Osamu Tatebe
Grid Technology Research Center, AIST

Design of File System Directory Services

Status of This Memo

This memo provides information to the Grid community about file system directory services. It does not define any standards or technical recommendations. Distribution is unlimited.

Abstract

This document discusses the design of file system directory services, which will be one of essential services for Grid file system or virtual file system in grid environment. It manages the namespace of federated and virtualized data from file system resources, access control mechanisms, and meta-data management. This document proposes a set of operations needed to be supported by file system directory services. For scalable, large-scale and distributed file system directory management, this document also discusses three types of federation of file system directory services.

Contents

1. Introduction

Data in a grid can be of any format and stored in any type of storage systems. There can be many hundreds of petabytes of data in grids, among which a very large percentage is stored in files. A standard mechanism to describe and organize file-based data is essential for facilitating access to this large amount of data. The Grid File System Working Group (GFS-WG) was established in GGF data area to standardize the mechanism through providing a virtual file system in grid environment.

Two major deliverables of the WG are (1) architecture of Grid File System Services and (2) specification of File System Directory Services that is one of essential services for Grid file systems. File system directory services will manage the namespace of federated and virtualized data from file system resources, access control mechanisms, and meta-data management [1]. It will provide features such as (a) virtualized hierarchical namespaces for files or potentially other type of data (such as live data feeds), (b) efficient and transparent file sharing, and (c) flexible management of inter-organizational data access controls, and (d) ability to describe and manage file-system and application-specific metadata.

This document intends to describe the design of the file system directory services. It proposes a set of operations needed to be supported by file system directory services. For scalable, large-scale and distributed file system directory management, it also discusses three types of federation of the file system directory services.

The overall architecture of Grid file system will be specified later in GFS-WG, which provides infrastructure of a virtual file system facilitating federation and sharing of virtualized data from file systems in the grid environment by using file system directory services.

2. File System Directory Services

File system directory services manage a hierarchical directory tree for a virtual file system using Directory PortType, which consists of virtual directories and junctions, having attributes such as access permission, and status such as opening and locking. A junction is an object that points to files and subtrees.

A virtual directory or a junction is specified by a *virtual file handle* (VFH). A virtual directory manages a list of a virtual name of a virtual directory or a junction and the corresponding VFH, which is called a *directory entry*. A junction includes a pointer to a location of a file data, a (virtual) directory or another object with access protocol. The pointer may point to a service to manage replica locations of the file data.

2.1 Operations of Directory PortType

Basic operations of Directory PortType are

1) Lookup operation to convert a hierarchical path name to the corresponding VFH under access permission control of a hierarchical directory tree,
2) Creation, removal, and rename operations for a virtual directory or a junction,
3) Operations for managing attributes or status of a virtual directory or a junction.

Table 1 shows a set of operations of Directory PortType. This list is not a complete set. Error codes or error status are not specified in this document. Detailed content of Attributes is not also covered in the document.

