Electrotechnical Laboratory, Techinical Report, TR-2001-4

Grid Data Farm for Petascale Data Intensive Computing*

Osamu Tatebe[1] Youhei Morita[2] Satoshi Matsuoka[3]
Noriyuki Soda[5] Hiroyuki Sato[2] Yoshio Tanaka[1]
Satoshi Sekiguchi[1] Yoshiyuki Watase[2] Masatoshi Imori[4]
Tomio Kobayashi[4]

[1] Electrotechnical Laboratory
[2] High Energy Accelerator Research Organization (KEK)
[3] Tokyo Institute of Technology/JST
[4] The University of Tokyo
[5] SRA Inc.

1 Introduction

High performance and data-intensive computing and networking technology has become a vital part of large-scale scientific research projects in areas such as high energy physics, astronomy, space exploration and human genome projects. One example is the Large Hadron Collider (LHC) project at CERN, where four major experiment groups will generate an order of Petabyte of raw data from four big underground particle detectors each year, data acquisition starting from 2006. Grid technology will play an essential role in constructing world-wide data analysis environments where thousands of physicists will collaborate and compete for the particle physics data analysis at the energy frontier. A multi-tier "Regional Centers" world-wide computing model has been studied by the MONARC Project[1]. It consists of Tier-0 center at CERN, multiple Tier-1 centers in participating continents, tens of Tier-2 centers in participating countries, and many Tier-3 centers in universities and institutes.

Grid Data Farm is a Petascale data-intensive computing project initiated in Japan. The project is collaboration among KEK (High Energy Accelerator Research Organization), ETL/TACC (Electrotechnical Laboratory / Tsukuba Advanced Computing Center), the University of Tokyo, and Tokyo Institute of Technology (Titech). The challenge will involve construction of a data processing framework that will handle hundreds of Terabyte to Petabyte scale data emanated by the ATLAS experiment of LHC. Both KEK and the Univ. of Tokyo will collaborate for building a Tier-1 regional center in Japan. The underlying hardware will be a thousands node scale PC cluster, each node facilitating a near-Terabyte of storage, and incoming data of approximately continuous 600Mbps bandwidth from CERN will be systematically stored and will be subject to intensive processing. The Grid Data Farm will facilitate the following features for collider data processing as well as serving as a framework for other types of data-intensive scientific applications:

*http://datafarm.apgrid.org/

- Global distributed filesystem for Petabyte scale data,
- Parallel I/O and parallel processing for fast data analysis,
- World-wide group-oriented authentication and access control,
- Thousands-node, wide-area resource management and scheduling,
- Multi-Tier data sharing and efficient access,
- Program sharing and management,
- System monitoring and administration,
- Fault tolerance / dynamic reconfiguration / Automated data regeneration or re-computation

Major components of the Grid Data Farm are the Gfarm client, the Gfarm server and the Gfarm (distributed) filesystem with Gfarm parallel I/O. The Gfarm filesystem consists of a thousands node scale PC cluster, each node with a local disk and possibly distributed over the Grid, and Petascale data are distributed across the disks in the Gfarm filesystem managed by the Meta Data Management System and the Gfarm Filesystem Daemon. The Meta Data Management System provides a mapping from logical file names to the distributed physical file components and also stores metadata such as a replica catalog and a history that is necessary to reproduce the data. The Gfarm filesystem daemon provides a facility of remote file operations with access control as well as remote program loading and resource monitoring. Large-scale distributed data are accessed by the Gfarm parallel I/O library and processed in parallel.

The Grid Data Farm middleware is based on Grid-based RPC (GridRPC), in particular an extended variant of our Ninf system[7, 5], and other lower level Grid service middleware, especially Globus[2]; it makes it easy for the users to register his analysis software and process massive amounts of data spread over multiple nodes in an easy way. Load balancing, Job scheduling, Fault Tolerance, and Data Maintenance are transparently or semi-transparently handled by the system. Users can interact with the system using GUIs or a simple shell front end; more sophisticated client program interaction is possible with GridRPC.

