A 64-bit, Scalable File System for Storage Area Networks

GYOUNG-BAE KIM, CHANG-SOO KIM, BUM-JOO SHIN
Internet Service Department
Computer and Software Technology Labs.
ETRI(Electronics and Telecommunications Research Institute)
161 Kajong-Dong, Yusong-Gu, Taejon, 305-350
KOREA

Abstract: - By merging network and channel interfaces, resulting interfaces allow multiple computers to physically share the storage devices in storage area network (SAN). In SAN, computers service local file requests directly from shared storage devices. Direct device access eliminates the server machines as bottlenecks to performance and availability. Communication is unnecessary between computers, since each machine views the storage as being locally attached. SAN provides us to very large physical storage up to 64-bit address space, but traditional file systems can't adapt to the file system for SAN because they have the limitation of scalability. In this paper, we describe the architecture and design features of a new scalable file system, SANtopia, which is being developed at ETRI as global shared cluster file system for 64-bit address space file in the SAN environment. SANtopia manages to deal in large file, large directories, and the large number of files using new metadata structure. SANtopia will perform well as a local file system as a traditional network file system, and as a high performance cluster file system running over SAN. SANtopia provides a key cluster enabling technology for Linux, helping to bring the availability, scalability, and load balancing benefits of clustering to Linux.

Key-Words: - Global File System, Storage Area Network, Cluster File System, Linux, SANtopia

1 Introduction

The growth of Internet allows increasing number of users to access vast amount of information stored at shared disks. However, long propagation delays between clients and servers, as well as hot spots of network and server load yield high latencies for data access in traditional storage system.

Recently, the advances of switching technology are allowed to order-of-magnitude improvements in the network latency and bandwidth through new technologies like Fibre Channel[1]. The Fibre Channel standard integrates both storages and networking capabilities into a single serial interface. Hundreds of Fibre Channel disks and host computers are able to combine in the shared-bandwidth loops or across switches, which is capable of maintaining several simultaneous gigabit transfers. This network-like connection among the disk drives and hosts has prompted a shift in the way storage systems are viewed. In contrast, today's parallel SCSI technology supports only about 8 devices per bus with each bus extending at most 25m making the technology effectively unscalable[2].

Storage Area Network (SAN)[3] is the combination of network attached FC storage devices and computers with the FC network adapters on a loop or fabric. Each computer accesses to all the drives effectively. SAN eliminates the bandwidth bottlenecks and the limitations of scalability imposed by the previous SCSI-based architectures and LAN connections between server and stored data.

SAN provides us to very large physical storage up to 64-bit address space. We needed a file system that could manage even petabytes of storage, but all of the file systems we know of are limited to either a few gigabytes or a few terabytes in size. These limitations stem from the use of data structures that don't scale and the use of 32 bit block pointers throughout the on-disk structures of the file system.

In this paper we introduce a new scalable file system, called SANtopia[4], for SAN in Linux. SANtopia is being developed at ETRI as the global shared cluster file system for 64-bits address space file in the SAN environment. SANtopia manages to deal in the large file, large directories, and large numbers of files using new metadata structure. SANtopia uses a new file system layout, which supports us to a 64-bits address space for large file system and modifies inode and directory structure for large file and directory. For fast recovery in file system failure, SANtopia adapts to metadata journaling. SANtopia will perform well as a local file system as a traditional network file system and as a high performance cluster file system running over SAN. SANtopia provides a key cluster enabling technology for Linux, helping to bring the availability, scalability, and load balancing benefits of clustering to Linux.

The rest of the paper is organized as follows: Section 2 reviews related works in the area of distributed file system for huge file and other SAN file systems. Section 3 describes the problems of huge global file system. Section 4 describes the SANtopia architecture and each component. In Section 5, we present the techniques of SANtopia system for globally shared file system in detail. For supporting the large file system, large directories, and huge file, we design a new file system layout, inode and directory structures. We conclude the paper in Section 6.

