4-1-2008

Sloth-NFS and the Possibility of using Fuzzy Control to Optimize Cache Management

Ryan C. Spring

Eric Freudenthal
University of Texas at El Paso, efreudenthal@utep.edu

Sloth-NFS and the Possibility of using Fuzzy Control to Optimize Cache Management

Ryan C. Spring and Eric A. Freudenthal
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79902
Email: rcs@spring-west.net, efreudenthal@utep.edu

Abstract—Modern software systems are complex and increasingly vulnerable to malicious attack. In order to apply bug fixes and protect against security weaknesses, workstation administrators must continuously patch operating system and application program installations. To simplify this process, administrators generally configure systems into clusters with common installations of operating systems and application programs in a manner that patches and other routine maintenance can be applied en masse. Two widely used approaches offer complementary advantages: Installations and updates are particularly convenient when all data including operating system and programs is served by traditional centralized network-connected file servers (networked file systems) because administrators need only maintain a single image that is distributed online to all workstations. In contrast, operating systems and application programs can be installed onto disk drives within each workstation. This *mass replication* of common data provides high aggregate bandwidth through parallelism since each disk operates independently. However, in contrast to the network file system approach, it is substantially more difficult to keep a large network of systems up to date when each system has an autonomous installation. Through the aggressive use of cooperative file caching, we expect that sloth-NFS is will provide the advantages of both approaches.

In this paper, we discuss the administrative problems with current distribution methods for program and operating system installations, analyze design decisions necessary in Sloth-NFS and present results from an initial simulator experiment.

I. Overview

Technical administrative costs are a major challenge for the modern enterprise. These costs are high, because many hours of administrator's time are required to perform a large number of tasks related to basic system maintenance. Key administration challenges include:

- **Adding Clients**: As the user's needs grow and change, needs for client machines change, administrators must add new client machines to the cluster.
- **Emergency Update and Patching**: To ensure system security and stability updates, administrators must apply them quickly to all systems with the affected software.
- **Routine Application Software Update**: As new versions of application software are released, administrators make them available on systems.
- **OS Reinstallation and Major Upgrade**:
  As new operating system software is released, administrators must update systems to use it.
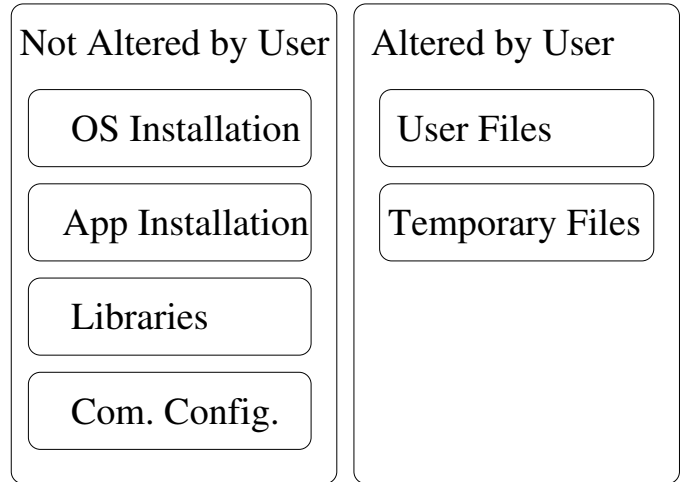

Fig. 1. File Types on a Network Client

To limit costs, systems are frequently configured as clusters with identical installations of operating systems and application programs. A feature of this approach is that it facilitates the application of administrative updates to the group en masse.

As shown in Figure 1, unlike user or temporary files, system installations, including operating systems, application programs, libraries, and common configuration data, should not be changed by ordinary users. Since such changes can cause security or other administrative problems, operating systems security mechanisms are frequently employed to prevent users from making these changes. In centralized installations, it is common to distribute these files in a read-only networked file system in a manner that only permits changes to be applied directly to the server by administrators.

Two widely used approaches to cluster configuration, which are organized into *clusters*. These two approaches offer complementary advantages: Read-only network file systems and Mass Replications.

A. Read-only Network Filesystems

In this section, we describe the distribution of operating system and application programs through the use of a read-only networked file system. In this approach, common application and operating system installations are stored on a central
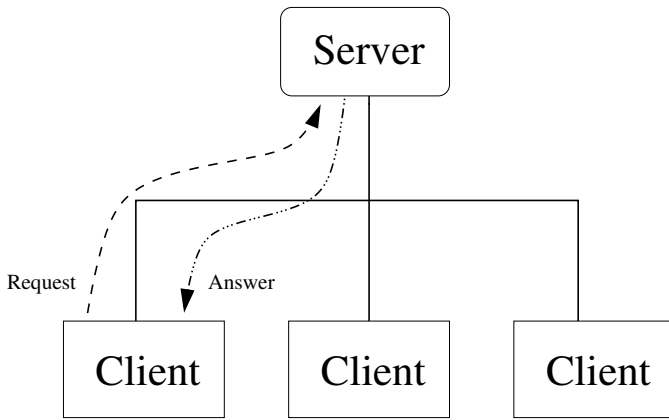
Fig. 2. A typical network filesystem

server, which makes them available to client workstations, through a network filesystem.

