ISSN:0975-9646

A Survey of File Systems and File Access Systems

R.Prathyusha[1], G.Praveen Babu[2]

[1]*M.Tech.(Comp.Networks & Info.Security) Student, School of IT, JNTUH, India*
[2] *Associate Professor of CSE, School of IT, JNTUH, India,*

I. INTRODUCTION

Remote access file systems enable an application that runs on a client computer to access files stored on a different computer. Remote file systems also often make other resources (remote printers, for example) accessible from a client computer. The remote file and resource access takes place using some form of Local Area Network (LAN), Wide Area Network (WAN), point-to-point link, or other communication mechanism. These file systems are often referred as network file systems. The components that make a files access system include file systems, file access protocol used to access the remote file system and an identity provider that contains the identities of all the clients accessing the file system.
This paper is a survey of the current state of the art in the design of file systems and file access protocols. It consists of four major parts: a brief survey of essential features of file systems, file access protocols and case studies of a number of contemporary remote file systems.

II. ESSENTAIL FEATURES OF FILE SYSTEMS

A. Space-Management

Space Management is one of the essential features of file system. File systems allocate space in a granular manner, usually multiple physical units on the device. The file system is responsible for organizing files and directories, and keeping track of which areas of the media belong to which file and which are not being used. File system fragmentation occurs when unused space or single files are not contiguous. As a file system is used, files are created, modified and deleted. When a file is created the file system allocates space for the data. Some file systems permit or require specifying an initial space allocation and subsequent incremental allocations as the file grows. As files are deleted the space that was allocated, eventually is considered available for use by other files. This creates alternating of used and unused areas of various sizes. This is free space fragmentation. When a file is created and there is not an area of contiguous space available for its initial allocation, the space must be assigned in fragments. When a file is modified such that it becomes larger, it may exceed the space initially allocated to it, another allocation must be assigned elsewhere and the file becomes fragmented.

B. Integrity

One significant responsibility of a file system is to ensure that, regardless of the actions by programs accessing the data, the structure remains consistent. This includes actions taken if a program modifying data terminates abnormally or neglects to inform the file system that it has completed its activities. This may include updating the metadata, the directory entry and handling any data that was buffered but not yet updated on the physical storage media. Other failures which the file system must deal with include media failures or loss of connection to remote systems. In the event of an operating system failure or "soft" power failure, special routines in the file system must be invoked similar to when an individual program fails. The file system must also be able to correct damaged structures. These may occur as a result of an operating system failure for which the OS was unable to notify the file system, power failure or reset. The file system must also record events to allow analysis of systemic issues as well as problems with specific files or directories.

C. MetaData

The length of the data contained in a file may be stored as the number of blocks allocated for the file or as a byte count. The time that the file was last modified may be stored as the file's timestamp. File systems might store the file creation time, the time it was last accessed, the time the file's metadata was changed, or the time the file was last backed up. Other information can include the file's device type (e.g. block, character, socket, subdirectory, etc.), its owner user ID and group ID, its access permissions and other file attributes (e.g. whether the file is read-only, executable,
A file system stores all the metadata associated with the file—including the file name, the length of the contents of a file, and the location of the file in the folder hierarchy—separate from the contents of the file.
Most file systems store the names of all the files in one directory in one place—the directory table for that directory—which is often stored like any other file.

D. Access Based Enumeration

Access-based enumeration displays only the files and folders that a user has permissions to access. It is a feature that was previously available as a downloadable package for the Windows Server® 2003 operating system (it was also included in Windows Server 2003 Service Pack 1). Access-based enumeration is now included in the Windows Server 2008 operating system, and you can enable it by using Share and Storage Management.There are several mechanisms used by file systems to control access to data. Usually the intent is to prevent reading or modifying files by a user or group of users. Another reason is to ensure data is modified in a controlled way so access may be restricted to a specific program. Examples include passwords stored in the metadata of the file or elsewhere and file permissions in the form of permission bits, access control lists, or capabilities. The need for file system utilities to be able to access the data at the media level to reorganize the structures and provide efficient backup usually means that these are only effective for polite users but are not effective against intruders.

E. Special features:

Each file system has some special features which are characteristic to it, which this paper describes and higlights as part of essential features itself.