2 Related Works

2.1 Global File System (GFS)

GFS[5] was developed in Minnesota for shared file system in 1995. At that time, GFS was primarily interested in exploiting the Fibre Channel technology to post-process large scientific datasets on Silicon Graphics (SGI) hardware. By the spring of 1998, GFS began porting their code to the open source Linux operating system. In addition, GFS had shed our narrow focus on large data applications and had broadened our efforts to design a general-purpose file system that scaled from a single desktop machine to a large, heterogeneous network enabled for device sharing. Because they had kernel source GFS could finally support metadata and file data caching, but this required changes to the lock specification, detailed in [6].

2.2 Frangipani

Frangipani [7] is a new scalable distributed file system that manages a collection of disks on multiple machines as a single shared pool of storage. The machines are assumed to be under a common administration and to be able to communicate securely. One distinguish feature of Frangipani is that it has a very simple internal structure-a set of cooperating machines use a common store and synchronize access to that store with locks. This simple structure enables us to handle system recovery, reconfiguration, and load balancing with very little machinery.

Frangipani is layered on top of Petal[8], an easy-to-administer distributed storage system that provides virtual disks to its clients. Like a physical disk, a Petal virtual disk provides storage that can be read or written in blocks. Unlike physical disk, a virtual disk provides a sparse 2^{64} bytes address space, with physical storage allocated only on demand. Petal also provides efficient snapshots to support consistent backup. Frangipani inherits much of its scalability, fault tolerance, and easy administration from the underlying storage system, but careful design was required to extend these properties to the file system level.

2.3 XFS

XFS[9] is the next generation local file system for SGI's workstations and servers. It is a general purpose Unix file system that runs on workstations with 16 megabytes of memory and a single disk drive and also on large SMP network servers with gigabytes of memory and terabytes of disk capacity. In this paper we describe the XFS file system with a focus on the mechanisms it uses to manage large file systems on large computer systems.

The most notable mechanism used by XFS to increase the scalability of the file system is the pervasive use of B+ trees[10]. B+ trees are used for tracking free extents in the file system rather than bitmaps. B+ trees are used to index directory entries rather than using linear lookup structures. B+ trees are used to manage file extent maps that overflow the number of direct pointers kept in the inodes. Finally, B+ trees are used to keep track of dynamically allocated inodes scattered throughout the file system. In addition, XFS uses an asynchronous write ahead logging scheme for protecting complex metadata updates and allowing fast file system recovery after a crash. We also support very high throughput file I/O using large, parallel I/O requests and DMA to transfer data directly between user buffers and the underlying disk drives. These mechanisms allow us to recover even very large file systems after a crash in typically less than 15 seconds, to manage very large file systems efficiently, and to perform file I/O at hardware speeds that can exceed 300 MB/sec.

3 Problems of Scalable File System

In designing and developing file system for huge file, we focused in on the specific problems with the traditional file systems that we felt we needed to address. In this section we consider the several problems of the specific scalability addressed in the design of SANtopia and why the mechanisms used in other file systems are not sufficient.

3.1 To Support Large File System

We needed a file system that could manage even petabytes of storage, but all of the file systems we know of are limited to either a few gigabytes or a few terabytes in size. Traditional file systems are limited to only 8 gigabytes in size. These limitations stem from the use of data structures that don't scale, for example the bitmap in EFS, and from the use of 32 bit block pointers throughout the on-disk structures of the file system. The 32 bit block pointers can address at most 4 billion blocks, so even with an 8 KB block size the file system is limited to a theoretical maximum of 32 terabytes in size

3.2 To Support Large File System

None of the file systems we looked at support full a 64-bits files. Traditional file system did not support sparse files at all. Most others use the block mapping scheme created for FFS. We decided early on that we would manage space in files with variable length extents (which we will describe later), and the FFS style scheme does not work with variable length extents. Entries in the FFS block map point to individual blocks in the file, and up to three levels of indirect blocks can be used to track blocks throughout the file. This scheme requires that all entries in the map point to extents of the same size. This is because it does not store the offset of each entry in the map with the entry, and thus forces each entry to be in a fixed location in the tree so that it can be found. Also, a 64-bits file address space cannot be supported at all without adding more levels of indirection to the FFS block map.