2 Software Architecture of Grid Data Farm

Figure 1 depicts a software architecture of the Grid Data Farm. Major components of the Grid Data Farm are the Gfarm client, the Gfarm server and the Gfarm filesystem. The Gfarm filesystem consists of the Gfarm Meta Database and the Gfarm pool that is a thousands node scale PC cluster, each node with a local disk and possibly distributed over the Grid. A large-scale distributed file, called Gfarm file, is divided into several fragments and distributed across the disks in the Gfarm filesystem. A Gfarm file, specified by the Gfarm file name or the Gfarm URL such as `gfarm:/path/name`, is accessed using the Gfarm parallel I/O library, and processed in parallel.

The Gfarm filesystem daemon (gfsd) runs on each node of the Gfarm pool to facilitate remote file operations with access control in the Gfarm filesystem

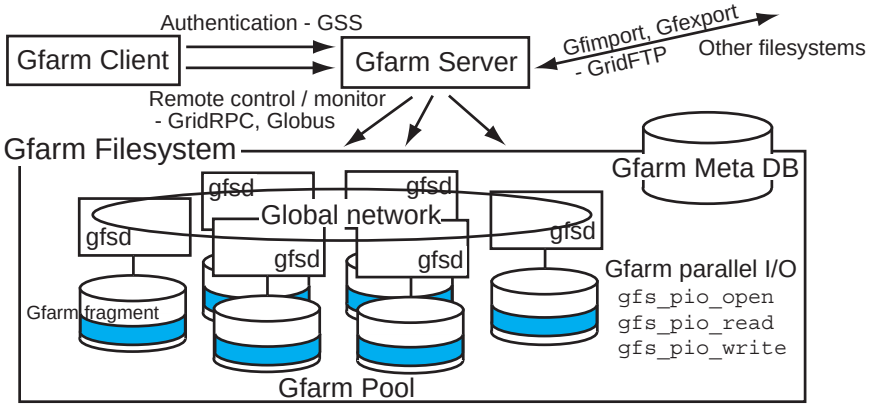

Figure 1: Software architecture of the Grid Data Farm

using a light-weight GFS RPC, as well as dynamic execution-loading from the Gfarm server; other roles are node resource status monitoring and control.

Metadata of files in the Gfarm filesystem is stored into the Gfarm Meta Database, which consists of a mapping from a logical Gfarm file name to physical distributed fragment file names and file status information including file size, protection, access/modification/change time and checksum as well as a replica catalog and a history. The history is needed to re-compute the data when a node or a disk fails, or to validate how the data is generated. Metadata is registered at the close operation of each Gfarm fragment and checked validity after all parallel processes terminate. When one of user processes terminates unexpectedly without registering metadata while the other processes correctly register metadata, metadata remains invalid and will be deleted by the system.

The Gfarm server is based on the network-enabled server[4] that is a major component of the GridRPC, enhanced with the Gfarm filesystem capability. The Gfarm server authenticates the Gfarm client using the Generic Security Service[3] for mutual authentication and single sign-on, and executes a parallel program that may be a user program registered by the Gfarm client, on the Gfarm pool spread over the Grid. The Gfarm server analyses input and output Gfarm files, and schedules Gfarm pool nodes to be executed by inquiring of the Gfarm Meta Database. The scheduling should consider physical locations of fragments of Gfarm files, the replica information and node status in the Gfarm pool.

The Gfarm client interacts with the Gfarm server and the Gfarm system using GUIs or a shell front end called the *Gfarm shell*; more sophisticated client program interaction is possible with GridRPC that makes it easy for the users to execute a remote procedure with the feature of dynamic Interface Description Language (IDL) loading and management. Users can register and execute his analysis software as well as monitor and administer the system using the shell.

There are several tools to interact with conventional filesystems or network streams by GridFTP[6] and so on. Gfimport imports and scatters large-scale data, and gfexport gathers and exports the data. The Gfarm system handles load balancing by redistributing the data based on program profiling.

3 Grid Data Farm Applications

The Grid Data Farm provides not only parallel file access but also parallel processing. In the typical usage of the Grid Data Farm, each node of the Gfarm pool accesses only local files, processes in parallel and creates new files locally.