III. Essential Features Of File Access Systems

A. Transparency

The distributed systems should be perceived as a single entity by the users or the application programmers rather than as a collection of autonomous systems, which are cooperating. The users should be unaware of where the services are located and also the transferring from a local machine to a remote one should also be transparent. Some of the different transparencies include Access Transparency, Location Transparency, Concurrency Transparency, Replication Transparency.

B. User Mobility

A user can walk to any workstation in the system and access any file in the shared name space. A user's workstation is personal only in the sense that he owns it. The file system must be independent of Client Operating System i.e. should support different network protocols that can be used for different clients. In addition it must support different types of Client devices.

C. Performance

Performance is measured as the average amount of time needed to satisfy client requests. This time includes CPU time + time for accessing secondary storage + network access time. It is desirable that the performance of a distributed file system be comparable to that of a centralized file system. The following methodologies increase the performance of the remote file system include Server Caching, Client Caching, and Multi-threading.

D. Security

Secure Authentication where the server and client interacting identify and trust each other that either of them are legitimate entities. There are a number of authentication methods like Shared Secret, OTP (One Time Password), Certificates, etc.

After a secure authentication session is established, the server and client exchange messages with a shared cipher that is agreed between the client and server during the Authentication session.

E. Visibility

Visibility is an essential file system feature by which the users will not know the existence of other files and folders inside a directory unless they are authorized to access them. For example, if there is a directory dir1 and there are folders user1 and user2 to which user1 and user2 have respectively access to, then when a user1 accesses dir1 through his client, he will be able to see only user1 and not user2.Similar is the case for user2 folder. This is a rare feature that is not implemented by all file systems.

IV. A Study Of File Systems

A. Ext4

1. Space Management: The ext4 file system can support volumes with sizes up to 1 exbibyte (EiB) and files with sizes up to 16 tebibytes (TiB). Delayed allocation is a performance feature (it doesn't change the disk format) found in a few modern filesystem and it consists in delaying the allocation of blocks as much as possible, contrary to what traditionally filesystems (such as Ext3, reiser3, etc) do: allocate the blocks as soon as possibleDelayed allocation, does not allocate the blocks immediately when the process write()s, rather, it delays the allocation of the blocks while the file is kept in cache, until it is really going to be written to the disk. This gives the block allocator the opportunity to optimize the allocation in situations where the old system couldn't. Delayed allocation plays very nicely with the two previous features mentioned, extents and multiblock allocation, because in many workloads when the file is written finally to the disk it will be allocated in extents whose block allocation is done with the malloc allocator. The performance is much better, and the fragmentation is much improved in some workloads.

2. Integrity: The journal is the most used part of the disk, making the blocks that form part of it more prone to hardware failure. And recovering from a corrupted journal can lead to massive corruption. Ext4 checksums the journal data to know if the journal blocks are failing or corrupted. But journal checksumming has a bonus: it allows one to convert the two-phase commit system of Ext3's journaling to a single phase, speeding the filesystem operation up to 20% in some cases - so reliability and performance are improved at the same time. (Note: the part of the feature that improves the performance, the asynchronous logging, is turned off by default for now, and will be enabled in future releases, when its reliability improves. Jourrnaling ensures the integrity of the file system by keeping a log of the ongoing disk changes. However, it is known to have a small overhead. Some people with special requirements and workloads can run without a journal and its integrity advantages. In Ext4 the journaling feature can be disabled, which provides a small performance improvement.

Write Barriers, a new feature, ensures that file system integrity even when power is lost to a device with write caches enabled.

3. Metadata: Timestamps in the extended file system arena prior to ext4 were seconds based. This was satisfactory in many settings, but as processors evolve with higher speeds and greater integration (multi-core processors) and Linux finds itself in other application domains such as high-performance computing, the seconds-based timestamp fails in its own simplicity. Ext4 has essentially future-proofed timestamps by extending them into a nanosecond LSB. The time range has also be extended with two additional bits to increase the lifetime by another 500 years.

3. Access Based Enumeration - This feature is not implemented in ext4 file system.

B. BrtFS

1. Space Management: BrtFS supports up to 16EiB. It supports a unique feature called Copy-on write to reduce the space allocation of a resource that is used by multiple processes. Copy-on-write (sometimes referred to as "COW") is an optimization strategy used in computer programming. Copy-on-write stems from the understanding that when multiple separate tasks use identical copies of the same information (i.e., data stored in computer memory or disk storage), it is not necessary to create separate copies of that information for each process, instead they can all be given pointers to the same resource. When there are many separate processes all using the same resource it is possible to make significant resource savings by sharing resources this way. However, when a local copy has been modified, the copy-on-write paradigm has no provision that the shared resource has in the meantime not been updated by another task or tasks. Copy-on-write is therefore amenable if only the latest update is important and occasional use of a slightly stale value is not harmful.

