Grid Datafarm Architecture for Petascale Data Intensive Computing[*]

Osamu Tatebe
National Institute of Advanced Industrial
Science and Technology (AIST)
o.tatebe@aist.go.jp

Youhei Morita
High Energy Accelerator
Research Organization (KEK)
youhei.morita@kek.jp

Satoshi Matsuoka
Tokyo Institute
of Technology
matsu@is.titech.ac.jp

Noriyuki Soda
Software Research
Associates, Inc.
soda@sra.co.jp

Satoshi Sekiguchi
National Institute
of Advanced Industrial
Science and Technology (AIST)
s.sekiguchi@aist.go.jp

Abstract

The Grid Datafarm (Gfarm) architecture is designed for global petascale data-intensive computing. It provides a global parallel filesystem with online petascale storage, scalable I/O bandwidth, and scalable parallel processing, and it can exploit local I/O in a grid of clusters with tens of thousands of nodes. Gfarm parallel I/O APIs and commands provide a single filesystem image and manipulate filesystem metadata consistently. Fault tolerance and load balancing are automatically managed by file duplication or re-computation using a command history log. Preliminary performance evaluation has shown scalable disk I/O and network bandwidth on 64 nodes of the Presto III Athlon cluster. The Gfarm parallel I/O write and read operations has achieved data transfer rates of 1.74 GB/s and 1.97 GB/s, respectively, using 64 cluster nodes. The Gfarm parallel file copy reached 443 MB/s with 23 parallel streams on the Myrinet 2000. The Gfarm architecture is expected to enable petascale data-intensive Grid computing with an I/O bandwidth scales to the TB/s range and scalable computational power.

1 Introduction

High-performance 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 raw data on the petabyte order from four large underground particle detectors each year, with data acquisition starting in 2006. Grid technology will play an essential role in constructing worldwide data-analysis environments where thousands of physicists will collaborate and compete in particle physics data analysis at new energy frontiers. To process such large amounts of data, a global computing model based on the multi-tier worldwide *Regional Centers* has been studied by the MONARC Project [11]. The model consists of a 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 research institutes.

The Grid Datafarm (Gfarm) is an architecture for petascale data-intensive computing on the Grid. Our model specifically targets applications where data primarily consists of a set of records or objects which are analyzed independently. Gfarm takes advantage of this access locality to achieve a scalable I/O bandwidth using an enhanced parallel filesystem integrated with process scheduling and file distribution. It provides a global, Grid-enabled, fault-tolerant parallel filesystem whose I/O bandwidth scales to the TB/s range, and which incorporates fast file transfer techniques and wide-area replica management.

2 Software Architecture of the Grid Datafarm

Large-scale data-intensive computing frequently involves a high degree of data access locality. To exploit this access locality, Gfarm schedules programs on nodes where the corresponding segments of data are stored to utilize local I/O scalability, rather than transferring the large-scale

[*]`http://datafarm.apgrid.org/`

data to compute nodes. Gfarm consists of the *Gfarm filesystem*, the *Gfarm process scheduler*, and *Gfarm parallel I/O APIs*. Together, these components provide a Grid-enabled solution to the class of data-intensive problems described above (and explained in detail in Section 3).

2.1 The Gfarm filesystem

The Gfarm filesystem is a parallel filesystem, provided as a Grid service for petascale data-intensive computing on clusters of thousands of nodes. Figure 1 depicts the components of the Gfarm filesystem, *Gfarm filesystem nodes* and *Gfarm metadata servers*, which provide a huge disk space in the petabyte range with scalable disk I/O bandwidth and fault tolerance. Each Gfarm filesystem node acts as both an I/O node and a compute node with a large local disk on the Grid.

The Gfarm filesystem is aimed at data-intensive computing that primarily reads one body of large-scale data with access locality. It provides a scalable read and write disk I/O bandwidth for large-scale input and output of data by integrating process scheduling and data distribution. For other files, the Gfarm filesystem works in almost the same way as a conventional parallel filesystem.

Note that we do not directly exploit SAN technology, which at a first glance might seem reasonable for facilitating storage on networks. The decision not to utilize SAN was based on several reasons. First, we need to achieve a TB/s-scale parallel I/O bandwidth, a range to which SAN technology is difficult and/or costly to scale. Second, because of the tight integration of storage with applications, as well as salient Grid properties such as scheduling, load balancing, fault tolerance, security, etc., having a separate I/O across the network independent from the compute nodes would be disadvantageous. Rather, we strove for tight coupling of storage to the computation to achieve both goals. Therefore, we adopt the “owner computes” strategy, or “move the computation to data” approach, rather than taking the other way round, for most data-intensive processing systems such as HPSS. We feel that for highly data-parallel applications our strategy is much more scalable, and is far better suited to the requirements of the Grid.