3.3 To Support Large Directories

Another area, which has not been addressed by other Unix file systems or Linux, is support for directories with more than a few thousand entries. While some, for example VxFS[11], at least speed up searching for entries within a directory block via hashing, most file systems use directory structures, which require a linear scan of the directory blocks in searching for a particular file. The lookup and update performance of these unindexed formats degrades linearly with the size of the directory. Others use in-memory hashing schemes layered over simple on-disk structures. These in memory schemes work well to a point, but in very large directories they require a large amount of memory. This problem has been addressed in some non-Unix file systems, like NTFS, by using B trees to index the entries in the directory.

3.4 To Support Large Number of File

While traditional file systems can theoretically support very large numbers of files in a file system, in practice they do not. The reason is that the number of inodes allocated in these file systems is fixed at the time the file system is created. Choosing a very large number of inodes up front wastes the space allocated to those inodes when they are not actually used. The real number of files that will reside in a file system is rarely known at the time the file system is created. Being forced to choose makes the management of large file systems more difficult than it should be. VxFS solve this problem by allowing the number of inodes in the file system to be increased dynamically. In summary, there are several problems with traditional file systems that we wanted to address in the design of SANtopia. While these problems may not have been important in the past, we believe the rules of file system design have changed.

3.5 To Support Fast Recovery

A file system with a crash recovery procedure that is dependent on the file system size cannot be practically used on large systems, because the data on the system is unavailable for an unacceptably long period after a crash. EFS and file systems based on the BSD Fast File System falter in this area due to their dependence on a file system scavenger program to restore the file system to a consistent state after a crash. Running fsck over an 8 gigabyte file system with a few hundred thousand inodes today takes a few minutes. This is already too slow to satisfy modern availability requirements, and the time it takes to recover in this way only gets worse when applied to larger file systems with more files. Most recently designed file systems apply database recovery techniques to their metadata recovery procedure to avoid this pitfall.

4 Architecture of SANtopia

Figure 1 shows the architecture of SANtopia, which supports the global file sharing in SAN environments. SANtopia consists of the four parts: the global file manager, the system manager, the global buffer and lock manager, and the logical volume manager.

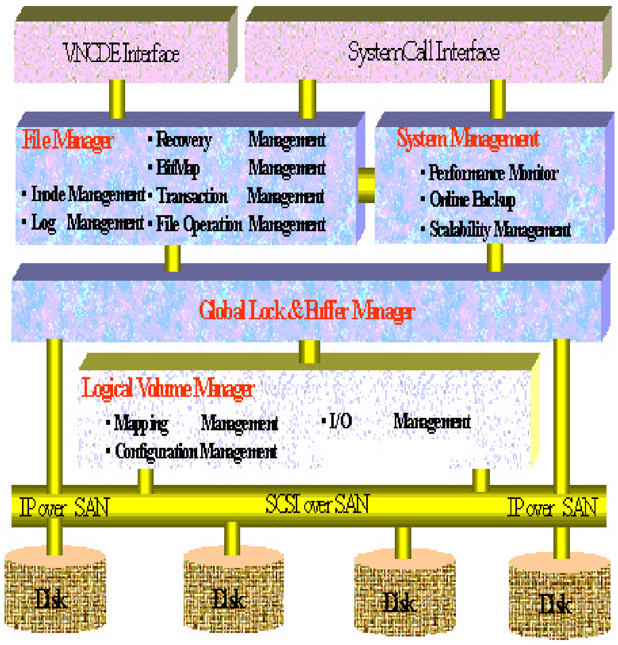

Figure 1. Architecture of SANtopia

The users require the operation of the SANtopia file system using VNODE operation and system call in VFS (Virtual File System). The global file manager of SANtopia processes metadata structures and its operation is for functions having a large volume of file sharing. Each operation is defined as a transaction that is a unit of file operation and recovery.

The system manager provides performance monitoring, on-line backup, and the system configuration of SANtopia. Using the system manager tools, the system manager is able to attach or detach some nodes, and add or delete the shared disks for a logical volume.

The global lock and buffer manager fulfills the function of the global buffer sharing to minimize the disk I/O. Especially, to reduce communication overhead between the local server and global server for buffer coherency, we integrated the lock manager, which controls file access, and the buffer manager, which provides file sharing, in SANtopia.

