Design of a Fault Tolerant Realtime Storage System for Multimedia Applications*

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

Abstract

We describe the design and implementation of the RIO (Randomized I/O) multimedia object server which manages a set of parallel disks and supports real-time throughput and statistical delay guarantees. The original implementation, on an SMP (Sun Microsystems E4000), has been ported to a cluster of PCs. The new version uses a similar philosophy with respect to allocation of data storage and scheduling of requests. However the parallel "shared nothing" architecture raises all the issues of latency, scalability, etc. that are familiar in parallel and distributed systems. Our requirements are particularly demanding since we expect to support a variety of multimedia applications with varying workload requirements and data reference patterns. 3D interactive applications, video, hypermedia and interactive scientific visualization of terabyte size data sets are among the applications we support. This paper details the design issues we addressed and the results of detailed simulation studies which dictated the design choices made.

1 Introduction

1.1 Motivation

Information systems are used increasingly for storage and retrieval of multimedia data, such as images, video, audio, 3D graphics, etc. These data types are extremely large (gigabytes) and have realtime delivery requirements. The realtime requirement affects the design of multimedia systems at different levels; data storage, indexing, processor and memory management, data communication, data display, etc. In this paper we concentrate only on storage system issues. Other aspects of multimedia systems are equally important, but are not discussed here. Our assumption is that multimedia objects are too large to fit entirely in main memory and need to be retrieved from disk on demand. Further, parallel disks are required for the bandwidth and storage capacities anticipated for multiple clients and for high performance applications.

*This research was supported in part by grants from Intel Corp., and NSF grant IRI-9527178.The second author is also with University of Sao Paulo; and his research is partially supported by a fellowship from CNPq (Brazil)

The design requirements for our system are summarized as:

- *scalable to hundreds of disks*
- *cost effective use of commodity PCs and communication hardware*
- *ability to provide statistical guarantee of maximum response time to serve a request for a data block (order of* 10^{-6} *probability of exceeding the delay bound)*
- *delay bound on the order of a few hundred milliseconds at most*
- *for cost effectiveness, the last two items must be achieved even with high disk utilization*
- *a simple interface to admission control, i.e., relatively simple means to characterize prospective sessions' workload*

Creating Random Access Patterns

New multimedia applications, e.g., 3D interactive virtual worlds, have I/O patterns which are much less predictable than conventional video or audio. In a 3D interactive virtual world application the user navigates through large 3D graphic models at variable speed and along user controlled paths. In order for the display engine to show the current world view, the graphical models of nearby 3D objects need to be continuously retrieved from disk as the user moves. The access pattern to storage objects thus depends on the speeds and paths selected by the user, which makes prediction imperfect at best. 3D virtual world models have been used for different applications such as architectural building design[1], urban city models[4], scientific visualization [5][2], etc.; and will be increasingly common in the future.

Not only do applications such as virtual environments produce unpredictable access patterns but we also want to concurrently support multiple user sessions in which different media types are being accessed. Further the fraction of the workload devoted to the different media types will fluctuate with time which adds another dimension to the predictability problem. Our solution is simple: we destroy all access patterns by random allocation of data blocks to the parallel disks! As illustrated in Figure 1 this converts all access patterns to a uniformly random access pattern to the parallel disks. Thus we have only one access pattern to deal with; albeit one that appears to present some difficulties. This may seem to be a rash decision and indeed we expect sometimes to sacrifice some performance but in return we gain predictability. In fact we will show later that there is actually very little sacrifice in performance for our applications; typically less than 10%.

For the moment, consider the issue of predictability. It may strike the reader as contradictory to randomly allocate data to obtain a reference pattern with no correlation between data accesses and then claims that this leads to predictability. The apparent paradox is resolved as follows. First, we want predictability, not determinism; so having a known stochastic distribution for the reference pattern is appealing. Having unknown behavior that is a function of what applications will be run (even future applications that do not currently exist) is a worse situation. Secondly, for soft realtime systems with statistical guarantees of delay bounds, we want to have response time distributions with "short tails". With just random allocation of data, the successive requests will be uniformly distributed over the disks and the tail of the response time distribution will be relatively "heavy" (i.e., the time τ, such that only one in a million requests suffer a larger latency than τ will be larger than desirable.). The "trick" we invoke is to use random replication of data and online load balancing and show that this can cause an exponential reduction of the tail of the response time distribution.

