GFARM V2: A GRID FILE SYSTEM THAT SUPPORTS HIGH-PERFORMANCE DISTRIBUTED AND PARALLEL DATA COMPUTING

Osamu Tatebe, Satoshi Sekiguchi, AIST, Tsukuba, Japan
Youhei Morita, KEK, Tsukuba, Japan
Noriyuki Soda, SRA, Nagoya, Japan
Satoshi Matsuoka, Titech / NII, Tokyo, Japan

Abstract

Grid Datafarm architecture is designed for facilitating reliable file sharing and high-performance distributed and parallel data computing in a Grid across administrative domains by providing a global virtual file system. Gfarm ™ v2 is an attempt to implement a global virtual file system that supports a complete set of standard POSIX APIs, while still retaining the parallel and distributed data computing feature of Grid Datafarm architecture. This paper discusses the design and implementation of Gfarm v2 that provides a secure, robust, scalable and high-performance global virtual file system.

INTRODUCTION

Recent research, development and standardization of Grid technologies make it possible to share resources such as CPU and storage across administrative domains. Fundamental problems for resource sharing such as authentication, authorization and security has been resolved to some degree. Problems are gradually shifting to core operating system functionalities such as process management, process scheduling, signal handling, and file systems, in order to exploit resources that are located in distant locations efficiently.

Our research group proposed the Grid Datafarm architecture for Petascale data-intensive computing facilitating distributed resources in wide area [1]. Main features of the architecture are to provide (1) a Grid file system that integrates local disks of compute nodes in Computational Grid, and (2) parallel and distributed computing associating Computational Grid and Data Grid.

A Grid file system is a global virtual file system that federates numbers of file systems (or file servers) in a Grid. Integration is achieved by a filesystem metadata server that manages a virtual human-readable namespace. Standardization regarding Grid file systems is ongoing in the Grid File System Working Group of the Global Grid Forum [2]. Physically, each file or each block in a file would be stored in some arbitrary file server in a Grid, while users and applications would access files via a virtualized file system without being concerned of the file location. As such, a Grid file system is a shared network file system scaled to Grid level, allowing easy and transparent sharing of file data without any modifications to existing applications.

In addition, in the Grid Datafarm architecture, local disks of compute nodes in a Computational Grid compose a Grid file system (Fig. 1); every file server provides not only storage but also computing resources. In other words, storages of file servers comprise a Grid file system or a Data Grid, while computing resources of file servers comprise a Computational Grid.

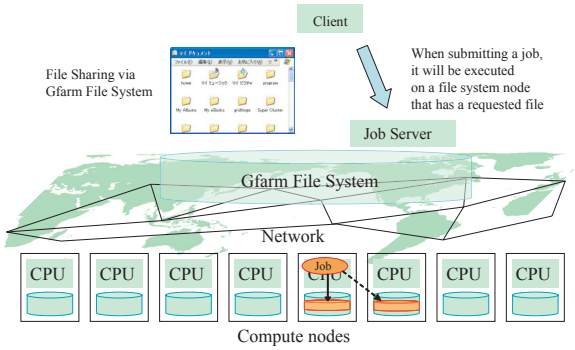

Figure 1: Grid Datafarm Architecture. Local disks of compute nodes in a Computational Grid composes a Gfarm file system. A job will be executed on a compute node that has one of file replicas of the requested file.

When a user submits a job to the Computational Grid, it will be scheduled and executed on one of the file servers (i.e., a compute node) that has a copy of the requested file depending on the CPU utilization. This scheduling policy is called *file-affinity* process scheduling, which enables scalable I/O performance as well as distributed data computing. The key issue is association of Computational Grid with Data Grid. Having a separate I/O across the network independent from the compute nodes would be disadvantageous for large-scale distributed system.

Grid Datafarm architecture, moreover, supports high-performance distributed and parallel computing for processing a group of files by a single program, which is a most time-consuming, but also a most typical, task in data-intensive computing such as high energy physics, astronomy, space exploration, and human genome analysis. Such a process can be typically performed independently on every file in parallel, or at least exhibit good locality. In order to facilitate this, in Grid Datafarm, an arbitrary group of

files possibly dispersed across administrative domains can be managed by a single Gfarm file. Each member file will be accessed in parallel in a new file view called *local file view* by a parallel process allocated by file-affinity scheduling based on replica locations of the member files. File-affinity scheduling and file view feature naturally derives the *owner computes* strategy, or *move the computation to data* approach for parallel and distributed data computing of member files of a Gfarm file in a single system image. This is the key distinction of Gfarm over other distributed file systems, where the data will be moved to computation by default.

