patterns and optimize data layout and scheduling accordingly, will be unable to support the dynamic workload that next generation multimedia applications create. What is needed is a server that can provide strong guarantees of performance for satisfying arbitrary requests without respect to any particular access pattern, while supporting hundreds of concurrent users.

For example, most multimedia servers manage only audio and video data types, and utilize data layout and scheduling strategies that exploit the sequential nature of video but limit client interactivity to only a subset of full VCR functionality. Such systems would be hard pressed to support the demands of a hypermedia news application, for example, where clients will be rapidly and unpredictably switching among text documents, still images and short movie clips on a continual basis.

---

* This research was supported in part by Intel Corporation, NSF Grant IRI-9527178, Microsoft Corporation, and Sun Microsystems.

† Also with the University of San Paulo, Brazil, his research is partially supported by a fellowship from CNPq.

At the UCLA Multimedia Lab, we are developing a large-scale, multi-user multimedia store server that will support both current and next generation multimedia applications. Our server utilizes randomized data allocation with a dynamic load balancing scheme that provides a statistically guaranteed delay bound for I/O performance. Furthermore, to support hundreds of concurrent clients, our server scales incrementally using a cluster of commodity computers. In this paper we present the essential background, design and implementation, and simulation studies of the storage component of our system, the "RIO Storage Server." We show that our system can guarantee with probability close to 1 that an arbitrary I/O request can be satisfied within a small delay bound of around 0.5 seconds, while obtaining system utilization between 90% and 99%.

The remainder of this paper is outlined as follows. Section 2 presents background material on the techniques used to achieve performance and scalability in current multimedia systems. Section 3 presents the fundamental ideas behind our multimedia storage server design. Section 4 presents the design and implementation of our server. Section 5 presents simulation studies of our server. Section 6 presents two closely related works from the literature. Finally, the paper concludes with a brief review and some comments on future work.

2 Background

There have been a number of multimedia servers developed for real-time delivery of audio and video data. Common techniques to achieve performance and scalability in these systems include clustering, striping, and object replication. The next three sections examine each of these characteristics in turn. For clustering, we briefly detail the specific challenges for building a scalable next generation multimedia server. For striping and object replication, we explain why they fail to provide the performance and scalability necessary under the increased and unpredictable workload of future multimedia applications.

2.1 Clustering

Clustering [3] is an attempt to provide the equivalent computing power of larger computers through the combination of many relatively inexpensive, commodity computers. In the same way that RAID [4] provides the opportunity for higher levels of performance through parallelism, clustered systems also have the potential to scale well beyond their monolithic counterparts and provide higher levels of availability and fault tolerance. From the client's perspective however, a clustered system presents a single system image and is indistinguishable from its single machine equivalent.

Unfortunately, the communication latency between nodes in a cluster impacts the system's ability to efficiently balance the workload, accurately present a single system image to clients, and outperform and out-scale a monolithic system.[5] Therefore, for next generation multimedia servers implemented as a clustered system, it is crucial for performance and scalability that synchronization and communication between nodes be minimized and that the workload be evenly distributed among nodes in the cluster.

2.2 Striping

Most conventional multimedia systems stripe data across multiple disks for the aggregate bandwidth and for load balancing [6]. Client requests are processed in cycles of constant duration, wherein data read in one cycle is transmitted to clients in the next. Furthermore, the system carefully schedules disk accesses to balance the load across all disks of the system. The advantage of striping is that it utilizes the bandwidth and storage capacity of all disks in the system while avoiding any contention.

The difficulty with striping is that its advantages degrade for applications with less than constant and predictable access patterns. For example, most systems that utilize striping support only constant-bit-rate video, since variable-bit-rate video would introduce fluctuations in the resource utilization of each stream. Furthermore, client interactivity is limited since synchronized cycles introduce potentially intolerable delays [7] under a dynamic workload. Finally, because of the variability of disk overhead, the length of a cycle is usually set large enough to ensure that all disk accesses are completed by the end of the cycle. The result is a worst-case cycle time with disks likely to be idle toward the end of the cycle—restricting the overall performance of the system.

2.3 Object Replication

Object Replication is the simplest and probably the most popular means of improving a system's performance, scalability and reliability. For multimedia applications, it invariably involves creating a duplicate of a popular multimedia object[1] on another server node, thereby increasing the systems capacity for serving that particular object. Clients now have a choice of nodes to connect to and some amount of load balancing can be obtained. A few multimedia systems even incorporate dynamic/predictive object replication algorithms that try to ensure that only the most popular objects are replicated [8, 9].

Unfortunately, the benefit of object replication is limited because the popularity of an object is not constant. For example, a movie that is popular now won't necessarily be popular next year, next month or even next week! Furthermore, the popularity of an object can vary from hour to hour, with for example, children's movies more popular in the afternoon but dramas more popular in the evening. Even with an accurate dynamic/predictive object replication algorithm, the resources required for the creation and migration of new replicas can reduce a system's ability to perform and scale. Therefore, object replication based on popularity may have limited effectiveness in an environment of future multimedia applications with large numbers of simultaneous clients, high levels of interactivity and large, complex multimedia objects.

3 Randomized I/O

Because of the unpredictable access patterns due to client interactivity and the size and complexity of multimedia data, next generation multimedia servers cannot use data layout and retrieval strategies such as striping that are optimized for sequential access patterns. Furthermore, for a server designed to support hundreds or even thousands of concurrent clients,

[1] "Object" refers to any particular type of multimedia object, including video and audio clips, 3D models, texture images, simple text files, etc., or any combination of these types. In most conventional multimedia servers however, "object" only refers to a video.

scalability should be an inherent part of the system and not obtained through ad hoc techniques such as object replication. The foundation of our multimedia server provides us with implicit scalability that is independent of the multimedia data type, while providing the dynamic real-time scheduling that interactivity requires.

Somewhat paradoxically, our randomized data allocation scheme, or "RIO" for Randomized I/O [10, 11], divides multimedia objects into blocks that are *randomly* placed across all disks in the system so that the client workload will be balanced over time. In this sense, RIO is both a data layout scheme and a load balancing technique but is completely independent of the multimedia data type stored. Figure 1 illustrates how RIO stores a single multimedia object which is divided into ten blocks that are randomly distributed across six disk drives, each with the capacity to store sixteen blocks.

