Gvu: A View-Oriented Framework for Data Management in Grid Environments

Pradeep Padala and Kang G. Shin
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122, USA
Email: {ppadala,kgshin}@umich.edu

Abstract

In a grid, data is stored in geographically-dispersed virtual organizations with varying administrative policies and structures. Current grid middleware provide basic data-management services including data access, transfer and simple replica management. Grid applications often require much more sophisticated and flexible mechanisms for manipulating data than these, including logical hierarchical namespace, automatic replica management and automatic latency management. We propose a view-oriented framework that builds on top of existing middleware and provides global and application-specific logical hierarchical views. Specifically, we developed mechanisms to create, maintain, and update these views. The views are synchronized using an efficient group communication protocol. Gvu (pronounced G-view) is built as a distributed set of synchronized servers and scales much better than the existing grid services. We conducted experiments to measure various aspects of Gvu and report on the results, showing Gvu to outperform existing grid services, thanks to its distributed nature.

I. Introduction

Grids [1] have become the favorite choice for executing data-intensive scientific applications. Scientific applications in domains, such as high energy physics, bioinformatics, medical image processing and earth observations, often analyze and produce massive amounts of data (sometimes of the order of petabytes). The applications access and manipulate data stored in various sites on the grid. They also have to distribute and publish the derived data.

The work reported in this paper was supported in part by the NSF under grant CCF 0444417

Let's consider how a typical scientific application interacts with the data grid.

1) A physicist participating in a high-energy physics experiment would like to execute a CMS (Compact Muon Solenoid) application.
2) The application requires various input files. It has to find the location of files using a catalog, index or database system where information about the location of the file is stored. The application usually uses a logical file name (LFN) to index into the catalog and find the physical location.
3) The files may have to be replicated at various sites in the grid for the application to find a nearby location to quickly access the file.
4) The physicist, having gathered all the information about the input files, runs the jobs on various sites. If a site doesn't have the required input data, the data is pre-fetched before the job starts.
5) The jobs execute using the input files and produce derived data.
6) The derived data needs to be distributed to various sites on the grid for usage by other scientists and for archival purposes.
7) Finally, the output data locations have to be published in a catalog so that other scientists can locate the data.

Using current middleware and grid file systems as they currently exist, the above scenario requires the application to perform complex interactions with grid services. Globus [2] middleware, one of the most popular grid toolkits, provides data management mechanisms including GridFTP [3] and RLS (Replica Location Service) [4]. GridFTP is an enhanced version of the popular File Transfer Protocol (FTP) that provides high performance using parallel streams, parallel file transfers, command pipelining, etc.

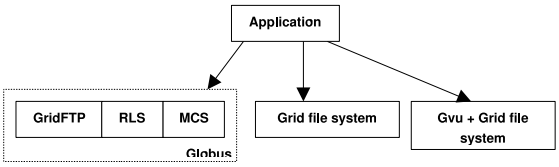


Fig. 1. Data-management mechanisms

To realize the above scenario, ad-hoc mechanisms using GridFTP, RLS, and metadata catalog services (MCS) [5] can be developed. Unfortunately, these mechanisms lack flexibility and power.

Therefore, the key research question is: *What are the data management requirements of typical workloads in grid environments and how do we provide flexible and powerful mechanisms for manipulating data?* Thain *et al.* [6] surveyed six scientific application workloads run in grid environments and concluded that traditional distributed file systems are inefficient for the *batch-pipelined* nature of these workloads.

Main characteristics of application data requirements are:

- Typical workloads include *pipeline-shared*, *batch-shared* and hybrid workflows;
- Pipeline-shared outputs require mechanisms for discovery of outputs by a reader, but the output need not be advertised to the same degree as in batch-shared data [6];
- The workflows use certain metadata (specific to the application) to find data on the grid;
- Most files are accessed as a whole, and block-level reads and writes are uncommon;
- Files are accessed with a *write-once and read-many* pattern; and
- Applications usually access a limited set of files distributed over the grid.

