Design of Cluster Safe File System

Gang Zheng, Kai Ding, and Zhenxing He

North China Institute of Computing Technology,
Beijing, 100083, China
{zhengg,dingk,hezx}@nci.ac.cn

Abstract. The Cluster Safe File System (CSFS) is a distributed file system that can provide features such as single name space and remote data mirror. Files in the file system are scattered on all or some of the nodes in the cluster, and all nodes in the cluster can access the CSFS file system by mounting it on a local directory, clients can access the file system via NFS, FTP, Samba, etc. Every file in the file system has its synchronized mirror on another node, and that is why we call it SAFE. And in fact we can easily expand it to support more than one synchronized mirrors.

1 Introduction

High Available clusters are more and more widely used to provide non-stop services, such as web services, commercial and banking system, etc, in which cases, temporary failure might cause great loss and catastrophic consequences. And in many case, data loss is more intolerable than service failure, so it is also very important for a HA cluster to provide HA for its data. The CSFS is such a cluster file system which provides a mirror for each file in the file system on another node in the cluster to prevent from data loss. The CSFS is a filtering file system which is based on other file systems such as ext3, reiserfs, etc. The CSFS is both safe and single file system..

The CSFS is not designed from scratch. It is based on some existing open source projects such as FailSafe, PVFS and Intermezzo. It is designed for our cluster system based on the SGI's Linux FailSafe.

2 Architecture

The CSFS system is a distributed file system.Its physical arrangement and data mirroring mechanism are transparent to users.

The architecture of CSFS is made up of five parts (see figure 1, the CMOND is the cluster monitor daemon which can monitor processes), Linux kernel module (CSFSKM), Linux virtual device (CSFSDEV), CSFS library (CSFSLIB), file system service daemon (CSFSD) and file system input/output daemon (CSFSIOD).

Every file in the file system has its synchronized mirror on another node. In figure 2, the file' on node2' is a mirror of the file on node2.

X. Zhou et al. (Eds.): APPT 2003, LNCS 2834, pp. 249–253, 2003.

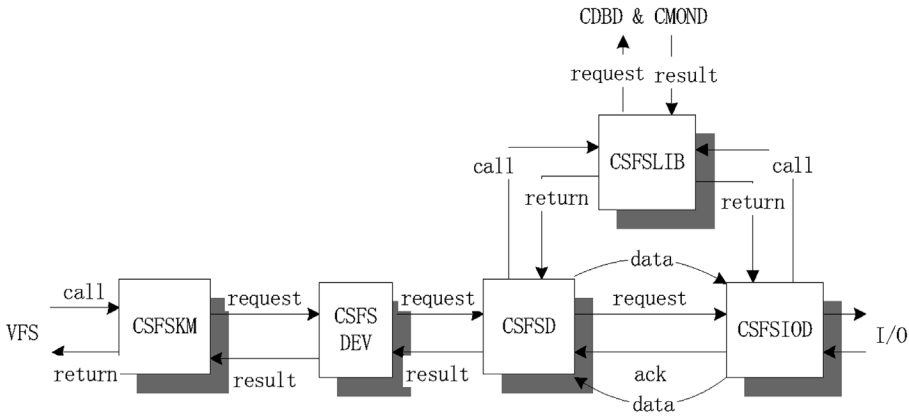

Fig. 1. The CSFS's Architecture

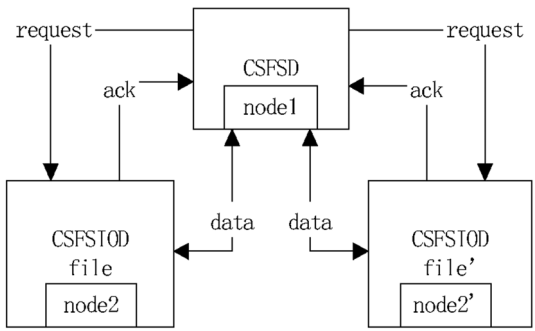

Fig. 2. File Mirror

The Linux kernel module (CSFSKM) generates file system operation request and Modification Log (Modification Log is generated only if the operation will modify the file system) for an operation, which are sent to and handled by the CSFSD. The CSFSD looks up the file or directory of the operation in the FailSafe's CDB (Cluster Database), and gets its two mirrored physical paths, then send the request to those nodes. And finally the CSFSIOD on those two nodes do the real I/O work.