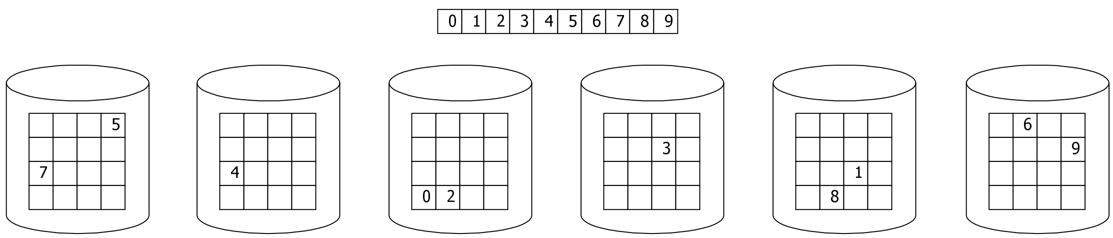

Figure 1: An example random distribution of a ten-block object across six disk drives.

Unfortunately, RIO loads balances well as the number of accesses *grows*, but may not load balance well over small intervals of time. To address this issue, we introduce the concept of block replication whereby a fraction of arbitrary blocks in the system are replicated, and where each replica of a block resides on a different randomly selected disk. When a client requests a block that is replicated, the system retrieves the copy of the block that resides on the least-loaded of the two disks. Figure 2 illustrates block replication with 20% of the blocks from Figure 1 replicated. Simulation studies in [11] show that with a properly chosen replication fraction, system utilization of over 90% can be obtained with a very low probability—one in a million—of violating the real-time continuous media requirement.

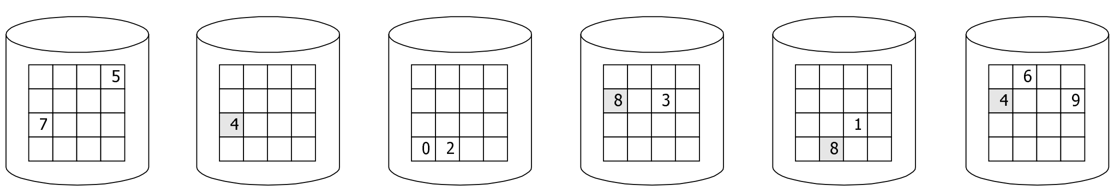

Figure 2: The same distribution of Figure 1 but with 20% replication (replicated blocks are shaded).

However, the clustered implementation of RIO reduces the short-term load balancing that block replication provides because the communication latency described in Section 2.1 lessens the accuracy of the workload information used to route block requests. Since a block can be replicated between any two disks in the system, each copy of a block may reside on two disks in the same node or on two disks in different nodes. In both cases, the system must decide from *which* disk to retrieve the block based on the current workloads of each disk. However, given the interactivity and scalability goals of the system and the inherent communication latency of a clustered implementation, it is possible that the system will incorrectly choose which block replica to retrieve.

To address this complication, we distinguish between two types of block replication in the clustered implementation of RIO: *intra-node replication* and *inter-node replication*. The goal is to "move" the responsibility of correctly choosing which of the replicated blocks to retrieve to the component of the system that has a more accurate knowledge of the relevant disk workloads. In intra-node replication, disk blocks are replicated on disks belonging to the same node, and the choice of which block to retrieve is made by the node itself since it can maintain a more precise measure of the workloads of its locally attached disks.[2] Figure 3 illustrates 20% intra-node replication given the same random distribution of blocks from Figure 1.

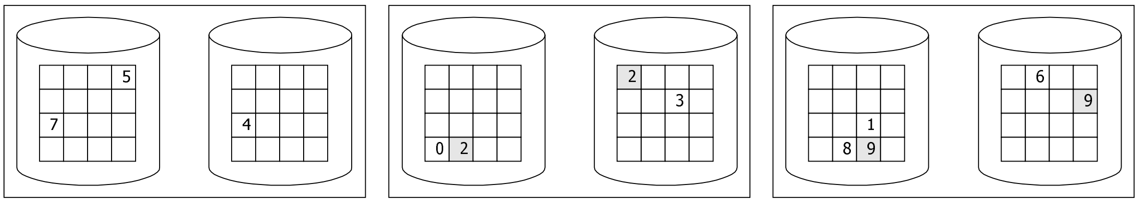

Figure 3: The same distribution of Figure 1 but with 20% intra-node replication (boxes represent node boundaries).

In inter-node replication, blocks are replicated on disks belonging to different nodes and the choice of which copy of the block to retrieve is made by the "Router", a component of the system that mediates between the clients and storage nodes of the system. Although each node can maintain a more accurate measure of its local disks' workloads, the Router must necessarily compare the workloads of two different disks on two different nodes to decide which block to retrieve. It is inevitable that this comparison will be imperfect, but there are a number of ways for nodes to efficiently communicate disk workloads to minimize the potential for error. We explore this topic more in Sections 4 and 5.4. Figure 4 illustrates 20% inter-node replication, given the same random distribution of blocks from Figure 1.

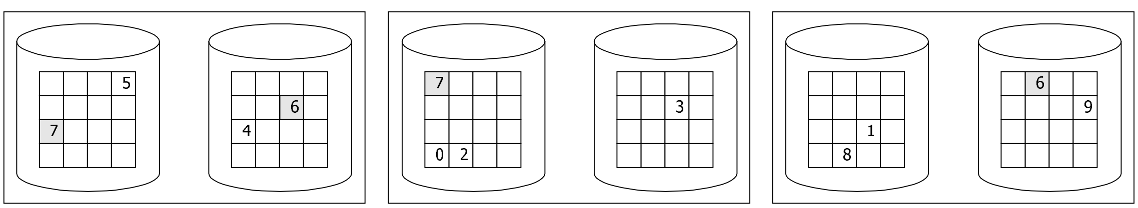

Figure 4: The same distribution of Figure 1 but with 20% inter-node replication

The problem of routing requests to the least loaded node for load balancing has been studied in the context of distributed systems [12, 13]. In [14], it is shown that most of the improvement in load balancing occurs when there are exactly two choices. Although our system provides two types of block replication, a block is only replicated using either intra-node or inter-node replication, and the percentages of total replicated blocks of each type can be specified at system initialization time. Section 5.3 shows through simulations that with the proper combination of intra-node and inter-node replication, a clustered implementation can guarantee with probability

[2] In the "monolithic" implementation of our system, all replication is effectively intra-node since there is only one node.

close to 1, that an arbitrary I/O request can be satisfied within a small delay bound of around 0.5 seconds while obtaining system utilization between 90% and 99%.