The logical volume manager enables the file manager to coalesce a large logical volume (storage pool) into a heterogeneous collection of shared storages. This manager has the several functions in LVM, 1) addition of a physical disk into logical volume, 2) software RAID, 3) mapping the logical address into the physical address, 4) transfer of the data page form the disk into the buffer.

5 Development of Scalable File System

The SANtopia file system has been designed with many improvements over the traditional file systems, such as the Unix or Linux file systems. There are a number of SAN file systems available today, with two prevailing architectures. One approach is to use a distributed lock manager and essentially treat the SAN node as a tight cluster, similar to a VAX cluster. GFS is a popular choice from that category, due to the fact that it is freely available on Linux. However, the cluster approach tends to be proprietary and homogeneous.

The alterative is a split data-metadata architecture. This architecture uses a single server per logical file system as the coordinator of all metadata traffic. File system metadata contains information such as the files name, allocation information, security and ownership specifications, and other information about the file. This information is typically very small and easily communicated over a LAN between peers. Using a LAN allows the benefit of presenting the storage as a network volume, so that most heterogeneous operability issues are eliminated.

5.1 Layout of SANtopia for Large File System

In the traditional file system, it divides the allocation area into inodes, directories, inode bit maps, and data blocks. The number of inode bit map entries for each inode is allocated one entry per 4KB data blocks in the Linux file system. This causes serious problems since the inode table will be full, in spite of the fact that the file system has free space, when a huge number of huge small files exist in file system.

For supporting large file systems, SANtopia uses 64-bit address space. In figure 2, the layout of the SANtopia file system consists of a boot area, super block, allocation area, and bitmap area. The boot area stores the booting images for the operating system. The super block stores the information on the file system such as the total size of the file system, block size, bitmap size, and several traditional super block data.

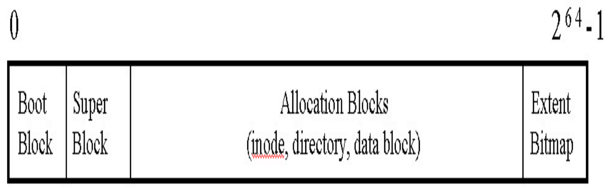

Figure. 2 file system layout for a 64-bit address

Therefore, there is no division in the allocation area among inodes, directories, and data blocks. Since information on the allocation area manages the bitmap table using two bits, we assigned the value of bit map as follows: 00 is a free allocation blocks (extent), 01 is an inode, 10 is a directory entry, and 11 is a data extent. Using this method, there is no limitation in the number of inode table entries.

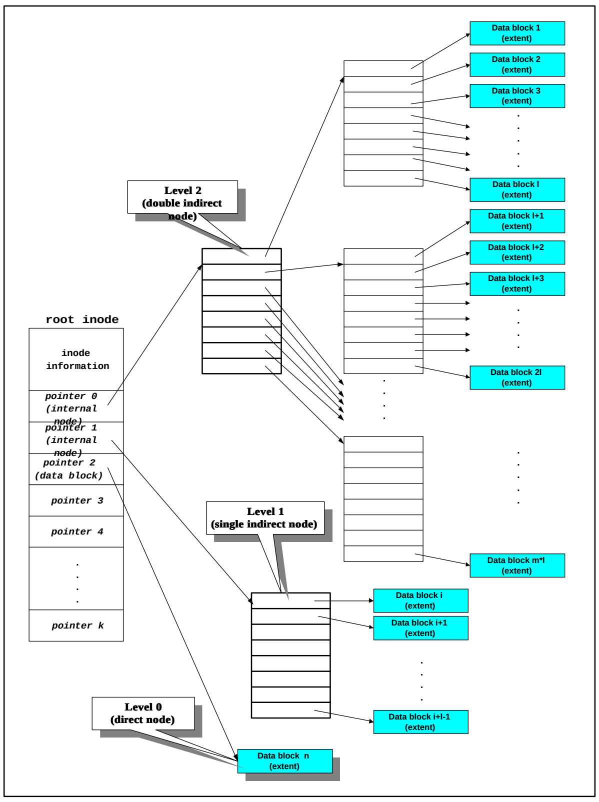

Figure 3. Multi-Level inode structure for large file

5.2 Multi-Level inode Structure for Large Files

SANtopia provides a 64-bits address space for each file. The support for 64 bits files means that there are potentially a very large number of blocks to be indexed for every file. In order to keep the number of entries in the file allocation map small, SANtopia uses an extent map rather than a block map for each file.