The centralized network filesystem approach is well suited for all four administrative tasks since only the filesystem being served by the server must be changed. The administrative tasks are easily performed as follows:

- **Adding Clients**: New systems must only be configured to boot from the network filesystem.
- **Emergency Update and Patching**: Administrators patch network filesystem. Clients instantly see the new patched installations on the central server.
- **Routine Application Software Update**: Administrators install the new version of the application software on the central server, and all subsequent file open operations will be directed to it.
- **OS Reinstallation and Major upgrade**: At times, the most efficient approach to distributing a major update is through the complete replacement of a filesystem. In these cases, existing network protocols can direct the client to reboot and mount the replacement filesystem.

The weakness of centralized network filesystem approach is its centralization, which leads to a network hot spot and server congestion. All clients must send and receive data from central server. As the number of clients increases, the load on the server increases as well as the load on the network in the vicinity of the server.

B. Mass Replications

The most common alternative to the use of a networked file system to distribute operating system and application programs is the *mass replication* of these files onto disks directly connected to each client. While this approach eliminates congestion at the server, it requires substantially more complicated infrastructure and effort to administer since patches and updates must be installed onto each system independently.

In a mass replication, common application and operating system installations are stored on a local storage device, which is periodically patched to update the various installations on the device.

With mass replication, the network hot spot problem is alleviated when each client has a current copy of all installation files. However, a centralized server can become congested if it is used to simultaneously provide installation images or updates to many clients.

- **Emergency Update and Patching**: Emergency update requires both the updating of the central copy of the installation and the application of patches at each of the clients. While there is substantial support for automatic dissemination of patches (usually via a centralized network filesystem), these systems are complex and it can be problematic if clients miss a critical patch or install patches in differing orders. Finally, the mass distribution of a large patch to many systems may saturate a centralized server, delaying installation and reducing overall system performance.
- **Routine Application Software Update**: Like emergency updates, non-critical updates require both the updating of the central copy of the installation as well as the initiation of replication. This process is less time-critical than emergency updates, but is similarly complex.
- **OS Reinstallation and Major Upgrades**: Installation requires both the updating of the central copy as well as the initiation of replication. This may be triggered by an emergency update and the time required to disseminate the new full installations may disrupt critical operations.

C. Summary

These two complementary approaches have complementary advantages and disadvantages. Network filesystems are maintainable, but not scalable. Mass replication schemes are less usable requiring more effort to maintain, but offer higher performance, especially in larger clusters where aggregate load to the central server may be much greater. We are investigation an alternate approach with the usability of network filesystems and the scalability of mass replication. We theorize this can be best achieved through a novel network filesystem with a cooperative cache. We call this system Sloth-NFS.

II. SLOTH-NFS

Sloth-NFS extends the NSF network filesystem with a cooperative cache that enables clients to cache blocks that they share to other clients. This cooperative cache is expected to permit the majority of file accesses to be satisfied by clients without server involvement.

Clients participating in the cache attempt to resolve accesses for chunks in the following order:

1) **Available in memory**: The chunk has been accessed by the client recently and a copy remains available in client's memory
2) **Available from client**: The chunk is available from another client, so it is transferred from there rather than the central server.
3) **Available from server**: The chunk is not available elsewhere, so it retrieved from the file server.
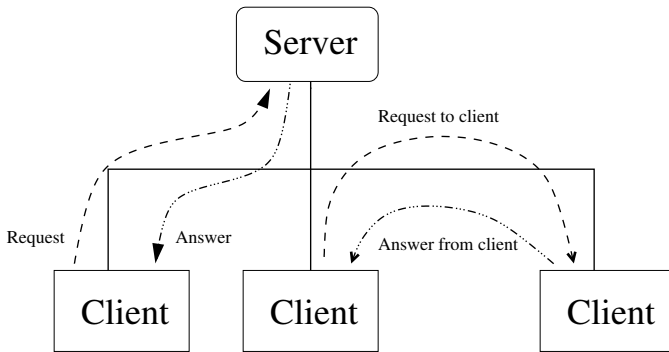
Fig. 3.   Sloth-NFS Operation

Sloth-NFS will have the administrative advantages of conventional networked filesystem. Furthermore, since, in Sloth-NFS, many requests will be satisfied by a large number of clients rather than only by a central server, we anticipate that client file access latency will be substantially reduced, especially in large installations.