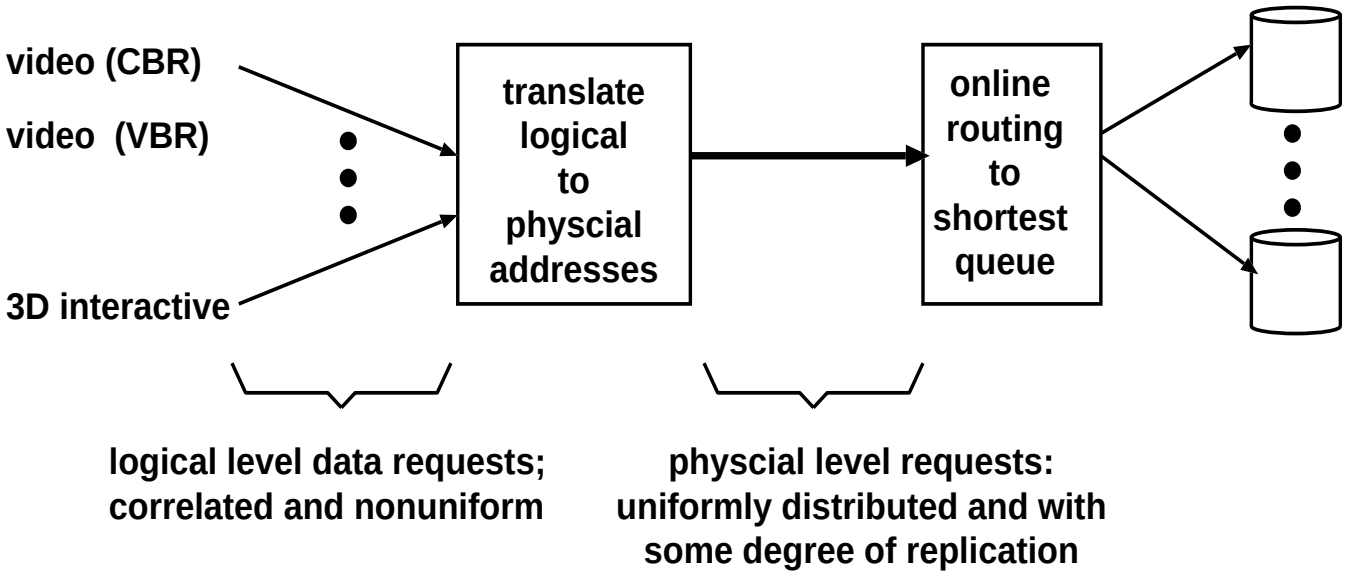

Figure 1: Mapping of all logical level reference patterns to random physical reference patterns and online load balancing based on random replication.

We note that replication for load balancing is not a new idea. In other applications the model is presented in terms of tasks that have to be routed to processing nodes. Probing of the state (e.g., queue lengths) at some randomly chosen set of processing nodes and routing a job to the one least loaded, Here we have to randomly replicate data blocks so that when a request is made to access a block, it is equivalent to randomly

selecting the storage nodes to probe and route to the least loaded. This mapping from processing task to storage blocks is trivial but our application of the idea in this study is still novel for several reasons:

- *we apply the replication and online load balancing to support real-time requirements rather than best-effort, i.e., gains in the mean values of performance metrics*
- *we consider the application in a parallel system in which clients (that originate requests) do not have a global view of the systems and therefore can not route requests to the least occupied disk. We compare and contrast several methods of implementation that effectively overcome this lack of global state information and achieve close to optimum performance.*
- *we consider not only the online scheduling of requests in a nominal system state with all components operational but also consider the tradeoffs between (a) performance in degraded mode (with a failed disk) and (b) the distribution of replicated data to decrease the mean time between data inaccessiblity (i.e., time until both copies of some data item is inaccessible).*

The original RIO prototype was implemented on a SUN E4000 machine, having 10 Ultrasparc processors, 1.25 Gbytes of shared memory, fourteen 4 Gbyte disks dedicated to multimedia data storage, with ATM and Ethernet connections to client machines. This prototype is operational and has been used to simultaneously support delivery of MPEG encoded videos, and 3D urban simulation city models. Other applications for realtime scientific visualization and medical VR are also under development. A port of the system to a cluster of workstations is underway. This paper describes design issues and summarizes simulation and measurement results with particular emphasis on the fault-tolerant issues for this soft-realtime system.

1.2 Organization

The main body of the paper (Section 2) presents simulation results which explore several major design issues. We first show some comparison of simulation results with measurement results which were taken on our current SMP prototype. The results here are mainly to validate the simulation of the disks. Next we describe a range of alternative data layout strategies and evaluate them via the simulation. The metrics used are (a) utilization, (b) delay bound, (c) probability of missing the delay bound. Throughout the paper we assume a probability of missing the delay bound is set to 10^{-6}. So most of the figures show the delay bound that can be achieved as a function of load. We first evaluate the data layout schemes assuming a

centralized scheduler that has perfect knowledge of the disk queues. We consider both normal operation and with one failed disk. We assume that one disk failure is common enough that the system should be designed to continue operation in this case and maintain all ongoing client sessions with the guaranteed QoS. We next introduce the issue of imprecise knowledge of the state of the disk queues and consider, via simulation, several approaches to this problem. One scheme which appears to be new, is to "concentrate" client requests in relatively few "router" nodes which then route requests to storage nodes. The router nodes each independently do their best to balance the load that they manage over the disks. They do this without any direct state information from the storage nodes; rather they use knowledge of where they recently sent disk block requests only. Simulations show the degree to which this is effective.