2.1.1 The Gfarm file

A *Gfarm file* is a large-scale file that is divided into fragments and distributed across the disks of the Gfarm filesystem, and which will be accessed in parallel. The Gfarm filesystem is an extension of a striping parallel system in that each file fragment has an arbitrary length and can be stored on any node.

A Gfarm file, specified by a *Gfarm filename* or a *Gfarm URL* such as `gfarm:/path/name`, is accessed using the

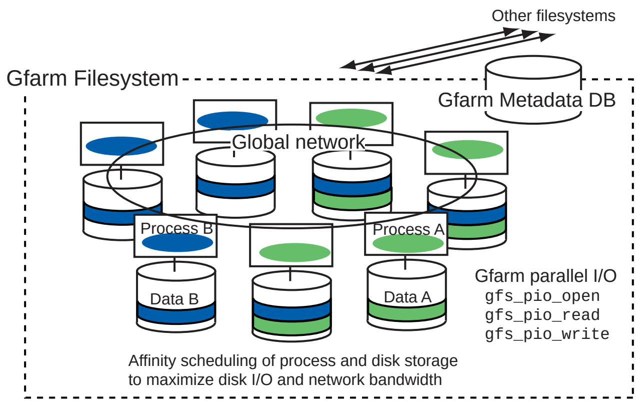

Figure 1. Software architecture of the Grid Datafarm

Gfarm parallel I/O library or Gfarm commands which provide a single-filesystem image.

Executable binaries for every execution platform can also be stored in the Gfarm filesystem. These executables can be accessed through the same Gfarm URL and selected depending on the execution platform.

Every Gfarm file is basically write-once. Applications are assumed to create a new file instead of updating an existing file. The Gfarm parallel I/O API supports read/write open, which is internally implemented by versioning and creating a new file. This is because 1) large-scale data is seldom updated (most data is write-once and read-many), and 2) data can be recovered by replication, or by re-computation using a command history log.

2.1.2 File replicas

Gfarm files may be replicated on an individual fragment basis by Gfarm commands and Gfarm I/O APIs manually or automatically. File replicas are managed by the Gfarm filesystem metadata. Since every Gfarm file is write-once, consistency management among replicas is not necessary. A replica is transparently accessed by a Gfarm URL depending on the file locations and disk access loads.

When a process needs to access data across nodes, there are two choices: replicate the fragment on its local disk, or access the fragment remotely by using a buffer cache. Which method is selected will depend on a hint for the Gfarm I/O API and node status.

File replicas are thus used not only for data recovery in the event of disk failure, but also to enable high bandwidth, low latency, and load balancing through their distribution over the Grid.

2.1.3 The Gfarm filesystem metadata

Metadata of the Gfarm filesystem is stored in the Gfarm metadata database. This consists of a mapping from a logical Gfarm filename to physically distributed fragment filenames, a replica catalog, and platform information such as the OS and CPU architecture, as well as file status information including file size, protection, and access/modification/change time-stamps.

Gfarm filesystem metadata also contains a file checksum and a command history. The checksum is mostly used to check data consistency when replicating, and the command history is used to re-compute the data when a node or a disk fails and to indicate how the data was generated.

Metadata is updated consistently with the corresponding file operations of the Gfarm filesystem. Generally, the metadata is referred to at the open operation and is updated and checked at the close operation. When one of the user processes terminates unexpectedly without registering metadata despite the fact that the other processes correctly registered their respective metadata, the metadata as a whole becomes invalid and will be deleted by the system.

2.1.4 Unified I/O and compute nodes

The *Gfarm filesystem daemon*, called *gfsd*, runs on each Gfarm filesystem node to facilitate remote file operations with access control in the Gfarm filesystem as well as user authentication, file replication, fast invocation, node resource status monitoring, and control.

In general, the parallel filesystem achieves high bandwidth when using parallel I/O nodes, which is limited by the network bandwidth or network bisection bandwidth. For petascale data, data rates in the GB/s range are too low to read data, since it would take more than 10 days to read the data. More bandwidth, in the TB/s range, is required, but has typically been costly to achieve.

Since each Gfarm filesystem node acts as both an I/O node and a compute node, it is not always necessary to transfer files from storage to compute nodes via the network. Gfarm exploits scalable local I/O bandwidth as much as possible by using Gfarm parallel I/O and the Gfarm process scheduler, and achieves TB/s rates by using tens of thousands of nodes, even though each node achieves rates of only tens of MB/s.

