Serverless Network File Systems

Himani Apte
CS 739: Distributed Systems
University of Wisconsin, Madison

February 22, 2006
Spring 2006

1 Overview

The paper presents a design for serverless network file system that can dynamically distribute control processing, data storage and caching among a set of cooperating workstations. The aim is to improve performance, scalability and availablility of such a peer-based distributed system.

The approach adopted in this paper is to build a monolithic system that handles all aspects of storage, caching, data layout and recovery. This is in contrast to the technique used by later distributed file systems such as Petal and Frangipani, which have a two-layer structure in order to reduce the complexity associated with building a single monolithic system.

2 Problem Statement and Assumptions

Most traditional distributed file systems have a centralized server that provides file system services to a large number of clients. Such a server tends to becomes a performance bottleneck as more clients are added, thereby limiting the scalability of the system. The file server is also a single point of failure in the system, which reduces system reliability. xFS avoids these problems by making the distributed file system serverless.

xFS assumes availability of fast switched network among the cooperating workstations of the distributed system. *The LAN is assumed to have a lower latency compared to the physical disk and hence data access from a remote machine's memory is expected to be faster than from the local disk.* The next generation multimedia and parallel applications have higher I/O requirements and expect better performance from the file system that cannot be satisfied by the current centralized file system designs. *However, this may not necessarily be true if the distributed file system uses multiple servers with appropriate load-balancing.*

The workloads are assumed to have good locality over multiple clients and workstations are expected to have large memory, so that cooperative caching gives a good performance. Also, xFS assumes that the workstations can mutually trust each other and hence does not provide any security primitives in the file system.

3 Methodology

xFS borrows ideas from various earlier systems to design a serverless file system. It uses RAID-style disk striping with a single parity disk to improve reliability and availability. The parity disk can be used to reconstruct data in case of single disk failure. *xFS appears to be using RAID-4 striping, i.e. all parity blocks are written to a single disk (as shown in Figure 1). This is definitely sub-optimal compared to RAID-5 striping which uses distributed parity. It seems that it may be possible for xFS to incorporate RAID-5 striping.*

RAID systems typically have a large performance overhead for small writes. *xFS solves this problem of small writes by writing data to disk as large log segments (as in LFS). However, stripe group reconfiguration remains to*

be an expensive operartion for xFS.

xFS incorporates the log-structured file system (LFS) design. Writes by each client are buffered in its memory and then written to disk as a contiguous log segment. This has a very good write performance and avoids the RAID small write problem. The log also provides a simple means for failure recovery as it can simply roll forward from the last checkpoint to re-build important data structures. The problems with LFS approach are complicated reads as the latest copies of the inodes must be tracked using imaps and overhead of log cleaners to create free disk space for new log segments.

xFS draws heavily on the design of Zebra striped network file system. The advantages are that the log is private to each client and parity computation is entirely local operation (thereby avoiding the expensive operation of tranferring log segments across clients). *With respect to Zebra, the new features incorporated in xFS are use of parallel log cleaner, stripe groups and a distributed manager design.*

4 Design

xFS uses the central theme of "anything anywhere" and leverages this location independence for load balancing and to increase locality.

4.1 Metadata and Data distribution

xFS distributes the four main tasks of a central server into different system components. The system's data blocks are distributed within stripe groups of storage servers. The metadata manager handles both the disk location and cache consistency metadata. Cooperative caching among clients replaces central server caches. Finally the cleaner component is responsible for recycling the disk space for log.

A number of mapping tables are used to manage the highly distributed data and metadata. Firstly a globally replicated manager map locates the manager associated with each file. *Global replication of the manager map to all of the managers and all of the clients in the system could become a performance bottleneck as the number of clients increase and if there are frequent changes in system load or the membership of managers. This could potentially limit scalability of the system.*

Each manager maintains an imap for its files in order to locate the disk log address of their inode. The file inode structure in xFS is very similar to that in FFS. *In order to avoid any two clients from using the same inode number, xFS must divide the inode space among its clients. This is done via a static partitioning.* The imap is collocated with a manager, cached in main memory and checkpinted on persistent storage.

The stripe group map maps a disk log address to a list of storage servers. The size of each stripe group is essentially less than the number of storage servers in the system. *This has the advantages of improving availability as one parity server is used per stripe group at the cost of reduction in disk storage space and bandwidth.* This also gives better write performance, allows optimal match between network and disk bandwidth and allows use of single cleaner per stripe group.

4.2 System operation

Figure 3 provides a detailed depiction of the steps involved in reading a data block in xFS.

Although the control flow of read operation has been neatly shown here, it does not elaborate on mechanisms needed to handle race conditions occuring due to multiple requests for the same data block from different clients and due to unpredictable network delays. Delegating a client to transfer a cached data block to another client makes the assumption that both clients and the network will be available through the entire operation.

The system optimizes its I/O path under the assumption that access to local memory is faster than access to remote machine's memory which in turn is faster than access from disk. Also, strong data locality assumptions exist in order to reduce the number of network hops.

xFS uses block level caching instead of file caching. *Finer granularity caching semantics give better performance and avoid false sharing as observed in SpriteFS. However, this also requires more complex cache consistency metadata to be maintained.*

xFS uses the first writer policy in order to collocate the manager with the client. This dramatically reduces the number of network hops for I/O operations. *The potential network hops for a data block read (for a small file)*

is five. However, as shown via simulation, 75% of the read requests are served from local cache, thereby reducing the avearge network hops to 2.9. This is expected to be higher for large files which must additionally access indirect blocks.

5 Implementation

The paper reports only a partial implementation of the xFS design. Both the distributed log cleaners and automatic crash recovery and dynamic reconfiguration components have not been implemented and hence it is not possible to empirically evaluate their design in terms of its usability.

6 Evaluation

The test environment consists of 32 workstations connected by high-speed switched Myrinet network with TCP throughput of 3.2 MB/s and disk bandwidth of 2.7 MB/s. The performance of xFS has been compared with that of NFS. *The NFS tests were performed for a single disk server with peak bandwidth of 5 MB/s, which is unfair as an NFS server would likely have multiple disks and processors. It would be more appropriate to add additional hardware to the NFS server for a fair comparison.*

The aggregate large write bandwidth is expected to have the best performance in such a system, however the bandwidth available to a single client is even less than 1 MB/s. The study of the storage server scalability does not give much insight into how to decide the stripe group size.

7 Conclusions

The xFS design brings out important considerations when developing a location independent serverless distributed system, such as collocation of data and manager, importance of cooperative caching, the use of stripe groups for scalability. A highly cited paper, many important research and commercial systems have drawn on the lessons learnt from xFS, including the now-famous peer-to-peer and cluster based systems.