Cluster-Based File Replication in Large-Scale Distributed Systems

Harjinder S. Sandhu and Songnian Zhou

Computer Systems Research Institute
University of Toronto, Toronto, ON, M5S 1A4
{hsandhu,zhou}@csri.toronto.edu

Abstract

The increasing need for data sharing in large-scale distributed systems may place a heavy burden on critical resources such as file servers and networks. Our examination of the workload in one large commercial engineering environment shows that wide-spread sharing of unstable files among tens to hundreds of users is common. Traditional client-based file cacheing techniques are not scalable in such environments.

We propose Frolic, a scheme for cluster-based file replication in large-scale distributed file systems. A cluster is a group of workstations and one or more file servers on a local area network. Large distributed systems may have tens or hundreds of clusters connected by a backbone network. By dynamically creating and maintaining replicas of shared files on the file servers in the clusters using those files, we effectively reduce reliance on central servers supporting such files, as well as reduce the distances between the accessing sites and data. We propose and study algorithms for the two main issues in Frolic, 1) locating a valid file replica, and 2) maintaining consistency among replicas. Our simulation experiments using a statistical workload model based upon measurement data and real workload characteristics show that cluster-based file replication can significantly reduce file access delays and server and backbone network utilizations in large-scale distributed systems over a wide range of workload conditions. The workload characteristics most critical to replication performance are: the size of shared files, the number of clusters that modify a file, and the number of consecutive accesses to files from a particular cluster.

1 Introduction

Efficient mechanisms for data sharing are essential to the performance of distributed file systems. Past research has focused on data management in local area environments, using cacheing techniques for improving file system performance. The success of distributed systems in local area environments has lead to their growing dominance in large-scale environments encompassing hundreds to thousands of hosts. This suggests a need to focus more research on the issues of efficient data sharing in large-scale distributed environments.

Since a distributed file system is essentially a means of sharing resources, as distributed systems grow larger, the amount of sharing may be expected to increase as well. This places a heavy burden on critical resources such as file servers and interconnection networks. Contemporary file systems, such as Sun's Network File System (NFS) [SGK+85], are unable to respond adequately to scale. Part of the reason for this is their reliance on client-based cacheing and synchronization mechanisms in which the client must contact the file server at file open time to check the validity of its cached data. Such techniques generate unnecessary network traffic and server load in circumstances where typically a file has not been updated since the last open by the client. Their scalability is thus limited.

One approach to scalable file systems is the Andrew File System (AFS) [HKM+88], intended to provide a unified computing environment for thousands of users. AFS replaces the client-based cache validation scheme with a call-back mechanism. Using call-back, it is the server's responsibility to notify the clients whenever a file changes, thus potentially eliminating unnecessary communication between the client and file server. Additionally, AFS performs whole file cacheing on client disks, and provides a mechanism for read-only replication. Since the file cache on local disk is typically larger than that in main memory, higher file access hit ratios may be expected. Within the academic environment in which AFS was developed, the primary need is to provide transparent and efficient file access throughout the system, as the users often move from location to location. The AFS approach may provide scalability to hundreds or more workstations in this type of environment.

A problem with AFS and other similar file systems is that they were designed almost exclusively for academic environments. Indeed, many of the past workload studies, upon which most of the existing distributed file systems are based, have concentrated on academic environments [OCH+85][BHK+91], and the applicability of these studies to other environments is questionable. In particular, it was found that read-only sharing of system files is common, but sharing of unstable user files is not, since in these environments users typically work in small isolated projects. Many commercial environments, on the other hand, tend to be oriented towards large projects, with a number of users working on a single project and actively sharing files. The ability to share data widely in these environments may be limited by the ability of file systems to respond adequately to scale under such circumstances.

We conducted a workload study in a large engineering computing environment comprising thousands of workstations for product development [SZ92]. Most of the computing facilities in this environment are distributed across a number of sites in one metropolitan area. Project-oriented workgroups are supported by clusters of 10-50 workstations and one or more file servers, and multiple clusters are joined by a variety of backbone networks and serial links. In our study, we concentrated on examining file system and network characteristics. Some of the findings of this study show that the problems we had expected do indeed exist, but that they are far beyond our expectations. For instance:

1. There is wide-spread sharing of user files, on the order of hundreds of users situated at workstations across multiple clusters. Such files include but are not limited to source libraries, applications, simulation input/output, and executables which are unstable (i.e., they may change from day to day or many times a day in some cases).

2. Many very large files (up to several megabytes in size) are also being shared widely, although a majority of files are on the order of 10 Kbytes in size.

3. Data is geographically distributed. Users commonly access data that exists in different sites around the metropolitan area.

4. The amount of data sharing causes excessive contention and delays on some file servers and interconnection networks.

5. Although sharing is wide spread, write sharing across multiple clusters is rare. Typically, one cluster is the producer of data, while the other clusters accessing that data act as consumers.

These observations would appear to be representative of an important type of environment: large distributed environments with a need for extensive wide-spread sharing. The need for wide-spread sharing arises from the fact that modern engineering and production environments often require large teams of people at different locations to cooperate closely on the same project.

Under conditions of wide-spread sharing of large unstable files, the traditional strategy of client cacheing from central file server sites (as shown in Figure 1) is inadequate for two reasons. First, even a low cache miss rate could be devastating to performance when tens or hundreds of clients are accessing and potentially modifying the same set of files. Not only must the server for these files handle many file access requests, but the client-server interaction required for coherence may become a serious factor in file server response time. Server congestion may be reduced by placing different actively shared files on different servers, but such load balancing is likely to be limited since it can only be done on groups of large files and file access patterns typically change over time.

