Implementation and Performance of Cluster–Based File Replication in Large-Scale Distributed Systems

by

James Y.C. Pang
Deepinder S. Gill
Songnian Zhou

Computere Science Research Institute
University of Toronto
Toronto, Ontario, Canada, M5S 1A1

August 1992

Technical Report

Abstract

Large scale distributed file systems supporting 100s to 1000s of hosts are becoming increasingly important as distributed systems rapidly grow in scale. Our examination of the workload in one large commercial environment shows that widespread sharing of unstable files is also common in such large systems. This increase in scale and sharing places a heavy load on critical resources such as file servers and networks, leading to degradation in file access performance. Traditional client-based file caching techniques are inadequate in such environments.

This paper describes the implementation and initial performance study of Frolic, a cluster-based dynamic file replication system. A cluster is a group of workstations and one or more file servers on a local area network. Instead of keeping copies of a widely shared file at each client workstation, these files are dynamically replicated onto the cluster file servers, so that they become locally available.

We compare the performance of Frolic implemented on top of NFS with native NFS involving two clusters. File sizes and the number of consecutive accesses to a file from a particular cluster are varied in the sequential workload; workstation think times are varied in the concurrent workload. The results show that cluster-based file replication, as implemented in our Frolic prototype, can improve the performance of distributed file systems for a workload with wide-spread sharing of unstable files.

1 Introduction

Large scale distributed file systems supporting 100s to 1000s of hosts are becoming increasingly important as distributed systems rapidly grow in scale. However, scale has a serious impact on the performance of the file system. The increase in system scale may place a heavy load on critical resources such as file servers and networks, leading to degradation in file access performance. This suggests a need to focus on the issues of efficient data sharing in large scale distributed environments.

Several file system designs, such as the Andrew File System (AFS) [3] and the more recent dynamic hierarchical caching scheme [2], have attempted to address the issue of scalability in large scale environments. These designs emphasize client caching so that client–server interactions are reduced. The key features of the AFS implementation are client disk caching, call back mechanism, and read–only replication. The design of this system is oriented towards academic environments where efficient access to files stored on remote file servers is needed, but sharing of unstable files is not common. However, a recent workload study of a large industrial file system found widespread sharing of unstable user files among hosts located in multiple clusters [1]. Client file caching as supported by AFS would be inadequate for such environments since frequent updates to the widely shared files would require invalidating a large number of client copies, causing substantial load on the file servers as well as traffic on the network.

A *scalable file system* has been understood to be one that supports a large number of users/hosts without substantial performance degradation[4]. However, this notion of scalability confines one's view to the size of the system only, without any consideration to the increased sharing in large systems. A *scalable file system* could thus be defined as one that continues to provide good access performance with increasing sharing of unstable files among users in large systems, as well as the growth in the number of users/hosts.

In this paper we describe the implementation and initial performance study of Frolic, a cluster-based dynamic file replication system [1]. This system specifically targets large scale commercial environments with extensive file sharing, although it is expected to also perform well in systems with limited file sharing as well. Instead of keeping copies of a widely shared file at each client workstation, these files are dynamically replicated onto the cluster file servers, so that they become locally available. The costs of maintaining consistency among the limited number of server copies should be much less than that among a large number of clients. The problems of network and server congestion as well as that of high access latency may be alleviated. Simulation studies using a statistical workload have shown that cluster–based file replication can significantly reduce file access times for a wide range of workload parameters [1].

Past research has concentrated mostly on using replication for fault tolerance and availability. In the Frolic system, we show that dynamic replication of unstable files can be effectively used for improving file access times in large-scale distributed file system environments.

Section 2 describes the Frolic architecture and our prototype implementation. Preliminary performance results are reported in Section 3. Conclusions and future plans are discussed in

Section 4.

2 Frolic Architecture and Implementation

2.1 Architecture

Figure 1 shows the basic architecture of a typical distributed file system such as AFS or NFS[5]. Files on each file server are directly and transparently accessible from every host. In contrast to these file systems, in the Frolic architecture, shown in Figure 2, clients access *all* files from the *local* cluster file server[1], although the file server may have to obtain a remote copy from another file server in order to satisfy a client's request. When a file is updated, its replica in other clusters are either updated or invalidated, depending on the consistency protocol.