2.2 The Gfarm process scheduler and Gfarm parallel I/O APIs

To exploit the scalable local I/O bandwidth, the Gfarm process scheduler schedules Gfarm filesystem nodes used by a given Gfarm file for affinity scheduling of process and storage. In this case, the scheduler schedules the same number of nodes as the number of the Gfarm fragments, taking into consideration the physical locations of fragments of Gfarm files, the replica catalog, and Gfarm filesystem node status, and uses "owner computes" heuristics to maximize the usage of the local disk bandwidth.

Moreover, the Gfarm parallel I/O APIs provide a local file view in which each processor operates on its own file fragment of the Gfarm file. The local file view is also used for newly created Gfarm files.

It is possible to maximize the usage of the local disk bandwidth to achieve a scalable I/O bandwidth when parallel user processes utilizing the local file view are scheduled for a large-scale Gfarm file.

In the case of Figure 1, process A is scheduled and executed on the four nodes where fragments of data A are stored. Process B is scheduled on three nodes out of six because data B is divided into three fragments. Each fragment has a replica and either the master or its replica is chosen as a compute node. File replicas are used not only for file backup, but also for load balancing.

When a Gfarm filesystem node that stores a Gfarm fragment is heavily loaded, another node might be scheduled. On this node, the process will then 1) replicate the fragment to its local disk, or 2) access the fragment remotely.

At the same time, each Gfarm file can also be accessed as a large file in the same manner as a standard parallel filesystem such as PVFS [9].

2.3 Fast file transfer and replication

A Gfarm file is partitioned into fragments and distributed across the disks on Gfarm filesystem nodes. Fragments can be transferred and replicated in parallel by each gfsd using parallel streams. The rate of parallel file transfer might reach the full network bisection bandwidth. After replicating files, the filesystem metadata is updated for a new replica.

In the wide-area network, several TCP tuning techniques [17] such as adjustment of the congestion window size, the send and receive socket buffer size, and the number of parallel streams, are necessary to achieve high bandwidth. Gfarm handles this through inter-gfsd communication, and we plan to incorporate GridFTP [16] as an external interface.

2.4 File recovery and regeneration

File recovery is a critical issue for wide-area data-intensive computing, since disk and node failures are common, rather than exceptional, cases. Moreover, temporal shortages of storage often occur in such dynamic wide-area environments.

The Gfarm filesystem supports file replicas which are transparently accessed by a Gfarm URL as long as at least one replicated fragment is available for each fragment.

When there is no replica, Gfarm files are dynamically recovered by re-computation. Files are recovered when they are accessed. The necessary information for re-computation, such as a program and all arguments, is stored in the Gfarm metadata, and the program itself and all arguments including the content of a file are also stored in the Gfarm filesystem. It is possible to re-compute the lost file by using the same program and arguments as were used for the original file generation.

The GriPhyN virtual data concept [3] allows data to be generated dynamically, and existing data retrieved, through an application-specific high-level query. Gfarm can support the implementation of the GriPhyN virtual data concept at the filesystem level by using a dynamic regeneration feature, when naming convention from a high-level query to a filename and a command history of the filesystem metadata is appropriately set up.

To regenerate the same data, the program must be free of any timing bug such as nondeterministic behavior. To ensure the consistency of re-computation, when a process has opened a file for writing, or both reading and writing, other processes cannot open that file.

Gfarm metadata is not deleted for regenerating the file later even when the Gfarm file is deleted.

As described above, Gfarm files can be recovered through file replicas and regeneration using a command history as far back as the filesystem metadata exists. The metadata itself is replicated and distributed to avoid any single-point-of-failure and achieve scalable performance over a wide area, though consistency management of updated metadata is often necessary.

2.5 Grid authentication

To execute user applications or access Gfarm files on the Grid, a user must be authenticated by the Gfarm system, or the Grid, basically by using the Grid Security Infrastructure [4] for mutual authentication and single sign-on. However, the problem here is that the Gfarm system may require thousands of authentications and authorizations from amongst thousands of parallel user processes, the Gfarm metadata servers, and the Gfarm filesystem daemons, thus incurring substantial execution overhead. To suppress this overhead, the Gfarm system provides several lightweight authentication methods when full Grid authentication is not required, such as within a trusted cluster.