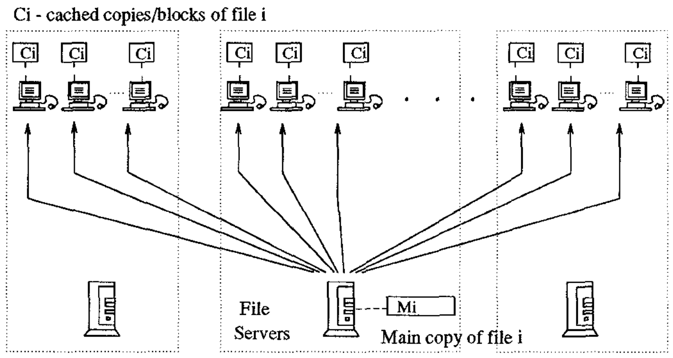

Figure 1: Traditional client cacheing from central servers.

The second reason for the inadequacy of traditional client-based cacheing is that, in a large-scale distributed environment, the distance between the requesting client site and the file server may cause unreasonable latency, since a data access may require traversing a complex interconnection network and large geographical distances. When a file is modified, the file server must send invalidation messages to possibly hundreds of clients over these same potentially large distances. Even if cache misses are infrequent, thousands of clients performing data access from file servers outside of their cluster result in unmanageable loads on interconnection networks. This is something we observed in the our workload study environment. Obviously, a better approach to scalable distributed file systems is needed for such environments.

In this paper, we propose the use of *dynamic cluster-based file replication* techniques to address these problems. In Section 3, we outline the important issues in cluster-based file replication, and propose some solutions. Section 4 presents a general workload model for studying the performance of file replication. Section 5 presents the results of simulation experiments that examine the performance of cluster-based file replication. We identify the critical parameters that impact the cost of replication and examine the conditions under which replication will perform well. Lastly, we compare the performance of the various replication strategies outlined in Section 3.

2 The Frolic Approach

Large-scale distributed systems tend to have an inherently clustered physical organization, as shown in Figure 2. The engineering computing environment discussed in Section 1 is a typical example. File systems designed for scalability (AFS, for example) also assume such a system architecture. Clustering is natural in such systems because it effectively overcomes the length and bandwidth limitations of LANs, simplifies administration and maintenance, and exploits locality in resource sharing. The scalability of such systems can be defined as the ability of the file system to support tens (and eventually hundreds) of clusters efficiently.

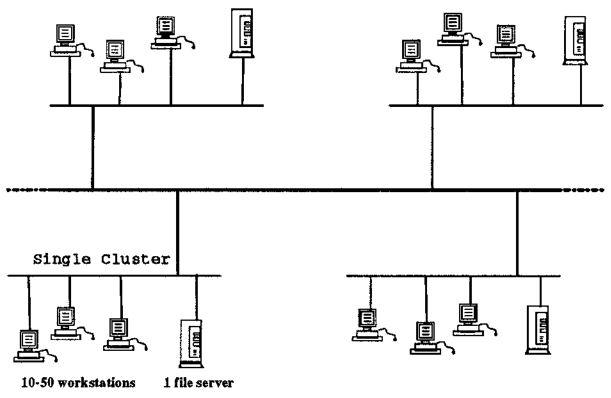

Figure 2: A clustered large-scale distributed system.

In the previous section, we outlined two major problems in large-scale distributed systems, namely, network and file server congestion, and high access latency. The source of these problems appear to be twofold: 1) for a shared file, the many-to-one relationship between potentially hundreds of clients and a single file server which must synchronize access to that file, and 2) the distance between the accessing site and data.

Our approach to scalability and performance in large distributed file systems, called *Frolic* (for File Replication Over Large Interconnected Clusters), involves dynamically creating and maintaining replicas of files on the *file servers* within the clusters that access those files. The general model is shown in Figure 3. It should be noted that Frolic is only a software technique; the physical hardware configuration of the system is the same as before. By replicating across clusters, we have effectively changed the n to 1 client-server relationship to an n to k to 1 client-server relationship (assuming there are n clients in k clusters accessing the file). This alleviates the problem of a central server being a potential bottleneck. By maintaining replicas within clusters where they are accessed, we also reduce access latency by insuring that most file access requests may be satisfied locally, although in some cases the local server will need to obtain a copy of the file before replying to the client.

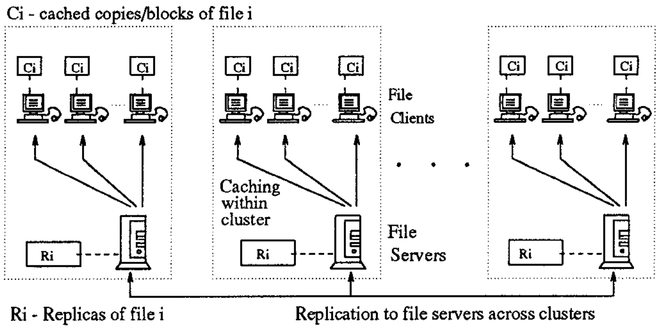
Ci - cached copies/blocks of file i
Ci
File Clients
Caching within cluster
File Servers
Ri
Ri - Replicas of file i
Replication to file servers across clusters

Figure 3: Dynamic cluster-based file replication.

In Frolic, replication is performed on the file servers and is transparent to client workstations. There are two levels of operation in Frolic: intra-cluster and inter-cluster. The protocol for client-server interaction within the cluster does not need to be modified, so any strategy for data management, cacheing, and cache validation may be used within the cluster. The file servers, however, must maintain consistency of the replicas among themselves. Such a clear hierarchical separation allows the Frolic approach to be applicable to any existing distributed file system. The next section examines replication strategies and the synchronization of replicas among clusters.

3 Algorithms for Replication

Our model of the system consists of file servers, one per cluster for simplicity, interacting with each other through some arbitrary interconnection network. Replicas are created and maintained on the servers on demand, that is, accesses to a file from another cluster causes a replica of that file to be created in the accessing cluster. The details of intra-cluster data management are ignored since they do not affect the behavior or performance of the replication algorithms. The impact of network failures and data loss is not discussed in this paper, since our primary focus is that of performance. Issues of availability and fault tolerance in relation to cluster-based file replication may be dealt with in some later work.