Finally, RIO schedules each disk in the system independently and asynchronously in a cyclic manner, wherein a maximum number of client requests are processed in each cycle. I/O requests for disk blocks received in one cycle are queued and serviced in the following cycle. Contrast this scheme with most conventional multimedia servers that use a synchronized cycle across *all* disks in the system—wherein disks are often idle towards the end of the cycle.

In summary, RIO, by virtue of randomly distributing multimedia data blocks across all of the disks in the system, eliminates access patterns that may result in poor load balancing. Furthermore, by replicating a fraction of the blocks using both inter-node and intra-node replication, short-term load balancing is enhanced and very close to optimal performance and utilization is achieved. Finally, the use of asynchronous real-time scheduling of requests at each disk in the system provides the dynamic storage system needed for full client interactivity. The crucial point is that RIO achieves this performance and load balancing *independently* of the multimedia data type.

4 Design and Implementation

For the development of a next generation, fully interactive multimedia server, we have decided on a clustered implementation that will provide the so-called "economy of scale" described in Section 2.1. The previous version of our multimedia server was implemented on a 10 processor Sun Ultra Enterprise 4000 with over 1 GB of RAM and 56 GB of raw disk storage. Our clustered implementation runs on 4 Intel "Balboa" computers, each with two Pentium Pro 200Mhz CPUs, 160 MB of RAM, and 2 Adaptec Ultra Wide SCSI adapters, each with 2 Seagate "Cheetah" Ultra Wide SCSI 4 GB disk drives.[3] Each node runs version 4.0 of Microsoft's Windows NT operating system.

All high level components in the RIO Storage Server are implemented as COM [15] objects and utilizes NT's RPC runtime support for communication.[4] The design of the RIO Storage Server consists of six high level components as illustrated in Figure 5. The next five sections discuss each of these components in more detail.

4.1 StorageDevice

Each node of the system has one or more StorageDevice components that each manage a single disk drive and implement the necessary block I/O operations. For performance reasons, we do not use any file system APIs, caching, or buffering provided by NT, and communicate directly with the physical storage device using a "pass-through" API with a combination of SCSI commands and "raw" I/O commands.

[3] Although we could have attached more than two disk drives to each adapter, doing so would increase the probability of SCSI bus contention.

[4] RPC communication is inherently synchronous. With version 5.0 of Windows NT we will be able to experiment with asynchronous RPC and its potential to improve performance.

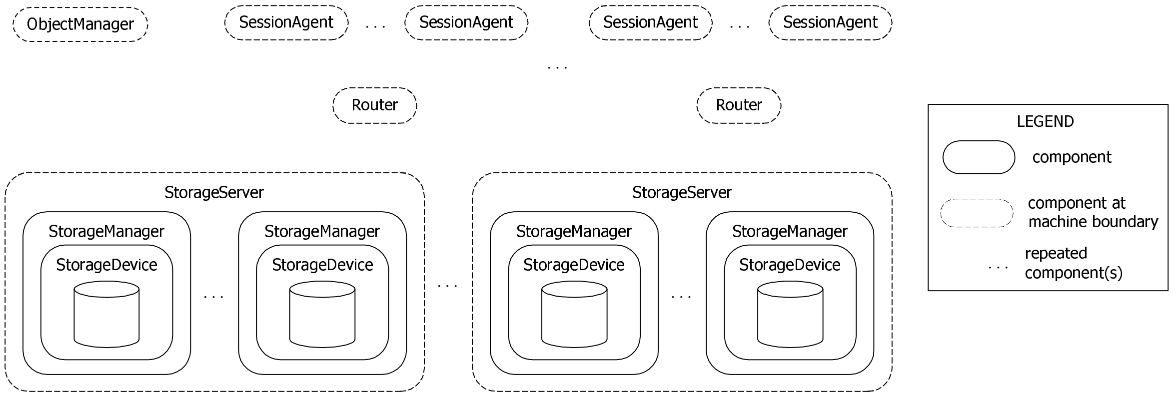

Figure 5: The component architecture of the RIO Storage Server

4.2 StorageManager

Each StorageManager instance manages a single StorageDevice and independently schedules the I/O requests received for blocks that reside on that StorageDevice as described in Section 3. Each StorageManager instance provides two queues for incoming requests, one for real-time requests and another for non real-time requests. The StorageManager periodically removes a maximum number of requests from each of the queues and invokes the StorageDevice to process each request accordingly. The StorageManager then processes requests according to the RTSCAN [11] algorithm, where requests are ordered in an alternating "elevator" scan sequence to amortize disk overhead.[5]

4.3 StorageServer

A StorageServer accepts and distributes incoming block I/O requests for all StorageManagers at a storage node and transmits read data blocks to the requesting SessionAgents. As each request is received, the StorageServer immediately passes the request to the appropriate StorageManager. With intra-node replication, if the block requested is replicated, the StorageServer gives the request to the least loaded of the two relevant StorageManagers. To do this, the StorageServer maintains a record of the request queue lengths for each of its StorageManagers.

4.4 ObjectManager

The ObjectManager is the entry point to the system and manages the creation and destruction of all multimedia objects in the system. Clients that wish to connect to the system, or create, resize, open, close, or destroy a multimedia object, do so through the ObjectManager. To accomplish this, it maintains a database of all devices attached to all nodes of the system, all multimedia

[5] The "cycle size", or maximum number of requests processed in each cycle, and the block size can be configured at system initialization time and are usually determined by the real-time performance guarantees required.

objects in the system and what blocks have been allocated to each, and which blocks on which devices on which nodes are available for allocation.

When a client wishes to create a multimedia object, the client contacts the ObjectManager and requests that an object is created of a given size. The ObjectManager allocates each block for that object according to the algorithm illustrated in Figure 6. This algorithm produces the inter-node and intra-node replication required for optimal load balancing as described in Section 3. Since the ObjectManager persistently stores the allocation information for each multimedia object, when a client requests that a particular object be resized or destroyed, the object need only allocate more blocks or deallocate previously allocated blocks appropriately.