2 Simulation Results

In order to evaluate the performance of the storage system for a larger number of disks than is currently available on the prototype, a simulator was developed and validated. In this section performance results obtained using the simulator are described.

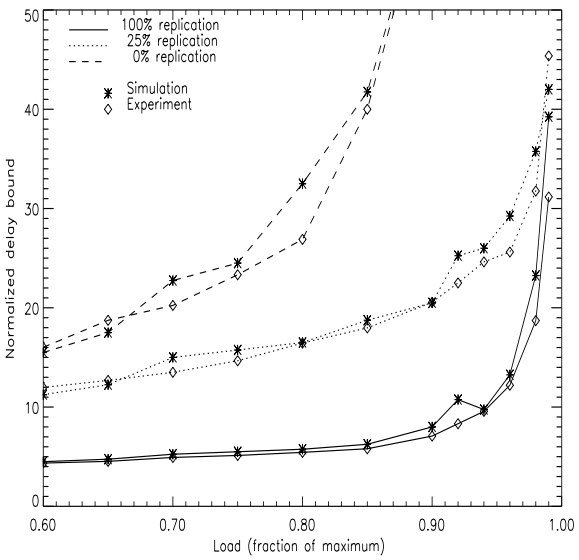

Figure 2: Simulation validation experiments.

In order to validate our simulator we compare the results generated by simulation with the experimental results obtained from measurements on the prototype on the Sun E4000. Clearly the results of the simulation

track very closely the measured results in Figure 2, confirming that our simulation is modeling the disk behavior accurately. Further simulations reported in this paper use the same disk models. It is anticipated that the PC cluster versions may show some differences due to systems software, bus speeds, etc. but these results give us confidence in the simulation being accurate enough for the design comparisons we need to make at this early stage.

In addition to illustrating the accuracy of the simulation, Figure 2 also shows the effectiveness of online load balancing with varying degrees of replication. For example we see that even as little as 25% replication provides a significant decrease in the delay bound that can be guaranteed with probability $1 - 10^{-6}$; particularly under relatively high load. (This observation was also made in [6] based on an analytic model of the same type of system we analyze here but with more restrictive assumptions.)

2.1 Fault tolerance

The use of partial replication as described in the last sections is an effective method for providing short term load balance among the parallel disks. However in order to provide tolerance to disk failures we need to consider alternative schemes. There are basically two approaches; one is to use full replication and the other is to use parity groups such as used in RAID schemes[8].

One possibility is to use RAID technology to provide fault tolerance and partial replication of data blocks to provide short term load balancing. In this approach, the total number D of disks in the system are divided into N_A disk arrays with $D_A = D/N_A$ disks each. Each disk array is a RAID level 5 [8] with parity group declustering [7] that stores data organized as parity groups of size G distributed on D_A different disks ($G-1$ data blocks and 1 parity block; $G \leq D_A$). A fraction α of the data blocks are then randomly replicated across all disks of the system. Replicating data blocks across all disks, instead of replicating them only within the same disk array allows balancing the load among disks in the disk array as well as across disk arrays. Parity blocks are not replicated, since they are only used when a disk has failed and, while in that mode, only to reconstruct data blocks that resided on the failed disk. Data blocks are read during both normal conditions and degraded operation. The ratio of useful data space and total storage space for this scheme is given by:

$$U = \frac{1}{1 + \alpha + \frac{1}{G-1}}$$

Note that this scheme has performance identical to that of a system that does not use RAID, when the system is operating under normal conditions (without failure) with only read operations, since parity groups

are only used if a disk fails. Later, after introducing alternative schemes we will analyze performance when a disk fails.

A second scheme is to use 100% replication for both load balancing and fault tolerance purposes. Since all data blocks are replicated, with any single disk failure all data is still available, i.e., accessible. However, using random replication across all disks does not support simultaneous failures of any two disks, since every pair of disk contains some data that is stored only on that disk pair. Imposing constraints on where data can be replicated can improve fault tolerance by allowing the system to survive at least some combinations of multiple disk failures. However these same constraints can also reduce the capacity for short term load balancing during both normal operation as well as degraded mode.