The workload in this model comprises file accesses to shared files, and a background workload representing the aggregate of all other activity in the system. *Shared files* are those that are accessed from more than one cluster during their lifetimes. A *file access* is a complete open-close session, consisting of a file open, zero or more read/write operations, and a file close operation. A file is considered to be modified if one or more write operations took place in the last access (i.e., open-close session) of that file.

In the algorithms we describe here, a file can only be modified by a single cluster at a time, this cluster being denoted the *owner* of the file. Another cluster wishing to modify the file must first acquire ownership of the file. Although this may appear to be overly restrictive, this rule can easily be relaxed without effecting performance significantly, since we expect concurrent writing across multiple clusters to be very rare. Enforcing this restriction, on the other hand, affords us the luxury of being able to avoid the concurrent writers problem completely. The only issue is that of ensuring that all replicas are kept up-to-date. Notice as well that this restriction is only enforced on replicas at the cluster level. If cluster *A* is currently the owner of a file, any number of users within that cluster may modify the file, using any type of consistency mechanism, but no users in any other cluster may do so until their cluster has acquired ownership. The owner must keep track of all the clusters that have valid replicas of a file. When ownership changes, this information is passed on to the new owner of a file.

Synchronization of replicas is performed at *file close time*. Our choice of file close consistency for synchronization of replicas is motivated by the goal of performance. The performance penalty of synchronizing replicas after every write operation will be high, and the benefits comparatively negligible given that most applications do not require such a high degree of consistency. AFS also uses file close consistency semantics, but again it must be emphasized that, in our case, the consistency semantics are applied only at the cluster level.

3.1 Issues in Replication

There are essentially two issues in file replication: 1) locating replicas, and 2) maintaining consistency among replicas. Locating (i.e., finding) a replica is an issue because the file no longer resides at a single fixed place in the distributed system, since replicas may be created and destroyed over time. A file server that does not have a replica of a particular file, or has an out-of-date copy of the file, must have some way of locating a valid replica of that file in order to obtain fresh data. The second issue, that of consistency, is to ensure that all replica sites are made aware of modifications to a file after that file is closed. In this section, we shall discuss some alternatives for dealing with both of these issues. Section 5.4 presents some simulation experiments that compare the effectiveness of these techniques.

3.2 Locating Techniques

There are a number of possible techniques for dealing with the issue of locating replicas, many of which have been applied to Distributed Shared Memory algorithms and are discussed by Li [Li89], and Stumm and Zhou [SZ90]. A analysis (more detailed than the one presented here) of their application to cluster-based file replication can be found in [San91].

- *Central Server.* A single file server (the Central Server) is made the *master* for a particular file or file subsystem. The Central Server is responsible for maintaining the most recent version of the file, and for resolving conflicts and enforcing consistency. The drawbacks of this method are: a) the Central Server may become a bottleneck, and b) the cost of sending all modifications of a file to the Central Server may be substantial.

- *Distributed Management with Multicast.* We can simplify the problem of locating a valid replica of a file to that of locating the cluster that is the owner of that file, since the owner is the only cluster guaranteed to have an up-to-date version. Since ownership may change over time, the simplest solution would be to multicast ownership change information to all concerned file servers. This technique has the disadvantage that multicast is expensive over many geographically distributed sites.

- *Distributed Management with Dynamic Location.* In order to reduce overhead, we want to avoid informing all replica sites every time a change in ownership occurs. A site wishing to locate the owner of a file must then contact the site it thinks is the owner, and that site in turn forwards the message to the site it thinks is the owner, and so on until the current owner is located. In the worst case, this forwarding technique requires searching through $N-1$ sites for an owner if there are N replica sites.

- *Locating Server.* This technique is a hybrid between centralized and distributed management techniques.

The Locating Server is a server whose sole responsibility is to keep track of the owner of a file. For each ownership change, the Locating Server is notified, and subsequent locating requests are directed to the Locating Server. Since the Locating Server handles only trivial requests, it is less likely to become a bottleneck than the Central Server described earlier. Additionally, the task of the Locating Server may be distributed throughout the system for different subdomains of the file space.

3.3 Consistency Schemes

There are two basic techniques for the synchronization of a replicated set of data: immediate update and invalidation. With *immediate update*, modification of a file causes all replicas to be updated as soon as the file is closed. The drawback with this technique is the potential number of wasted updates. That is, replicas may be updated many times before they are actually used by another site and the cost of sending updates after every modification may be prohibitive. With *invalidation*, on the other hand, a modification to a file causes all other replicas to be invalidated at file close time. Updates are then requested on demand (i.e., at file open time) by a site with an invalid replica, thus eliminating the problem of wasted updates. However, invalidation also has drawbacks. First, requiring replica sites to acquire updates on demand leads to higher file access latency, whereas, with immediate update, updates can be performed asynchronously. Second, although we are not explicitly concerned with fault tolerance, it is obvious that the chances of losing data are higher with invalidation, since, using this technique, it is possible that only a single replica of the file exists at a time.

A third technique, which we call *partial update*, represents a compromise between invalidation and immediate update. Using partial update, a site other than the current owner is sent updates to the file, but all other sites are invalidated. If the alternate site is selected to be one that frequently accesses the file, then this may reduce average latency while incurring relatively low overhead cost. Moreover, it has the additional advantage that the chance of data loss is greatly reduced (compared to invalidation) since now at least two sites will always have valid replicas.

4 Workload Model

In this section, we describe the workload model which will be used as a basis for evaluating the benefits of Frolic and the different replication algorithms. File servers, file server disks, and networks constitute the only resources in our system model. They are modeled as single server queueing centres. Workstations are not modeled explicitly, but are represented only by the workload they generate on the rest of the system. The effects of workstation cacheing are taken into account by reducing the effective workload induced by workstations to the rest of the system.

The workload model consists of file operations to shared files and a background workload. Recall that, by our definition, *shared files are those that are accessed from more than one cluster during their lifetime.* The load generated by all operations not directly related to accesses to shared files is represented as a background workload that simply inflates the utilization of each of the resources. Cacheing at the file server is represented implicitly in the model by incorporating those requests which would result in a file server cache hit into the background workload.