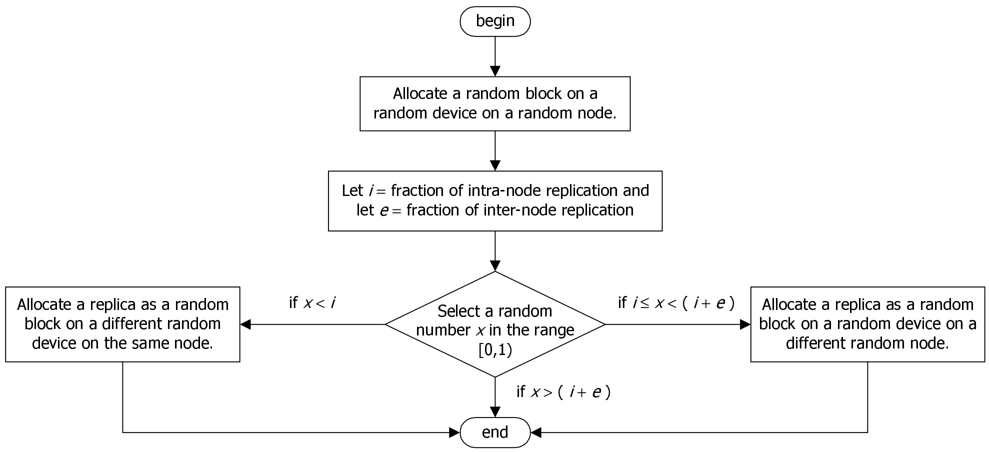

Figure 6: The randomized block allocation algorithm.

4.5 Router

Each Router instance passes block requests received from a non-overlapping set of clients to the appropriate StorageServers. If the requested block is replicated using inter-node replication, the Router chooses the least loaded of the relevant StorageServers to send the request to. There are a number of algorithms that can be used for making this choice, but Section 5.4 explores this topic in more detail using simulations. For simplification, we assume that a Router compares the cumulative loads of StorageServers and not the loads of specific disks at StorageServers.

4.6 SessionAgent

The SessionAgent is an application-specific component that mediates between the end-user and the RIO Storage Server. It communicates with the ObjectManager for object creation and destruction, sends block I/O requests to its appropriate Router, and receives read blocks directly from the StorageServers. Thus, the SessionAgent effectively abstracts out the randomized block I/O semantics of the system and provides the single system image necessary in a clustered implementation. For the system to support new types of multimedia objects and interaction models, only a new type of SessionAgent needs to be developed—the remainder of the system remains unchanged. Preferably, the SessionAgent instance runs on the client machine [16], but may run on any node of the system if necessary.[6]

Figure 7 illustrates a SessionAgent for accessing a 3D virtual reality model. As the end-user "flies through" the model via a 3D viewer on the client machine, the viewer continually sends telemetry information to the SessionAgent, including the current position, trajectory and velocity. The SessionAgent uses this information to access a spatial index of 3D objects for that particular model, and determines what new objects if any the viewer may need to render in the future. If any objects are needed, the SessionAgent uses the block allocation information for each object originally obtained from the ObjectManager to issue block requests to the appropriate StorageServers. Once all blocks are received for an object, the SessionAgent can return the object to the 3D viewer for rendering.

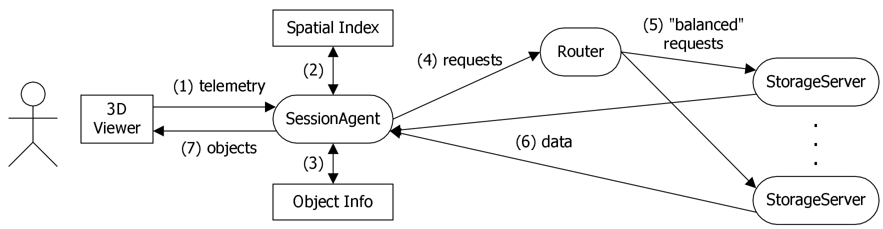

Figure 7: A SessionAgent for accessing a 3D virtual reality model.

5 Simulation Studies

In this section we present the simulation results for the clustered implementation of the RIO Storage Server. The purpose of these simulations is twofold: to easily test design alternatives and to obtain performance numbers that will assist in determining the optimal system configuration. Given the interactivity and scalability goals of the system, many system parameters have to be obtained empirically since we want to provide strong statistical guarantees of performance.

To validate our simulator, we compared in [11] the performance results obtained with experimental performance data from our "monolithic" implementation. For brevity, we do not reproduce the results here. However, we observed that the simulation results were very close to the experimental results, confirming that our simulation is modeling the system performance and disk behavior accurately. The next section describes the simulator in detail and the following three sections present simulations of specific aspects of our clustered system.

5.1 Simulator Description

The simulator architecture is equivalent to the system architecture shown in Figure 5, except that a Traffic Generator replaces a set of SessionAgents for each Router instance. Each Traffic Generator instance generates a continuous sequence of requests with an exponential inter-arrival time distribution—a Poisson process. Although an individual client's request sequence in a typical multimedia application does not follow a Poisson distribution, the superposition of

[6] In our monolithic implementation of RIO, the equivalent of the SessionAgent runs only on the server. To support hundreds of simultaneous clients, its resource utilization could be a detriment to overall server performance.

several independent sequences tends to follow a Poisson process when the number of sequences is relatively large. Since we are assuming a relatively large number of concurrent users, a Poisson arrival process is thus a reasonable assumption.

Requests generated by the Traffic Generator are sent by the Router to the appropriate StorageServer and then by the StorageServer to the appropriate disk queue. We assume that the time for sending requests from the Traffic Generator to the Routers and then to the StorageServers is negligible when compared to the time that the request spends waiting on queues plus the time to complete the actual disk I/O operation. This assumption was verified in our monolithic implementation.

In RIO the size of data block is configured at system initialization time. The larger the data block, the higher the disk I/O efficiency that can be achieved. However larger data blocks require larger memory buffers and increase the probability of reading superfluous data for applications that have random data access patterns and/or small object granularities. The selection of the right block size therefore needs to consider these tradeoffs but is beyond the scope of this paper. Our simulation uses a block size of 128 KB although the results are presented normalized by the mean disk I/O operation time, thus the results will be similar for other block sizes.