Gfarm^{TM} v1 is a prototype implementation of the Grid Datafarm architecture[1]. It provides a subset of the POSIX standard API required for data-intensive computing. Gfarm v1 has demonstrated its architectural advantages and ease of programming for data-intensive applications [3]. However, we found that there are several weaknesses in the system, some from the lack of feature typically found in distributed file systems, some being robustness and dependability, and some more fundamental to the architecture itself.

This paper discusses the design and implementation of Gfarm v2 that attempts to overcome such weaknesses. Gfarm v2 aims to provide a POSIX-compliant global virtual file system facilitating features of Grid Datafarm architecture for Petascale data-intensive computing. It can be used as a general-purpose network file system for Grid or virtual organization, allowing existing applications to share files securely and dependably, and to access files efficiently across administrative domains.

RELATED WORK

There are several high-performance file systems that support more than a thousand clients and/or file system nodes. Lustre [4] supports more than a thousand clients in a cluster system. Lustre consists of numbers of file servers or Object Storage Targets (OSTs), metadata servers, and numerous clients. Between file servers and clients, high-speed interconnects such as Gigabit Ethernet, Elan3, and Myrinet are assumed. Each file (or object) can be placed in any OST. Lustre does not facilitate replica management; instead, it uses writeback cache to improve write performance. Collaborative read cache is being planned to improve read performance. The major difference from the Grid Datafarm architecture is that Lustre separates file system nodes from clients (or compute nodes). This reflects the fact that Lustre assumes an operating environment of large clusters and intra-enterprise high bandwidth connectivity, where moving data to computation makes sense, in contrast to Gfarm where data may reside distributedly in wide area, making exploitation of local high bandwidth by moving computation to data essential.

The Google File System [5] supports more than a thousand storage nodes. All files are divided into fixed-size chunks, and each chunk can be placed in any storage node (chunkserver). All chunks have three replicas by default to prevent data loss on failures. All I/O operations are implemented by user client library with no client or server cache. Google File System does not provide complete POSIX API, but tunes itself to operations that support Google's data processing needs. The difference from the Grid Datafarm architecture is that Google File System also separates I/O from clients, and it divides a file into fixed-size chunks. The latter is disadvantageous for parallel and distributed data computing since data access cannot be localized.

OVERVIEW OF GFARM V1

Gfarm v1 is a prototype implementation to realize the Grid Datafarm architecture. It is an open source software available at http://datafarm.apgrid.org/. It consists of Gfarm I/O library, an I/O server; *gfsd*, and a set of metadata servers; *gfmd* and *slapd*.

Gfarm I/O library provides interfaces for accessing the Gfarm file system. It includes Gfarm file read/write, file replication, parallel I/O, parallel file transfer, and file-affinity process scheduling. Parallel I/O provides the *file view* feature: *index file view* and *local file view*, for distributed and parallel data computing.

A system call hooking library is provided for existing binary programs to access Gfarm file system as if it were mounted at /gfarm. By loading the system call hooking library before program execution, necessary system calls of existing programs can be trapped without any modification. The system call hooking library determines from the access path or a file descriptor whether it is a file in Gfarm file system or not. If it is a file in Gfarm file system, the corresponding Gfarm APIs are called to access the file. Otherwise, it call the system call as usual.

Gfsd is an I/O daemon running on every file system node. It facilitates file access and file replica creation for the file system nodes. In Grid Datafarm architecture, a file system node is also assumed to be a compute node, and gfsd has the ability to execute a remote program on the file system node. Moreover, gfsd manages status information and CPU load average of the file system node for scheduling.

The metadata server consists of gfmd and slapd. Every file system metadata including directory, file status information, replica catalog, is managed by slapd, which is an ldap server developed by the OpenLDAP project [6]. Gfmd is a process manager that is used by the Gfarm remove execution commands.

Gfarm file system can be accessed by not only the Gfarm I/O library but also standard protocols such as scp, GridFTP, and SMB using the system call hooking library.