2. Integrity: Integrity in BrtFS is achieved through a number of methods like Checksums on data and metadata, Online data scrubbing for finding errors and automatically fixing them for files with redundant copies and Snapshots. BTRFS's snapshotting capability is to atomically freeze changes to a subvolume, and then transfer these snapshots to a backup volume. The atomic freezing provided by BTRFS should allow transactionally operating systems such as database files and raw file system images to be repairable to a consistent state from a snapshot. When performing incremental backups, we will be working with two snapshots, one of them representing the time of the earlier backup, and the other representing the current backup. When the backup run completes, we can discard the earlier of the snapshots to prepare for the next incremental run.

3. Metadata-Timestamps in the extended file system arena prior to ext4 were seconds based. BrtFS has essentially future-proofed timestamps by extending them into a nanosecond LSB. The time range has also be extended with two additional bits to increase the lifetime by another 500 years.

4. Access-Based Enumeration-This feature is not implemented in BrtFS file system.

C. ReFS

1. Space Management - ReFS is designed to work well with extremely large data sets — petabytes and larger — without performance impact. ReFS is not only designed to support volume sizes of 2^64 bytes (allowed by Windows stack addresses), but ReFS is also designed to support even larger volume sizes of up to 2^78 bytes using 16 KB cluster sizes. This format also supports 2^64-1 byte file sizes, 2^64 files in a directory, and the same number of directories in a volume. The first new concept for storage management in Windows Server 2012 is to Storage Pool: a disk group, even of different types of them that, once collected in a pool, they are "virtualized" and made available as a unique resource for the creation of a Storage Space accessible by the system.The Storage Pool once created can be extended by adding additional disks, thus extending the potential size of the Storage Spaces that use it.

2. Integrity: ReFS stores data in a way that protects it from many of the common errors that can normally cause data loss. When ReFS is used in conjunction with a mirror space or a parity space, detected corruption—both metadata and user data, when integrity streams are enabled—can be automatically repaired using the alternate copy provided by Storage Spaces. In addition, there are Windows PowerShell cmdlets (Get-FileIntegrity and Set-FileIntegrity) that you can use to manage the integrity and disk scrubbing policies.

3. MetaData-A reliability mechanism that increases the performance of ReFS is "integrity streams". Just like metadata, integrity streams will use allocate-on-write semantics to reduce the chance that a failure while writing to disk will result in corruption of the only good copy of the file's content. Integrity streams are not appropriate for all types of files; applications that require control over the physical file structure (e.g. databases) should disable this feature.

4. Access Based Enumeration-ReFS provides access-based enumeration but this feature is provided as optional. So the end users can see only the files and folders they have access to.

D. ZFS

1. Space Management : ZFS supports upto 16EB of data.ZFS introduces a new concept called Zvols.These are like special objects in a ZFS pool of storage and can be very useful for storage virtualization because you can snapshot them and present them to other systems as block devices through protocols like iSCSI, FCoE/FC and Infiniband when used with a SCSI target framework like SCST or LIO. As a storage appliance / SDS software developer, this is a really cool feature though it may not be as important for general use case - *2. Integrity*, Data integrity is achieved by using a (Fletcher-based) checksum or a (SHA-256) hash throughout the file system tree. Each block of data is checksummed and the checksum value is then saved in the pointer to that block—rather than at the actual block itself. Next, the block pointer is checksummed, with the value being saved at its pointer. This checksumming continues all the way up the file system's data hierarchy to the root node, which is also checksummed, thus creating a Merkle tree In-flight data corruption or phantom reads/writes (the data written/read checksums correctly but is actually wrong) are undetectable by most filesystems as they store the checksum with the data. ZFS stores the checksum of each block in its parent block pointer so the entire pool self-validates.