3 Grid Datafarm Applications

The Grid Datafarm supports large-scale data-intensive computing that achieves scalable disk I/O bandwidth and scalable computational power by exploiting the local I/O bandwidth of cluster nodes. Data-intensive applications include high energy physics, astronomy, space exploration, human genome analysis, as well as business applications such as data warehousing, e-commerce and e-government. The most time-consuming, but also the most typical, task in data-intensive computing is to analyze every data unit such as a record, an object, or an event within a large collection. Such an analysis can be typically performed independently on every data unit in parallel, or at least have good loci of locality. Data analysis for high energy experiments, the initial target application of Gfarm, is the most extreme case of such petascale data-intensive computing [12].

3.1 High Energy Physics Application

Data analysis in typical high energy experiments is often characterized as “finding a needle in a hay stack”. Each collision of particles in the accelerator is called an *event*. Information on thousands of particles emerging from the collision point is recorded by the 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 physically interesting ones, which are recorded into the storage media at a 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 mean finding events with certain special characteristics that occur on an order of several tens out of 10^{16} collisions.

Each event data consists of digitized numbers from subdetectors such as a calorimeter, silicon micro-strips, and tracking chambers. This initial recording of the event results in *RAW data*. In the ATLAS experiment, the amount of RAW data is approximately 1 to 3 Mbytes per event, corresponding to several petabytes of data storage per year. The digitized information in the RAW data is reconstructed into physically meaningful 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 the Tier-0 regional center at CERN. For the event reconstruction rate to keep up with the data taking, at least 150 K to 200 K SPECint95 processing power is required at the ATLAS Tier-0 center.

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

Because events are independent of each other, we can analyze the data independently on each CPU node in parallel. Only in the last stage of the analysis will a small set of statistical information need to be collected from every node. The data-parallel, distributed, and low-cost CPU-farm approach has been very popular 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 up against a new technical challenge regarding the design and maintenance. How to effectively distribute the large quantity of data to each CPU also remains a problem. Gfarm is designed to enable the handling of the large quantity of data localized in each CPU while the integrity of the data set is ensured by the filesystem metadata.

In the ATLAS data analysis software, object database technology will be used to store and retrieve data at various stages of analysis. One of the candidates for this task is a commercial database package, Objectivity, which has already been employed in production by the BaBar experiment at SLAC, and is already a core part of the software development in the CMS experiment of LHC. Gfarm has been designed to accommodate Objectivity as well, using system call trapping [12].

4 Gfarm Parallel I/O API

The Gfarm parallel I/O API enables parallel access to the Gfarm filesystem to achieve a scalable bandwidth by exploiting local I/O in a single system image in cooperation with Gfarm metadata servers.

All Gfarm files are divided into several indexed fragments and stored into several disks on Gfarm filesystem nodes. The Gfarm parallel I/O API provides several file views, such as a global file view and a local file view. The local file view restricts file accesses for a specific file fragment and exploits access locality.

Gfarm achieves a high bandwidth even for access in the global file view as a parallel filesystem, and also Gfarm achieves highly scalable bandwidth for access in the local file view for each file fragment. The Gfarm process scheduler schedules the same number of Gfarm filesystem nodes as the number of fragments of a given Gfarm file for affinity scheduling. Each node has its own file fragment in the local file view that is expected to be on its local disk. The local file view can also be applied to newly created files, which makes it possible to achieve a scalable bandwidth for writing to exploit the local I/O bandwidth.

The APIs described in this section are just a subset of the current interfaces for our first prototype; still, they reflect our design philosophy and architectural decisions. For a full description, refer to [14, 2].

4.1 File Manipulation

4.1.1 Opening and creating a file

```
char* gfs_pio_open(char *url, int flags,
    GFS_File *gf);
char* gfs_pio_create(char *url, int flags,
    mode_t mode, GFS_File *gf);
```

gfs_pio_open opens the Gfarm URL `url`, and returns a new Gfarm file handle `gf`. Values of `flags` are constructed by a bitwise-inclusive-OR of the following list. Exactly one of the first three values should be specified:

GFARM_FILE_RDONLY Open for reading only.

GFARM_FILE_WRONLY Open for writing only.

GFARM_FILE_RDWR Open for reading and writing.

The following may be specified as a hint for efficient execution:

GFARM_FILE_SEQUENTIAL File will be accessed sequentially.

GFARM_FILE_REPLICATION File may be replicated to a local filesystem when accessing remotely.

GFARM_FILE_NOT_REPLICATION File may not be replicated to a local filesystem when accessing remotely.

gfs_pio_create creates a new Gfarm URL `url` with the access mode `mode`, and returns a new Gfarm file handle `gf`. `Mode` specifies the file permissions to be created, and is modified by the process's `umask`.

gfs_pio_open and gfs_pio_create has individual file pointers among parallel processes.

