File Systems

Timothy Myers
Tennessee Tech
344 East 12th Street Apt. 4
Cookeville, TN 38501
931-206-2255
temyers240@gmail.com

ABSTRACT

In this paper I will try to give a brief overview of a few of the major file systems that may blur the line between file storage and file delivery in some cases. These range from the NFS which is primarily responsible for allowing the use of a native file system on another computer, to the impressively designed Google File System which manages the many, many terabytes of storage needed by Google. Like all file systems, they must keep track of how to store break up and store the abstraction we call files into pieces of binary data and to be able to retrieve those pieces again so we humans can use them. One of the reasons that these file systems are so different from the ones seen normally on storage devices is the fact that most of the file systems discussed here are used in a distributed computing environment, where many computers are accessing the same information, possibly simultaneously.

Keywords

File Systems, Google File System, Andrew File System, Network File System, ZFS.

1. INTRODUCTION

Out of the many things that a computer might find odd about a human, one would be our fascination with names, and another would probably be our desire for things to be in order, whether that be in alphabetical or chronological or otherwise. File Systems in general were created to help bridge the fundamental gap between humans and computers. They allow for a possibly large number of chunks of data strewn throughout the storage area of the computer to be seen as a human as a linear list of information with an actual name, able to be remembered by a person. Out of systems being discussed, the Network File System (NFS) is probably the most unusual, as it does not actually keep track of bytes of data on disk itself, instead the NFS is a protocol[] which enables a client machine to interact with a native file system on a server machine. The other file systems on the list, the Google File System (GFS), the Andrew File System (AFS), and the Zettabyte File System (ZFS), all of which have the capabilities of a local file system, such as Ext(2|3|4), NTFS, or FAT, as well as having the ability to be used in a distributed environment, for example a college, where students would like to be able to access their working files from as many places as possible. In fact that environment is where the AFS began[]. While this makes these file systems very useful in certain situations, the overhead involved in network access and availability limit their use for everyday users.

2. Google File System

I do not know and cannot pretend to know the sheer amount of information that the Google traffics in everyday, but even if they only retained a tiny fraction of the information available to be found on Google in the form of cached search results and other data, their need for storage space is enormous. Both because of the amount of information and because of the nature of the very broadly based customer base that Google is serving, Google created it's own file system which was designed to deal with some of the unique problems that arise in the situation that Google has placed themselves, in terms of availability, responsiveness, and concurrency[].

One of the big realizations that Google had in terms of data availability was the realization that hardware failure is more than an incident, and more even than a fact of life. Namely, they realized that hardware failure is happening *right now* to their systems and especially their storage systems. To help with this the GFS uses redundancy in its data handling. The GFS stores files as fixed sized chunks on chunkservers, whose job is to store these chunks of data, but each chunk is replicated on to multiple different chunkservers, in different server racks even[]. The number of redundant copies is three by default, but can be changed, usually upward, if needed for a specific set of chunks.

As far as responsiveness is concerned, the convention is to have one master server per GFS cluster, and possibly 1000 or more chunkservers per cluster[]. The master holds all file metadata and also has all of the knowledge concerning what files are comprised of what chunks, and chunkservers have those chunks[]. In a situation like a large server, the master could become a large bottleneck if all reads and writes are done through the master. So Google has created their file system such that when a master server is contacted about a file, it replies with the correct chunkserver and chunk, so the client can interact directly with the chunkserver directly[], thus cutting some network traffic overhead and keeping the information more available to clients.

Upon analyzing the types of operations done on the data held by Google, the designers of the GFS realized that the vast majority of writes done to files were appending data to the end of files and very little was actually writing in a random access fashion[]. To compound the act of writing to files in the GFS is the fact that a particular cluster of the GFS may have literally thousands of clients interacting with the cluster simultaneously. Because of this the GFS has an operation called record append. This operation is guaranteed to be atomic within the GFS itself[]. This allows for multiple clients to append data to a record (seemingly) at the same time, without the client needing much in the way of synchronicity.