We use a statistical workload model whose parameters are based upon three primary sources: a) measurements and traces performed using a locally developed tracing package called Snooper [ZS90], b) the case study in which we examined the workload of a large-scale commercial distributed environment [SZ92], c) and other published work in the area. In general, we try not to limit ourselves to observations of workload characteristics in our workload study, as they constitute only a single data point and would thus limit the generality of the results. Instead, we define the most important workload parameters and attempt to study replication under a wide range of workload conditions by varying these parameters.

4.1 Service Demands

Our model considers the following types of file operations: open, close, block read/write (BRW), replica update (or just *update*), and server-to-server messages (for invalidation and locating). The resource demands in the model are based on measurements published by Molloy [Mol90] for a network of HP 9000's, as well as measurements from traces conducted at Toronto on a network of DEC Vaxen [ZS90]. For the backbone network, measurements of the service demand were not available, so we estimate a base cost, and then perform simulation for a range of values to determine the sensitivity of the results to this base cost.

Table 4.1 shows a matrix of service demands imposed by each type of operation on each resource. We define a *local operation* as being one which is handled within the cluster in which it originated, and a *remote operation* as being one that transcends two clusters. For open, close, and read/write, the service demand on each resource is the aggregate of both the request and the reply phases of the operation. For replicated files, if a local replica of a file exists, then only local operations are needed. If the local replica has become stale, a local open is preceded by a replica update operation. If the replica is modified during an open-close session, the other replicas are either invalidated (using server-to-server messages) or updated (using a replica update operation), depending on which consistency mechanism is being used.

	local cluster			BBn	remote cluster		
operation	Lan	fs	disk	Net	Lan	fs	disk
Local							
BRW	7	15	27	-	-	-	-
open	1	10	3	-	-	-	-
close	0.1	4	-	-	-	-	-
Remote							
BRW	7	-	-	16	7	15	27
open	1	-	-	1	1	40	3
close	0.1	4	-	0.5	0.1	4	-
replica update	7	15	27	16	7	15	27
server-to-server msg	0.5	4	-	0.5	0.1	4	-

Table 1: Average service demands (in ms) per resource for file operations. Block operations and replica update are in 8 Kbyte units.

As an example of the service demand matrix parameterization, a local block read/write originating on a user workstation and served by the file server within the same cluster has the following service demands: 7 ms on the LAN and 15 ms on the file server (for both the request and the reply), and 27 ms on the file server disk for those requests that result in a file server cache miss. A remote BRW requires the same 7 ms on the local LAN, 16 ms on the backbone network (BBn Net in the table), another 7 ms on the remote LAN, 15 ms on the remote file server, and 27 ms on the remote disk. For simplicity, the partial overlaps in the use of resources for file operations are not simulated. Such overlaps have no effect on resource contention, and we do not expect the *relative* results to be affected by this simplification.

While most of these numbers are based upon actual measurements, the cost of using the backbone network was only estimated. We assume it to be 2.5 times the cost of using a LAN to take into account the longer distances, multiple gateways, and often lower bandwidth of backbone networks. In our experiments, this cost was varied in order to determine the sensitivity of the results to this parameter.

An important design issue in Frolic is whether to perform replica updates on a block basis or on a whole file basis. Performing updates on a block basis implies that the modified blocks of a file can be identified, so that only those blocks need to be sent to update a stale replica. Performing updates on a whole file basis means that the entire file is transferred to a stale replica site. For the sake of generality, we choose to study the whole file update technique. If block updates are feasible, the performance of cluster-based replication would be better. In the service demand matrix, the unit cost of performing a replica update is shown as the average cost of transferring an 8 Kbyte block. The cost of replicating a file would be this value multiplied by the number of 8 Kbyte blocks in this file. In our workload study [SZ92], we found a high variance among the average sizes of different types of shared files: 8 Kbytes for executables, 22 Kbytes for object files, and 68 Kbytes for source libraries. In the model, we choose a default average file size of 24 Kbytes (for simplicity, we choose average file size to be a multiple of the block size). In Section 5.3.1, we examine the impact of choosing whole file transfer for replica update, as well as the effect of varying the average file size.

A few other workload parameters need to be mentioned here. The first is ***workload intensity***, which we characterize by the number of accesses to shared files per cluster per second. Our default value for workload intensity is 2.0 accesses per cluster per second, estimated from the request rate to files in the Snooper trace study and the expected rate of access to shared files. Most of our simulation experiments were conducted for a range of workload intensities ranging from very light, where there is almost no activity on the system, to very heavy.

A second parameter, related to the file size, is the portion of the file accessed in a typical open-close session. This was not examined in our workload study, so for an initial estimate we rely on observations in other published work that most of a file is accessed in a typical open-close session [OCH$^+$85] [Kur88]. Our initial estimate of the portion accessed is an average of 16 Kbytes (2/3's of an average 24 Kbyte file), or 2 BRW's. A multiple of the block size is chosen for simplicity. In the experiments, we examine the effects of varying the portion accessed from a minimum of 8 Kbytes (one block) to the whole file.

The background workload, which includes all activities not directly related to shared files, is characterized by an expected utilization on each of the system resources in the absence of accesses to shared files. For the local area case, the expected utilizations are about 30% on the servers, 12% on the LAN's, and about 80% on the disks. These values correspond roughly to observations of local area environments made by Molloy [Mol90] as well as in our workload study [SZ92]. For the backbone network, we estimated the background load utilization to be around 5%. This agrees with observations in our workload study in which the background networks had an average utilization of 20-30% and 60-80% of this was due to file access traffic.

4.2 File Reference Patterns

The file reference characteristics of an environment essentially determine the effectiveness of any data management scheme. For cluster-based replication, some of the most important file reference parameters are:

1. *degree of sharing*: the number of *clusters* that access a file over its lifetime.

2. *number of writers*: the number of *clusters* sharing a file that will also modify it during its lifetime.