Data analysis in typical high energy experiments is often characterized as "finding a needle in a hey stack". Each single collision of particles in the accelerator is called an "event". Information of thousands of particles emerging from the collision point is recorded by surrounding particle detectors. In the LHC accelerator, there will be 10^9 collisions per second. The events are then processed and filtered "on-line" to pick up only physically interesting ones, which are recorded into the storage media at the rate of 100 Hz for later "off-line" analysis. During the first year of the accelerator run, an order of 10^{16} collisions will be observed and 10^9 events will be recorded. Discovering a Higgs particle, depending on its unknown mass, will be of finding of events with certain special characteristics with an order of several tens out of 10^{16} collisions.

Each event data consists of digitized numbers from sub-detectors such as calorimeter, silicon micro-strip and tracking chambers. This initial recording of the event is called "RAW" data. In ATLAS experiment, the size of RAW data is approximately 1 to 3 Mbytes per event, corresponding to several Petabytes of data storage per year. The digitized information in RAW data is reconstructed into physically meaning full analog values such as energy, momentum, and the geometrical position in the detector. In ATLAS, typical event reconstruction will take about 300 to 600 SPECint95 per event, which will take place mainly at Tier-0 regional center at CERN. To keep up the event reconstruction rate with the data taking, at least 150 to 200K SPECint95 processing power is required for ATLAS Tier-0.

Physics data analysis such as Higgs particle search, B-quark physics and top-quark physics will be based on the reconstructed event summary data (ESD) at Tier-1 centers distributed over the world.

Because each event is not related each other, we can analyze the data independently on each CPU node in parallel. Only at the last stage of the analysis, a small set of statistical information needs be collected from each node. The data-parallel, distributed, and low-cost "CPU farm" approach has been very favorable and successful in high energy physics data analysis for the past decade. However, building a large-scale CPU farm with an order of 1,000 CPUs brings us a new technical challenge in the design and the maintenance. How to distribute the large quantity of data to each CPU effectively also remains to be a problem. Gfarm is designed to solve the handling of the large quantity of data localized into each CPU while the integrity of the data set is ensured by the meta database.

In ATLAS data analysis software, object database technology will be used to store and retrieve data at various stages of analysis. A commercial database package Objectivity is one of the candidates to achieve this task. It has already been employed in production by BaBar experiment at SLAC, and is already a core part of the software development in CMS experiment of LHC. Gfarm has been demonstrated to co-work with Objectivity.

4 Example

Users can interact with the Gfarm system with web-based GUIs, the Gfarm shell, GridRPC and GridRPC applications. This section describes an example of user interface using the Gfarm shell.

At first, an user should log in to the Gfarm server.

```
% gfarm gfarm.apgrid.org
Trying gfarm.apgrid.org...
Connected to gfarm.apgrid.org
210 gfarm.apgrid.org Grid Data Farm server (Version 0.01 alpha) ready.
login (tatebe):
331 Password required for tatebe
Password:
230- *** Welcome to Grid Data Farm ***
230 User tatebe logged in
gfarm>
```

This example shows the login process to the Gfarm server `gfarm.apgrid.org` with the plain-text authentication like FTP. In the current implementation, the Gfarm shell supports plain text authentication and rsh or ssh authentication, while the Generic Security Service will be supported soon. The user can execute remote program with `exec` command.

```
gfarm> exec foo --gfarm_in gfarm:in --gfarm_out gfarm:out
```

This example executes the program `foo` with the input Gfarm file `gfarm:in` and the output Gfarm file `gfarm:out`. In this case, the Gfarm server creates a list of hosts that store fragments of the Gfarm file `gfarm:in` inquiring to the Gfarm Meta Database, and executes the program `foo` in parallel. The output Gfarm file `gfarm:out` will be created.

5 Gfarm Parallel I/O API

The Gfarm parallel I/O API provides the facility of Gfarm file access in cooperation with a Gfarm Meta Database. All Gfarm files are divided into several indexed fragments and stored into several disks. The Gfarm parallel I/O API accesses each fragment explicitly and in parallel. The API is assumed to be called not only on the Gfarm pool nodes but also on the Gfarm server.