Figure 1 shows various data-management mechanisms currently available in grid environments. On one hand, data-management facilities like replica location and metadata management can be provided by different services that can be combined in various ways depending on the application. On the other hand, one can develop a unified grid file system that provides a consistent file-system-like interface to the application. Researchers [7], [8], [9] have worked on providing a file system-like interface to the data on the grid (a detailed comparison is provided in the next section). Although there is no consensus on the grid file system interfaces, these efforts have succeeded in providing uniform access to heterogeneous storage systems distributed over a grid. Certain key features that are missing are global hierarchical name space and application- and user-specific views of the data.

Why do we need a global hierarchical name space? If we consider the scenario explained earlier, jobs of an application running on different sites can see others' files as they are created in a global hierarchical tree.

Why do we need a logical name space? Data in a grid is stored in various sites at different physical locations. It would be more flexible for an application to refer to the data using a logical name instead of a complicated physical name that might change over time. In a single administrative domain, creating this logical hierarchical name space is easy. NFS (Network File System) [10] already provides a simple, though inflexible, mechanism for doing this. In a grid, the data is scattered in different virtual organizations (VOs). The key research question is: *How do we provide this view without losing the autonomous nature of the sites and still maintain flexibility?*

Consider a grid file system that provides logical global hierarchical name space. Would that solve all the problems in the above scenario? Not completely. Consider the situation where an application manipulates thousands of files and produces many more files. How do we allow flexible access by other applications which want to use the same data? With the existing tools, this would be a nightmare. The Virtual Data System (VDS) [11] provides a convenient way of maintaining and querying *recipes* for data derivations [11], but it does not provide a way of finding and creating files. A view that contains only the files manipulated by a particular application will solve this problem. Consider the following scenario to clarify the usage.

1) A physicist participating in a high-energy physics experiment would like to execute a CMS application.
2) The application runs various jobs on various sites. Each job, after starting, requests Gvu to provide all the files related to the experiment. This is similar to a
   ```
   find . -type f -group myexperiment
   -name "*"
   ```
   command in traditional UNIX file systems.
3) Gvu returns a logical view, which appears the same to all the jobs.
4) The jobs query the VDS to find out the recipe to generate a new dataset. The input files are accessed using the logical view and created in the logical view. Gvu redirects the access to the logical files to appropriate physical locations.

In this paper, we develop mechanisms for creating the global hierarchical namespace and application-specific views on top of it. We first review the existing mechanisms for manipulating data in a grid or distributed system. We next describe the architecture of Gvu. We then provide the details of the implementation. We conclude with experiments demonstrating the usage and performance impact of

Gvu.

II. Related Work

There is a vast volume of literature on distributed file systems solving various problems that occur in distributed data sharing. CIFS and NFS (v2 and v3) [10] provide a global namespace, but the naming is only at a local domain level. They also have security weaknesses that are not suitable in wide-area grids. NFSv4 has many enhancements and provides a global physical view of the system. An effort called GridNFS, taken up by CITI at the University of Michigan, to customize NFSv4 for grids is still in its infancy.

Other distributed file systems including AFS [12], Coda [13], and GFS [14] are distributed file systems that are designed for multiple clients to access files by using file caches, and do not perform very well in the data-intensive computing environments that are commonly seen in grids. It is interesting to note that AFS provides a global physical view of the distributed system. The physical view is quite inflexible and does not allow sites to export application-specific views.

In the grid realm, the focus has been on providing high-performance data access. Grid-specific data access mechanisms including GridFTP [3], LegionFS [9], and Gfarm [7] succeed in this respect. Gfarm provides highly scalable and high-bandwidth read/write operations by integrating process and data scheduling. It also provides replica management and supports file fragments, but creates a static view of the global namespace similar to AFS and leaves it to the user to handle it. The centralized metadata database used in Gfarm might become a bottleneck. It is also unclear how Gfarm servers interact with each other.

The Storage Resource Broker (SRB) [8] developed by SDSC provides some interesting capabilities to grid data management. SRB provides a uniform interface to heterogeneous data resources and provides replica management. The metadata catalog (MCAT), which is a part of SRB, provides a way of accessing the data sets using attributes. The key feature of this system with respect to our work is the usage of logical names. SRB fails, though, in providing a hierarchical view of the logical names, and has no concept of application-specific views.