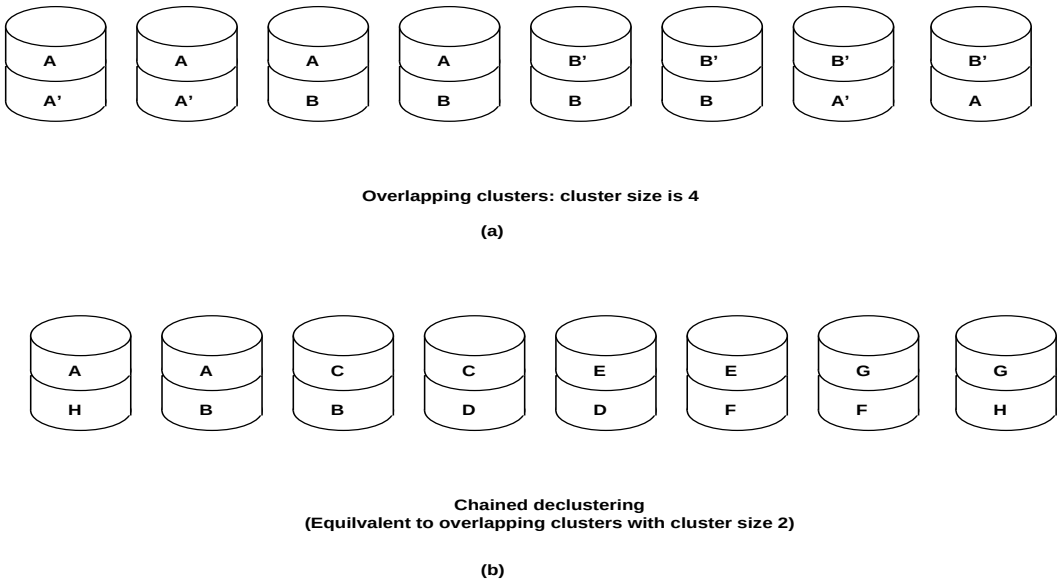

Figure 3: Replication using overlapping clusters; limiting case (cluster size = 2) equivalent to chained declustering.

One such 100% replication scheme is chained declustering [3], in which half the data which is stored on a given disk i is replicated in the next logically consecutive disk $i + 1$. This scheme is illustrated in Figure 3(b). Chained declustering allows multiple disk failures, provided only that no two failures occur on logically consecutive disks. Chained declustering pays for increased fault tolerance with less flexible short term load balancing as our simulation results will show. A generalization of chained declustering, which we call *overlapping clusters*, allows the system designer to trade off fault tolerance and realtime performance. In the overlapping clusters scheme each disk can be thought of a two logical volumes, each with holding half

the storage capacity of a physical disk. Logical volumes are are divided into overlapping clusters of size C and offset by C/2 with respect to the neighboring clusters as illustrated in Figure 3(a). The two copies of a replicated data block are then constrained to be located within one cluster. This scheme allows multiple disk failures provided that no two failed physical disks imply the failure of two logical volumes in the same cluster. This scheme lies between random replication and chained declustering since it supports more disk failures than random replication and less than chained declustering, while providing more flexibility for short term load balancing than chained declustering but not as good as random replication.

Overlapping clusters can be implemented in a cluster of PCs using dual ported disks as illustrated in Figure 4.

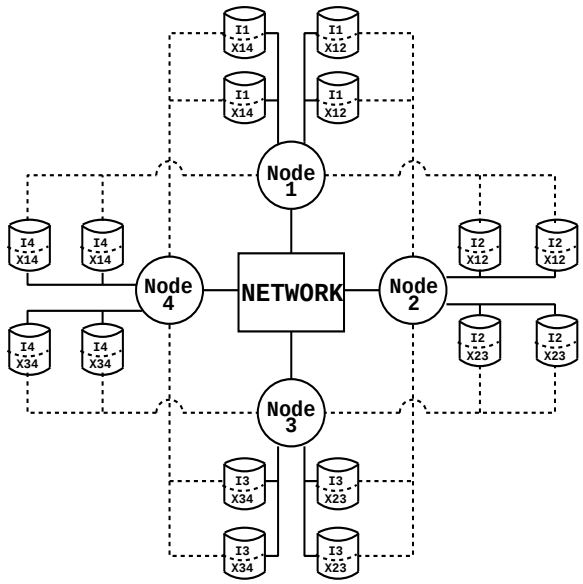

Figure 4: Cluster architecture supporting overlapping clusters and load balancing in degraded mode.

2.2 Scheduling with perfect knowledge of disk queue status

The first set of simulation results compares the data allocation schemes described above both in normal operation and with one failed disk. To separate considerations we assume in this first set of experiments that requests are routed to disks with perfect global knowledge of the disk queues. Later we will consider how scheduling can be done when, due to the distributed nature of the PC cluster, only imprecise knowledge of the disk queues is available.

