International Journal of Open Publication and Exploration (IJOPE), ISSN: 3006-2853
Volume 6, Issue 1, January-June, 2018, Available online at: https://ijope.com

Data-Intensive Computing on Grid Computing Environment

Palak Raina[1], Hitali Shah[2]

[1,2]Institute of Technology, Nirma University, India

ABSTRACT

Grid computing raises challenging issues in many areas of computer science, bioinformatics, high energy physics and especially in the area of distributed computing, as Computational Grids cover increasingly large data, networks and span many organizations. In Grid computing environment data-intensive applications involve large overhead costs due to a concentration of access to the files on common nodes. To avoid this problem in traditional distributed filesystems, users have to distribute the file access manually. However, such solution has some difficulties for users in the Grid environment. We propose a data management mechanism for data-intensive computing on Grid filesystem. Our technique improves the file access performance by automatically scheduling the file access and the data management on the filesystem. The filesystem is based on dynamically configured node groups corresponding to the network topology. Utilizing the configuration, it monitors file access to detect concentrated situations, creates the file replica, and schedules its placement and access. We applied the proposal technique to the Gfarm, a filesystem that scales to the Grid. We emulate real application workloads using a job scheduler and confirmed a speedup of factor 3.7 compared with a filesystem without automatic file access distribution techniques.

Keywords—Grid computing, Data management, Data Integration, Heterogeneous resources, Data intensive computing, Gfarm.

INTRODUCTION

Data-intensive applications on the Grid, such as highenergy physics, astronomy, life-science, etc., require to share and process a large amount of data generated by experiments or observations among multi-sites. In the environments, network shared filesystems, such as NFS, AFS, and cluster filesystems[4][3][5], are widely used to provide data accessibility and single system image, but there are some difficulties in using traditional distributed filesystems on the Grid . Some data-intensive applications involve large overhead costs due to a concentration of access to the files on common nodes. To avoid this problem in exsisting distributed filesystems, users have to distribute the file access manually. However, such solution is difficult for users in the Grid environment. To solve this problem, we propose a data management mechanism for data-intensive computing on Grid filesystem. Our technique improves the file access performance by automatically scheduling the file access and the data management on the filesystem. The filesystem is based on dynamically configured node groups corresponding to the network topology. Utilizing the configuration, it monitors file access to detect concentrated situations, creates the file replica, and schedules its placement and access. We applied the proposal technique to the Gfarm, a filesystem that scales to the Grid. We emulate real application workloads using a job scheduler and confirmed a speedup of factor 3.7 compared with a filesystem without automatic file access distribution techniques.

Grid Computing

There has been a surge of interest in grid computing, a way to enlist large numbers of machines to work on multipart computational problems such as circuit analysis or mechanical design. There are excellent reasons for this attention among scientists, engineers, and business executives. Grid computing enables the use and pooling of computer and data resources to solve complex mathematical problems. The technique is the latest development in an evolution that earlier brought forth such advances as distributed computing, the Worldwide Web, and collaborative computing. Grid computing harnesses a diverse array of machines and other resources to rapidly process and solve problems beyond an organizations available capacity. Academic and government researchers have used it for several years to solve large-scale problems, and the private sector is increasingly adopting the technology to create innovative products and services, reduce time to market, and enhance business processes. The term grid, however, may mean different things to different people. To some users, a grid is any network of machines, including personal or desktop computers within an organization. To others, grids are networks that include computer clusters, clusters of clusters, or special data sources. Both of these definitions reflect a desire to take advantage of vastly powerful but inexpensive networked resources. In our work, we focus on the use of grids to perform computations as opposed to accessing data, another important area known as data grid research [8].

Different Systems

Grid computing is akin to established technologies such as computer clusters and peer-to-peer computing in some ways and unlike them in others. Peer-to-peer computing, for example, allows the sharing of files, as do grids, but grids enable users to share other resources as well.Computer clusters and distributed computing require a close proximity and operating homogeneity; grids allow computation over wide geographic areas using computers that are heterogeneous.