Weaknesses of Gfarm v1

Being the first prototype/experimental implementation of the Grid Datafarm architecture, there are several weaknesses of Gfarm v1 with respect to functionality, robust-

[1]Gfarm is a registered trademark in Japan.

ness, security, and flexibility. Gfarm v1 has been developed to investigate the requirements of large-scale data-intensive computing, and to show the effectiveness of the Grid Datafarm architecture. As such functionalities found commonly in a distributed file system but were considered not important for those specific purposes, including file open in read-write mode, and advisory file locking were not designed or implemented[2]. On the other hand, as we expanded the application fields, it became clear that some traditional distributed file system feature, as well as some enhancements, were necessary.

In Gfarm v1, it is the role of the Gfarm I/O library to generally maintain consistency between metadata and the corresponding physical file. Since it is a user-level library, however, unexpected application crash can easily break the consistency. Another case is that a file owner can modify or delete physical files directly on file system nodes bypassing the Gfarm I/O library. This also causes the inconsistency between metadata and the physical file. Although Gfarm v1 provides a maintenance command to check and fix the inconsistency, this is cumbersome and error-prone.

Gfarm v1 does not impose sufficient access control for metadata access. Also, there is no access control for a group since there is no group management.

Grid Datafarm architecture supports managing a group of files (collection, or container) as a single Gfarm file for parallel and distributed data computing. To support this feature, Gfarm v1 has a special type of metadata for a group of files that has any number of member files. On the other hand, for every type of grouping, we have had to construct support for it internally, which lacked flexibility and was in fact quite cumbersome.

DESIGN AND IMPLEMENTATION OF GFARM V2

The goal of Gfarm v2 is to provide a POSIX compliant, robust, dependable and secure network file system as well as to support more than ten thousand clients and file server nodes with scalable file I/O performance. POSIX compliance includes supporting read-write file open mode and advisory file locking. It also aims to be a substitute for NFS and AFS, while retaining the parallel data processing capability.

Opening Files in Read-write Mode

When a file has several file replicas, consistency among the file replicas needs to be maintained when it is modified. Semantics of consistency among file replicas supported by Gfarm v2 is the same as AFS.

1. If there is no advisory file locking, updated file content can be accessed only by a process that opens it after a writing process closes.
2. Otherwise, up-to-date file content can be accessed in the locked region among processes that lock it. Note that this is not always ensured when a process writes the same file without file locking.

To ensure the semantics of consistency, file replicas to be accessed are selected as follows when opening a file.

1. When opening a file in read mode,
   (a) select any file replica of the file.
2. When opening a file in write mode,
   (a) if there is a process that opens the file in write mode, select the file replica already opened in write mode,
   (b) if there are several processes that open the file in read mode, select any file replica or one of file replicas opened in read mode,
   (c) if there is no process that opens the file, select any file replica.

The selection is done by the metadata server. This selection ensures two different file replicas cannot be opened in write mode at the same time.

Metadata server deletes all invalid metadata and all invalid file replicas when closing a file that is opened in write mode. The reason why all possible invalid metadata and all possible invalid file replicas are not deleted at file open time is that there is a case such that a file is not modified even though it is opened in write mode. Regarding deletion of file replicas whilst one of them is still held open by some process, there is no particular problem since it is still accessed by a valid file descriptor.

Advisory File Locking

Gfarm v2 supports advisory file locking in POSIX. A read lock and a write lock are supported for the whole file or a region of a file.

The basic policy to implement the advisory file locking is that all processes access the same file replica when the file is locked. Moreover, to ensure access to the up-to-date file content, client cache is disabled in the locked region.

Fig. 2 describes mechanism to implement advisory file locking with an example. There are two file system nodes; FSN1 and FSN2. A file /grid/jp/file2 has two file replicas stored on FSN1 and FSN2. Process 1 running on FSN1 opens a file /grid/jp/file2 in read-write mode. Since one of file replicas is stored on the same node, the local file replica is selected to be accessed. Process 2 running on FSN2 opens the same file in read-only mode. Since one of the file replicas is stored on the same node, the local file replica is selected to be accessed. Note that the modification of file replica on FSN1 does not need to be reflected on to the file replica on FSN2 since Process 2 opens the file before Process 1 closes it.

Process 2, then, requests a read lock to the metadata server. Since Process 1 already has the file replica open on FSN1 of the same file in read-write mode, it needs to

[2]Read-write mode has been supported since the version 1.0.4.