We use a RTSCAN cycle size of 1 in our simulations since this value produces the minimum possible delay bound guarantee. For cycle sizes greater than 1, the disk throughput increases slightly, but the delay bound also increases because of the variance introduced by the reordering of requests in an RTSCAN cycle. However, they will have the same general behavior as the results for a cycle size of 1.

The total time to process a disk I/O operation is composed of many components, including seek time, rotational latency, disk transfer time, etc. Although a detailed model of the disk could be used to simulate this complexity, we observed through simulations and experiments that a simple normal distribution is an adequate approximation. We therefore assume a normal distribution of disk I/O operation time for individual data block requests for the simulations described in the following sections. The selected block size of 128 KB and cycle size of 1 results in a mean disk I/O operation $\mu = 35.25$ ms and a standard deviation $\sigma = 5.20$ ms.

In each experiment we simulate the system for a period of time sufficiently large to generate approximately 10^7 requests. We then measure the delay of each request and estimate the delay distribution by computing a delay histogram. The delay bound that is “guaranteed” by RIO is defined such that the probability of a request being delayed by a value greater than the delay bound is less than or equal to 10^{-6}. We thus estimate the delay bound from the histogram obtained in each simulation and plot this value as a function of the system load on the graphs presented in the following sections.

5.2 Single Node Performance

In this section we summarize previous performance results obtained by simulating a single node system in [11] as a basis for studying the performance in a clustered implementation. Figure 8 shows the delay bound that can be "guaranteed" with probability $1-10^{-6}$, normalized by the mean service time of a single block I/O as a function of the system load, which is normalized to a percentage of the maximum possible load. The maximum load is given by the sum of the throughput of all disks, and thus its absolute value will vary for different numbers of disks. Note that the disk throughput is a function of the selected block size and cycle size, and is a fraction of the disk bandwidth as described in Section 5.1.

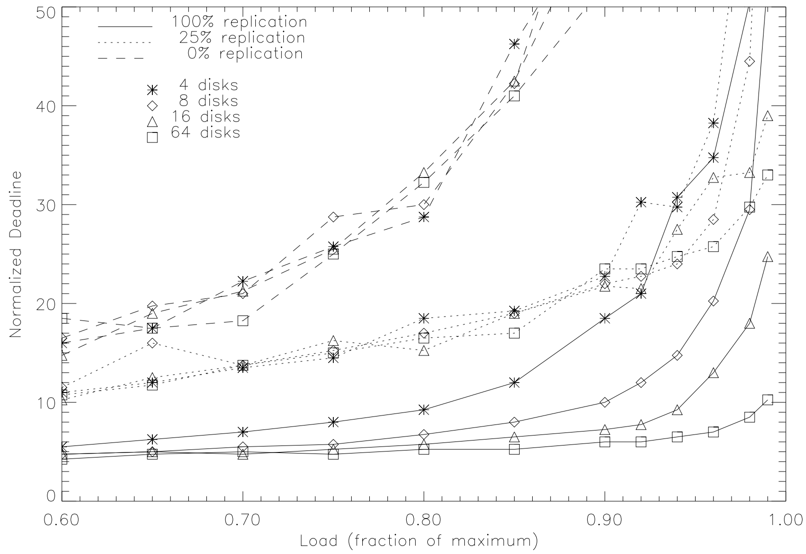

Figure 8: Single node performance

We observe from Figure 8 that as we increase the fraction of replicated blocks, the system can provide lower delay bounds due to the improved load balancing among disks. Furthermore, as we increase the number of disks, the delay bound decreases—especially with higher loads and full replication.

This can be explained as follows. With high levels of replication the system will tend to equally distribute the total number of requests among the system disks. The average number of requests that arrive on the system in any time interval T is λT, where λ is the average arrival rate. For a Poisson arrival process, the standard deviation of the number of requests in T is given by $\sqrt{\lambda T}$ and thus the standard deviation normalized to the mean (coefficient of variation) is given by $\left(\sqrt{\lambda T}\right)^{-1}$. But for the same relative load of Figure 8, the absolute load is proportional to the number of disks N with $\lambda = N\lambda_D$, where λ_D is the average load per disk. Therefore, the coefficient of variation of the number of requests arriving in T is proportional to $\left(\sqrt{N}\right)^{-1}$. If the load is equally divided among the system disks, the coefficient of variation of the disk queue size is also proportional to $\left(\sqrt{N}\right)^{-1}$. Thus, a system with a higher number of disks will have a lower standard deviation on the queue size for the same average load per disk. Thus, the delay bound, which is a function of the tail of the queue size distribution, is lower for systems with higher numbers of disks.

This observation is important when designing a clustered multimedia storage system. For a system with only intra-node replication, each node will perform as well as if it were an independent system but with a delay bound larger than that of monolithic system when viewed as a whole system. This suggest that (1) the number of disks attached to each node should be maximized *before* a new node is added to the system and (2) some degree of inter-node replication is needed to reduce the delay bound of the system. In the next two sections we examine the effects of intra-node and inter-node replication in more detail.

5.3 Intra-Node vs. Inter-Node Replication

In this section we determine the optimal combination of intra-node and inter-node replication by comparing two different cluster configurations, one with 4 nodes and 4 disks per node and another with 8 nodes and 8 disks per node, with a single node with the same number of disks. Figure 9 shows the performance results with different combinations of both intra-node and inter-node replication. These results assume a single Router that has *perfect* knowledge of the total number of requests in each node and therefore, always sends each request to the node with the lowest cumulative load at that moment.