3. *cluster locality*: the number of consecutive accesses to a file from a particular cluster before any other cluster requests it.

For each of these characteristics, we derive a general model for representing them and also arrive at some default parameters, shown in Table 2. In our simulation experiments, we varied each of the parameters and attempted to determine the sensitivity of the results to these parameters.

The sharing patterns among clusters were examined in our workload study, and it was found that while there were a large number of files being shared, most files were only accessed from a few clusters during the one week examination period. The degree of sharing was found to be hyper-geometrically distributed, with a large number of files being shared among two clusters, fewer files being shared among three clusters, fewer still being shared among four clusters, and so on. In our workload study, the shared files we classified are mostly user files such as source libraries, applications, and executables. Although system files, such as those in '/bin', are widely used, they are of less interest since they are read-only, so that once replicated they will not cause any further transfer or consistency overhead. We model the degree of sharing using a hyper-geometric distribution whose mean and variance are parameters in our workload model. The default mean is set to 4 (which is higher than the average degree of sharing for a set of source files in our workload study).

Past workload studies in local area environments have noted the read-mostly aspect of files [OCH+85] [BHK+91] [BR90] [Kur88]. This observation appears to hold true for larger environments as well, with those files that are written to being modified usually by only one cluster during its lifetime. In our workload study, we observed that for files which are modified, they are usually modified by only a single cluster/writer (recall that if a cluster is a writer, any number of users within that cluster may modify the file). Few files have two writers, and fewer still have three. The number of writers per file is also modeled as a hyper-geometric distribution (hyper-geometric being chosen so that we could vary both the mean and the variance of the distribution and study their impacts). In our model, since we are interested primarily in unstable shared files, all files have at least one writer. The default mean for the number of writers distribution is 2, which is higher than what we observed in our workload study. We chose this mean to be higher than expected in order that we do not create an inherent bias towards replication. Experiments in Section 5.3.2 examine the effect of different numbers of writers per file.

For cluster locality, we set the default value to 4 accesses per cluster and later varied this value in the experiments. No measurements of cluster locality were available, so this default value is only an estimate. Section 5.3.2 examines the impact of cluster locality on file replication.

Parameter	Default Value
Number of clusters	10
Workload intensity	2 opens/sec/cluster
Average File Size Average Portion Accessed	24 Kbytes 16 Kbytes
Average degree of sharing	4 clusters
Average number of writers	2 writers per file
Average degree of locality	4 accesses per cluster

Table 2: Default workload parameters.

Table 2 shows a summary of the default parameters. Each of these parameters was varied in our experiments. However, while one parameter was being varied, all others were set to the default values shown in this table. The duration of time simulated for the experiments is about 6 hours. The workload is uniform over this time period (i.e. burstiness in the workload was not modeled).

5 Simulation and Results

In this section, we present the results of a series of simulation experiments studying the performance of Frolic, and comparing the different replication algorithms. The simulator is implemented in Csim [Sch88], a C-based process-oriented simulation tool, and uses the workload model described in the previous section. We first compare the performance of Frolic to that of a non-replicated file system using the default parameters shown in Table 2. We then present some experiments designed to determine the behavior of file replication over a range of workload conditions. While a number of techniques for locating and consistency have been discussed, all of the experiments use the Locating Server technique coupled with invalidation for consistency. Section 5.4 presents a comparison of the performance of the different replica management and consistency schemes presented in Sections 3.2 and 3.3.

The primary measure of performance is the *average file access time*, which is the sum of the durations of all file operations in an open-close session, including the open request, a number of read/write operations, and the close request. This is equivalent to the real-time duration of an open-close session with the time between operations removed. When replication algorithms are used, access time also includes the time required for a possible *update* (file transfer) when a new replica is required at the local cluster or a replica has become stale. Synchronization at file close time (i.e., invalidation of replicas) does not directly contribute to the access time, since it is performed asynchronously with respect to file access. However, an increase in the number or cost of synchronization operations will increase network and server utilization, and that will in turn affect the average access time. The benefit of using file access time as a response time measure is that factors which do not directly contribute to the relative performance of file replication are excluded.

The no replication case used for comparison in these experiments is based upon a simplified model of traditional client cacheing from a central server, in which file access requests to access blocks of a file arrive at the server for that file from the workstations. The hardware configuration of the system is assumed to be the same, and files are distributed across the system such that a file may exist with equal probability on any one of the clusters that share that file. The effects of cacheing at the workstations is taken into account implicitly by adjusting the workload intensity according to the expected workstation cache hit ratio.

5.1 The Benefits of Replication

In the first set of experiments, the simulation was run with and without replication for varying workload intensities. The workload intensity is controlled by the arrival rate of access requests per cluster to shared files. The values of all other parameters are the defaults described in Table 2 in the previous section.

Figure 4 shows a comparison of the file replication and no replication case for different workload intensities. 90% confidence intervals are shown as vertical bars in the graph. We also show, for the sake of comparison, the unrealizable *pure-local* case. Pure-local is when all accesses are satisfied locally (within the cluster), and no inter-cluster interaction is required. It is in a sense a lower bound on access time for any cluster-based data management strategy.

Given our workload assumptions, file replication appears to improve file access times for any workload intensity. At low workload intensity, the increase in the percentage of file accesses that are local due to replication causes a significant reduction in average file access times. This is shown as well by Table 3, which shows the percentage of local and remote file opens for files shared among clusters, as well as the percentage of file opens that require update and synchronization operations before and after replication of these files.

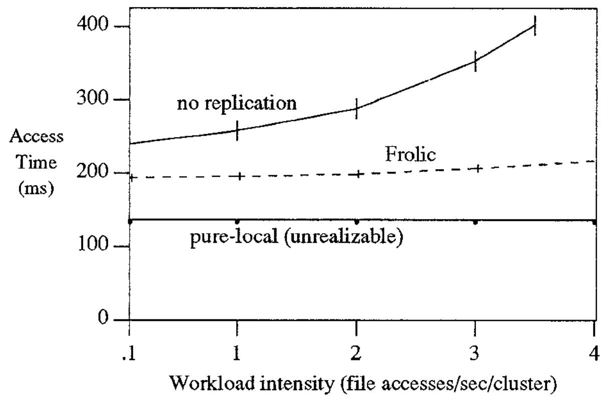

Figure 4: Performance of file replication for varying workload intensities.