The APIs described in this section are current interfaces, and subject to change. Almost all functions return a constant address of a status message so that it is easy to check both the error code and the error message.

5.1 Definitions

Gfarm file and Gfarm fragment A *Gfarm file* is a logical file specified by a Gfarm URL or a Gfarm file name. Physically, a Gfarm file is divided into several indexed *Gfarm fragment*s and stored into several disks. Each Gfarm fragment can be specified by the Gfarm URL and the index.

Gfarm URL or Gfarm file name A *Gfarm URL* or *Gfarm file name* is a path name of the Gfarm filesystem. Every Gfarm URL has a prefix "`gfarm:`" and the following Gfarm URLs are valid.

```
gfarm:~username/path/name
gfarm:/path/name
gfarm:relative/path/name
```

Gfarm file handle A *Gfarm file handle* is an opaque object created by gfs_pio_open, gfs_pio_create, gfs_pio_open_local and gfs_pio_create_local, and freed by gfs_pio_close. All operations on an open file reference the file through the file handle.

local disk *Local disk*s are usually assumed to be connected directly to a PC with SCSI, IDE and so on, and can be accessed faster than remote disks. However, this assumption is not a requirement.

5.2 Error code

Error code is not an integer but a constant pointer of characters that contains an error message. Currently the following error codes are defined.

```
GFARM_SUCCESS (= NULL)
GFARM_ERR_NO_MEMORY
GFARM_ERR_NO_SUCH_OBJECT
GFARM_ERR_ALREADY_EXISTS
GFARM_ERR_PERMISSION_DENIED
GFARM_ERR_INVALID_ARGUMENT
```

5.3 Initialization and finalization

```
char* gfarm_init(void);
char* gfarm_finalize(void);
```

gfarm_init initializes the execution environment of the Gfarm system and establishes a connection to a Gfarm Meta Database. gfarm_finalize terminates the execution environment and disconnect a connection to the Gfarm Meta Database.

5.4 File Manipulation

5.4.1 Opening and creating a file

```
char* gfs_pio_open(char *url, int index, char *host,
                   int flags, GFS_FILE *gf);
char* gfs_pio_create(char *url, int index, char *host,
                     mode_t mode, GFS_FILE *gf);
```

gfs_pio_open opens the Gfarm fragment identified by the Gfarm URL `url` and the index `index` on the Gfarm pool node `host`, and returns a new Gfarm file handle `gf`. When the `host` is not specified, it is obtained by the Gfarm Meta Database. `flags` is one of `GFARM_FILE_RDONLY` or `GFARM_FILE_RDWR` which request opening the file read-only or read/write, respectively. gfs_pio_create creates a new Gfarm

fragment specified by the Gfarm URL `url` and the index `index` with the access mode `mode` on the Gfarm pool node `host`, and returns a new Gfarm file handle `gf`. `mode` specifies the permissions to use. It is modified by the process's `umask`.

The following functions are intended for a special but typical case such that every node has at least one Gfarm fragment.

```
char* gfs_pio_set_local(int index, int size);
char* gfs_pio_open_local(char *url, int flags, GFS_FILE *gf);
char* gfs_pio_create_local(char *url, mode_t mode, GFS_FILE *gf);
char* gfs_pio_local_paths_get(char *url,
                              int *npaths, char ***paths);
```

gfs_pio_open_local opens the local Gfarm fragment specified by the Gfarm URL `url`, and returns a new Gfarm file handle `gf`. `flags` is one of `GFARM_FILE_RDONLY` or `GFARM_FILE_RDWR`. gfs_pio_create_local creates the local Gfarm fragment specified by the Gfarm URL `url` with the access mode `mode`, and returns a new Gfarm file handle `gf`. gfs_pio_set_local sets the index `index` of the local Gfarm fragment and the total number of fragments `size`. gfs_pio_local_paths_get returns a list of path names of local Gfarm fragments and the total number of the local fragments of the Gfarm file specified by the Gfarm URL `url`.

5.4.2 Closing a file

```
char* gfs_pio_close(GFS_FILE gf)
```

gfs_pio_close closes the Gfarm file handle `gf`, and updates or checks the file size and the checksum of the Gfarm Meta Database. The checksum is used to verify the identity of a master file and the replica.