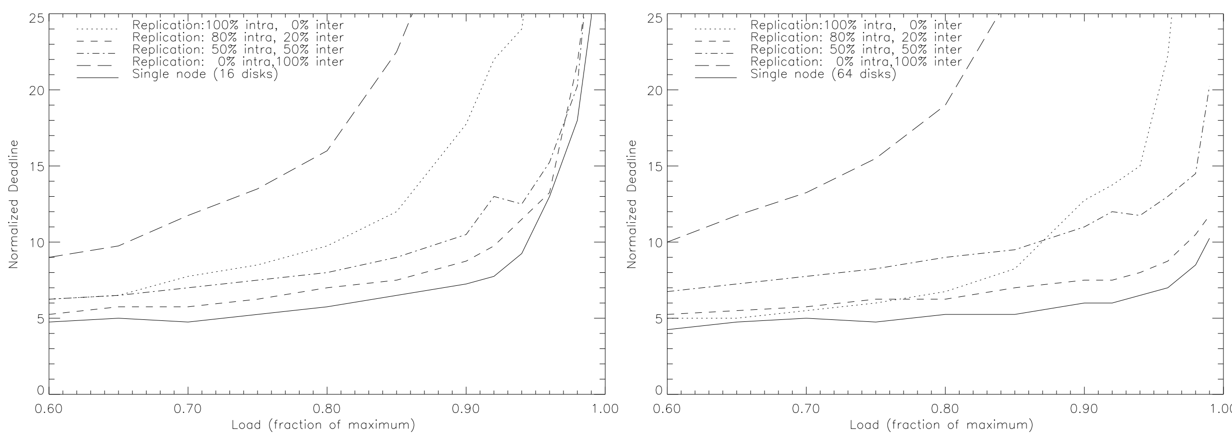

Figure 9: Intra-node vs. inter-node replication with 4 nodes and 4 disks per-node (left) and with 8 nodes and 8 disks per-node (right)

As expected, using only intra-node replication provides better performance than using only inter-node replication, since load balancing across nodes is done using only the aggregate load on a node while load balance inside a node uses the more accurate knowledge of individual disk queue lengths. However, in both cases optimal performance is obtained when the ratio of intra-node and inter-node replication is 80% to 20%. We also observe that using the optimal ratio of intra-node and inter-node replication provides performance just slightly worse than a monolithic system with the same total number of disks, which satisfies our requirement that the clustered implementation provide the approximate equivalent performance of the monolithic implementation.

5.4 Routing Schemes

In the previous simulations we assumed the system had a single Router that had perfect knowledge of the current workload at each node. In a clustered implementation this is not possible because of the communication latency between nodes. Also, for scalability reasons, we might have more than one Router, which complicates load balancing between nodes since each Router can balance only a fraction of the total load and do not have an accurate knowledge of the workload created by other Routers.

Since Section 5.3 showed that some amount of inter-node replication is needed to obtain optimal performance, it is critical that Routers select the correct StorageServer to pass a request to. In the following two sections we discuss two “routing schemes”, ACK-BASED and SLIDING-WINDOW, that can be used to make this selection and we present performance simulations for each. We also vary the number of Routers to demonstrate its effect on overall performance. All simulations use 100% replication, with 80% intra-node and 20% inter-node.

5.4.1 ACK-BASED

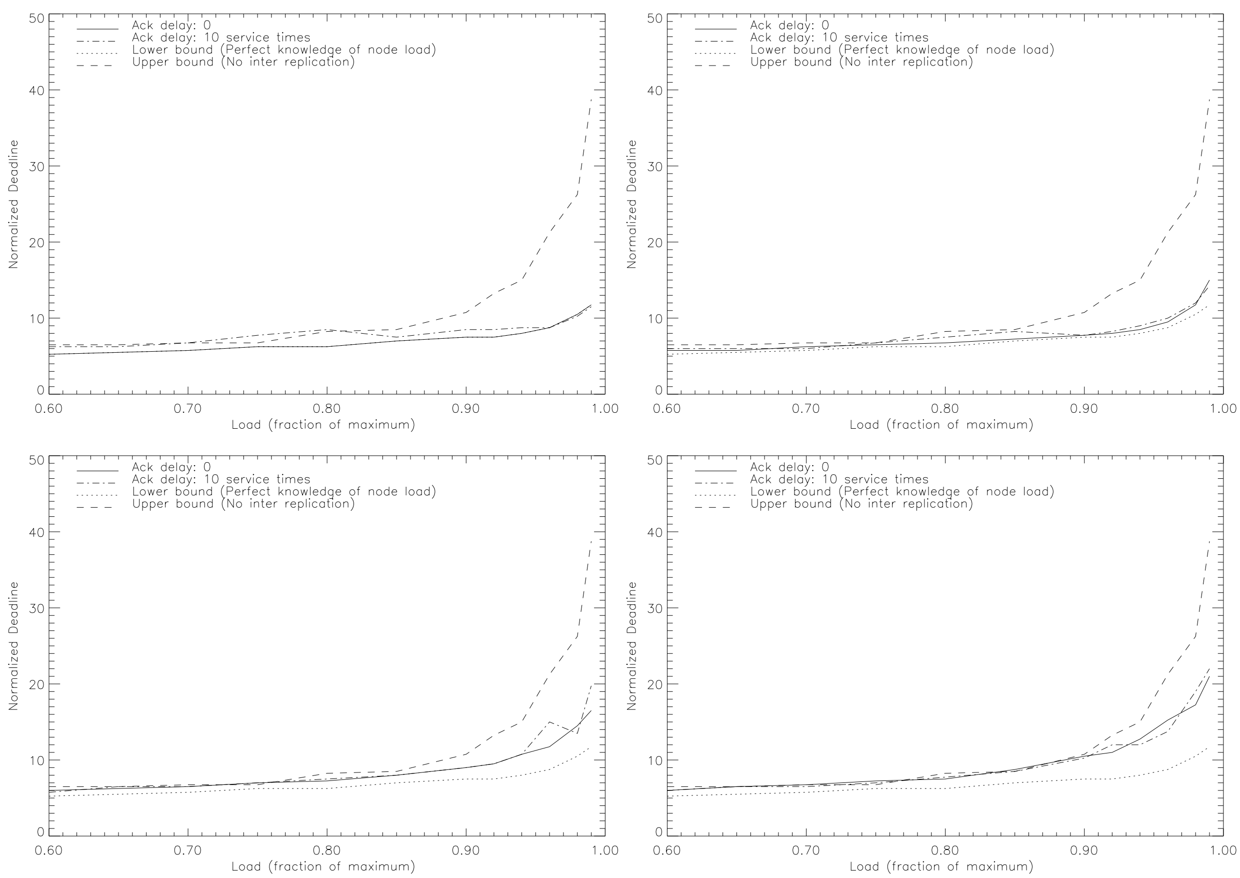

Figure 10: Performance of 8 nodes with 8 disks each using ACK-BASED routing with 1 Router (top-left), 8 Routers (top-right), 64 Routers (bottom-left), and 256 Routers (bottom-right)

The basis of ACK-BASED routing is that if each individual Router balances its fraction of the load, then the total aggregate load will tend to be balanced. This is easily implemented by maintaining a counter for each StorageServer at each Router. When the Router sends a request to a StorageServer, it increments the corresponding counter. When the StorageServer has completed the request, an acknowledgement or "ACK" is sent back to the submitting Router, which then decrements the appropriate counter. Thus, assuming no communication delay, the set of counters at a Router precisely represents its load at each StorageServer and a Router compares the appropriate counter values when faced with an inter-node replication choice. Unfortunately, this scheme imposes an additional overhead of one message per request for sending an ACK.