In [15], we provided a detailed comparison of grid file system features in a survey submitted to the GFS-WG (Grid File Systems Working Group). This work is in progress and currently compares Gfarm and SRB, two of the most popular grid data management mechanisms. GFS-WG recently released RNS (Resource Namespace Service) specification, which is still in draft form. It describes many of the features that we envisioned earlier in this work.

In the realm of traditional file systems, semantic file systems provide capabilities very similar to what we are envisioning. A file can be thought of as a poor man's database with basic query functionality. On the other end of the spectrum we have feature-rich databases that can provide sophisticated query capabilities to data. Semantic file systems are somewhere in the middle, providing more semantics than traditional file systems and allowing somewhat sophisticated queries to be executed. For example, a semantic file system can associate metadata with audio and video files, and allow queries like *find all the music by Michael Jackson*.

A survey [16] of semantic file systems lists various semantic file systems and their features. MIT's SFS [17] automatically extracts attributes from files with file type specific *transducers*. The concept of virtual directory provides a query-based view of the file system. Our approach extends this idea to the grid, where the queries have to be answered by different file systems. We propose an architecture in which some sites might have a semantic file system.

Key to our approach is efficient query execution using file semantics. We draw ideas from Essence [18] in building our system. Essence was originally used to find files with certain attributes in a local file system.

III. Design of Gvu

We have considered the following issues in designing Gvu.

- *Distributed vs. Centralized*: Since the metadata and files are distributed over the grid, Gvu should not be centralized, but use a set of distributed servers that are synchronized.
- *View ownership*: The logical view exported by a site is owned by the site administrator, but the application and user-specific views created on top of the global view are owned by the respective applications or users.
- *Fault-tolerance*: Gvu should tolerate faults in a Gvu server. Currently, Gvu handles the crash failure of any number of Gvu servers. If a Gvu server goes down, the user will still be able to access the metadata related to the files on the corresponding site, but won't be able to access the files.
- *Performance*: Since the Gvu servers are synchronized over a wide-area network, it is important to keep the communication among the Gvu servers to minimum. We have implemented batching of metadata updates to improve performance.
- *Scalability*: Gvu should be scalable with the number of clients. Currently, Gvu provides better scalability than RLS and MCS combined because of its distributed nature. We are running experiments to

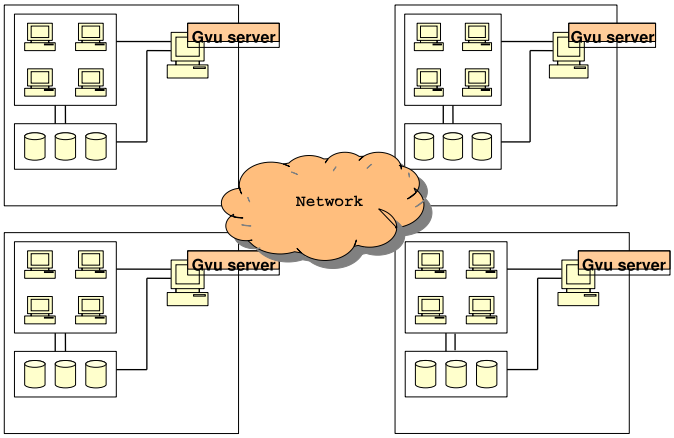


Fig. 2. General top-level organization of Gvu servers

quantitatively measure the scalability and will report the results in future.

- *Consistent, flexible and powerful API*: Gvu provides a familiar file-system-like interface. Once the view is created, the interaction with the view is very similar to the interaction with a traditional file system.

The following subsections detail design of Gvu.

A. Gvu Servers

Each site on the grid runs a Gvu server that maintains the local logical view (explained in the next section) for that site. Figure 2 shows a grid with Gvu servers. Note that the Gvu server usually runs on the gatekeeper machine, which has access to local schedulers, local clusters and local file systems.