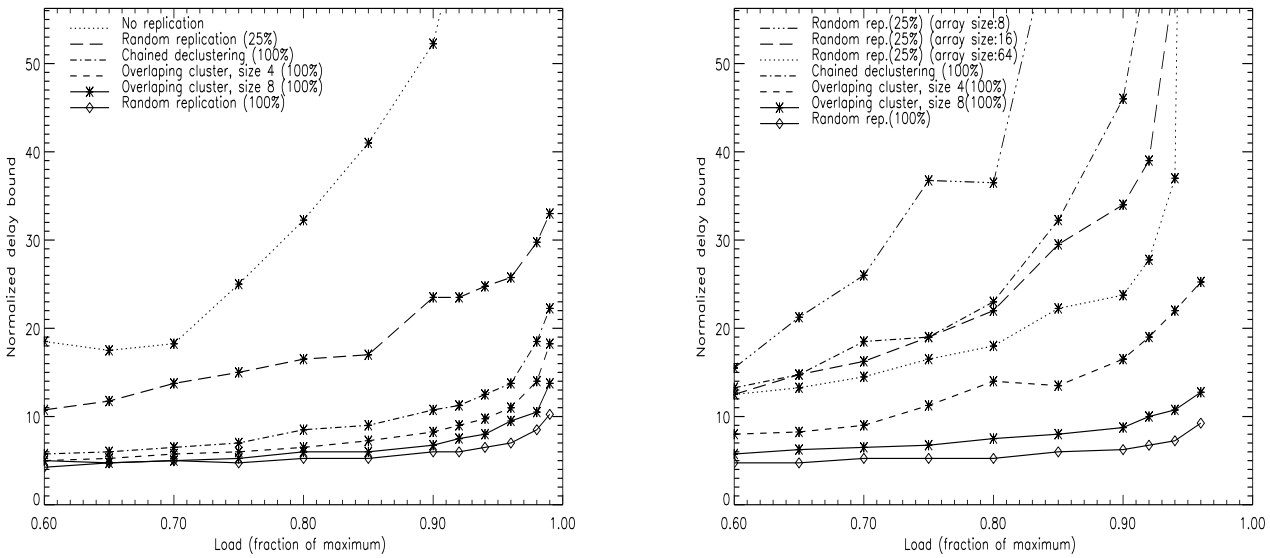

Figure 5: Perfect knowledge online router; nominal performance vs. with one failed disk.

Unless otherwise stated, all simulations are for systems with 64 disks. When RAID is discussed, unless otherwise stated, the parity group size is 5, i.e., 4 data blocks and a parity block.

Figure 5 (left) shows the delay bounds that can be obtained with several of the choices of data layout that have been described above. First, the value of random replication is clear by considering the three curves that correspond to random replication of 0%, 25% and 100% of the data. It is clear from these curves that the delay bound that can be guaranteed with probability $1 - 10^{-6}$ grows considerably with load when there is no replication but is relatively flat when replication is used, even out to 90% utilization or more. The implication is that, if admission control will allow utilizations up to 90%, then with no replication the delay bound will have to be set at about 50 times the disk service time; a rather large value. Note that if a lower delay bound is guaranteed to a session when the utilization is lower, then admission control would be held to that guarantee and if further clients requested service, they might have to be denied service in order to meet the delay guarantee made to the earlier session. For this reason we assume that the delay bound is chosen according to the maximum utilization that will be permitted. So, for example, if the largest utilization allowed is 95%, with 100% random replication we could guarantee a delay bound of about 5 times the average disk service time. With 25% random replication the delay bound would be about 22 times the average disk service time.

Also from Figure 5 (left) we see that chained declustering and the overlapping cluster schemes work reasonably well and, as expected, the larger the clusters are, the more effective it is for short term load balancing and the flatter the curves are for a larger range of utilizations.

Figure 5 (right) shows the same type of plot but with one failed disk. In this part of the figure there are a couple of new curves. These correspond to the use of RAID arrays with some additional random replication of data blocks for load balancing. We compare, for example, various size RAID arrays (remember the total number of disks is 64 in all cases) and with 25% random replication of data blocks. We choose to only look at 25% replication since, with $G = 5$, there is already 25% overhead for replication and by randomly replicating an additional 25%, the total is already up to 50%. Any further replication would be close enough to 100% that it would seem simpler and more effective to just use a 100% replication scheme and be done with it. However, if storage capacity is scarce, the use of RAID with some random replication for short-term load balancing is a possible alternative. (We note that in recent years disk storage capacity is increasing significantly faster than transmission or access rates so that, increasingly disks will be limited by bandwidth as opposed to storage.)