Algorithm	Local Access	Remote Access	Update and local Access	Synch-ronize
no replicate	23%	77%	-	-
replication	84%	3.0%	13.%	3.0%

Table 3: Percentages of *shared* file accesses that are local, remote, or requiring Update or Synchronize.

Without replication, the percentage of local opens is low. Given that our mean degree of sharing is 4, and assuming that each of the clusters accesses the file with the same frequency, we would expect about 1/4 of the opens to be local, since only one of the clusters actually has a copy of the file locally. With replication, the percentage of local opens increases dramatically. The corresponding overhead due to consistency requirements is relatively low; only 13% of opens in our simulation require updates at file open time. The synchronization traffic (of performing invalidation at file close time) is even lower; only 3% of opens required synchronization operations. The 3% percent remote accesses for the replication case is due to the fact that replication is triggered only after a few initial requests for remote files.

As the workload intensity increases, there is a significant increase in average file access time without replication, but the replication case is not affected much at all. Again, the reason for this is the localization of file access traffic, and the low synchronization overhead. In

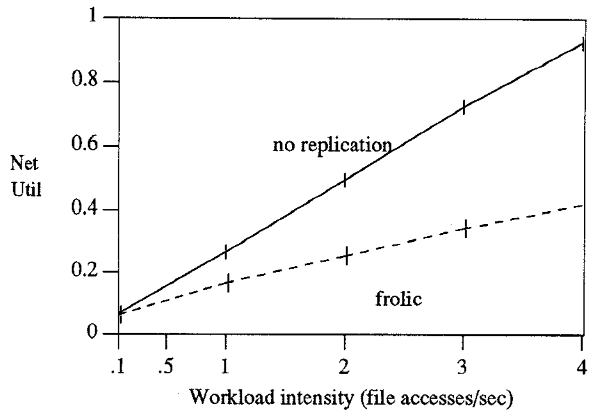

Figure 5: Backbone network utilization for varying workload intensities.

the no replication case, the increase in workload intensity causes dramatic increase in backbone network utilization (Figure 5). The increased backbone network utilization in turn induces greater file access latency. Using replication, there is a smaller increase in backbone network traffic.

Reduction in backbone network load, however, is not the primary cause of performance improvements under moderate load. This is illustrated in Table 4 which shows file access times with different backbone network costs (the default cost was 16 ms). This table indicates that even when network delays and congestion are negligible (e.g., a service demand of 1.6 ms), replication still performs better than the no replication case. This performance gain can only be attributed to the implicit load balancing of file servers with replication, and the resulting reduction in file server congestion.

Algorithm	Backbone Network Cost 1.6 ms	8 ms	16 ms	32 ms
no replication	232	250	288	909
replication	188	192	202	238
% improvement	19%	23%	30%	74%

Table 4: File access times for varying backbone network costs.

5.2 The Effects of Scale

It is apparent that, based upon our default workload parameters, using replication improves performance significantly for any workload intensity. In fact, without replication, the system becomes unusable under high workload intensity. This is shown in our next experiment as well (Figure 6). Here we study the effect of system scale on file replication, by increasing the number of clusters from to 30 (from the default of 10).

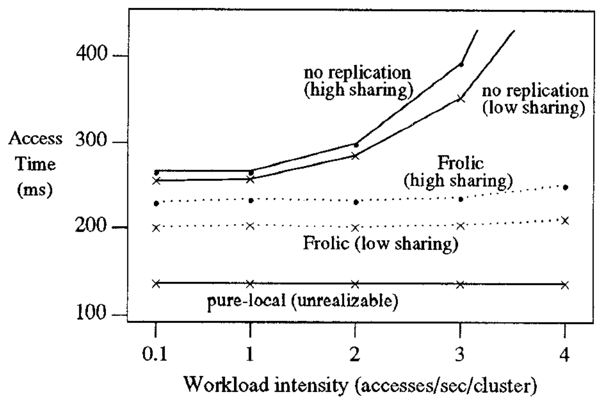

Figure 6: Performance for a larger system (30 clusters).

The effect of increasing the number of clusters in the system is similar to that of increasing the workload intensity. Since it is unclear how the degree of sharing is correlated to system scale, we consider two bounding cases. The case in which the average degree of sharing increases by the same factor as system scale (high sharing) is shown along with the case in which the average degree of sharing remains constant as scale increases (low sharing). Comparing Figure 6 with the 10 cluster system shown in Figure 4, it becomes evident that the overhead for file replication is not affected severely by an increase in scale. On the other hand, without replication, the system becomes unusable for workload intensities higher than 3 accesses per cluster per second. These observations were apparent in our workload study environment as well, where the scale of the system and amount of file sharing contributes to very high peak utilizations on backbone networks and file servers.

5.3 Analysis of Workload Sensitivity

5.3.1 File Size and the Portion Accessed

In the next set of experiments, we consider file replication under a range of workload conditions. We first examine file size and the portion of the file accessed in a typical open-close session. File size is critical to replication performance in that it directly impacts the cost of updating stale replicas, due to our assumption that replica updates are performed on a whole file basis. The portion of a file accessed is also important because replication will only be beneficial if sufficient data in a file is used in a typical access.

Figure 7 shows the case in which we vary the average portion of a file accessed in a typical open-close session. We compare the performance of the no replication case with that of Frolic for several different average file sizes. Note that the performance of the no replication case depends only on the number of blocks of a file accessed, and not on the size of the file.