The servers communicate with each other using a reliable, fault-tolerant group communication protocol. There has been a substantial volume of research on providing reliable, fault-tolerant group multicast. For example, Newtop [19] is a truly decentralized peer-to-peer system using Lamport's clocks which provides total ordering of messages even in a failure-prone environment. It was designed for static groups, which makes it more appropriate for a grid. It allows dynamic addition and removal of nodes in the group. Shima and Takizawa [20] developed a fault-tolerant group communication protocol. They assume fault-tolerant groups of processes that are replicated in clusters. Litiu and Prakash [21] provide a framework called *distview*, in which a server pool (called Corona) maintains the shared information. The publishers (clients) can submit data to the server pool and subscribers can receive the data either in synchronous or asynchronous mode. The communication protocol provided has all the properties that we require for synchronizing the Gvu servers. We have decided to use `distview`, because of its features and support for distributed collaboration. Other reasons for using it include the source availability.

```
<?xml version="1.0" encoding="utf-8"?>
<dir name="/">
    <dir name="pp" src="/home/ppadala">
        <file name="gvu.pdf" src="/tmp/gvu.pdf" />
        <attribute key="exp" value="bigbang" />
        <attribute key="group" value="umichphys" />
    </dir>
    <file name="fstab" src="/etc/fstab">
        <attribute key="type" value="system" />
    </file>
    <file name="password" src="/etc/passwd" />
</dir>
```

Fig. 3. A sample logical view in XML

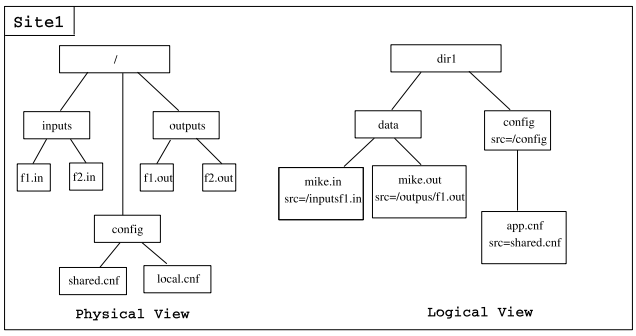
```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://gvu.umich.edu"
    xmlns="http://gvu.umich.edu"
    elementFormDefault="qualified">

    <xs:element name="dir" type="dirtype" />
    <xs:complexType name="dirtype">
        <xs:attribute name="src" type="xs:string" />
        <xs:sequence>
            <xs:element name="dir" type="dirtype" />
            <xs:element name="file" type="filetype" />
        </xs:sequence>
    </xs:complexType>
    <xs:simpleType name="filetype" type="xs:string">
        <xs:attribute name="src" type="xs:string" />
    </xs:simpleType>
</xs:schema>
```

Fig. 4. XML schema for the configuration file

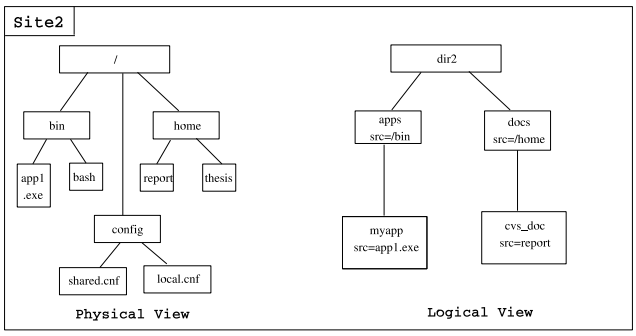
B. Logical Views

Each administrator of the site creates a local logical view that may not necessarily correspond to the physical view. The logical view is specified using a configuration file written in XML. An example configuration file can be seen in Figure 3. The schema for the XML file is shown in Figure 4. The schema does not specify all the types of attributes that can be added to the files and directories. It is left to the user and system administrator's discretion. Note that the files and directories in the logical view can correspond to arbitrary places in the physical view. The attributes for the files are read from the extended attributes stored by the on-disk file system. The logical views are synchronized among the servers using `distview`.