Figure 10 shows the simulation results for various numbers of Routers. The dashed and dotted lines represent the worst-case and best-case delay bounds that would be achieved with random routing (in other words, no inter-node replication) and with a single Router that has perfect knowledge of node load, respectively. The solid line represents the performance achieved when each Router has perfect knowledge of its fraction of the load only. The dashed-dotted line of shows the performance when the delay of sending an ACK is 10μ. We observe that when the number of Routers is small, performance is just slightly worse than the best case, even if there is a delay on the transmission of ACKs to Routers.

5.4.2 SLIDING-WINDOW

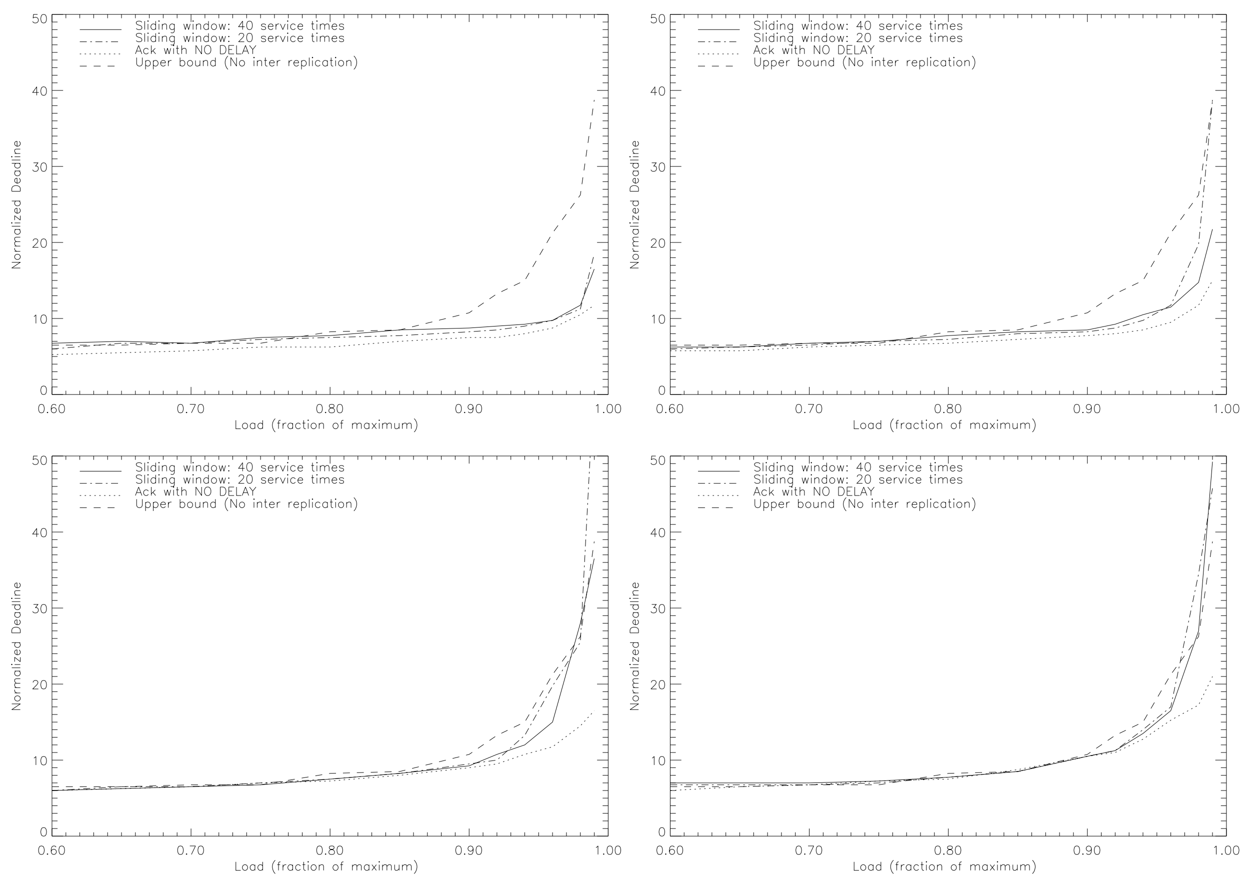

Figure 11: Performance of 8 nodes with 8 disks each using SLIDING-WINDOW routing with 1 Router (top-left), 8 Routers (top-right), 64 Routers (bottom-left), and 256 Routers (bottom-right)

Rather than decrement a counter after the receipt of an ACK, a Router can simply assume that after some fixed worst case time, a request must have been successfully processed since the system provides delay bound guarantees for processing requests. Therefore, for each request sent, after this worst case time has elapsed, the Router can safely decrement the appropriate counter. Thus, the counters of a Router form "sliding windows" of its workload at each of the StorageServers. Although this scheme is less accurate than ACK-BASED routing, the advantage is that no additional messages are required, and this is an important consideration given the scaling goals of the system.

The simulation results for this routing scheme are presented in Figure 11. The dashed line represents the worst-case delay bound as in Figure 10, but for comparison purposes, the dotted line now represents the performance of ACK-BASED routing with no communication delay. We show results for a sliding window of size 20µ and 40µ. We observe that the performance of SLIDING-WINDOW routing is only slightly worse than ACK-BASED routing and degrades gracefully as the number of Routers increases. Furthermore, a sliding window of size 40µ performs slightly better than 20µ at higher levels of utilization and slightly worse at lower levels.

