Introduction

This report describes the basic foundations of distributed file systems and one example of an implementation of one such system, the Andrew File System (AFS).

The first part of the report describes the conditions on which distributed systems started to evolve and why. It also discusses some common requirements for distributed systems such as concurrent updates, different kind of transparency and security.

The second part explains the history of AFS since its creation in the 1980s, and its usage today. It also goes more into detail and describes the basic terms used in AFS systems such as cells and volumes. The core components of AFS such as server and client services are described and in the last part AFS is compared to other systems and its advantages are identified.

Distributed file systems – an overview

What is a file system?

File systems are abstraction that enable users to read, manipulate and organize data. Typically the data is stored in units known as files in a hierarchical tree where the nodes are known as directories. The file system enables a uniform view, independent of the underlying storage devices which can range betwees anything from floppy drives to hard drives and flash memory cards. Since file systems evolved from stand-alone computers the connection between the logical file system and the storage device was typically a one-to-one mapping. Even software RAID that is used to distribute the data on multiple storage devices is typically implemented below the file system layer.

What is a distributed file system?

During much of the 1980s, people that wanted to share files used a method known as sneakernet. A sneakernet is the process of sharing files by copying them onto floppy disks, physically carrying it to another computer and copying it again.

As computer networks started to evolve in the 1980s it became evident that the old file systems had many limitations that made them unsuitable for multiuser environments.

At first many users started to use FTP to share files. Although this method avoided the time consuming physical movement of removable media, files still needed to be copied twice: once from the source computer onto a server, and a second time from the server onto the destination computer. Additionally users had to known the physical addresses of every computer involved in the file sharing process.

As computer companies tried to solve the shortcomings above, entirely new systems such as Sun Microsystem's Network File System (NFS) were developed and new features such as file locking were added to existing file systems. The new systems such as NFS were not replacements for the old file systems, but an additional layer between the disk file system and the user processes.

Summary of possible features of a distributed file system.

Concurrent updates

The file systems in the 1970s were developed for centralized computer systems, where the data was only accessed by one user at a time. When multiple users and processes were to access files at the same time a notion of locking was introduced. There are two kinds of locks, read and write. Since

two read operations don't conflict, an attempt to set a read lock on an object that is already locked with a read lock is always successful. But an attempt to set a write lock on an object with a read or write lock isn't.

There are two philosophies concerning locks: mandatory locking and advisory locking. In mandatory systems operations actually fail if a lock is set. In advisory systems it is up to the programs to decide if the lock is to be honored.

Transparency

An early stated goal for some of the new distributed file systems was to provide the same interface as the old file systems to processes. This service is known as access transparency and have the advantage that current stand-alone systems could be migrated to networks seemingly easy and old program wouldn't need to be modified.

Location transparency makes it possible for the clients to access the file system without knowledge of where the physical files are located. For example this makes it possible to retain paths even when when the servers are physically moved or replaced.

Scaling transparency is the ability of incremental growth of the system size without having to make changes to the structure of the system or its applications. The larger the system, the more important is scaling transparency.

Distributed storage and replication

One possibility in distributed systems is to spread the physical storage of data onto many locations. The main advantage of replication is increased dependability. Multiple copies increases the availability of the data by making it possible to share the load. In addition, it enhances the reliability by enabling clients to access others locations if one has failed. One important requirement for replication is to provide replication transparency, users should not need to be aware that their data is stored on multiple locations. Another use for distributed storage is the possibility to disconnect a computer from the network but continue working on some files. When reconnected the system would automatically propagate the necessary changes onto the other locations.

Security

In a multi-user environment a desirable feature is to only allow access to data to authorized users. Users must be authenticated and only requests to files where users has access right should be granted. Most distributed file systems implement some kind of access control lists. Additionally protection mechanisms may include protect of protocol messages by signing them with digital signatures and encrypting the data.

Andrew File System

History

Andrew File System (AFS) is a file system that once was a part a larger project known as Andrew. Andrew was a project of Carnegie Mellon University (CMU) to develop a distributed computing environment on the Carnegie Mellon campus in the mid 1980s. AFS was originally developed for a computer network running BSD UNIX and Mach.