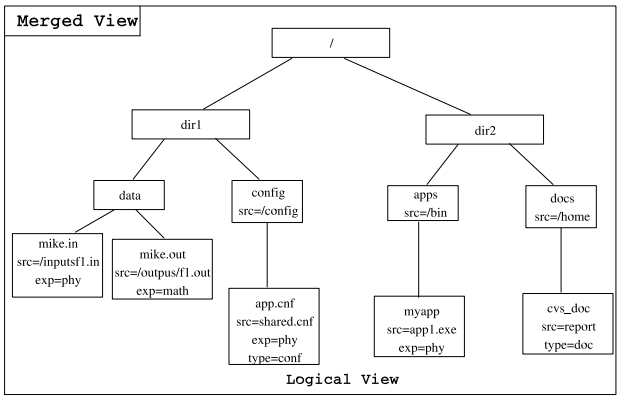
Figures 5(a) and 5(b) show two sites and how their logical views are formed. Figure 5(a) shows the physical and exported logical view of `site1`. Note that only a few directories and files are exported to the grid. The administrator can also specify attributes in the logical view. The attributes are not shown in the first two figures for clarity. Note that the original `outputs` directory is exported as



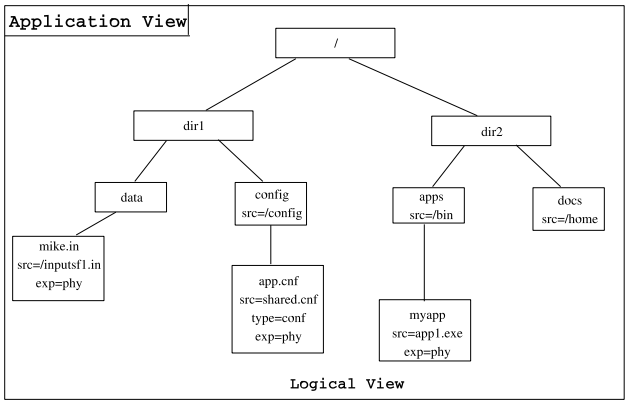
(a) Physical and exported logical view of site1



(b) Physical and exported logical view of site2



(c) The merged logical view



(d) Application-specific view

Fig. 5. Gvu views

`data` in the logical view. Similarly, `site2` exports `bin` directory as `apps`. A few of the features provided by Gvu for the creation of views are worth mentioning. The user can create a logical directory without any corresponding physical directory, but the files in the directory have to be specified with fully qualified physical paths. In Figure 5(a), the data directory is a logical directory. If a directory has a corresponding src attribute, meaning it has a physical directory, then the files under the directory are assumed to be under the corresponding physical directory, unless its absolute path is specified.

After the Gvu servers are initialized with respective configuration files, the views are merged and a global view is formed. Figure 5(c) shows the merged global view. For clarity, `src` attribute is not shown in the figure. One important question while merging is: *What should Gvu do when name conflicts occur?* There is no single answer to this question. Gvu can either provide unique names automatically or ask the administrator to change the logical views. We leave the decision to the site administrator.

When an application queries the Gvu to

```
get all the files related to
experiment "phy"
```

Gvu returns an application-specific view as shown in the Figure 5(d), which is formed by running the appropriate query on the global view.

C. View Synchronization

To support distributed collaboration, views created by different users have to be synchronized. For example, when two users run the query explained in the previous section, they should both see the same view and any changes done by a user should be seen by all the users sharing the view.

The synchronization is achieved using `distview` which provides mechanisms to share Java objects in a distributed system. The objects are required to implement certain interfaces that specify how the objects have to be synchronized. More details are given in the implementation section.

D. View Security

Security is an important issue on the grid due to different administrative domains and policies. GSI (Grid Security Infrastructure) [22] is the de-facto standard for providing security on the grid. We discuss how GSI provides grid-wide authentication with different local authentication mechanisms, policies and how it affects the security of Gvu. Each user on the grid belongs to a VO (Virtual Organization), which provides the user a unique identity that can be used to access grid resources. Each VO on the grid can selectively grant permission to use

resources to various users (identities). In current grids, this is done by mapping an identity to a local user in a global configuration file called `gridmap-file`.

How do the authentication mechanisms affect Gvu views? There are two issues related to Gvu security: access control of files and access control of views. We have implemented security by wrapping Gvu calls with GSI. GSI can map an identity to a local user and Gvu can check the permissions of files to see whether a user has enough privileges to access the file. Adding access control to views is tricky and complicated. Some of the issues are: How do we set the access control list for a local logical view? How can we add access control for application or user-specific views? Where do we store them? One possible solution is to create a security configuration file similar to `gridmap-file` that specifies access control for local logical views. Access control lists for user or application specific views can be maintained by the local Gvu server. We leave more detailed analysis and implementation as future work.