We have presented two routing schemes that provide accurate load balancing among StorageServers. Because both routing schemes perform better when the number of Routers is small, we propose that the number of Routers in the system be limited to a small number of instances. However, we do not see this as restricting the scalability of the system because: (1) the functionality of the Routers is trivial compared to that of the StorageServers, (2) each Router instance runs on a dedicated node, and (3) ten Routers should provide enough "routing" capability for hundreds of StorageServers—together providing a system with potentially thousands of disk drives.

6 Related Work

Although we refer the interested reader to our previous work [11] for more detail on RIO, [17] and [18] both present schemes for real-time multimedia servers that utilize random data placement. In [17], issues in clustered storage servers are explored using queuing system models. The authors note that although random placement provides long-term load balancing, short-term imbalance is possible. Although they use random placement in their models, they do not consider the use of replication as a means of addressing short-term imbalance. Furthermore, they assume simple clients with sequential access semantics.

In [18], the author proposes the use of replicated, randomly distributed blocks as a means of obtaining higher performance, reliability and scalability. The system proposed proceeds in rounds wherein a set number of blocks are retrieved from each disk. However, the resulting synchronized cycles would result in idle disks at the end of each cycle since its length must bound the worst case I/O time. Furthermore, the system retrieves at most one block for each user, limiting both the varieties of multimedia data types and client interactivity supported.

Conclusion

We have presented the essential background, design and implementation, and simulation studies of the "RIO Storage System", a next generation, large-scale, multi-user, multimedia storage server. In review, we enumerate the important advantages of our system:

1. *Support for all multimedia data types:* The performance guarantees and load balancing is completely independent of the multimedia data type stored.

2. *Support for interactivity:* Any and all access patterns at the application level are mapped to the same random access pattern at the physical level.
3. *Implicit and incremental scalability:* RIO utilizes the storage and bandwidth capabilities of all nodes and disks in the system.
4. *Asynchronous disk scheduling:* Since each disk processes requests independently, there is no need for system-wide synchronized cycles that would only hurt performance.
5. *Statistical guarantees of service:* Through the appropriate combination of both intra-node and inter-node replication, the system can provide very strong statistical guarantees of performance.

We have shown through simulations that our clustered implementation can guarantee with probability close to 1, that an arbitrary I/O request can be satisfied within a small delay bound of around 0.5 seconds, while obtaining system utilization between 90% and 99%. Furthermore, we have achieved this level of performance through the innovative use of block-based replication, intra-node and inter-node replication, and efficient routing algorithms. Thus, our clustered storage server provides the foundation upon which to build a scalable, next generation multimedia server.

For future work, we are exploring a number of areas, including implementing higher-levels of functionality such as admission control and an adaptive quality of service scheme that can utilize the idle resource allocations of other clients. Furthermore, we are also looking at a number of alternative routing schemes that might provide even better performance. One idea we are considering is to implement predictive workload models in the Routers that will provide more accurate estimates of the StorageServer loads. Finally, we are considering how to integrate fault tolerance into the system, which is especially important considering the scalability goals of the system.

References

[1] W. Jepson, R. Liggett, and S. Friedman, “Virtual Modeling of Urban Environments”, *Presence: Teleoperators and Virtual Environments*, Vol. 5, No. 1, MIT Press, 1996.

[2] W. Karplus and M. R. Harreld, “The Role of Virtual Environments in Clinical Medicine: Scientific Visualization”, *Proceedings First Joint Conference of International Simulation Studies (CISS)*, Zurich, Switzerland, 1994.

[3] G. P. Pfister, *In Search of Clusters: The Ongoing Battle in Lowly Parallel Computing*, Prentice Hall, 1998.

[4] D. A. Patterson, G. Gibson and R. H. Katz, “A Case for Redundant Arrays of Inexpensive Disks (RAID)”, *SIGMOD ’88*, 1988.

[5] R. Friedman and D. Mosse, “Load Balancing Schemes for High-Throughput Distributed Fault-Tolerant Servers”, *Symposium on Reliable Distributed Systems*, 1997.

[6] B. Ozden, R. Rastogi and A. Silberschatz, "Disk Striping in Video Server Environments", *IEEE International Conference on Multimedia Computing and Systems*, 1996.

[7] S. Ghandeharizadeh, S. H. Kim, W. Shi, and R. Zimmermann, “On Minimizing Startup Latency in Scalable Continuous Media Servers", *Multimedia Computing and Networking 1997*, February 1997.

[8] A. Bestavros, "Demand-Based Document Dissemination to Reduce Traffic and Balance Load in Distributed Information Systems", *Proceedings of SPDP'95: The 7th IEEE Symposium on Parallel and Distributed Processing*, San Antonio, Texas, 1995.

[9] C. Shahabi, M. H. Alshayeji and S. Wang, "A Redundant Hierarchical Structure for a Distributed Continuous Media Server", *Proceedings of the IDMS'97*, September 1997.

[10] S. Berson, R. R. Muntz and W. R. Wong, "Randomized Data Allocation for Real-Time Disk I/O", *Compcon '96*, 1995.

[11] R. Muntz, J. R. Santos and S. Berson, "A Parallel Disk Storage System for Realtime Multimedia Applications", To appear in *International Journal of Intelligent Sciences, Special Issue on Multimedia Computing Systems*, 1998.

[12] D. L. Eager, E. D. Lazowska and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE Transactions on Software Engineering*", 1986.

[13] R. Friedman and D. Mosse, "Load Balancing Schemes for High-Throughput Distributed Fault-Tolerant Servers", *Symposium on Reliable Distributed Systems*, 1997.

[14] M. D. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing", *Ph.D. Dissertation*, University of California at Berkeley, Computer Science Department, 1996.

[15] Microsoft Corporation, *The Component Object Model Specification*, Version 0.9, October 1995.

[16] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, "Using Smart Clients to Build Scalable Services", *Proceedings of the USENIX 1997 Annual Technical Conference*, 1997.

[17] R. Tewari, R. Mukherjee and D. Dias, "Design and Performance Tradeoffs in Clustered Video Servers", *International Conference on Multimedia Computing and Systems*, 1996.

[18] J. Korst, "Random Duplicated Assignment: An Alternative to Striping in Video Servers", *Proceedings of ACM Multimedia*, 1997.