The design goal of AFS was to create a system for large networks. To scale well the system minimizes client-server communication by using an extensive client-caching scheme. Instead of transferring recent data for each request, client stores whole files in their cache.

In 1989, the original design team of AFS left CMU and founded a company named Transarc Corporation that commercialized AFS. In the 1990s a number of ports of AFS were done by Transarc's customer to other systems, for example a group at MIT ported AFS to Linux. IBM acquired Transarc in the 1990s and continued to produce commercial implementations of AFS. In 2000 IBM decided to release a branch of the source code under the IBM Public License. The current development of AFS is now done in a project named OpenAFS. Today the OpenAFS implementation supports a number of platforms such as: Linux, Apple MacOSX, Sun Solaris and Microsoft Windows NT.

AFS today

Today AFS is mature and is not only used for research purposes, it is deployed in some large (5000 workstations+) networks. A number of known universities such as Stanford, MIT and KTH uses AFS in their computer networks. Even research labs such as CERN and large companies such as Intel and Morgan Stanley uses AFS.

OpenAFS is in active development, the most recent version (1.4.1) was released 2006-04-23 and added support for MacOSX 10.4.

AFS - Technical details

This overview will primarily focus on the OpenAFS implementation in a UNIX environment (www.openafs.org).

AFS communication is implemented over TCP/IP (default server ports: UDP 7000-7009). Rx, an RPC protocol developed for AFS, is used for communication between machines. The communication protocol is optimized for wide area networks, and there are no restrictions on geographical locations of participating servers and clients.

Cells

The basic organizational unit of AFS is the *cell*. An AFS cell is an independently administered collection of server and client machines. Usually, a cell corresponds to an Internet domain, and is named thereafter. Since AFS is most commonly used in academic and research environments, it is not unusual for cells to also participate in a global AFS filetree. However, this is not mandatory.

Servers and clients can only belong to a single cell at a given time, but a user may have accounts in multiple cells. The cell that is first logged in to is the *home cell*; other cells are called *foreign*.

From the user's point of view, the AFS hierarchy is accessed from /afs/[cell name] through normal file system operations. Since AFS is location transparent, the path to a file in the tree will be the same regardless of the location of the client. Whether the file is actually accessible or not depends on the user's credentials and the file's access permissions, as set by Access Control Lists.

AFS clients

The client-side component of AFS is the Cache Manager. Depending on AFS version, it may be run as a user process (with a few changes in the kernel), or implemented as a set of kernel modifications (OpenAFS). The responsibilities of the Cache Manager include retreiving files from servers, maintaining a local file cache, translating file requests into remote procedure calls, and storing callbacks.

Several key observations of common file access patterns inspired the design of AFS:

- Commonly accessed files are usually a few kilobytes in size
- Reading is more common than writing

- Sequential accesses outnumber random accesses
- Even if the file is shared, writes are usually made by only one user
- Files are referenced in bursts

The strategy chosen for AFS was to cache files locally on the clients. When a file is opened, the Cache Manager first checks if there is a valid copy of the file in the cache. If this is not the case, the file is retrieved from a file server. A file that has been modified and closed on the client is transferred back to the server.

Over time, the client builds up a "working set" of often-accessed files in the cache. A typical optimal cache size is 100 MB, but this depends on what the client is used for. As long as the cached files are not modified by other users, they do not have to be fetched from the server when subsequently accessed. This reduces the load on the network significantly. Caching can be done on disk or in memory.

The conceptual strategy used is that of whole file serving and caching. However, since version 3 of AFS, data is actually served and cached in chunks of 64 kilobytes.

To ensure that files are kept up-to-date, a callback mechanism is used. When a file is first opened, a so-called Callback Promise sent to the client together with newly opened files. The Callback Promise is stored on the client and marked as *valid* by the Cache Manager. If the file is updated by another client, the server sends a Callback to the Cache Manager, which sets the state of the Callback Promise to *cancelled*. The next time an application accesses the file, a current copy must be fetched from the server. If a file is flushed from the cache, the server is notified so that the callback can be removed.