Grids are usually heterogeneous Networks and Data

Grids are usually heterogeneous networks. Grid nodes, generally individual computers, consist of different hardware

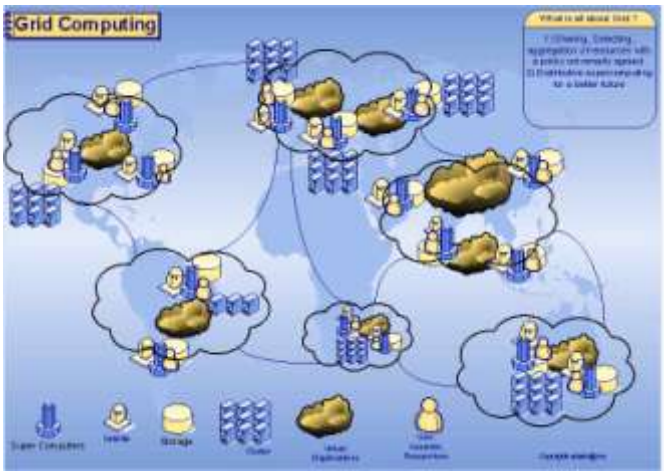

Fig. 1: Grid System

and use a variety of operating systems, and the networks connecting them vary in bandwidth. Realizing the vision of ubiquitous parallel computing on a grid will require that we make grids easy to use, and this need applies both to the creation of new applications and to the distribution and management of applications on the grid itself. To accomplish this goal, we need to establish standards and protocols such as open grid services architecturewhich allows communication across a network of heterogeneous machinesand tool kits such as Globus, which implement the rules of the grid architecture.

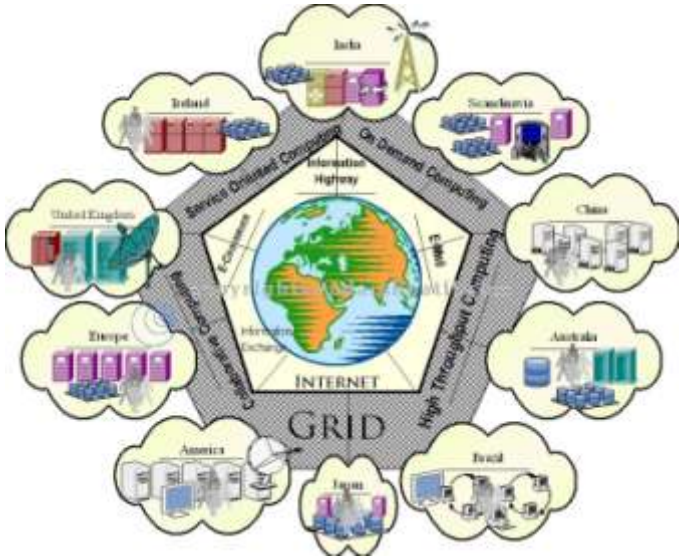

Fig. 2: Heterogeneous Network

Data Management

Data Services A grid fundamentally consists of two distinct parts, compute and data:

Compute gridprovides the core resource and task management services for grid computing: sharing, management, and distribution of tasks based on configurable service-level policies.

Data gridgridprovides the data management features to enable data access, synchronization, and distribution of a grid. Efficient access to and movement of huge quantities of data is required in more and more fields of science and technology. In addition, data sharing is important, for example enabling access to information stored in databases that are managed and administered independently. In business areas, archiving of data and data management are essential requirements.

Objectives

Data services are used to move data to where it is needed, manage replicated copies, run queries and updates, and transform data into new formats. They also provide the capabilities necessary to manage the metadata that describes OGSA data services or other data, in particular the provenance of the data itself. Data services requirements include:

Data access: Easy and efficient access to various types of data (such as database, files, and streams), independent of its physical location or platform, by abstracting underlying data sources is required. Mechanisms are also required for controlling access rights at different levels of granularity.

Data consistency: OGSA must ensure that consistency can be maintained when cached or replicated data is modified.

Data persistency: Data and its association with its metadata should be maintained for their entire lifetime. It should be possible to use multiple persistency models.

Data integration: OGSA should provide mechanisms for integrating heterogeneous, federated and distributed data. It is also required to be able to search data available in various formats in a uniform way.

Data location management: The required data should be made available at the requested location. OGSA should allow for selection in various ways, such as transfer, copying, and caching, according to the nature of data[4].

Problem of Filesystems on the Grid

There are a lot of advantages in the use of the network filesystem to federate resources on the Grid. It can provide not only data sharing among multi-sites but also single system image. The requirements of the filesystem on the Grid are mainly considered Security and Scalability. NFS based filesystems combined with security mechanisms [6] support secure data sharing on a wide area network. However, these systems basically consist of a single node, causing possible performance bottleneck. Therefore, filesystems of this kind arent suitable for the usage on the Grid in the point of scalability. Striping parallel filesystems[4][3][5] are mainly used in HPC cluster environments to gain better I/O performance. All files are divided into fixed-size chunks, and each chunk can be placed in any storage node. However, the performance