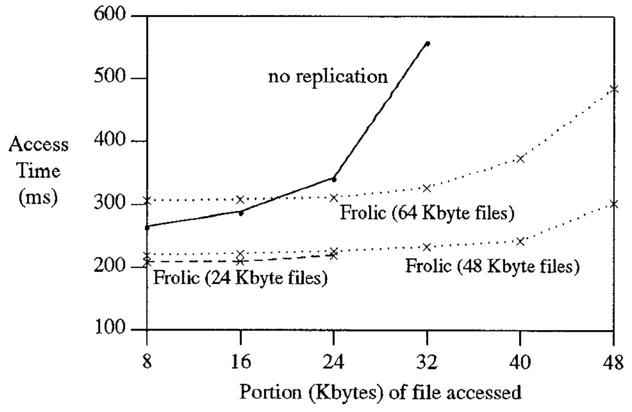

Figure 7: Varying portion accessed and average file size.

The results in Figure 7 indicate that replication performs significantly better than the no replication case for most combinations of the file size and portion of the file accessed. In the no replication case, increasing the average amount of data accessed from all files quickly saturates the backbone network, and results in poor performance. With replication however, access times are much better for files of up to 48 Kbyte's in size. For large average file sizes (e.g., 64 Kbytes), replication performs worse when very little of the file is used in a typical access. However, for 64 Kbyte files, if more than 24 Kbytes of the file is used in a typical access, replication is significantly better than the no replication case.

If we view this same graph (Figure 7) from the point of view of increasing file size for a fixed portion accessed, it is evident that the performance of file replication degrades as average file size increases. This serves to illustrate an important point: that whole file transfer on update may lead to poor performance for large files. The alternative is to identify the modified blocks of the file and transfer only those to the stale replica sites. We noted earlier that cluster-based replication would probably perform better if block-based updates were used, but we chose to model whole file updates because this was more general. The additional complexity (in file system design) of block-based updates may be warranted if there is a significant number of large files that are shared.

It should be noted as well that we do not expect average file sizes to be as high as 64 Kbytes. In our workload study [SZ92], we observed that although there are many files over one Megabyte in size, the mean of the file size distribution is quite low; most files are on the order of 10 to 16 Kbytes. Thus, given these types of workload conditions, using whole file update with replication would work well for most files.

5.3.2 Cluster Locality and Number of Writers

The next experiment shows the effects of cluster locality and the number of writers per file (*nw* in the graphs). The file size and portion accessed are both set to their default values.

Files in the simulation are classified according to their degree of cluster locality, and by the number of clusters which write to that file during the file's lifetimes. The same files are examined with and without replication. Figure 8 shows the result of this experiment. Again, the no replication case is unaffected by both the degree of cluster locality and the number of writers.

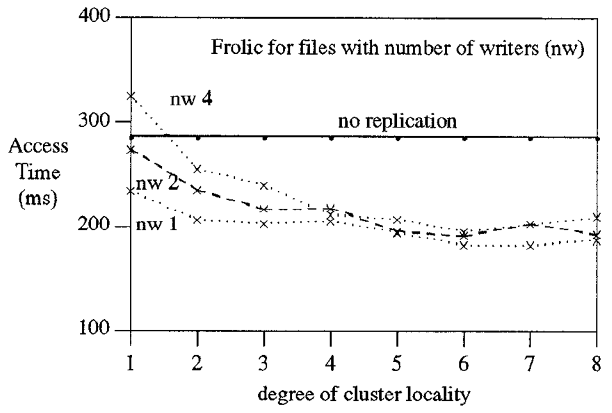

Figure 8: Varying number of writers (*nw*) and degree of locality.

Replication of read-only files is always beneficial, but read-only files were not considered in these simulations. More writers per file means that files are likely to be updated more often, and more synchronization is required. A decrease in the degree of locality (and thus an increase in the synchronization and update traffic) has a similar effect; more synchronization will occur when locality is lower. However, Figure 8 shows that, unless average cluster locality is very low (less than two) and the number of writers is higher than four, replication results in a significant reduction in file access times. This illustrates the robustness of the replication techniques for varying degrees of locality and numbers of writers per file.

The next experiment shows the influence of cluster locality and file size upon file access time. The average portion of a file accessed is set back to its default value of 16 Kbytes. The results are shown in Figure 9.

The results follow intuitively from our earlier experiments. Smaller files (24 Kbytes or less) benefit from replication even for low cluster locality. The performance for larger files is dependent on having sufficient cluster locality in file access. For a 64 Kbyte file, for instance, repli-

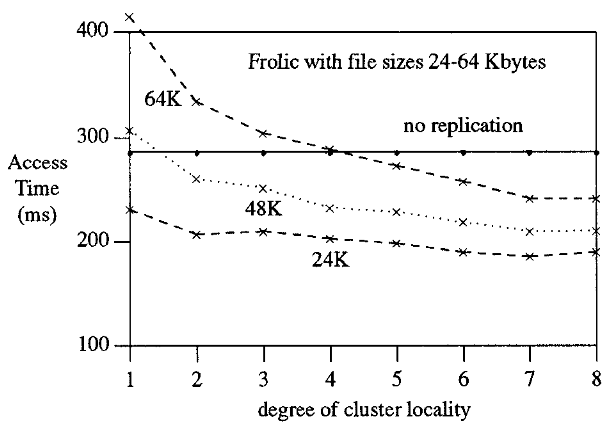

Figure 9: Varying file sizes and cluster locality.

cation becomes beneficial if the cluster locality is higher than about four, given that only about 16 Kbytes of that file are actually touched during a typical file access.

In our workload study, we found that the single writer mode of sharing was most common (recall observation 5 in Section 1), and that most files were generally small. Thus, based upon our experiments, we conclude that unless locality is poor or the number of writers high, cluster-based file replication will improve performance significantly.

5.4 A Comparison of Consistency and Locating Algorithms

Now we examine the trade-offs between using invalidation, immediate update, or partial update for consistency. These options were discussed in Section 3.3. A comparison of these algorithms is shown in Figure 10.