In the case where the client has been rebooted, all stored Callback Promises need to be checked to see if their statuses have changed. To achieve this, the Cache Manager issues a Cache Validation Request.

If multiple clients happen to concurrently open, modify, and close the same file, the update resulting from the last close will overwrite previous changes. That is, the AFS implementation does not handle concurrent updates, instead delegating this responsibility to the applications.

AFS uses advisory file locking. Byte ranges can not be locked, only whole files. For these reason, AFS is not suited for situations where many users simultaneously need to write to a file, for instance in database applications.

Volumes

The basic unit of storage in AFS is the volume, a logical unit corresponding to a directory in the file tree. For example, it is common to map users' home directories to separate volumes. System files that are not needed at boot time can also be placed in volumes.

A volume must fit in a single physical partition, but is usually much smaller. In contrast to having partitions as the smallest unit of storage, the use of volumes offers several advantages:

- Volumes can be moved to different locations on a live system without affecting availability
- Volumes can be replicated, offering the possibility of load-balancing and increased availability
- File system quotas can be set on a per-volume basis
- Handling of backup and recovery is simplified

Volume replication

Volumes that are frequently accessed by many clients can be replicated. This entails placing read-only copies of a volume on different servers, and is thus most useful for volumes whose contents are frequently accessed but seldom changed, such as UNIX libraries and programs.

Which file server is contacted when a client tries to access a file depends on server load and availability, and is handled by the Cache Manager without client intervention.

When a file from a replicated volume is retreived, only one Callback Promise per volume is obtained. Any change to the volume will break the Callback Promise.

AFS servers

Server machines can have several roles, which may be combined:

- Acting as a simple file server
- Acting as a server for the replicated AFS databases
- Acting as a binary distribution machine, for delivering updated versions of the server software
- Acting as a system control machine that distributes common configuration files and also acts as a primary time synchronization site

When there is only one server in a cell, it will take on all of the mentioned roles. It is recommended that if a cell has more than one server machine, it is best to run multiple database servers, to gain the advantages of database replication. Since the administrative databases are often accessed, doing this helps distribute the load on the servers. However, having more than three database server machines is not necessary.

Simple file servers can be added as necessary as the demand for storage space increases. Only one system control machine is needed in each cell, and there should be as many binary distribution machines as there are different server system types.

The system can be administered from any client workstation in the cell.

Server processes

The AFS server implementation is modularized, and not all types of server processes are run on each server. Here is a short summary of the different processes and their functions:

The **Basic OverSeer Server (BOS)** makes sure that the other server processes are running correctly, and will restart failed processes. Its primary function is to minimize system outtages, and to reduce the need for manual administrator interventions. After a File Server or Volume Server fails, the BOS will invoke a special process known as the *Salvager*, which repairs disk inconsistencies caused by failure.

The **File Server** is responsible for delivering and storing files, maintaining a hierarchical file structure, handling copy/move/create/delete requests, and keeping track of file stats (size, modification time). It is also involved in creating links between files and granting advisory locks. To determine if users are authorized to perform requested actions, the File Server communicates with the Protection Server.

The **Authentication Server** maintains the *Authentication Database*, which stores user password and server encryption keys. It employs Kerberos to provide services for mutual authentication between clients and servers, and to grant time-limited tokens that clients use to prove that they have been authenticated. An option for the administrator is to use an external Kerberos implementation, such as MIT's.

The **Protection Server** maintains the *Protection Database*, and is a key component in the implementation of Access Control Lists. ACLs are applied on the directory level, and constitute a refinement of the normal UNIX permission system. There are seven different types of access permissions (lookup, insert, delete, administer, read, write, and lock). Up to 20 different ACL entries can be applied to each directory. For fine-grained access control, users can create ACL groups. These enable setting permissions for many users (or IP addresses) at the same time.

The **Volume Server** provides facilities related to volume management, such as creation, relocation, deletion and replication, and preparation for backup and archival of volumes.