From the results in Figure 5 (right) it is clear that the various schemes retain the same relative performance as with no failed disk. However the differences are more pronounced, i.e., the curves separate significantly. Again, if we wanted to allow some maximum utilization and also, not to have to drop any ongoing sessions when a disk fails, we can read the delay bound that can be guaranteed from the curves in the righthand figure. So for example, with overlapping clusters of size 8, and allowing up to 90% utilization, the delay bound that could be guaranteed (with one chance in a million of missing) is about 8 times the average disk service time. For 100% random replication, the delay bound, under the same circumstances is about 6. So with the overlapping cluster scheme we have achieved close to the same performance (with and without failure) as with completely random replication. The overappling clusters also has somewhat better probability of surviving a second disk failure without loss of data availability. [1]

For reference Table 1 shows the relative mean time between "data inaccessible" failures (failures where a fraction of data is unaccessible) for all cases. We assume that the mean repair time of disks is much smaller than the mean time between disk failures in the system. If this is the case the mean time between "data inaccessible" failures are dominated by double failures, and it is proportional to the number of disks in which

[1] We do not expect that the system will continue to operate without dropping sessions in the (rare) case of a second disk failure before the first is repaired. However, as long as data does not become inaccessible, the rebuild of a failed disk can be fairly efficient. When data becomes inaccessible from disk, the system must resort to tertiary storage and the rebuild time can be much, much longer.

Scheme	Mean time between "data inaccessible" Failures
Chained declustering	$\frac{K}{2}$
Random replication (100%) (System with D disks)	$\frac{K}{D-1}$
Disk array (size 8)	$\frac{K}{7}$
Disk array (size 16)	$\frac{K}{15}$
Disk array (size 64)	$\frac{K}{63}$
Overlapping clusters (size 4)	$\frac{K}{5}$
Overlapping clusters (size 8)	$\frac{K}{11}$

Table 1: Relative mean time between "data inaccessible" failures

a second failure (after a failure in a specific disk) would cause unavailability of data. Note that if the number of disks is very large this assumption may not hold and we may have to consider multiple failures with more than 2 failed disks, but in general this is a reasonable approximation. Table 1 shows the approximate mean time between "data inaccessible" failures relative to a constant K that can be computed as function of the mean time to failure of a disk and the mean time to repair a disk and is the same for all of the systems. The time to data inaccessibility is inversely proportional to the number of disks for which a second failure would cause inaccessibility of some portion of data.

2.3 Decentralized disk scheduling

Figure 6 illustrates the major factor that we have to deal with in the clustered version of the object server. The number of clients is expected to be large (e.g., hundreds) and the number of storage nodes to be up to a few tens. Clients are not expected to directly access storage nodes for several reasons. One is that they should not have to be aware of the details of the replication and load balancing. Also there are potentially

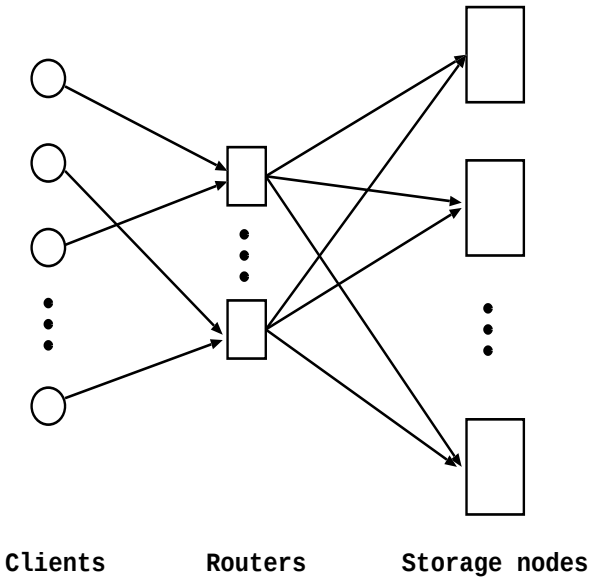

Figure 6: Client requests go through "routers" which choose replica to be fetched.

so many of them that it would be costly to keep them up to date on the load on the storage nodes, let alone the disks. Our proposed solution is to have a front end set of nodes that interface to clients. While these nodes will also perform other functions (e.g., admission control, authorization, etc.) we concentrate here on their function in routing client requests to achieve short-term load balancing and thereby, relatively short delay bound guarantee. (Incidentally, when sessions are initiated they are assigned to some router with an aim to approximately equally load the routers.)

The routing function is relatively simple so we expect only between one and ten routers at most. The obvious question is what kind of information the routers will have in order to make routing decisions. In this subsection we present some initial simulation results investigating alternatives.

We use the terms *node routing* or *two level load balancing* to mean routing to a storage node to balance the aggregate load at storage nodes. The assumption being that, when there are multiple disks per storage node, the local disk scheduler does routing to the exact disk and uses available replication to do short-term load balancing locally. It should be clear where the two level load balancing terminology comes from. We use the term *disk routing* or *one level load balancing* to mean that the router nodes have estimates of disk queues and make the decision as to which disk (when there is a choice) will serve a particular request.