In the SANtopia file system, we designed a new inode scheme, called MLI (Multi-Level Inode) for huge sized file. The level of MLI is dynamically up when file size was increased and there is no a free pointer in mid-level. Figure 3 shows our MLI structure.

5.3 Directory management for large directories and large numbers of file

One serious problem encountered when we adapted traditional file systems to the SAN environment was that they did not perform well with large directories. Most traditional file systems such as the Unix file system (UFS) and Linux file system (Ext2) were designed for small sized file systems. So they have a limitation in the number of files possible in the directory. Also, they store files in the directory as an unsorted linear list of directory entries. This is satisfactory for small directories, but it greatly increases the directory operations time for larger ones. On average, we must search through half of the directory to find any random entry presents in the directory. In the traditional directory structure, large directories can take up megabytes of disk space, so this is costly not only in terms of the CUP time but also I/O time. Two alternatives are proposed for large directory. One approach is using a B-tree for entries in the directory, such as in the xFS file system. The other approach is Extensible Hashing [12], such as GFS.

The SANtopia file system is adapted to Extensible Hashing due to its directory structure. Extensible Hashing provides a way of storing a directory entry so that any one entry can be found very quickly. However, since the basic Extensible Hashing method is not designed for large directory, we modified its traditional data structure for an extent-based large directory and refer to it as an Extent-based Multi-Level Extensible Hashing (EBMLEH).

Figure 4 shows our EBMLEH structure.

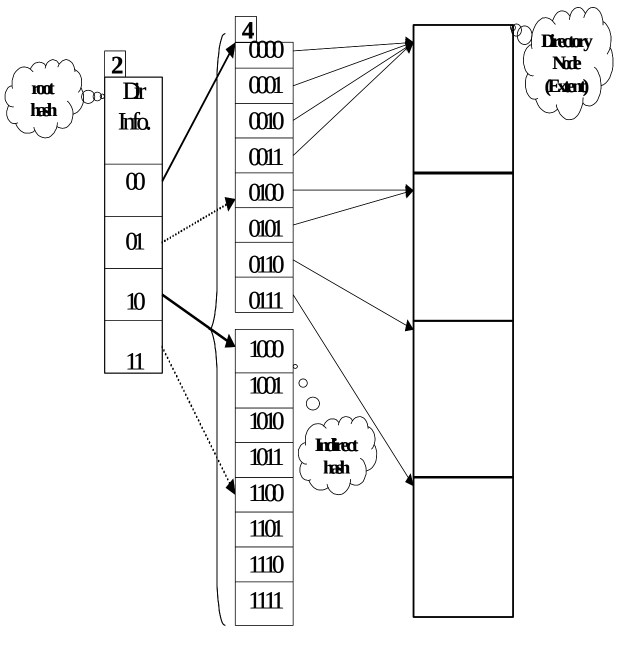

Figure 4. Extent-Based Multi-Level EH

The EBMLEH uses a multi-bit hash of each filename. A subset of the bits is used as a unique index in a hash table. Each pointer of the leaf hash table points to a leaf extent block that contains the directory entries. The EBMLEH has multi-level hashing structures that contain a root node and indirect nodes that are sets of hash values and block pointers in the hashing tables. Each node has an indicator that represents the bit size of the hash value to be compared in the hash table.

The small size directory only uses root hash nodes, the pointer of which points directly to the directory entries block. However, in a large directory, the capacity of the root node will reach its limitation when the number of directory entries keeps increasing. The EBMLEH uses indirect hash nodes in that case. In Figure 4, for example, if the capacity of the root node is 4 and indicator of the hash bits is 2 in the root node, the number of directory entries has already exceeded its limitation, so that the root node has 4 indirect hash nodes. The root node points to the indirect hash node and each indirect hash nodes point to a real block that stores directory entries. Two steps are needed to find any particular directory entry in figure 4. The number of directory entries that can be pointed to in the directory blocks is increased rapidly when the root node splits into indirect hash nodes in EBMLEH.