2.1 CSFS Kernel Module (CSFSKM)

The CSFSKM is a bridge between the VFS and CSFSD. The CSFSKM gets an operation from VFS, translates it into request, and send it to the CSFSD via the CSFSDEV. After the result has arrived from the CSFSD, the CSFSKM gets it from the CSFSDEV and returns it to the VFS. If the request from the VFS belongs to writing requests, the CSFSKM will generate Modification Logs (ML), and fill it into the request that will be sent to the CSFSD.

2.2 CSFS Device (CSFSDEV)

The CSFSKM runs in the kernel space, while the CSFSD runs in the user space. So the CSFSKM need to communicate with the CSFSD via the CSFSDEV.

2.3 CSFS Library (CSFSLIB)

The CSFS's directory entries that are information about mapping logical paths into physical paths are stored in the FailSafe's CDB (Cluster Data Base), which will ensure that all nodes in the cluster will have exact copies of that information.

The CSFSLIB is a function library that provides interfaces to the CDB and the CMOND, etc. The CSFSD can use those functions to access the CDB

The mapping information between logical path and physical path of files and directories in the CSFS is stored in the CDB. Its format is

" logical_path → hostname1:physical_path1:hostname2:physical_path2".

The logical_path is the file or directory's path in the single name space. The hostname is the node the file or directory is on. The physical_path is the local path of the file or directory on the node which it is stored. The hostname2:physical_path2 is the mirror of the hostname1:physical_path1. We also did some optimization to reduce the entries in the CDB. If only a directory or file are not on the same node with its parent directory will we create an entry in the CDB to record the mapping information, otherwise, we don't need it.

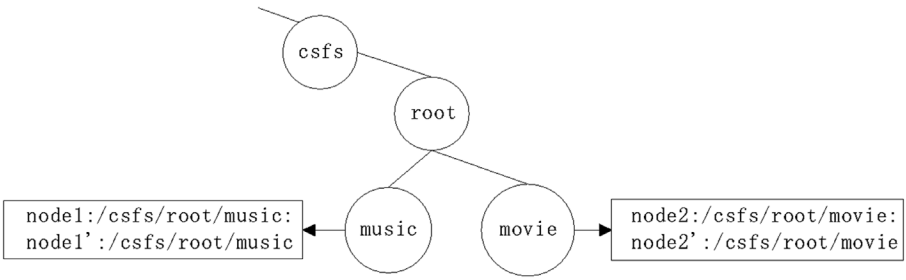

Fig. 3. Logical Directory Structure 1

We create an entry in CDB named "csfs", and an entry named "root" under the node "csfs" which is the root of the CSFS's single name space. Every child entry under the "root" corresponds to a file or directory in the single name space. We store a file or directory's physical paths as the value of the CDB's entry. If the value is empty, it means that the file or directory corresponding to this entry and its parent directory are on the same node.

On the assumption that there are four machines which are referred to as node1, node1', node2 and node2' respectively, and two directories in the CSFS, one is music, which is stored on node1 and node1' with the local path "/csfs/root/music" and the other is movie, which is stored on node2 and node2' with the local path "/csfs/root/movie". (see figure 3, the paths in the rectangle are the value of the corresponding CDB entry that stores the two mirrored physical paths).

2.4 CSFS Daemon (CSFSD)

The CSFSD's responsibility is receiving and handling the CSFSKM's requests. The CSFSD parses a request from CSFSKM, calls functions provided by the CSFSLIB to look up the file or directory of the request in the CDB and get its two mirrored physical paths, then send the request to those nodes' CSFSIOD.

Operation types of the request are divided into two classes, writing requests and reading requests. Reading requests are requests which don't modify the file system. Contrasting to reading requests, writing requests will modify the file system.

2.5 CSFS I/O Daemon (CSFSIOD)

The CSFSIOD receives requests from the CSFSD, does the real I/O work, and finally returns the results to the CSFSD by which the requests are sent. The content of a result message is similar to that of a result message from the CSFSD to the CSFSKM.

3 Reading Requests

Reading requests include null, getmeta, lookup, rlookup, readlink, getdents, statfs, export, hint, fsync. When the CSFSKM sends the reading request to the CSFSD, which doesn't include a ML, the CSFSD only sends the request to one of the two CSFSIODs on those two mirror nodes respectively. If the CSFSIOD returns success, the CSFSD returns success. Otherwise, the CSFSD sends the request to the other CSFSIOD. If the other CSFSIOD returns success, the CSFSD returns success. Otherwise, the CSFSD returns failure and send a notification to report the situation.