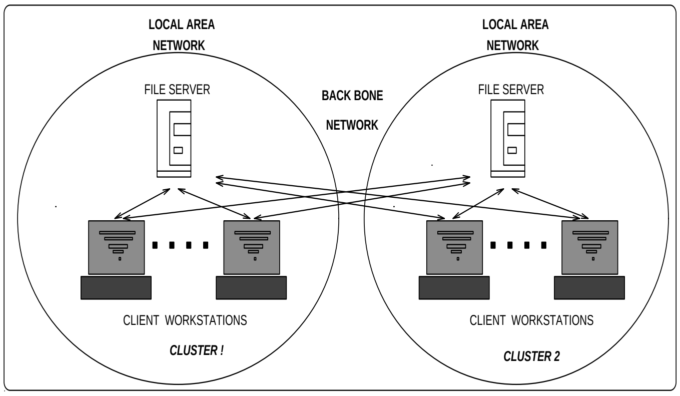

Figure 1: Basic architecture of a typical distributed file system (e.g. NFS, AFS)

In the Frolic system, there are two levels of file system interactions: intra-cluster and inter-cluster. Intra-cluster file operations comprise file access requests from client workstations to the local cluster file server. Inter-cluster file operations comprise mechanisms for replicating and maintaining consistency among the cluster file servers.

2.2 Prototype

The design of the prototype is motivated by the need for simplicity and portability so that we can easily and quickly assess the performance of Frolic in an actual system. Thus, we have decided to implement the prototype without changing the kernel or the underlying file system, although these alternative schemes may offer higher performance.

Figure 3 shows the structure of the prototype, which consists of a user-level library, *cl-frolic*, and the Frolic server processes, *clientfrolic* and *frolicfrolic*. Inter-host communication is done using the Sun-OS RPC. An invalidation scheme with pull-based update is used to

[1] Without loss of generality, we assume one file server in each cluster.

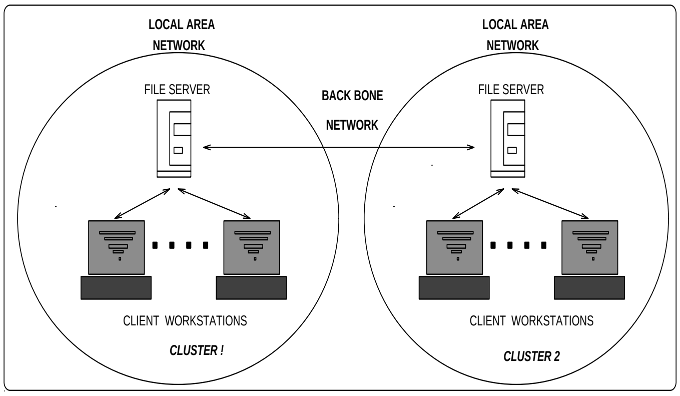

Figure 2: Basic architecture of a Frolic file system

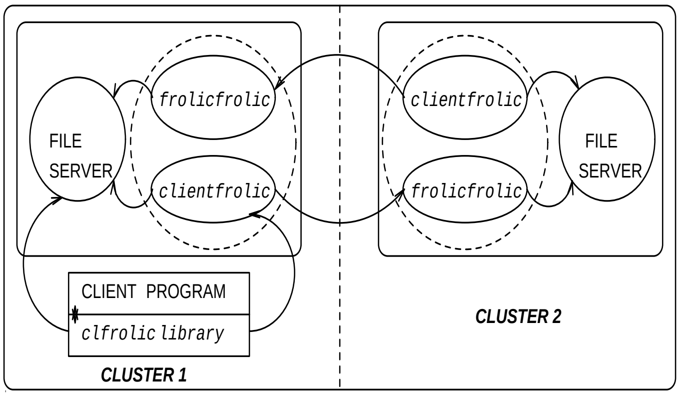

Figure 3: Frolic prototype structure

maintain consistency. The two Frolic server processes execute on the file server machine. Client applications are linked with the *clfrolic* library which traps file system calls. If a file requires Frolic server processing, an RPC call is made to the *clientfrolic* process which in turn makes an inter-cluster RPC call to the *frolicfrolic* process on another cluster when necessary.

An alternative to trapping the file calls at the client is to trap the operations at the file server machine. With this scheme, both the *clfrolic* library and the RPC call between the client and the Frolic server are no longer required, and thus applications do not have to be re-linked. We chose not to implement this scheme since it would involve modifying the kernel and/or the file server code.

In the Frolic prototype, files that are shared between clusters are identified by a bit in the file's record (inode); a second bit in the inode indicates whether a shared file is valid or invalid. The shared files reside under directory points which are identified in prefix tables maintained by the *clfrolic* library and the Frolic server processes.

Using these prefix tables, the *clfrolic* library and the *clientfrolic* process perform intra-cluster and inter-cluster file name mappings, respectively. Thus, the client and Frolic hosts do not have to share the same file name space. As a result, the prototype can be run on file systems such as NFS. A uniform global name space can, however, be set up by having the same prefix entries in the *clfrolic* and Frolic server prefix tables.