Figure 7 shows a comparison of two things: one is the effect of "two level" versus "one level" routing

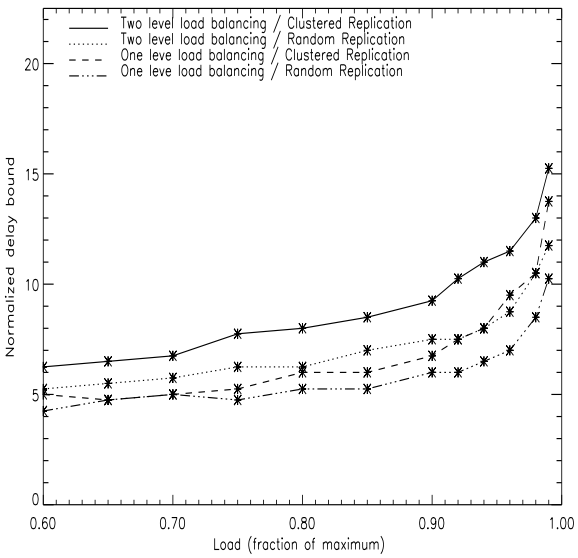

Figure 7: Node (two level) routing vs. Disk (one level) routing: clustered replication vs. random replication.

and the other is random replication versus overlapping clusters. In these simulations we still assume that the router has exact knowledge of state information that it utilizes, i.e., the aggregate number of requests queued at a node for two level routing and the number of requests at each individual disk for disk routing. Later we look at several schemes for maintaining approximate state information at the routers.

Obviously one would expect the two level routing to be less effective than the disk (one level) routing. The results in Figure 7 bears this out. Also the "full" random replication is more effective than the overlapping clusters. So for example, with overlapping clusters (for fault tolerance) and two level scheduling (which may be more practical) the delay bound at 90% utilization goes from between 5 and 6 times average disk service time to about 9 or 10 times the average disk service time; not quite double.

Figure 8 illustrates, for two level scheduling, and several size systems (number of disks) and cluster sizes the delay bounds that can be achieved with and without a failed disk. We note that we can guarantee less than 10 times the disk service time as a delay bound, even with a failed disk, if the cluster size is at least 8. The delay bound that can be achieved is sensitive to cluster sizes below 8.

A "sliding window" router

One possibility is to have the storage nodes periodically send update information about their state to the routers. This raises questions of latency and overhead that are problematic. We decided to explore

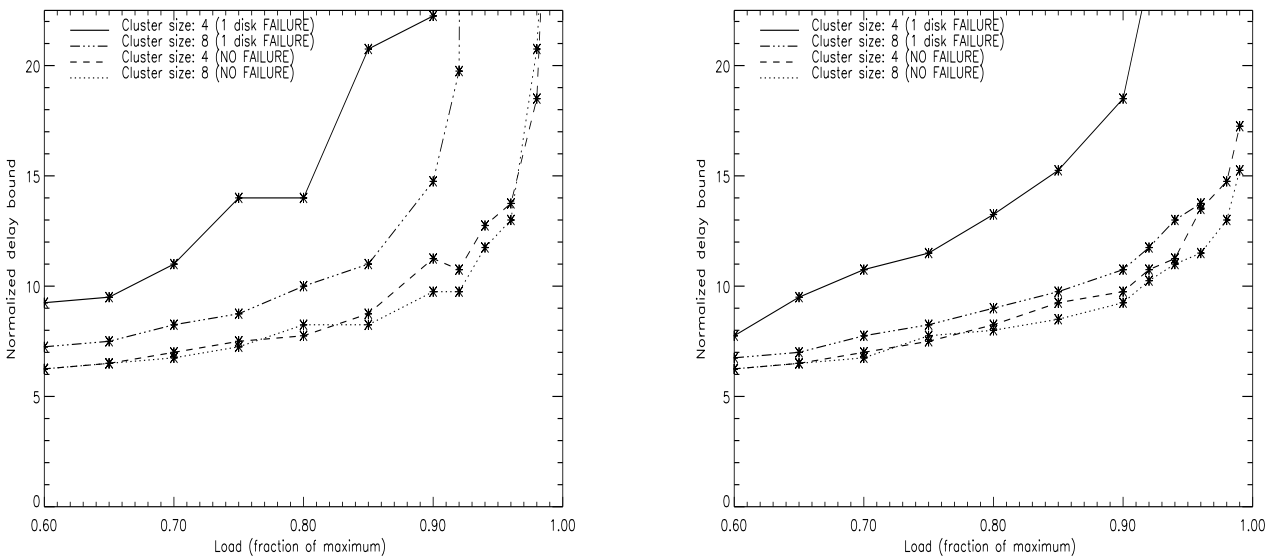

Figure 8: Performance of 16 disk system (left) and 64 disk system (right) with two level routing and different size clusters.

a different approach. Basically, if the number of routers is small enough such that each handles a not insignificant percentage of the aggregate load, then it may be effective to have each router just balance the load that it imposes on the storage serves, i.e., not worry about the load that other routers place on storage nodes. In this case, perhaps a router could just keep track of the requests it has recently sent to the various storage nodes, use this as an estimate of their current load, and route to the least loaded based on this estimate.