As for the read operation, because the data being transferred could be too large to be filled in the result message's buffer, so we decide to create another TCP connection between the CSFSD and the CSFSIOD, which is used to transfer data especially. The getdents operation is similar to read operation.

4 Writing Requests

Writing requests include setmeta, creat, remove, rename, symlink, mkdir, rmdir, mkdirent, rmdirent, read, write, which are attached with a ML. The CSFSD must send the request to both of CSFSIODs simultaneously on those two mirror nodes. The CSFSD waits for the CSFSIODs on both nodes to finish their jobs or time out. If both nodes finish their jobs successfully, CSFSD returns success. If one of them fails or time out, the CSFSD will send a notification to the administrator and then returns success. If both nodes fail or time out, the CSFSD will send a notification to report the situation and return failure.

In terms of the write operations, we also create other TCP connections between the CSFSD and those two CSFSIODs, which are used to transfer data especially.

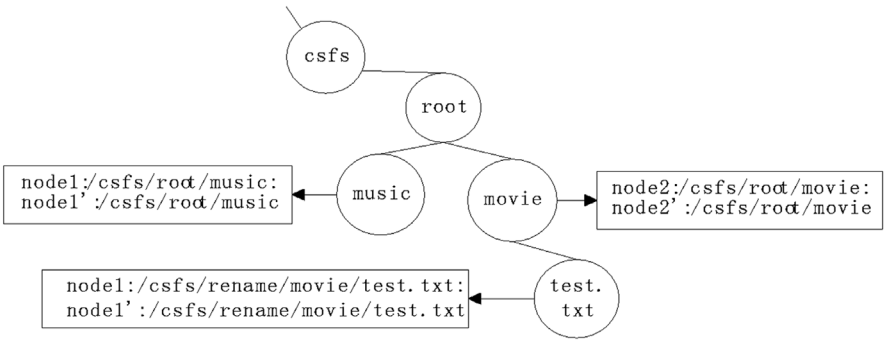

Fig. 4. Logical Directory Structure 2

5 Moving Files

In common condition, the implementation of rename is easy. But some times, the rename operation could lead to move files from one node to another node, because the old physical path and the new physical path are not on the same node. For example, we suppose that there is a file named test.txt under the directory /csfs/root/music on node1 and node1' (figure 3). After the user executes "rename /music/test.txt /movie/test.txt", the CSFSD receives and parses the request, then maps the logical paths into physical paths. It finds out that the physical path of /movie is on node2 and node2'. It means that we need to move the file test.txt from node1 and node1' to node2 and node2'. Because of the expensive cost, We can solve it by the following method. The CSFSD sends the request "rename /csfs/root/music/test.txt /csfs/rename/movie/test.txt" to the CSFSIODs on node1 and node1' simultaneously. The new physical path is made up of /csfs/rename and the file or directory's logical path, so that the new physical path is unique under the directory /csfs/rename. After the CSFSIODs on node1 and node1' return success, the CSFSD creates a logical path "/movie/test.txt" and sets its physical path as "node1:/csfs/rename/movie/test.txt:node1':/csfs/rena-me/movie/test.txt", then delete the logical path "/music/test.txt" in the CDB. (We create the directory /csfs/rename on every node, under which all the files or directories that should have been moved out of this node). After that, the logical directory structure looks like the figure 4.

References

1. Mallik Mahalingam, Christos Karamanolis, Lisa Liu, Dan Muntz, Zheng Zhang. *Data Migration in a Distributed File Service*. Computer Systems and Technology Laboratory,HP Laboratories Palo Alto ,HPL-2001-128, May 23rd, 2001.
2. Philip H.Cams, Walter B.Ligon III, Robert B. Ross Rajeev Thakur. *PVFS: A Parallel File System for Linux Cluters*. October 2000.
3. Peter J.Braam. *InterMezzo: File Synchronization with Intersync*.
4. Joshua Rodman, Steven Levine. *Linux Failsafe Administrator's Guide*.2000.
5. Joshua Rodman, Lori Jhonson. *Linux Failsafe Programmer's Guide*. 2000.