4.1.2 File view

File view is a current set of data visible and accessible from an open file.

When a Gfarm file is used for the Gfarm parallel scheduler or is newly created, it may have a local file view such that each process accesses its own file fragment using the following API.

```
char* gfs_pio_set_view_local(GFS_File gf,
    int flags);
```

gfs_pio_set_view_local changes the process's view of the data in the file specified by the Gfarm URL `url` to the local file view. `flags` can be specified in the same way as are the flags for a hint of gfs_pio_open. When the file is a new file, the order of file fragments is the same as the order of process ranks.

The following API is used to explicitly specify a specific file fragment.

```
char* gfs_pio_set_view_index(GFS_File gf,
    int nfrags, int index, char *host,
    int flags);
```

gfs_pio_set_view_index changes the process's view of the data in the file specified by the Gfarm URL `url` to a file fragment with the index `index`. When the file is a new file, it is necessary to specify the total number of file fragments `nfrags` and the filesystem node `host`. When the file exists, `GFARM_FILE_DONTCARE` and `NULL` can be specified for `nfrags` and `host`, respectively.

4.2 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`.

4.3 Trapping system calls for porting legacy or commercial applications

To utilize a Gfarm filesystem from legacy or commercial applications whose source code is not available or cannot be modified, such as the Objectivity object database, system call trapping of file I/O operations is provided so that these applications can be readily parallelized in Gfarm. In this case, thousands of files are automatically grouped into a single Gfarm file when they are created with the trapped open, write, and close syscalls, which will be used for parallel process scheduling and automatic replica creation for dynamic load balancing and fault tolerance under the Gfarm filesystem management.

The open and creat syscalls check whether the given pathname is a Gfarm URL. When it is a Gfarm URL, the file view is changed to the local file view by gfs_pio_set_view_local and the file descriptor is registered to indicate the Gfarm file for the subsequent read and write syscalls.

5 Gfarm Commands

The Gfarm commands facilitate shell-level manipulation of the Gfarm filesystem, which provides most UNIX file manipulation commands and Gfarm administration commands. For a full description, see [14, 2].

`gfls`, `gfmkdir` and `gfrmdir` can be used to manipulate Gfarm filesystem metadata. `gfrm`, `gfchmod`, `gfchown`, `gfchgrp`, and `gfcp` access and modify file metadata and Gfarm fragments on a Gfarm filesystem.

`gfimport` imports and scatters large-scale data from other filesystems or from the network, while `gfexport` gathers and exports the same sort of data. Since the most effective means of scattering data or a file is basically application-dependent, typical cases such as block striping and line-oriented partition are provided and these can be used as a skeleton code for application-dependent partitioning. To permit efficient interaction with other conventional filesystems or network streams, an adaptor for GridFTP [16] is currently being developed.

6 Performance Evaluation

The basic performance of Gfarm parallel I/O is evaluated on the Presto III Athlon cluster at Tokyo Institute of Technology, where each node of the cluster consisted of a dual AMD Athlon MP 1.2GHz processor, 768MB memory, and 200GB HDDs. There are a total of 128 nodes, and 256 processors, interconnected with Myrinet 2000 and Fast Ethernet. The Linpack HPC benchmark achieves 331.7 GFlops out of a theoretical peak performance of 614.4 GFlops.

6.1 Disk I/O bandwidth

Gfarm provides scalable I/O bandwidth for reading a primary large-scale file using affinity scheduling and local file view, and scalable I/O bandwidth for writing new files in local file view.

Figure 2 shows an excerpt of a program for measuring the Gfarm parallel I/O bandwidth for writing. This program creates a new Gfarm file `fn` and changes the file view to the local file view, which is expected to create a fragment of the Gfarm file on each local disk if sufficient space is available. The data `buf` is written to the new file in parallel using gfs_pio_write. Finally, the Gfarm file is closed with gfs_pio_close, which also registers the filesystem metadata of the Gfarm file. For simplicity, Figure 2 calls gfs_pio_write only once using the whole buffer size; however, an actual benchmark program will repeatedly call gfs_pio_write with the same 64KB buffer.