One odd thing about the chunk size for the GFS is the large size of 64 MB[], which is much larger than the size for file pieces on other file systems. However, when it is seen that Google commonly deals with multi-GB files[], this chunk size becomes more reasonable. Another possible advantage of the large chunk size is the likelihood that any writing or reading that would need to be done would all happen within one singe chunk, and would not require multiple requests to the master server.

3. Andrew File System

The Andrew File Search (AFS) is a file system that was designed at Carnegie Mellon University as part of a campus wide initiative to provide a distributed computing environment. The AFS was included in an attempt to provide universal access to personal files by students, faculty, and staff[]. One thing that makes this an interesting contrast from the GFS is that while both were designed with a specific issue in mind. However, while the GFS was designed for a corporation to use over its enormous distributed network, the AFS was designed for use with the specific environment of a university, and in [] it is revealed that several possible shortcomings in AFS for very large systems were simply worked around because of the smaller collegiate setting. I hope to go ever a few of the very interesting features of the AFS here.

One of the most powerful aspects of the AFS is the access control lists (ACLs) that the file system was designed to implement. Once the file server authenticates the user trying to access files, that user is given a set of access privileges, in addition to the standard unix read, write, and execute permissions on files. These access controls include[]:

reading any file in the current directory

writing to any file in the current directory

inserting new files in the current directory

deleting files from the current directory

looking up files in the current directory

locking files in the current directory

being able to change the access control lists

These, along with the standard unix permissions allow for rather fine grain control on what the users are capable of doing with their files and in particular to the server.

Another interesting feature of the AFS is the fact that it does only operates on whole files. Most file systems split files up for storage and only transfers sections of files when access is required, but the AFS transfers the entire file to the client when the file is requested. The client then works on the local copy kept on the local machine[]. When the file is later closed, AFS transfers the entire file back to the file server which then updates the copy of the file on the server. These files are also cached on the client machine as well, allowing the AFS to participate in much less network traffic[].

The AFS also has a concept called Logical Volumes. From []:

“A typical logical volume would be a single user's files, or a particular release of the system binary files.”

While a user would not generally be aware of the logical volume made up of the files in his home directory, the person administrating the file server would use that logical volume to do things such as making a backup, or cloning, those files[]. This could also be used for distributing a new software release by cloning the logical volume with the releases binaries[]. These logical volumes can even be grouped together with “mount points” which link one part of the AFS to another in much the same way as unix allows for a separate file system to be grafted into the directory tree.

4. Zettabyte File System

The Zettabyte File System (ZFS) is a file system that was developed by the former Sun Microsystems, and was first seen in their Solaris operating system. It's community proclaims that the design of ZFS has removed 20 years of assumptions that are no longer valid concerning file systems[], and while I cannot verify that statement by any means, I will say that the list of features in ZFS are quite impressive and noteworthy. It has advancements that are not seen in other file systems, such as internal support for volume management, constant time snapshots, and even constant data error discovery and even data error recovery built into the file system itself. Here I hope to provide an adequate view on some of these revolutionary features.

One of the new control mechanisms that administrators have over the actual block devices making up the ZFS is a construct called a pool. One of these pools “sits” on top of possibly many local block devices and accept instructions for them[]. As an example, the administrator would create a pool by assigning block devices to the pool, and then he would tell the pool to create directories and export them onto the systems directory tree. The software of the pool actually places the files on the individual disks and can even set quotas or allow for space reservations on individual directories[]. Set up correctly, these pools can create the same affect as almost all versions of RAID, without RAID software, and done entirely within the ZFS[].

The ZFS does seem to carry a large amount of meta-data, but due to this it is able to create a very large amount of read only data snapshots in constant time with respect to the size of the file system[] or so it claims. The system it uses to accomplish this feat is the fact that each block of data in the ZFS is timestamped with the time of it's creation. This allows for one traversal of the file system tree to find all of the files created at or before a certain point in time, which is the snapshot[]. While this is a very impressive system, I am still skeptical of the advertised constant time snapshots.