5.5 File access

The Gfarm parallel I/O API provides blocking, noncollective operations and uses individual file pointers.

```
char* gfs_pio_read(GFS_FILE gf, void *buf, int size,
                   int *nread);
```

gfs_pio_read attempts to read up to `size` bytes from the Gfarm fragment referenced by the file handle `gf` into the buffer starting at `buf`, and returns the number of bytes read `nread`.

```
char* gfs_pio_write(GFS_FILE gf, void *buf, int size,
                    int *nwrite);
```

gfs_pio_write writes up to `size` bytes to the Gfarm fragment referenced by the file handle `gf` from the buffer starting at `buf`, and returns the number of bytes written `nwrite`.

```
char* gfs_pio_seek(GFS_FILE gf, file_offset_t offset, int whence);
```

gfs_pio_seek repositions the offset of the Gfarm fragment referenced by the file handle `gf` to the argument `offset` according to the directive `whence` as follow:

SEEK_SET The offset is set to `offset` bytes.

SEEK_CUR The offset is set to its current location plus `offset` bytes.

SEEK_END The offset is set to the size of the file plus `offset` bytes.

```
char* gfs_pio_flush(GFS_FILE gf);
```

gfs_pio_flush forces a write of all buffered data for the Gfarm fragment referenced by the file handle `gf`.

```
int gfs_pio_getc(GFS_FILE gf);
int gfs_pio_ungetc(GFS_FILE gf, int c);
char* gfs_pio_putc(GFS_FILE gf, int c);
```

gfs_pio_getc reads the next character from `gf` and returns it, or `EOF` on end of the fragment or error. gfs_pio_ungetc pushed `c` back to `gf`, where it is available for subsequent read operations. gfs_pio_putc writes the character `c` to `gf`.

```
char* gfs_pio_getline(GFS_FILE gf, char *s, size_t size,
                      int *eofp);
char* gfs_pio_puts(GFS_FILE gf, char *s);
char* gfs_pio_putline(GFS_FILE gf, char *s);
```

gfs_pio_getline reads in at most one less than `size` characters from `gf` and stores them into the buffer pointed to by `s`. Readings stops after an `EOF` or a newline. If a newline is read, a `'\0'` is stored. If an `EOF` is read though no character is read, `eofp` is set. gfs_pio_puts writes the string `s` to `gf`. gfs_pio_putline writes the string `s` and a trailing newline to `gf`.

6 Gfarm Commands

The Gfarm commands facilitate manipulation of the Gfarm system. The commands can be executed on a Gfarm server and each Gfarm pool node. Table 1 is a list of major Gfarm commands. This list includes UNIX file manipulation commands and Gfarm administration commands. `gfls`, `gfmkdir` and `gfrmdir` manipulate file metadata of the Gfarm Meta Database. `gfrm`, `gfchmod`, `gfchown`, `gfchgrp` and `gfcp` access and modify file metadata and Gfarm fragments on Gfarm filesystem. `gfcd`, `gfchdir` and `gfpwd` can be used with a Gfarm shell.

`gfdf` reports number of free disk blocks and files on the entire Gfarm filesystem. `gfsck` checks the consistency between metadata of the Gfarm Meta Database and each Gfarm fragment and also between master data and the replica. The consistency between metadata and the corresponding Gfarm fragments can be broken by unexpected node or disk failure as well as unexpected user program termination. Since metadata is registered only after every Gfarm fragment is closed and every process is terminated, or at the checking point, the inconsistency is settled by deleting or removing the lost Gfarm fragments.

`gfimport` imports and scatters large-scale data from other filesystems or network. We plan to utilize GridFTP to retrieve data from network. `gfexport` gathers and exports the data to other filesystems or network. `gfdigest` outputs the message digest. `gfredist` redistribute Gfarm fragments for load balancing.