For reading, parallel processes are scheduled by the Gfarm file `fn`, then each process opens the file and changes the file view to the local file view. In the performance measurement, each process is scheduled on the node where the corresponding fragment is stored, although this is not always the case in general use. The data is read from the file in parallel using gfs_pio_read. Finally, the Gfarm file is

```
write_test(char *fn, void *buf, int size)
{
    GFS_File gf;
    gfs_pio_create(fn, GFS_FILE_WRONLY,
                   mode, &gf);
    gfs_pio_set_view_local(gf, lflag);
    gfs_pio_write(gf, buf, size, &np);
    gfs_pio_close(gf);
}
```

Figure 2. An excerpt to measure Gfarm parallel I/O bandwidth

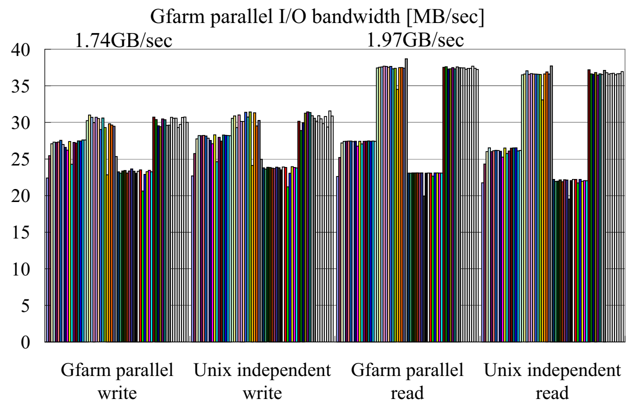

Figure 3. Gfarm parallel I/O performance

closed with gfs_pio_close, which checks for data error by using the md5 checksum.

Figure 3 shows the Gfarm parallel I/O performance with a total of 640 GB of data on 64 cluster nodes. Each process accesses 10 GB of data, which is much more than the 768 MB of main memory, to measure the disk I/O performance while minimizing the influence of memory buffering. The performance is measured from the open operation to the close operation in Figure 2, which includes the overhead of accessing the Gfarm filesystem metadata and calculating the md5 checksum.

The `Gfarm parallel write` writes a total of 640 GB of data in parallel and achieves an aggregate bandwidth of 1.74 GB/s. Each node achieves a bandwidth of 27.2 MB/s on average. Presto III cluster node has either of two kinds of disks; Seagate ST380021A and Maxtor 33073H3, which shows a different disk I/O performance.

The `Unix independent write` shows the combined bandwidth of an independent write syscall on each node. Note that the performance difference is negligible even though the bandwidth of the Gfarm parallel write includes the overhead of accessing the metadata and calculating the md5 checksum.

The `Gfarm parallel read` reads 640 GB of data in parallel and achieves a total bandwidth of 1.97 GB/s, with each node achieving a bandwidth of 30.8 MB/s on average. Since a simple read syscall on each node achieves a bandwidth of 29.9 MB/s, so the difference is again negligible.

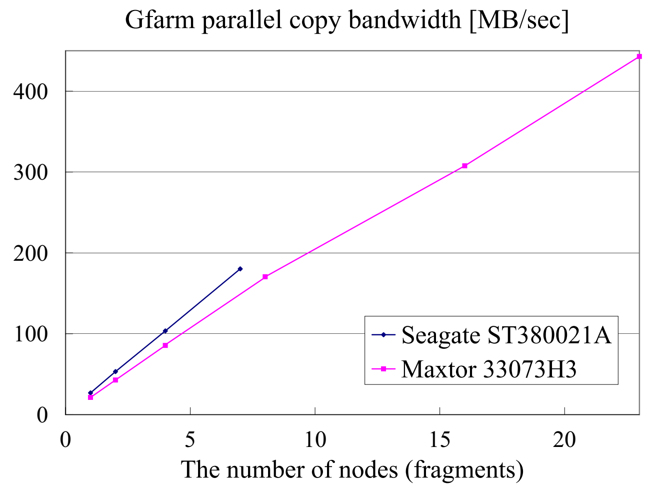

Figure 4. Gfarm parallel file replication performance

The performance measurement has therefore shown that the Gfarm parallel bandwidth scales at least up to 64 nodes, and the overhead of accessing the metadata and calculating the checksum is not significant.

6.2 Parallel file replication and copy

A Gfarm file is partitioned into fragments and distributed across the disks on Gfarm filesystem nodes. Fragments can be transferred and replicated in parallel.

Figure 4 shows the bandwidth to replicate a Gfarm file with a fragment size of 10 GB using the Gfarm command `gfrep`. File size of Gfarm files increase in proportion to the number of nodes or fragments.

A Gfarm file is replicated in parallel through a Myrinet 2000. The Myrinet 2000 has a bandwidth of about 130 MB/s, however the copy bandwidth of each stream is limited by a disk I/O bandwidth of 26 MB/s for Seagate or 21 MB/s for Maxtor. As shown in Figure 4, Gfarm parallel file replication achieves 443 MB/s using 23 parallel streams on the Myrinet 2000.