TABLE I. Supported commands in the Gvu server

Session related commands	
connect	Starts a session with the local Gvu server
disconnect	Ends the current Gvu session
View related commands	
create view	Creates a view for a particular query
delete view	Deletes a view with a particular id
ls	List the contents of the view in human readable tree format
ls xml	List the contents of the view in raw XML
set view	Set the current view to the specified id
create file	Create a file in the current view
delete file	Delete a file in the current view
read file	Read a file using GridFTP
Attribute related commands	
attr add	Add an attribute to a file
attr delete	Delete an attribute from a file
Administrative commands	
sync	Sync the tree corresponding to the local Gvu server to the disk
close	Close the current connection between client and server
shutdown	Shutdown the local Gvu server

IV. Implementation

A. Gvu servers and views

We have implemented Gvu in the Java programming language using `JDK 1.4.2`. Java is chosen, because `distview` is written in Java and Globus toolkit (GT3) is heading towards Java-based web services. XML libraries provided with the standard Java distribution are used to parse and execute queries on the views represented in XML.

The `distview` toolkit provides mechanisms to share any objects in a distributed system. The shared objects have to implement certain functions including `notifyStateChange`, `notifyStateUpdate` and `setValue`. We have designed a class called `GvuTree` that implements these functions and can be shared using the `distview` toolkit. The `GvuTree` object is self-sufficient and can identify the global tree, application, user-specific views and respective mappings using a global hash table.

B. Synchronizing the view to the disk

Whenever an update is made to the view, the update is immediately reflected in the in-memory GvuTree and `distview`. As a result, the update will be seen almost immediately on all the servers. Each server runs a SyncThread that keeps checking the GvuTree for any modifications to its local logical views and updates the contents of the disk. This may involve creation or deletion of files and attributes. This mechanism provides good performance for creation and deletion of files. This is similar to the buffer cache mechanisms employed in traditional file systems. If a GvuServer tries to read a file before the file is written to the disk on the server that owns the file, the application is blocked until the write completes. The SyncThread updates the contents of the disk every 30 seconds. The periodicity of sync can be tuned and the value of 30 secs is used, because it matches with the sync periodicity of pdflush kernel thread.

C. Gvu Shell

We have implemented a small shell that allows the users to interact with Gvu servers. The shell connects to the local Gvu server and interacts with it to create and update views. A list of implemented commands are given in the Table IV-A.

D. Query execution

One of the key aspects in creating views is the efficient execution of queries. Currently, Gvu supports simple queries and complex queries containing boolean operations. We have implemented a simple XML node matching algorithm for executing the queries. This can be extended in various ways using well-known techniques used in XML databases.

E. An example scenario

Figure 6 shows an example scenario of interactions between a client application and the Gvu system. The

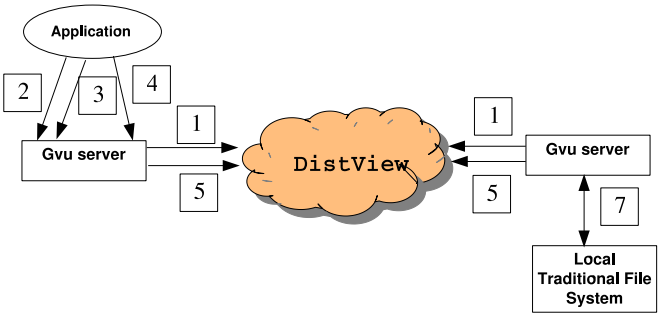


Fig. 6. Client-server communication

numbers in the figure are explained below.