The ZFS also performs a copy-on-write for each bit of data written into the file system[]. This seems to usually to be a copy to another physical device, but not necessarily need to be so. This copy-on-write facility is combined with another feature to provide the assumption of correct data and even data healing in case of an error. This other feature is the checksumming of all blocks of data written to the file system[]. Because, of this system, and the top level control of the pool, bad data, that is data for which the checksums do not match is not given to the requesting application[]. No only that but when a bad block is encountered, the pool will find a good copy of the bad block, give the good block to the application, and then will use the good block to repair any bad blocks encountered in the proccess.

5. Network File System

The Network File System (NFS) is quite different from the other file systems in that it does not actually play with where data goes onto a physical disk. The NFS is a protocol that allows for a client computer to use the native file system on a NFS server like

it is local to the client machine. It is accompanied by software on both the server and client machines. This software is necessary to turn local file management calls on the client into Remote Procedure Calls (RPS) that are sent across the network to the server which translates the RPC into the specific command required for the local file system being shared on the server machine[]. In this way even file systems we have already discussed, such as ZFS[], can be accessed using NFS.

The NFS is currently in version 4. Version 4 of NFS added several new features into the mix. One of the small changes is now the file handles used to communicate between the client and server are encoded in 8 bit unicode (UTF-8) as an attempt to handle some internationalization[]. However, one of the larger changes are new operations, a couple of which are called OPEN and CLOSE. The OPEN operation is especially interesting, one because it encapsulates several individual operations needed in previous versions of NFS[], but also because it introduces state to the NFS protocol. Version 3 and earlier of NFS were stateless protocols which meant that no special information was retained about each client by the server, which makes recovering from an error or power outage simple and quick. However, the OPEN operation causes stateful information to be kept about a client that has opened a file and which then has either a read delegation or a write delegation from the server to mandate how that client can treat that file. This state information is discarded when the CLOSE operation is run and the delegations given to the client are revoked[].

Also in contrast to earlier versions of the protocol is the ability of NFS to use access list controls (ACLs). Much like the ones mentioned in the discussion of the AFS, these allow for the users accessing files through the clients to be controlled in their file modification[]. This is especially important when the two local file system which are interacting through NFS do not share the same security model, such as is the case with the ext family of file systems and the NT File System.

6. CONCLUSION

The file systems of the world exist to help we humans to find access information in a way meaningful to us, even when the words we read are actuality bits strewn about a storage device. These systems started out rather primitively, and the commonly used File Allocation Table (FAT) is a rather primitive system, and as we have seen here the idea of what features can and should be provided have grown immensely. Fortunately, not only have the ideas grown, but also the implementation to match the thrilling ideas. As wonderful as it would be just to stay where we are technologically, and play with the great implementations before us, we are facing a world of great change. Like always, our technology will need to grow to meet that change, and I for one, look forward to what the future holds.

7. REFERENCES

[1] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck “Network File System (NFS) version 4 Protocol" RFC 3530, April 2003. [Online]. Available: http://www.faqs.org/rfcs/rfc3530.html

[2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles* (SOSP '03). ACM, New York, NY, USA, 29-43. DOI=10.1145/945445.945450 http://doi.acm.org/10.1145/945445.945450

[3] Howard, J.H. Feb. 1988. An Overview of the Andrew File System. In *Proceedings of the USENIX Winter Technical Conference.* Dallas, TX

[4] NFS Overview Retrieved November 17, 2010 from A Performance Comparison of NFS and iSCSI for IP-Networked Storage http://www.usenix.org/events/fast04/tech/full_papers/radkov/radkov_html/node3.html

[5] Jeff Bonwick, Bill Moore. ZFS The Last Word in File Systems. From OpenSolaris ZFS Community Group http://hub.opensolaris.org/bin/view/Community+Group+zfs/docs/zflast.pdf