Although `gfrep` includes the overhead of invoking the copy operations and updating the Gfarm filesystem metadata, the copy bandwidth scales at least up to 23 nodes on the Presto III cluster.

7 Related Work

MPI-IO [10] is the standard interface for parallel file access, however it does not define the local file view provided by Gfarm filesystem, which is a key issue to maximize local I/O scalability.

PVFS [9] is a striping parallel filesystem that utilizes the local disks of a Linux cluster, and which supports UNIX/POSIX I/O APIs and MPI-IO as well as native PVFS APIs. Since the striping filesystem does not take into account the affinity of process and disk storage, the bandwidth is often limited by the network bandwidth as reported by [9]. On the other hand, Gfarm parallel I/O achieves a scalable parallel I/O bandwidth by utilizing the affinity of process and disk storage as much as possible. Moreover, the Gfarm filesystem supports Grid security and fault tolerance by file duplication or re-computation on a cluster of clusters with thousands of nodes on the Grid.

HPSS [5] is a hierarchical mass storage system with parallel I/O that uses striping disk caches and parallel movers as are typically used by parallel FTP, MPI-IO, and DFS. Since HPSS does not support any form of disk-side or mover-side computation, all data must be moved through a network before computation, which means that the system bandwidth is also limited by the network bandwidth.

Distributed filesystems, such as NFS, AFS, Coda, xFS [7] and GFS [13], target situations where many distributed clients efficiently access files by using file caches, etc. Unfortunately, distributed filesystems cannot achieve sufficient bandwidth for write operations requiring a GB/s bandwidth, which is typically needed for data-intensive computing.

Several systems that are equipped with Grid-aware file accesses, such as GridFTP [16], Legion I/O [18] and Kangaroo [15], enable access to remote files on the Grid in a similar manner to distributed filesystems, albeit in a more loosely coupled manner for wide-area networks. GridFTP [16] is an FTP extension to the Grid Security Infrastructure [4] and facilitates adaptive parallel streams to maximize the bandwidth in a wide-area network. Kangaroo copes with recoverable errors as much as possible to ensure highly reliable execution, and hides latency by using a local disk as a disk cache. The Gfarm filesystem aims to maximize the bandwidth both between systems in a wide-area network and within a system by minimizing data movement and effectively sending computation to the nodes.

Globus replica management [6] provides metadata management of replicated files on a Grid, choosing the best possible replica to allow efficient access from a remote site. It consists of low-level Replica Catalog APIs that manage the metadata and high-level Replica Management APIs. The Replica Catalog APIs provide low-level replica catalog manipulations such as creating a replica entry, and do not ensure consistency between a metadata and a physical file. The Replica Management API is designed to cope with this issue using GridFTP and Replica Catalog APIs, which are available in the Globus Toolkit 2 release [1].

8 Implementation Status and Development Schedule

The initial prototype system implemented almost all Gfarm parallel I/O APIs and several indispensable Gfarm shell-level commands [14] including sufficient system call trapping to utilize the Gfarm filesystem with the Objectivity object database on Linux, Solaris, NetBSD, and Tru64. A light-weight authentication has been implemented based on a secret shared key assuming a trusted environment for delivering the key. The Gfarm metadata server uses the OpenLDAP server. The Gfarm filesystem daemon has facilities for fast remote-file manipulation, fast remote execution, third-party file transfer between the Gfarm filesystem daemons, and load average monitoring. It has been deployed on the Presto III and other smaller scale clusters and is currently being tested using Monte Carlo simulation data.

The current schedule for the Grid Datafarm project is as follows. It will be closely synchronized with the CERN LHC Data Challenge practice to ensure the functionality and the scalability of the product.

Second prototype system (2002 – 2003): Process scheduling will be incorporated with load balancing and fault tolerance using runtime replica creation. File recovery using re-computation will be fully supported. Security will be enhanced for the Grid environment via GSI and a bridge to GSI. Scalability up to thousands of nodes will be achieved with a fast-startup mechanism in a similar manner to that for the multipurpose daemon (MPD) [8]. Multiple Gfarm metadata servers with replicated metadata will be consistently operated to provide fault tolerance for the metadata database and efficient metadata access from different sites.