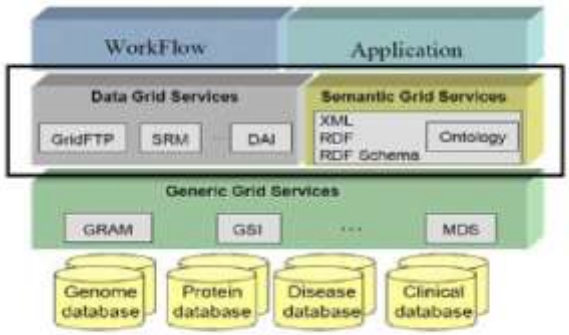

Fig. 3: Data Management

of these filesystems can often be limited by the network bandwidth, and most of these filesystems dont support security mechanisms that should satisfy the requirements on the Grid. Grid Datafarm[7] is an architecture for petascale data intensive computing on the Grid. This system not only provides data sharing on the Grid but also schedules programs on nodes where the corresponding segments of data are stored to utilize local I/O scalability, rather than transferring the large-scale data to compute nodes. However, users need to manage data in the filesystem manually to improve I/O performance in the heterogeneous resources. Moreover, even if one assume the use of the filesystems such as Grid Datarfarm, the situation

that a concentration of file access from many clients to a single node and that access to file on remote sites may occur, which can be a performance bottleneck for applications running on the filesystems.

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 data to compute nodes. Gfarm consists of the Gfarm filesystem, the Gfarm process scheduler, and Gfarm parallel I/O APIs.

The Gfarm file system

The Gfarm filesystem is a parallel filesystem, provided as a Grid service for petascale data-intensive computing on clusters of thousands of nodes. Figure 4 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

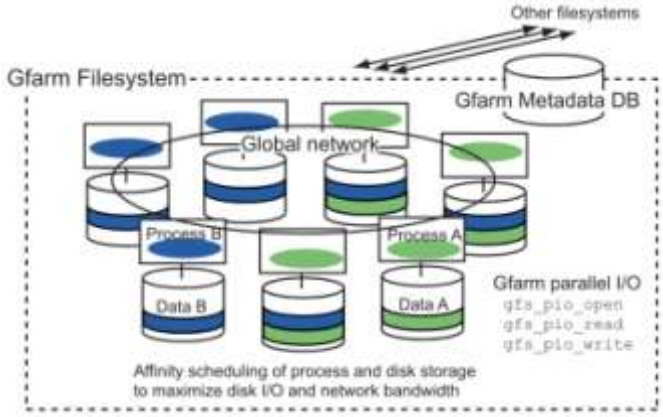

Fig. 4: Software architecture of the Grid Datafarm Gfarm [3]

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/sscale 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.

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 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 recomputation using a command history log. 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.

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 recompute 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.

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.

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 , 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].

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

File recovery and regeneration

File recovery is a critical issue for wide-area dataintensive 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 recomputation. Files are recovered when they are accessed. The necessary information for recomputation, 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 recompute 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 recomputation, 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.

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.

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].

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 109 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 offline analysis. During the first year of the accelerator run, an order of 1016 collisions will be observed and 109 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 1016 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 Tier1 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 largescale 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].

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.

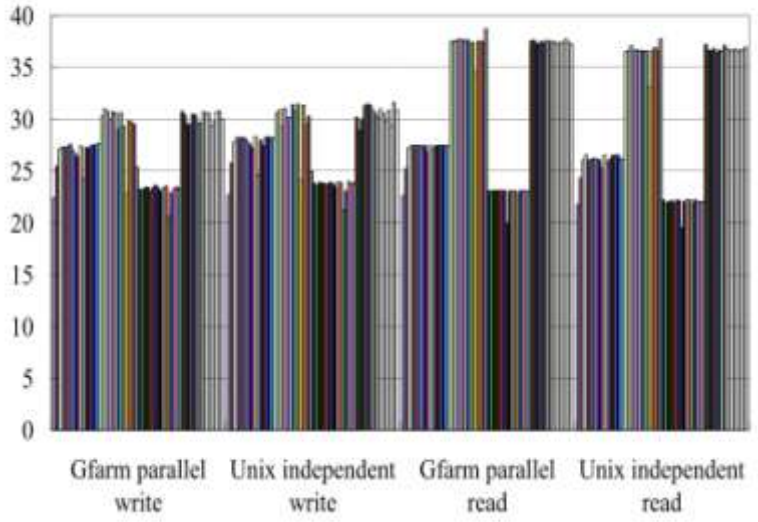

Fig. 5: Gfarm parallel I/O performance [3]