3. Access Based Enumeration: This feature is not supported by ZFS.

V. A STUDY OF FILE ACCESS SYSTEMS

A. *NFS* **-** Network File System

1. *Security* - NFS v3, unlike the other file access protocols, is an exported file system. This means that access and security are enforced at the NFS client, and not the NFS server. As a result, NFS is easily hacked if not on a dedicated secure network. NFS v3 is a stateless protocol like HTTP and FTP, so suffers performance since it must assert current state with each operation (for example, it does not define Open and Close file, only Read and Write). However NFSv4 has authentication mandated, but is still not widely spread.
2. *Transparency* - NFS provides only Location Transparency, i.e. the location of the known from the file. But NFS does not provide Location independence, i.e. the physical storage location cannot be changed without changing the location of the file.
3. *User-Mobility* - NFS can be used by all Unix-operating system clients and some non-Unix clients.
4. *Performance* - To improve the performance, NFS systems use delayed write. But they don't free delayed written block until the server confirms that the data have been written on disk. So, here, Unix semantics are not preserved. NFS does not handle client crash recovery like Unix. Since, servers in NFS are stateless; there is no need to handle server crash recovery also. To improve the performance, the NFS systems cache the file/directory attributes. For large caches, bigger block-sizes are beneficial.

B. CIFS (Common Internet File System)

1. *Security* - CIFS provides security optionally like NFS. CIFS servers support both anonymous transfers and secure, authenticated access to named files. File and directory security policies are easy to administer. However, in data communication CIFS does not provide encryption capabilities.
2. *Transparency* - CIFS provides location transparency, i.e users do not have to mount remote file systems, but can refer to them directly with globally significant names (names that can be located anywhere on the Internet), instead of ones that have only local significance (on a local computer or LAN). Distributed File Systems (DFS) allows users to construct an enterprise-wide namespace. Uniform Naming Convention (UNC) file names are supported so a drive letter does not need to be created before remote files can be accessed.
3. *User-Mobility* - CIFS can be used only on windows clients.
4. *Performance* - CIFS servers are highly integrated with the operating system, and are tuned for maximum system performance.

C. *Lustre* **-** Lustre is a unique distributed client server protocol. It specifically breaks the functions of a file system up at the protocol layer in order to gain huge scalability for great numbers and very large files (like seismic data for petroleum exploration).

1. *Security* - Lustre implements process authorizations groups as they provide more security from root setuid attacks, provided hardened kernels are used. New features of Lustre are file encryption, careful analysis of cross realm authentication and authorization issues and file I/O authorization. Lustre can also be configured to use /etc/password,/etc/group in environments where small clusters can be configured.
2. *Transparency* - Supports Linux Systems. Lustre files can be re-exported using NFS or CIFS (via Samba) enabling them to be shared with a non-Linux client.
3. *User-Mobilty* - Lustre supports only Unix Platforms.
4. *Performance* - There are few features which increase the performance like a robust failover and recovery mechanism, making server failures and reboots transparent. Lustre uses a modified version of the ext4 journaling file system to store data and metadata. This version, called ldiskfs, has been enhanced to improve performance and provide additional functionality needed by Lustre. Version interoperability between successive minor versions of the Lustre software enables a server to be upgraded by taking it offline (or failing it over to a standby server), performing the upgrade, and restarting it, while all active jobs continue to run, experiencing a delay while the backup server takes over the storage.

D. AFS (Andrew File System)

1. *Security* **-** AFS uses Kerberos for authentication, and implements access control lists on directories for users and groups. Each client caches files on the local file system for increased speed on subsequent requests for the same file. This also allows limited file system access in the event of a server crash or a network outage.

2. *Transparency* - Users of the data are unaware of the location of the read-only copy; administrators can create and relocate such copies as needed. The AFS command suite guarantees that all read-only volumes contain exact copies of the original read-write volume at the time the read-only copy was created.

4.3. *Performance* - Andrew file system is more scalable when the file reads are 6 times greater than file writes, files are referenced in bursts, and sequential access is more common than random access.

3. *User-Mobility* - Andrew File System supports heterogeneous systems like Linux, Mac and Microsoft Windows.

VI. CONCLUSION

The above file systems and file access systems presented are a brief study of what are the essential features required for an ideal file system and file access systems.

REFERENCES

1. http://en.wikipedia.org/
2. http://www.lasr.cs.ucla.edu/classes/cs111_online.spring13/readings/distributed_FS_survey.pdf
3. http://crystal.uta.edu/~kumar/cse6306/papers/mantena.pdf
4. http://wiki.lustre.org/index.php/FAQ_-_OS_Support
5. http://www.unf.edu/~sahuja/cis6302/filesystems.html