Deployment (2004 –): Gfarm will be fully deployed on the production platform to analyze petascale online data. The current cluster design is as follows. Each Gfarm node will have a 5-TByte Raid-5 drive with 28 200-GByte low-power 2.5” HD drives, 4-way over-5-GFlops 64-bit CPUs, over-20-GByte RAM, and a multi-channel, multi-gigabit LAN. The node is a 1U box, 200-250 W power/box with active cooling. The Gfarm cluster will consist of 20 chassis, 4 petabytes, 16 TFlops, 200 KWatts, each chassis with 200 TByte, 160 CPUs/40 U and 10 KWatts, which also has a 3-PByte tape storage and a direct multi-gigabit link to the network fabric.

9 Summary and future work

The Grid Datafarm provides a huge disk space of over a petabyte with scalable disk I/O, network bandwidth, and fault tolerance that can be used as a Grid service for petascale data-intensive computing using high-end PC technology. The idea is to utilize local I/O bandwidth as much as possible. To achieve this, we have parallel computation move to the data, and not vice versa as is with other efforts, abiding by the "owner computes" approach. Storage and computation are tightly integrated to facilitate smooth and synergetic scheduling, load-balancing, fault-tolerance, security, etc., which are all necessary properties for the computation to scale to the Grid.

Preliminary performance evaluation showed that scalable disk I/O and network bandwidth on the 64 nodes of the Presto III Athlon cluster could be achieved. The Gfarm parallel I/O write and read achieved 1.74 GB/s and 1.97 GB/s, respectively, when using 64 nodes. The Gfarm parallel file copy achieved 443 MB/s with 23 parallel streams on the Myrinet 2000.

We are now trying to evaluate and improve the performance of the Gfarm system on a cluster with hundreds of nodes, and will also evaluate the performance between Gfarm clusters connected via a gigabit-scale wide-area network using a Gfarm parallel copy and GridFTP. We believe the Gfarm architecture can achieve a scalable I/O bandwidth of over 10 TB/s with corresponding scalable computational power on the Grid at the same time.

Currently, Gfarm parallel I/O APIs are provided as a minimal set based on the synchronous POSIX model. We plan to add nonblocking interfaces and to integrate the MPI-IO interface.

The goal of the Grid Datafarm project to build a petascale online storage system by 2005 that is synchronized with the CERN LHC project. Moreover, the Grid Datafarm system will provide an effective solution to other data-intensive applications such as bioinformatics, astronomy, and earth science.

Acknowledgments

We are grateful to the reviewers for their valuable comments. 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] *Globus Toolkit 2 release*. http://www.globus.org/gt2/.

[2] *Grid Datafarm*. http://datafarm.apgrid.org/.

[3] *Grid Physics Network*. http://www.griphyn.org/.

[4] *The Grid Security Infrastructure Working Group*. http://www.gridforum.org/security/gsi/index.html.

[5] *HPSS: High Performance Storage System*. http://www.sdsc.edu/hpss/.

[6] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Proceedings of IEEE Mass Storage Conference*, 2001.

[7] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. In *Proceedings of the Fifteenth ACM Symposium of Operating Systems Principles*, pages 109–126, 1995.

[8] R. Butler, W. Gropp, and E. Lusk. A scalable process-management environment for parallel programs. Technical Report MCS-P812-0400, ANL, April 2000.

[9] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, 2000.

[10] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, July 1997. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html.

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

[12] Y. Morita, O. Tatebe, S. Matsuoka, N. Soda, H. Sato, Y. Tanaka, S. Sekiguchi, S. Kawabata, Y. Watase, M. Imori, and T. Kobayashi. Grid data farm for Atlas simulation data challenges. In *Proceedings of International Conference on Computing of High Energy and Nuclear Physics*, pages 699–701, 2001.

[13] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The global file system. In *Proceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, 1996. http://www.globalfilesystem.org/.

[14] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, H. Sato, Y. Tanaka, S. Sekiguchi, Y. Watase, M. Imori, and T. Kobayashi. Grid data farm for petascale data intensive computing. Technical Report ETL-TR2001-4, Electrotechnical Laboratory, March 2001. http://datafarm.apgrid.org/pdf/gfarm-ETL-TR2001-4.pdf.

[15] D. Thain, J. Basney, S.-C. Son, and M. Livny. The Kangaroo approach to data movement on the grid. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing*, pages 325–333, 2001.

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

[17] B. L. Tierney. *TCP Tuning Guide for Distributed Application on Wide Area Networks*. http://www-didc.lbl.gov/tcp-wan.html.

[18] B. S. White, A. S. Grimshaw, and A. Nguyen-Tuong. Grid-based file access: The Legion I/O model. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 165–173, 2000.