In addition to the prefix table, each Frolic server also maintains a list of valid/invalid file information. If a file is valid, the list entry contains the names of the clusters which have requested a valid file from that server. If a file is invalid, the list entry contains the name of the cluster which has the valid file. This list resides in Unix shared memory since both Frolic server processes may update it. Locks are used to serialize access to the same list entry.

The specific operations of Frolic are dictated by the pull-based consistency scheme. The *clfrolic* library traps the file operations *open*, *close*, *creat*, *unlink*, *mkdir*, and *rmdir*; *reads* and *writes* are not intercepted since we are doing whole file replication at *open* time. When a file is opened, the *clfrolic* library makes a direct Unix open system call on behalf of the client. Subsequently,

- if the open is successful and the inode bits indicate that the file is valid or non-shared, the *clfrolic* library returns immediately to the application program. Thus, negligible overhead is incurred for file opens not requiring Frolic server interaction.

- if the file is invalid, an RPC call is made to the *clientfrolic* process. The *clientfrolic* process looks up its prefix table and list of invalid files for the name of the invalidating cluster, and then initiates a pull-update. The server containing the valid file records the requesting cluster's name in the file's entry in its list of valid replica.

- if the file is not found in the local cluster, and its path name indicates that it is in a shared directory, the *clientfrolic* process probes its peers in the other clusters to determine if the file exists. If a valid copy is found, a replica is obtained as in the invalid case, and intermediate directories are created if necessary.

At file close time, if the file has been opened for update, the *clientfrolic* process looks up its list of valid replica, and sends invalidation messages asynchronously to the clusters with a valid replica. Note that when a cluster updates a file for the first time (i.e., after obtaining a valid replica or after system startup), invalidation messages must be sent to all the clusters. An alternative using a distribution of valid sets [6] to avoid broadcasting may be implemented in the future.

Mkdir, *rmdir*, and *unlink* of shared files are propagated asynchronously to the other clusters.

The consistency semantics of this pull-based scheme are dependent on the underlying file system. For example, with AFS, file close consistency semantics are guaranteed. With NFS, however, the consistency semantics of the prototype is timing dependent.

3 Preliminary Measurement Studies

Our initial performance studies are conducted on two clusters connected by a backbone network. Each cluster consists of Sun Sparc IPC workstations served by NFS. A null RPC call on average takes 4 ms within a cluster and 17 ms across the clusters. An RPC transfer of 8K bytes takes 25 ms within a cluster and 55 ms between the two clusters.

3.1 Sequential Workload

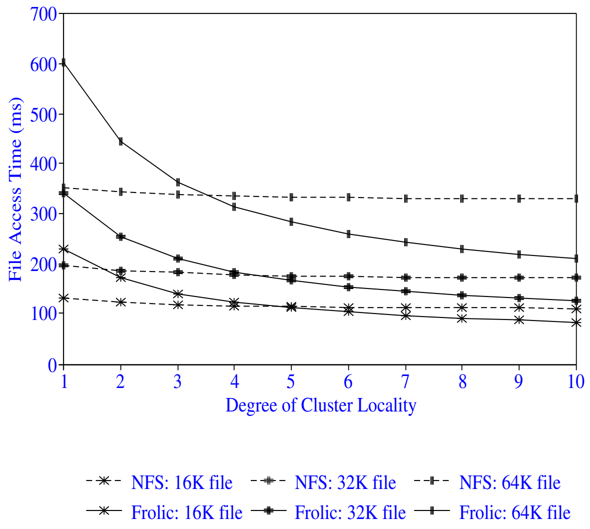

Figure 4: Performance of Frolic for varying degrees of cluster locality

We first compare the performance of Frolic implemented on top of NFS versus native NFS for the sequential single writer, multiple reader (SWMR) access pattern involving one file. Without concurrent accesses, network and server congestion is also absent. The file size and degree of *cluster locality* are varied in our study. We define *cluster locality* as the number of consecutive read accesses to the file from *different* hosts in the same cluster before the other cluster writes to it. In Frolic, since the file is replicated, a modification to the file causes an invalidation request to be sent to the other cluster. With native NFS, the file is placed in the same cluster as the writer so that all the writes are to the local file server. Each reader accesses the file only once after the write; Frolic and NFS read times for a file that is cached on a workstation have been verified to be the same so we exclude them from the workload. We have also verified that reads in a cluster to a file that is located on the local file server is the same for NFS and Frolic; thus, readers in the same cluster as the writer are not considered. Whole file updates and reads are performed in the workload. The measure of performance used in our study is the *average file access time*, which is the sum of the durations of all file operations in an open-close session; the times between file operations are excluded. This average includes the write access within the cluster and the subsequent sequential read times from the remote cluster. Since we want to exclude the effects of network congestion, the times are measured during periods of low network utilization.