Table 1 Set of operations of Directory PortType

Interface	Description
`VFH getrootVFH()`	Returns a virtual file handle of the root directory.
`VFH lookup(VFH fh, String path)`	Returns a virtual file handle of a virtual directory or a junction specified by a relative path from the virtual directory specified by `fh`. When `path` is an absolute path, `fh` is ignored.
`Direntry[] getdents(VFH fh)`	Returns an array of directory entries of the virtual directory specified by `fh`. `Direntry` is a structure consisting of a virtual name and the corresponding virtual file handle.
`Int mkdir(VFH fh, String name, Attr[] attrs)`	Creates a new virtual directory `name` under the virtual directory specified by `fh`, and sets attributes `attrs` to the virtual directory. This function returns `0` or an error code.
`Int rmdir(VFH fh, String name)`	Removes the virtual directory `name` in the virtual directory specified by `fh`.
`Int access(VFH fh, Int mode)`	Determines whether the virtual directory or junction specified by `fh` can be accessed with the specified mode. `Mode` is one or more of `READ`, `LOOKUP`, `MODIFY`, `EXTEND`, `DELETE`, and `EXECUTE`.
`Attr[] getattrs(VFH fh, String[] attrkeys)`	Returns an array of attributes of the virtual directory or junction specified by `fh`. Obtained attributes can be specified by `attrkeys`.
`Int setattrs(VFH fh, Attr[] attrs)`	Sets one or more attributes specified by `attrs` to the virtual directory or junction specified by `fh`.
`Int addjunction(VFH fh, String name, Attr[] attrs)`	Adds a new junction `name` in the virtual directory specified by `fh`, and sets attributes to the junction.
`Int removejunction(VFH fh, String name)`	Removes the junction `name` in the virtual directory specified by `fh`.
`Int link(VFH destfh, VFH fh, String name)`	Creates the “hard” link `name` for the junction specified by `destfh`, in the virtual directory specified by `fh`.
`Int rename(VFH srcdir, String oldname, VFH destdir, String newname)`	Renames the junction `oldname` in the virtual directory specified by `srcdir` to the junction `newname` in the virtual directory specified by `destdir`.
`Int open(VFH fh, Int flags)`	Creates server state for opening the junction specified by `fh`.
`Int close(VFH fh)`	Releases server state for opening the junction specified by `fh`.

`Int lock(…)`	Requests a record lock for the byte range.
`Int lookt(…)`	Tests the lock as specified in the arguments.
`Int lockf(…)`	Unlocks the record lock specified by the parameters.

Almost all operations need `lookup` operation to obtain a VFH at first. This means at least two round-trip interactions are required, while they could be critical overhead in wide area networks. One of solutions is introducing a COMPOUND procedure such as NFSv4 [2].

3. Federation of File System Directory Services

This section discusses federation of file system directory services for scalable, large-scale and distributed file system directory management. There are three major ways; (1) junction for remote file system directory services, (2) mounting file system directory services, and (3) remote virtual file handle in directory entries.

3.1 Junction for remote file system directory services

The target of a junction may be a remote virtual directory managed by different file system directory services. This provides loosely coupled integration of file system directory services.

Because the targeted virtual directories or targeted services do not know junctions that target to, there is no way to traverse to the parent directory. This also means that there is no way for the targeted service to inform services that have a junction that targets to even when the targeted service is lost or changed, which results in the inconsistency among file system directory services.

Every lookup request needs to start from the service that provides the root directory. This may result in the access concentration.

3.2 Mounting file system directory services

This approach assumes that a client system or library mounts several virtual file systems provided by file system directory services using a mount table or maps provided by *file system table services*. A mount table or maps consist of mount points and mounting file systems.

File system table services provide similar functionality of a mount table of `/etc/fstab`, or maps of `automount` in Unix systems. A mount table is mostly used for small-scale federation of file system directory services, while maps using such as a domain name system (DNS) are used for large-scale federation.

There is no access control mechanism when reading a mount table or maps of file system table services.

Using file system table services, a directory tree can be flexibly managed by changing a mount table or maps because there is no direct relationship among file system directory services. On the other hand, a client library has responsibility to control access permission between file system directory services, or when crossing a mount point. Moreover, moving a junction or a directory, or making a “hard” link is only possible within the same file system directory services, and it is not possible across file system directory services because of the same reason.

Unlike the case of junction for remote file system directory services, a lookup request is not necessary to start from the service that provides the root directory. This mitigates the access concentration.

3.3 Remote virtual file handle in directory entries

The virtual file handle in a directory entry may be a remote virtual directory provided by different file system directory services. This enables to link different file system directory services each other, and to provide single large-scale file system directory services using several distributed file system directory services. This is considered to be tightly coupled integration of file system directory services.

When creating or removing a remote virtual directory managed by different file system services, or moving a virtual directory between different file system services, directory entries of both related services need to be consistently modified. To ensure the consistent modification, transaction across different services is required.