We have done some extensive simulations of a scheduler based on this idea. A router, in this scheme, keeps track of the number of requests that it has sent to each node during the last w seconds (where w is the window size) and uses this count as the current load on that node. Our results indicate that a relatively "short memory" (i.e., small w) at the router works best. Figure 9 shows a sampling of these results. The lower curve in each plot is for random replication and perfect knowledge of the state of the disk queues; this is included for reference. From these figures it appears that the sliding window scheduler works almost as well as having precise knowledge of the aggregate load at storage node up to about 85 or 90 % utilization.

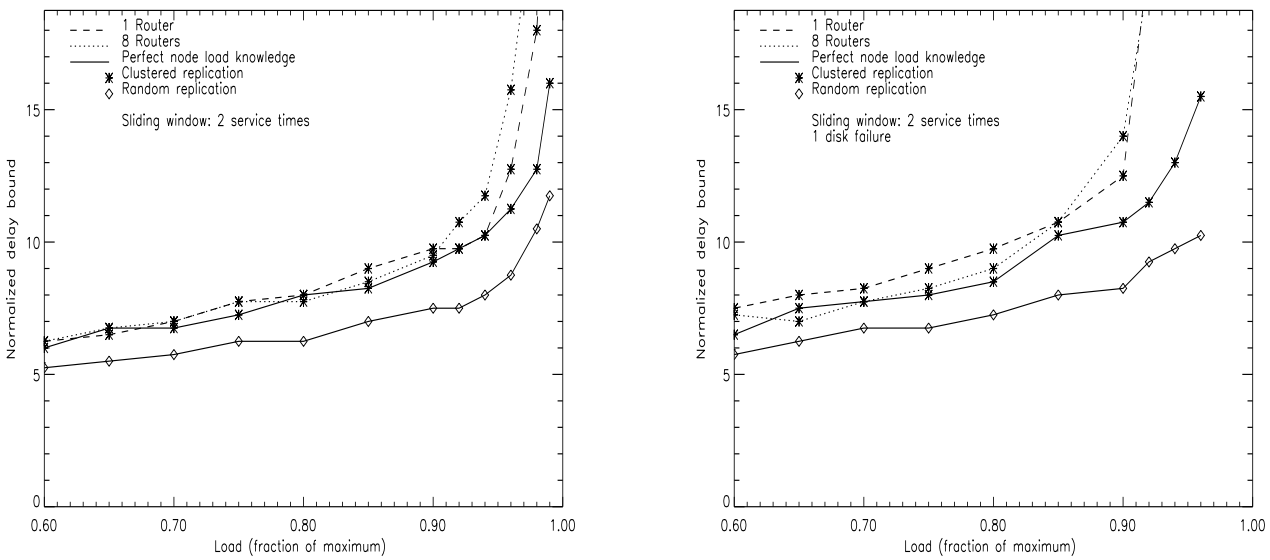

Figure 9: Sliding window scheduler performance.

3 Conclusion

We have completed a rather extensive simulation study of various data layout and scheduling strategies for a clustered, realtime object server. Replication is advocated for both fault tolerance and to achieve a relatively high utilization in combination with a relatively low delay bound guarantee. Several approaches to routing disk requests to storage nodes were studied. The sliding window scheduler was shown to be quite simple and effective and avoided issues of state update information having to be propagated to the routers.

References

[1] Funkhauser, T., Sequin, C., Teller, S., "Management of Large Amounts of Data in Interactive Building Walkthroughs", *ACM SIGGRAPH Proc. of the 1992 Symposium on Interactive 3D Graphics*, 1992.

[2] W. Gekelman, D. Leneman, J. Maggs, "Experimental Observation of Alfven Wave Cones", *Physics of Plasmas*, 1, pp.3775-3783, 1994.

[3] H. Hsiao, D.J. Dewitt, "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines", *Proc. of Data Engineering*, pp 456-65, 1990.

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

[5] W. Karplus, M.R. Harreld, "The Role of Virtual Environments in Clinical Medicine: Scientific Visualization", *Proc. First Jint Conference of International Simulation Societies (CISS)*, Zurich, Switzerland, pp. 13-17, September 1994.

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

[7] R. Muntz, J. C-S. Lui, "Performance Analysis of Disk Arrays Under Failure", *Proc. VLDB*, Brisbane, Australia, pp. 162-173, 1990.

[8] D.A. Patterson, G. Gibson, R.H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *SIGMOD 88*, pp.109-116, 1988.