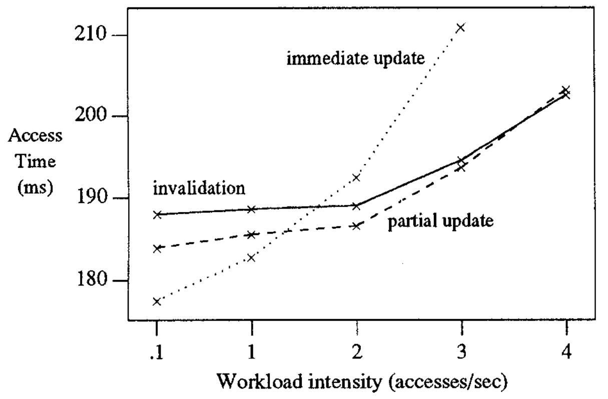

Figure 10: A comparison of the Consistency techniques.

The results are again intuitive. For low workload intensity, immediate update is better since the benefit of always having valid local replicas at file open time outweighs the additional overhead of wasted updates. For moderate to heavy workloads, however, the opposite is true. The demand on network bandwidth causes a significant increase in file access times when immediate update is used, so that invalidation appears more desirable. Partial update falls in between the two extremes.

The only surprising aspect of this graph is that the difference in performance between invalidation and immediate update is not more than 5 to 15 percent for most workload intensities. The reason for this is that the degree of sharing between clusters is generally low (only about four clusters typically share a file), so that the difference in performance between these two techniques is low as well. For more wide-spread sharing among clusters, we would see a larger difference.

The last set of experiments compare the performance of the different replica management techniques that were discussed in Section 3.2. The results were again somewhat surprising in that the difference in the performance of the algorithms is almost indistinguishable (and therefore not shown). There are several reasons for this. First, as mentioned earlier, the degree of sharing is generally low, with only a few clusters sharing a file. Second, most files have very few clusters that modify them during their lifetimes, so that file ownership is fairly stable. When there is only a single writer per file, for instance, all of the different replica management techniques behave similarly. Lastly, the predominant cost in replication is the cost of updating stale replicas. Locating the owner is relatively inexpensive since it requires sending only small messages and thus places small service demands on resources.

6 Conclusions

In this paper, we proposed Frolic, a scheme for cluster-based file replication in large-scale distributed systems. The motivation for this work was based upon the observation that traditional approaches to data management in large-scale environments will perform poorly under conditions of wide-spread sharing of unstable files. In our workload study [SZ92], we found that in the environment that we studied, wide-spread sharing of unstable files among tens or hundreds of users situated across multiple clusters was common. This may be typical of many other large engineering computing environments as well.

Using simulation techniques and a statistical workload model, we showed that cluster-based file replication can significantly improve performance in such environments. The default workload parameters were extracted from measurements and characterization of real workload, but the experiments show that replication is beneficial under a wide range of workload conditions. File replication has several benefits: 1) it reduces reliance on critical resources such as backbone networks and central file servers; 2) it reduces the distance between the accessing site and the data; and 3) it balances the load on file servers. Our experiments show that reduction of file server load and access latency is the primary benefit of file replication under light to moderate load, whereas reduction in network congestion is the primary reason for greater benefits under heavy load.

We identified several aspects of the workload most critical to file replication performance: the file size and the portion of the file accessed, the number of writers per file, and the cluster locality of replicated data. In general, file replication will perform well when enough of the file is accessed in a typical open-close session, the number of writers per file is small, and the cluster locality of replicas is sufficiently high.

Lastly, we examined a number of different algorithms for dealing with the issue of locating replicas and maintaining replica consistency. Our results did not show any significant performance advantage to using one locating technique over another, mainly because the degree of sharing among clusters was generally low and the predominant mode of access was a single cluster modifying the file and many clusters reading it. For consistency schemes, we found that invalidation performed better than immediate update under moderate to high workload intensity, due to the elimination of wasted updates.

Frolic is a practical technique for distributed file system design; we plan to do an implementation study of it in a distributed environment. The Frolic approach to scalable distributed file systems has the advantage over other designs (such as AFS) in that: a) the disk space requirements are low, since there is only a single replica per cluster; b) no particular model of data management within the cluster is imposed; hence, Frolic can be implemented on top of an existing distributed file system; and c) the natural clustering inherent to large-scale distributed systems is used.

Acknowledgments

We would like to thank the referees of this paper for their many valuable comments. Harjinder Sandhu was supported in part by an Ontario Graduate Scholarship.

References

[BHK+91] M. Baker, J. Hartman, M. Kupfer, K. Shirriff, and J. Ousterhout. Measurements of a distributed file system. In *Proc. 13th ACM Symposium on Operating System Principles*. ACM, October 1991.

[BR90] P. Biswas and K.K. Ramakrishnan. File access characterization of VAX/VMS environments. In *10th Int. Conf. on Distributed Computing Systems*, November 1990.

[HKM+88] John Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[Kur88] Q. Kure. *Optimization of File Migration in Distributed Systems*. PhD thesis, University of California Berkeley, 1988.

[Li89] K. Li. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems, Vol 7, No 4*, November 1989.

[Mol90] M. Molloy. Validation of MVA models for client/server systems. Hewlett-Packard, 1990.

[OCH+85] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace-driven analysis of the Unix 4.2 BSD file system. In *Proc. 10th ACM Symposium on Operating System Principles*. ACM, December 1985.

[San91] H.S. Sandhu. File replication and performance in large-scale distributed systems. Master's thesis, University of Toronto, January 1991.

[Sch88] H. Schwetman. Using CSIM to model complex systems. In *Proceeding of the 1988 Winter Simulation Conference*, 1988.

[SGK+85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network file system. In *Usenix Conference and Exhibition*, Portland, OR, Summer 1985.

[SZ90] M. Stumm and S. Zhou. Algorithms implementing distributed shared memory. *Computer*, May 1990.

[SZ92] H.S. Sandhu and S. Zhou. A case study of file system workload in a large-scale distributed environment. Technical report, University of Toronto, 1992. In preparation.

[ZS90] S. Zhou and C. Siebenmann. Snooper user guide, January 1990. University of Toronto.