To create a remote virtual directory in different file system directory services, an additional argument of a Grid service handle (GSH) of a remote factory service is required by `mkdir`. In this case, `mkdir` will create a service instance using the remote factory service that is used to create a remote virtual directory, and create a link each other. To ensure the consistency of each link, it is necessary to add two types of operations; `createdir` and `commitdir`, which are assumed to be called within `mkdir`. After creation of a service instance using the remote factory service if necessary, `mkdir` calls `createdir` to obtain a new VFH, and adds the VFH to a directory entry temporally. After successful call of `commitdir` that checks whether the temporal entry exists in directory entries of the parent directory, `mkdir` makes the temporal entry formal.

`Int mkdir(VFH fh, String factoryGSH, String name, Attr[] attrs)`	Creates a remote virtual directory `name` in remote services created by `factoryGSH` under the virtual directory specified by `fh`, and sets attributes `attrs` to the virtual directory.
`VFH createdir(Attr[] attrs, long timeout)`	Creates a temporal virtual directory with attributes `attrs` having the lifetime `timeout`, and returns the virtual file handle.
`Int commitdir(VFH fh, VFH parentfh, String name)`	When the parent directory specified by `parentfh` has a directory entry of `name` and `fh`, makes the temporal directory formal.

When removing a remote virtual directory, `rmdir` requires an additional operation to keep consistency of links of both services; `destroydir`, which removes a virtual directory after checking whether there is no entry for oneself in the parent directory. `Rmdir` first deletes a directory entry for `name` temporarily, and call `destroydir` to remove the remote virtual directory. When `destroydir` successfully removes the remote virtual directory, `rmdir` deletes the directory entry permanently.

`Int rmdir(VFH fh, String name)`	Removes the virtual directory `name` in the virtual directory specified by `fh`.
`Int destroydir(VFH fh, String childname)`	Removes the virtual directory specified by `fh` after checking whether there is no entry for oneself `childname` in the parent directory.

Moving a virtual directory or a junction to different file system directory services also requires a transaction across services.

This approach provides complete access control mechanism using a hierarchical directory tree because of direct federation of file system directory services instead of relying on a client library. `getrootVFH` can return the VFH of the root directory even though it is managed by different file system directory services.

Unlike the case of junction for remote file system directory services, a lookup request is not necessary to start from the service that provides the root directory. This mitigates the access concentration.

4. General problem

There are several problems not covered or not settled in this document.

- Replication of file system directory services – To enhance fault tolerance and reliability, redundant management of file system directory services is indispensable. The consistency model required by file system directory services is needed to be investigated.
- Backup – To prepare an unforeseen disaster, backup of data in file system directory services may be required. Moreover, even every file data itself may be required to be back up.
- Consistency problem between access permission of a junction and the target
- Removal or modification of a file data without notification to file system directory services
- Consistency problem between file data replicas
- Interoperability issue with NFSv4 and CIFS

5. Summary and conclusion

This document intended to describe the design of the file system directory services, which will be one of essential services for Grid file system or virtual file system in grid environment. It manages the namespace of federated and virtualized data from file system resources, access control mechanisms, and meta-data management.

This document proposed a set of operations needed to be supported by file system directory services. For scalable, large-scale and distributed file system directory management, it also discussed three types of federation of the file system directory services.

Further detailed discussion for specification and evaluation by implementing file system directory services are needed with respect to performance, consistency, scalability, and reliability. The evaluation needs to consider functionality of a client library, especially, with and without client attribute cache.

Author Information

Osamu Tatebe
Grid Technology Research Center, AIST
1-1-1 Umezono, Tsukuba
Ibaraki 3058568 Japan
o.tatebe@aist.go.jp

Intellectual Property Statement

Full Copyright Notice

References

[1] Leo Luan and Ted Anderson, “Grid Namespace for Files”, GGF working draft, GGF8, 2003
https://forge.gridforum.org/projects/gfs-wg/document/Grid_Namespace_for_Files/en/1
[2] S. Shepler, et al., “Network File System (NFS) version 4 Protocol”, RFC3530, 2003