Table 1: Gfarm commands

`gfls`	List contents of directory
`gfmkdir`	Make directories
`gfrm`, `gfrmdir`	Remove directory entries
`gfchmod`	Change the permission mode of a file
`gfchown`, `gfchgrp`	Change file ownership
`gfcp`	Copy files
`gfcd`, `gfchdir`	Change working directory
`gfpwd`	Return working directory name
`gfdf`	Displays number of free disk blocks and files
`gfsck`	Check and repair file systems
`gfimport`	Import a file to Gfarm filesystem
`gfexport`	Export a file on Gfarm filesystem
`gfdigest`	Output message digest
`gfredist`	Redistribute a Gfarm file on Gfarm filesystem
`gfreg`	Register a file to Gfarm filesystem
`gfsched`	Create a host file

`gfreg` registers a file to the Gfarm filesystem. This function is only for legacy applications not to use Gfarm parallel I/O API. `gfsched` creates a list of hosts that stores fragments of given Gfarm URLs.

7 Implementation Status

Current status and schedule of the Grid Data Farm project are as follows, which will be closely synchronized with the CERN LHC "Data Challenge" practice to ensure the functionality and the scalability of the product.

7.1 Development schedule

Initial prototype system (2000 – 2001): The initial prototype system has such facilities as metadata management, data streaming, load balancing and the GridRPC. It will be deployed on a small (approximately 100 nodes) cluster and tested using Monte-carlo simulation data.

Second prototype system (2002 – 2003): Scalability is enhanced and fault tolerance facility is introduced in the second prototype system.

Full Production Development (2004 – 2005)

Deployment (2005 –): The Grid Data Farm system will be deployed on massive (several thousands nodes) PC clusters. It will be used to analyze peta-scale online data.

7.2 Target system example

Target system requires Petabyte of online storage. We are planning to adopt high-end PC technology to build the system. Each Gfarm pool node will have a 6Tbyte Raid 5 drive with 25 300GByte low power HD drives, 4-way over

10GFlops SMT 64bit CPUs, over 20GByte RAM, multi-channel, multi-gigabit LAN. The node is 4U, 500W power/box and active cooling capability. Total Gfarm system will be 20 chassis, 1.2 Petabytes, 8TFlops, 100KWatts, each chassis will be 60TByte, 40CPUs/40U and 5KWatts, and also the system has a 3PByte tape storage and direct multi-gigabit link into the network fabric.

8 Summary

Petabyte-scale data intensive computing wave of computational science surges over the high-performance computing. Grid and clustering technology offers viable solutions. Existing Grid infrastructure can be utilized, but further research and development required. The Grid Data Farm builds on the Grid technology of Grid-based RPC such as Ninf[7, 5] and lower level Grid service such as Globus[2] and provides Petascale global filesystem and parallel I/O to cope with such challenge. The Grid Data Farm project attempts to build a Petascale online storage system until 2005 synchronized with the CERN LHC project, though the Grid Data Farm system provides an effective solution to other data intensive applications such as bioinformatics, astronomy and earth science.

References

[1] MONARC Collaboration. Models of Networkd Analysis at Regional Centres for LHC experiments: Phase 2 report. Technical Report CERN/LCB-001, 2000. http://www.cern.ch/MONARC/.

[2] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

[3] John Linn. *Generic Security Service Application Program Interface Version 2, Update 1*, January 2000. RFC-2743.

[4] Satoshi Matsuoka, Hidemoto Nakada, Mitsuhisa Sato, and Satoshi Sekiguchi. Design issues of network enabled server systems for the Grid. *Lecture Notes in Computer Science*, 1971:4–17, 2000. Proceedings of Grid Computing – GRID 2000.

[5] Hidemoto Nakada, Mitsuhisa Sato, and Satoshi Sekiguchi. Design and implementations of Ninf: towards a global computing infrastructure. *Future Generation Computing Systems*, Metacomputing Issue, 1999.

[6] The Globus Project. *GridFTP: Universal Data Transfer for the Grid*. http://www.globus.org/datagrid/deliverables/C2WPdraft3.pdf.

[7] Satoshi Sekiguchi and Mitsuhisa Sato. World-wide computing infrastructure: Global and local partnership. *Proceedings of the second AIZU International Symposium on Parallel Algorithms / Architecture Synthesis*, pages 25–30, 1997.