III. PREVIOUS WORK

The first widely used network filesystem was NFS (network filesystem) [1] introduced by Sun Microsystems in 1984. NFS has become the standard production network filesystem for use in Unix/Linux networks. It is a central server filesystem in which all file accesses are answered by a central server.

It has served as the basis of many advanced research network filesystems and will serve as the basis for Sloth-NFS.

Scalability issues with NFS led to the development of the Andrew filesystem at Carnegie Mellon that caches filesystem chunks in disks installed within clients. Though Andrew clients cache chunks on clients it is not cooperative: neighboring clients can't retrieve chunks from each other's caches.

Others have considered the augmentation of a networked filesystem with a cooperative cache. NFS-CC [2] is a research network filesystem based on traditional NFS with the addition of server-based redirection. Files accesses may either be requested and retrieved from the server, or the server may redirect the request to another client that provides the chunk from a local cache.

Since every request for a chunk in NFS-CC must be communicated to the central server, this server remains a resource bottleneck. This effect can be aggravated if the server is not co-located with clients and instead installed in a distant server room since all requests must be transmitted over this high-latency connection. We hypothesize that this approach, while less centralized then traditional NFS and thereby offering better performance then it, is still more centralized then is optimal. Thus there is still room for improvement with the more decentralized approach like Sloth-NFS

An alternative variant of NFS with cooperative caching is Shark [3] Similar to NFS-CC the Shark server is still the first recipient of all file access requests and the chunks are cached at clients (cachers). However, it adds an extra layer of indirection involving the Coral Distributed Hash Table. In Shark, the client first queries the Shark server for a list of chunks that comprise a file. The clients then look up the $chunk \rightarrow cacher$ mapping in Coral. If one exists, the chunk is retrieved from the cacher. If not, it is retrieved from the Shark server.

While we draw much inspiration from some details of this system, our goal is considerably different. Our target is local networks, where communication latency is low and bandwidth is high. In contrast, Shark targeted wide area networks such as the Internet, where latency is less critical.

IV. DESIGN

As is common in cache design, our work will address mechanisms and policy for *placement*, *replacement*, and *lookup*. There are three key design challenges in the cooperative cache of Sloth-NFS: Placement, Replacement and Lookup.

Placement is the dynamic determination of which clients will cache a chunk. We are considering three approaches:

1) **Lazy**: File chunks are cached at the participants that needed the chunk recently.
2) **Load Driven**: File chunks are cached at participants that haven't needed the chunk, but which have available memory capacity to cache them.
3) **Hybrid**: Participants cache chunks in both lazy and load driven manners.

The replacement challenge involves determining which chunks a client removes from its cache when space is needed.

1) **Local**: Replacement in a participant's local portion of the cache is decided exclusively based solely on local state.
2) **Global**: Replacement in a participants local portion of the cache is decided based on the full state of the system. However, dissemination of the full system state may not be practical.
3) **Hybrid**: The replacement in a participants portion of the cache is a multi-criteria decision based on full knowledge of local state and partial (and potentially incorrect) knowledge of external state.

The lookup challenge involves determining if and which client(s) are caching a chunk.

1) **Broadcast**: All clients broadcast each lookup request to all participants. All participants caching the requested chunk respond. In this scheme, there is no central index of contents.
2) **Central Server**: There is a central index of contents stored on a central server, all clients unicast their lookup request to a central server, this server responds to the request.
3) **Gossip**: There is no coherent indexing protocol, instead clients exchange partial knowledge at each communication.

V. SIMULATOR

A simulator was implemented to test the validity of the Sloth-NFS approach. Only initial proof-of-concept experiments have been performed that ignore the cost of lookup.
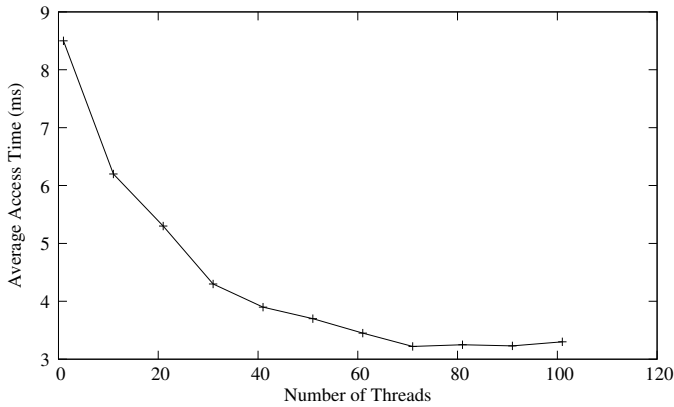
Fig. 4. Experiment 2 Results