The **Volume Location Server** maintains the *Volume Location Database*, which keeps track of which servers different volumes reside on. It is a key component in the transparent access of data, since the Cache Manager contacts it to determine from which file server to fetch a file.

The **Update Server** makes sure that all servers have up-to-date versions of the server processes and common system configuration files.

The **Backup Server** maintains the *Backup Database*, and is used to create backup tape archives of volumes.

Database replication

The contents of the four types of AFS databases may change frequently, and it is of vital importance to system performance that all database servers have identical copies of the databases. To ensure this, AFS employs Ubik, a replicated database mechanism implemented with RPC. Only one server, the synchronization site, may accept updates to the databases. It runs a process known as the Ubik coordinator. Changes are immediately propagated to other servers. During this time, read/write access to the databases is not possible. Unless a majority of the other servers receive and acknowledge the changes, the update is rolled back.

A list of which servers are actually synchronized needs to be maintained. If necessary, the Ubik coordinator may be moved to a different server through an election process, to maximize availability.

Clock synchronization

AFS in general, and its database replication in particular, is dependent on server clock synchronization. This is achieved by using the Network Time Protocol Daemon. One server acts as a synchronization site, gleaning the time from an external source. The other servers update their clocks against the synchronization site.

Discussion

The prominent features of AFS are location transparency, its performance and the built-in security:

Location transparency

AFS has automatic file location tracking by system processes and Volume Location Database. NFS mount points for tracking file's physical location must be set by administrators and users. Users only need to know the pathname to a file and its AFS cell.

Performance & scalability

One of Andrew File System's biggest advantages is the notation of volumes. Compared to normal file system partitions they can be moved from one computer to another without service outage. This can be useful in many cases, for example if one of two volume replication fails, the still working volume can be copied to another server immediately. It also makes AFS scalable, new servers can

be added and populated without interruption.

Finding the right balance when selecting what directories should be included in a volume may however be difficult.

During normal operation, the load of remote procedure calls on AFS servers should be much less than a comparable NFS server. This is because AFS only sends remote procedure calls on operations such as open and close. The callbacks helps to maintain the consistency of client caches where NFS uses getattributes calls by the clients much more often.

In addition to reducing the network traffic the cache system enables the system to perform subsequent accesses to file data almost as fast as it would be to a local disk. Another advantage is that if the file server would crash, the user can continue working on the open files that are in the local cache.

The scalability of AFS is limited by the performance of the server that is most-frequently accessed. Replication can help reduce this load by spreading the files on more than one server. However, this only works for read-only files, read-write files can't be replicated in AFS. This is one of AFS major drawbacks.

The initial AFS specifications aimed for a client/server ratio of 200-to-1. Currently, there are AFS sites running at this ratio, but the recommendation is 50-to-1. According to tests done with the Andrew benchmark, AFS has better performance than NFS when the number of clients increases.

Security

AFS uses Kerberos to authenticate users, this improves the security in many different aspects. Passwords are not sent over the network in plain text. Also, the authentication is mutual - both the user and the service provider's identities are verified.

The use of Access Control Lists enables the system's users to set detailed access rights for their own directories.

Conclusion

AFS has got some very pleasing qualities such as very good scaling and (for the most of the time) good performance thanks in part to its client-caching. The drawbacks of AFS is that it isn't user friendly for administrators and there may be some performance problems when heavy used read-write files exists in the system. The drawbacks seems insignificant compared to the advantages when comparing AFS with the competition, at least historically. So why didn't AFS become the most popular distributed file system? There are not an easy answer to that question, maybe it was lack of an early big company for marketing, or the fact that one had to pay a good deal of money to get an AFS license or its early lack of multi-platform support. Perhaps AFS will have a brighter future now when its open source, the future will tell.

References

- Operating Systems Concept, 7th edition.
- http://en.wikipedia.org/wiki/Sneaker_net
- http://en.wikipedia.org/wiki/OpenAFS
- AFS Administrator's Guide - http://www.openafs.org/pages/doc/index.htm
- AFS Frequently Asked Questions - http://www.angelfire.com/hi/plutonic/afs-faq.html