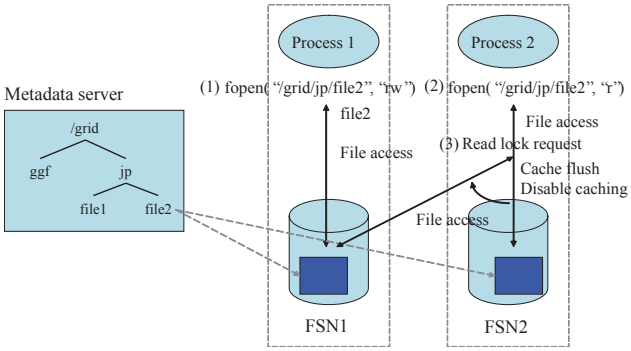

Figure 2: Advisory file locking. Process 1 opens /grid/jp/file2 in read-write mode. Process 2 opens the same file in read mode, and requests a read lock. In this case, Process 2 flushes and disables the client cache, and changes the file replica to be accessed from FSN2 to FSN1, since the file replica on FSN1 is already opened in read-write mode.

change the file replica to be accessed. Before changing the file replica, it flushes the client file cache, and disables it in the locked region to ensure access to the up-to-date content.

Consistent Update of Metadata

Gfarm v1 maintains consistency between metadata and the corresponding physical file using the Gfarm I/O library. Since it is a user-level library, when application unexpectedly crashes before closing a file, file status information including file size cannot be updated.

Gfarm v2 changes this metadata update mechanism. In Gfarm v2, Gfarm I/O library basically does not update metadata directly. Instead, it is updated by gfsd. When a file is closed, Gfarm I/O library sends a close request to gfsd. Gfsd updates the metadata after closing the file. Gfsd also updates the metadata when the connection from a client is broken. This ensures the consistent metadata update even on unexpected application failure.

The other possibility that breaks the consistency between metadata and the corresponding physical file is direct access and modification of the physical file without notifying the metadata server. In Gfarm v1, the access to physical files is allowed by a file owner in the Gfarm file system. Because of this, the file owner is able to modify the physical file accidentally, which would cause the inconsistency.

In Gfarm v2, every physical file is owned by gfsd. This disables direct file modification by users. Access permission is only controlled by the metadata in Gfarm file system.

Generalization of File Grouping Model

Gfarm v1 introduces a special type of metadata to manage a group of files. Although introducing a special type enabled the management of group of files, it lacked flexibility in many ways. For example, it had a restriction such that a specific member file should belong to only one group not multiple groups. Moreover, it is not possible to have a group of groups of files.

In the case of astronomical data analysis for the Subaru telescope [3], image data of the prime focus camera consists of $10 \times N$ files in N shots since it is a mosaic CCD camera consisting of 10 CCD detectors. During the data analysis, there are three cases; 10 files can be executed in parallel, N files can be executed in parallel, and $10 \times N$ files can be executed in parallel. Because this, we need three kinds of grouping for the same physical files.

For flexible grouping, Gfarm v2 does not introduce a special metadata type but exploits the filesystem directory structure to manage a group of files. All files under a directory including its subdirectories form a group of files. Moreover, exploiting symbolic links and hard links, it is possible to realize the above three kinds of grouping for the same file set using standard file operations.

SUMMARY AND FUTURE WORK

Gfarm v2 aims at being a global virtual file system having scalability up to more than ten thousand clients and file system nodes. It also aims to be POSIX compliant, secure, robust, and dependable. This paper discussed its design and implementation.

We are still implementing the Gfarm v2, and have a plan to release the first version in March, 2005. We would like to evaluate it especially regarding the scalability up to more than ten thousand nodes. We need to investigate several research issues including data preservation and efficient algorithm of automatic replica creation.

ACKNOWLEDGEMENTS

We thank the members of the Gfarm project of AIST, KEK, Tokyo Institute of Technology, and the University of Tokyo for taking the time to discuss many aspects of this work with us, and for their valuable suggestions. We also thank the members of Grid Technology Research Center, AIST, for their cooperation in this work.

REFERENCES

[1] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda and S. Sekiguchi, "Grid Datafarm Architecture for Petascale Data Intensive Computing", Proc. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (2002)

[2] https://forge.gridforum.org/projects/gfs-wg/.

[3] N. Yamamoto, O. Tatebe and S. Sekiguchi, "Parallel and Distributed Astronomical Data Analysis on Grid Datafarm", Proc. 5th IEEE/ACM International Workshop on Grid Computing (2004).

[4] http://www.lustre.org/.

[5] S. Ghemawat, H. Gobioff and S. Leung, "The Google File System", Proc. 19th ACM Symposium on Operating Systems Principles (2003).

[6] http://www.openldap.org/.