The simulator was implemented using multiple asynchronous threads. Each thread simulates a client system, asynchronously requesting file chunks. Each of these simulated accesses proceeds as follows:

1) **Select a block to access**: A Zipf distribution is used to randomly select the block to be accessed.
2) **Determine access location**: Based on global information, determine if the chunk will be obtained form the central server or a client.
3) **Update timing data**: Aggregate statistics of access latencies are updated.
4) **Update cache directory**: Adjust simulator data structures to reflect the presence of the chunk at the thread now containing the data. Client caches have a capacity of one hundred blocks and thus acquisition of a new block can result in the eviction of the least recently accessed block within its memory.

Zipf distributions [4],[5] have been observed to approximate the distribution of file and web accesses in a variety of contexts [6]. We are simulating a cluster of workstations that presumably will be performing similar tasks, and thus, in this initial experiment, an identical distribution is assumed for all clients.

Figure 4 indicates the average client remote access latency for systems of various sizes at that request blocks randomly drawn intervals in the range $(0:5]$ seconds on a cooperative cache with lazy placement and local eviction policy. Requests satisfied by the central server incur a 20ms delay (roughly corresponding to a random disk transfer), and requests satisfied by another client incur a 2ms delay (roughly corresponding to a round-trip transaction on a local network).

As expected, latency decreases as as new clients are added due to the increase in available cache. Average Response time flattened out at approximately 3ms, this represents a large improvement over what is available from a file server.

VI. PROJECT SUMMARY AND FUTURE WORK

The majority of the work on Sloth-NFS is still to be done. We will use the simulator to rapidly explore various combinations of placement, replacement and lookup schemes.

We will also augment the simulator to simulate subtleties of network and server load.

We also wish to prove or disprove our assumptions in the simulator. We will gather empirical data about file access distributions and determine both 1.) whether file accesses are Zipf distributed and 2.) whether all clients follow the same distribution or different distributions or have different distribution parameters.

Finally, we have begun implementation of the prototype system as a user mode filesystem, using the usermode filesystem kit [7] provided as part of the SFS [8] distribution. Our initial implementation will be with lazy placement, local replacement and central server lookup. We don't expect the performance here to be optimal, but it will provide an easily optimizable baseline implementation, onto which we can later add DHT and Gossip lookup functionality.

For this optimization, we need to control the parameters of our implementation. For this optimization, we cannot directly apply traditional techniques of optimal control, since we do not know the exact equations describing the dynamics of a system: this dynamics depend on the behavior of various users. It is therefore natural to use intelligent control techniques which have been explicitly designed for optimization under uncertainty; see, e.g. [9].

VII. SYNOPSIS

In this paper, we analyzed the administration shortcomings of current options for distribution of common programs and operating system installations. We proposed an alternate system Sloth-NFS, that will address the shortcomings of network filesystems to allow them to better serve as the method of choice to distribute these installations.

We presented the design choices we face and presented our simulator, which allows us to investigate the options for theses choices without building full implementations. We finally presented our plans for further investigation both in the simulator as well as in a prototype implementation.

REFERENCES

[1] B. Callaghan, B. Pawlowski, and P. Staubach, "NFS version 3 protocol specification," Network Working Group, RFC 1813, june 1995.
[2] Y. Xu and B. D. Fleisch, "NFS-cc: tuning NFS for concurrent read sharing." *IJHPCN*, vol. 1, no. 4, pp. 203–213, 2004.
[3] S. Annapureddy, M. J. Freedman, and D. Mazières, "Shark: scaling file servers via cooperative caching," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2005, pp. 129–142.
[4] B. Mandelbrot, *Fractals: Form, Chance and Dimension*. San Francisco, CA USA: W.H. Freeman and Company, 1977.
[5] ——, *The Fractal Geometry of Nature*. New York, NY USA: W.H. Freeman and Company, 1982.
[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *INFOCOM*, 1999, pp. 126–134.
[7] D. Mazieres, "A toolkit for user-level file systems," in *USENIX Technical Conference*, Boston, MA, June 2001.
[8] D. Mazires, "Self-certifying file system," Ph.D. dissertation, May 2000.
[9] H. T. Nguyen, *A First Course in Fuzzy and Neural Control*. Boca Raton, FL, USA: CRC Press, Inc., 2002.