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 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 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. Figure 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, 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

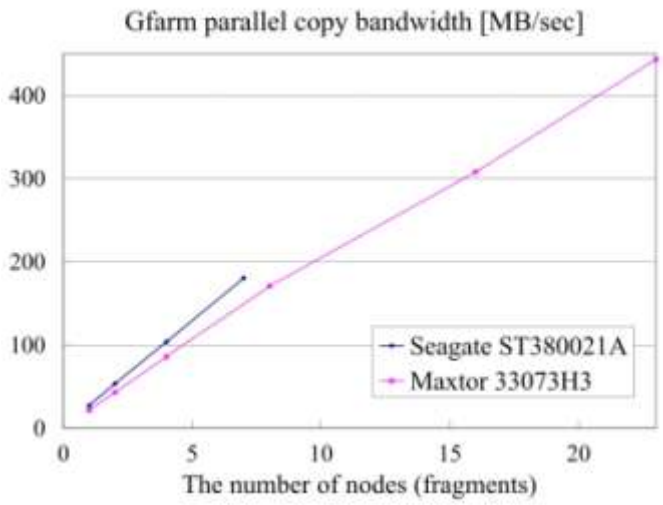

Fig. 6: Gfarm parallel file replication performance [3]

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. 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.

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 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 , 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.

CONCLUSION

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 MPIIO 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 dataintensive applications such as bioinformatics, astronomy, and earth science.

Acknowledgment
I am grateful to the Nirma University for providing access to IEEE and ACM digital library which helped me to access different research papers.Special thanks to Dr.Madhuri Bhavsar who helped me throughout the entire process of writing the term paper.

REFERENCES

[1]. Ioannidis, Yannis. ”Data management over the GRID: heaven or hell?.” Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on. IEEE, 2004.
[2]. Tatebe, Osamu, et al. ”Grid datafarm architecture for petascale data intensive computing.” Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on. IEEE, 2002.
[3]. Sato, Hitoshi, and Satoshi Matsuoka. ”Data management on grid filesystem for data-intensive computing.” Applications and the Internet Workshops, 2007. SAINT Workshops 2007. International Symposium on. IEEE, 2007.
[4]. Miceli, Chris, et al. ”Programming abstractions for data intensive computing on clouds and grids.” Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. IEEE Computer Society, 2009.
[5]. Sravan Kumar Pala, “Advance Analytics for Reporting and Creating Dashboards with Tools like SSIS, Visual Analytics and Tableau”, IJOPE, vol. 5, no. 2, pp. 34–39, Jul. 2017. Available: https://ijope.com/index.php/home/article/view/109
[6]. Globus Toolkit 2 release. http://www.globus.org/gt2/.
[7]. Yu, Jia, and Rajkumar Buyya. ”A taxonomy of scientific workflow systems for grid computing.” ACM Sigmod Record 34.3 (2005): 44-49. Grid Datafarm. http://datafarm.apgrid.org/.
[8]. Grid Physics Network. http://www.griphyn.org/.
[9]. The Grid Security Infrastructure Working Group . http://www.gridforum.org/security/gsi/index.html.
[10]. HPSS: High Performance Storage System . http://www.sdsc.edu/hpss/. [10] Cao, Junwei, et al. ”Gridflow: Workflow management for grid computing.” Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on. IEEE, 2003.
[11]. Sravan Kumar Pala, “Synthesis, characterization and wound healing imitation of Fe3O4 magnetic nanoparticle grafted by natural products”, Texas A&M University - Kingsville ProQuest Dissertations Publishing, 2014. 1572860. Available online at: https://www.proquest.com/openview/636d984c6e4a07d16be2960caa1f30c2/1?pq-origsite=gscholar&cbl=18750
[12]. Figueiredo, Renato J., Peter Dinda, and Jose Fortes. ”A case for grid computing on virtual machines.” Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on. IEEE, 2003.
[13]. Anderson, David P. ”Boinc: A system for public-resource computing and storage.” Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on. IEEE, 2004.
[14]. Menasc, Daniel, and Emiliano Casalicchio. ”QoS in grid computing.” Internet Computing, IEEE 8.4 (2004): 85-87.
[15]. Azzedin, Farag, and Muthucumaru Maheswaran. ”Towards trust-aware resource management in grid computing systems.” Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on. IEEE, 2002.
[16]. Gfarm. http://datafarm.apgrid.org/.
[17]. Lustre. http://www.lustre.org.
[18]. SAGA Project Page, http://saga.cct.lsu.edu.
[19]. NIMBUS http://workspace.globus.org/.