Figure 4 shows the file access times for varying degrees of cluster locality and file sizes. In our two clusters, it is found that a local write costs about the same as a remote read[2]. Thus, for native NFS, the average access times are constant for varying degrees of cluster locality. In Frolic, invalidation and replication add to the cost of the write[3] and the first read in the cluster. However, these costs are amortized over a number of readers, causing the average access time to decrease and approach the time for a single local read session. As shown in Figure 4, Frolic access times are lower than NFS times for more than four readers for a wide range of file sizes. For larger file sizes and degrees of cluster locality the access times for Frolic are significantly lower than native NFS access times.

3.2 Concurrent Workload

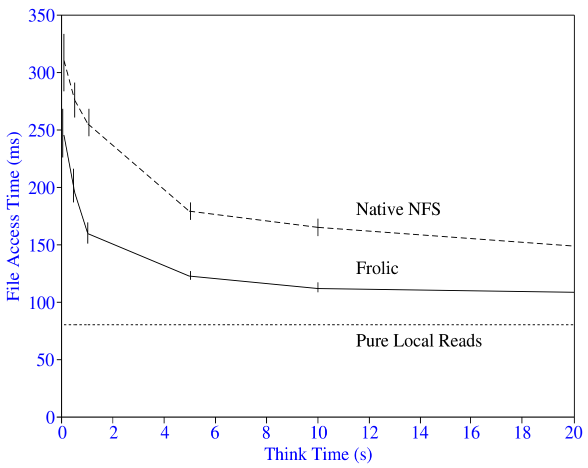

Figure 5: Performance of Frolic for varying client think times (32K byte files, 8 degrees of cluster locality, 18 clients)

Our preliminary concurrent workload consists of nine client workstations and one file server in each of the two clusters. Each client randomly chooses a file to access. For native NFS, since each file server contains half of the files, each client has an equal probability of accessing a remote/local file; as in the sequential workload, writes are to the local file server only. After accessing the entire file, the client pauses for a specified think time which we vary from 0.1 s to 20 s. The clients are initially synchronized so that the time between file requests in the *system* is Z/N where Z is the think time and N is the number of clients in the system. All files are 32K bytes and the degree of cluster locality is fixed at 8; i.e., a write occurs only after

[2]This fact suggests that our clusters may be too close for replication to offer significant benefits since no benefits from replication is expected in the extreme case of two clusters of workstations in the same LAN.

[3]Frolic writes are performed synchronously in the sequential and concurrent workloads.

8 clients in each of the two clusters have read the file once. As in the sequential workload, reads to cached files are excluded.

Figure 5 shows the file access times for various think times. The vertical bars indicate 90% confidence intervals. As a reference, the *pure local reads* case shows the time for a local read. Frolic access times are significantly lower than native NFS for a wide range of think times. For longer think times, the workload is essentially sequential; note that the access times are lower than the times in Figure 4 since this workload includes local reads. For short think times, multiple clients must wait for the Frolic server to replicate files, resulting in longer access times. For native NFS, the access times are even greater due to the backbone network becoming congested.

4 Conclusion and Work In Progress

The preliminary studies show that cluster-based file replication, as implemented in our Frolic prototype, can improve the performance of distributed file systems for a workload with widespread sharing of unstable files. Currently, we are engaged in:

- evaluating Frolic's performance on more than two clusters, and under bursty workload and network conditions,
- evaluating alternative Frolic designs and implementations, including multi-threaded Frolic server processes and trapping of file system calls at the file server,
- comparing the performance of Frolic with native AFS, and
- conducting workload studies in a large-scale commercial environment and studying the feasibility of using Frolic in such a system.

References

[1] H.S.Sandhu and S.Zhou. Cluster-Based File Replication in Large-Scale Distributed Systems. *Proceedings of ACM Sigmetrics Performance '92 Conference, 20(1),* May 1992.

[2] M.Blaze and R.Alonso. Dynamic Hierarchical Caching in Large-Scale Distributed File Systems, *Proc. 12th Intl. Distributed Computing Systems Conference,* June 1992.

[3] J.H.Howard, M.West, M.L.Kazar, S.G.Mennees, D.A.Nichols, R.N.Sidebotham, and M.Satyanarayanan. Scale and Performance in a Distributed File System, *ACM Transactions on Computer Systems, 6(1),* Feb. 1988.

[4] M.Satyanarayanan. The Influence of Scale on Distributed File System Design. *IEEE Transactions on Software Engineering, 18(10),* Jan. 1992.

[5] R.Sandberg, D.Goldberg, S.Kleiman, D.Walsh, and B.Lyon. Design and Implementation of the Sun Network File System. *Proc. USENIX,* Summer, 1985.

[6] K.Li and P.Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems, 7(4),* November 1989.