5.4 Metadata Journaling for Fast Recovery

Metadata journaling provides the fast restart of file system in the event of a system crash. Using database journaling techniques, SANtopia can restore a file system to a consistent state in a matter of seconds or minutes, versus hours or days with non-journaled file systems such as HPFS, ext2, and traditional UNIX file systems. In the event of system failure, these file systems rely on restart-time utilities (that is, fsck), which examine all of the file system's meta-data (such as directories and disk addressing structures) to detect and repair structural integrity problems. This is serious problem in SAN file system, because it is a time-consuming and error-prone process, which, in the worst case, can lose or misplace data.

So, the journaling of SANtopia provides a log-based, byte-level file system that was developed for transaction-oriented, high performance systems. Scalable and robust, its advantage over non-journaled file systems is its quick restart capability. SANtopia uses techniques originally developed for databases to log information about operations performed on the file system meta-data as atomic transactions. In the event of a system failure, a file system is restored to a consistent state by replaying the log and applying log records for the appropriate transactions. The recovery time associated with this log-based approach is much faster since the replay utility need only examine the log records produced by recent file system activity rather than examine all file system metadata.

6 Conclusion

In this paper, we present the architectural and design features of SANtopia, which allows multiple machines to access and share disk and tape devices on a Fibre Channel or SCSI storage network in a Linux system. It will perform well as a local file system, as a traditional network file system running over IP environments, and as a high performance cluster file system running over storage area networks. SANtopia provides a key cluster enabling technology for Linux, helping to bring the availability, scalability, and load balancing benefits of clustering to Linux.

The mechanisms in SANtopia for satisfying the requirements of huge file systems also make it a high performance general-purpose file system. The use of dynamic multi-level inode structure and extent-based extensible hashing throughout the file system efficiently manages very large files and large directories in the file system. Using metadata journaling, SANtopia eliminates many of the metadata update performance problems in the traditional file systems.

References:

[1] Alan F. Benner. Fibre Channel: Gigabit Communications and I/O for Computer Network. McGraw-Hill, 1996.

[2] Friedhelm Schmidt. The SCSI Bus & IDE Interface. Addison-Wesley, second edition, 1998.

[3] Randy H. Katz, “High-Performance Network and Channel Based Storage”, Proceedings of IEEE, Vol.80, No.8, pp.1238-1261, 1992.

[4] G. B. Kim, C. S. Kim, and B. J. Shin, “Global File Sharing System for SAN”, The 3rd Int. Conf. On Advanced Communication Technology, pp.870-874, Feb. 2001.

[5] Kenneth W. et al., “Implementation Journaling in a Linux Shared Disk File System”, Proc. Of the 8th NASA Goddard Conference on Mass Storage System and Technologies in cooperation with the 17th IEEE Symposium on Mass Storage Systems, March 2000.

[6] Steve Soltis et al. “The Global File System”, In The Fifth NASA Conference on Mass Storage System and Technologies, pp.319-342, Colleage Park, Maryland, March 1996.

[7] Chandramohan A. Thekkath, Timothy Mann, Edward K. Lee. “Frangipani: A Scalable Distributed File System”, ACM Operating Systems Review, Vol. 31, no.5, Dec. 1997.

[8] Chandramohan A. Thekkath, Timothy Mann, Edward K. Lee, “Petal : Distributed Virtual Disks”, In Proc. 7th Intl. Conf. In Architectural Support for Programming Languages and Operating Systems, pp.84-92, Oct. 1996.

[9] http://oss.sgi.com/projects/xfs/

[10] Comer, D., "The Ubiquitous B-Tree," Computing Surveys, Vol. 11, No. 2, June 1979, pp.121-137.

[11] Veritas Software, http://www.veritas.com

[12] Michael J. Folk et al., File Structures, Addison-Wesley, March 1998.