1) When the Gvu servers #1 and #2 start, they add their local logical views to `distview`.
2) When the application starts, it queries the local Gvu server for a logical view corresponding to a particular query. It is similar to doing

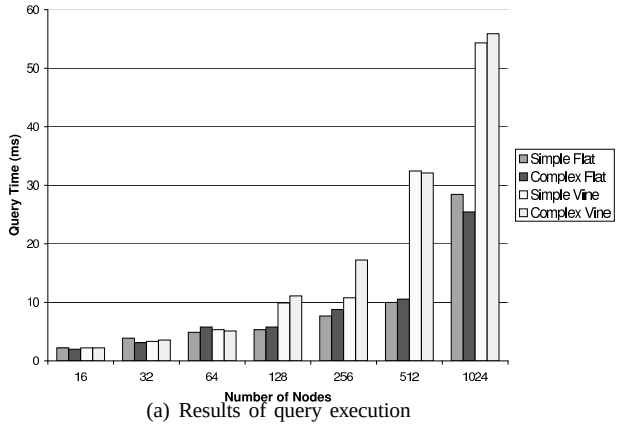
   ```
   find . -type f -group "os"
   -name "*"
   ```

   on a standard UNIX system.
3) The local Gvu server runs the query on the global view (which is kept up-to-date by `distview`) and returns an application-specific view to the user. Note that the view appears the same irrespective of the location where the query is executed.
4) The application creates a file in the logical view.
5) Local GvuServer updates `distview`.
6) `Distview` broadcasts the new contents of the GvuTree to all the servers containing the view.
7) The file is created at the appropriate server.

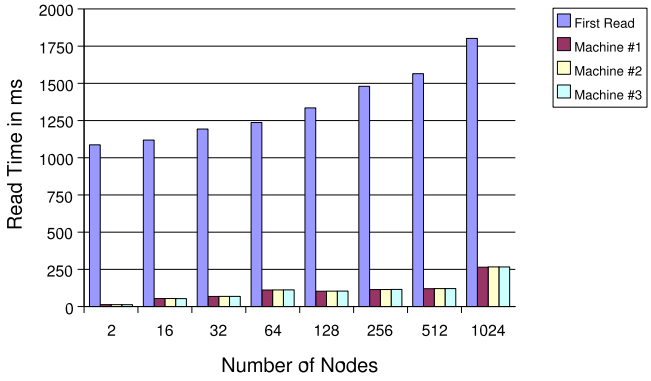
The application can use the logical view to perform various data-management operations. When the application reads a file in the logical view, the read request is forwarded to the appropriate site's Gvu server.

V. Experimental Results

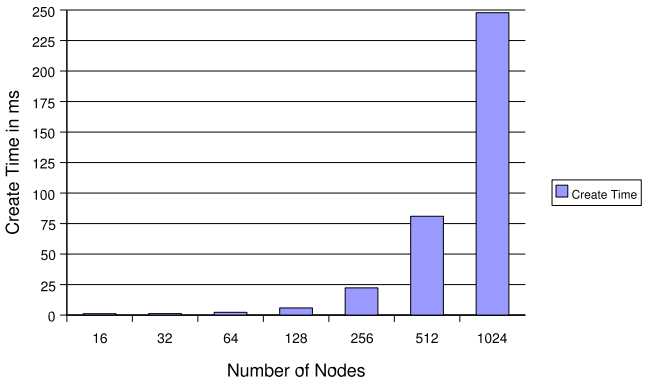
We have conducted experiments measuring various aspects of Gvu. We have set up two separate testbeds for the experiments. The first testbed is a small grid created in the RTCL (Real Time Computing Laboratory) at the University of Michigan. We used this testbed for debugging and for conducting experiments that didn't depend on the wide-area nature of a real grid. We used Grid3 production grid for our real-world scenarios and for understanding the impact of wide-area network on Gvu.



(a) Results of query execution



(b) GvuTree initial read results



(c) Average creation time

Fig. 7. RTCL grid experiment results

TABLE II. Creation and deletion time for one file

Creation time	9.78ms
Deletion time	1.34ms

A. Experiments on the RTCL Grid

All the machines we used had a similar hardware configuration with dual Pentium III processors running at 500 MHz and 512 MB of RAM. They are running Linux 2.4.21 with SMP enabled. We used the extended attributes provided by the ext2 file system for storing attributes for files. The machines are connected by 100Mbps Ethernet LAN, and the load on the machines was minimal when we ran the experiments.

One of the machines is designated as the Corona server. Four machines are designated as Gvu servers and two other machines are used as clients running GvuShell.

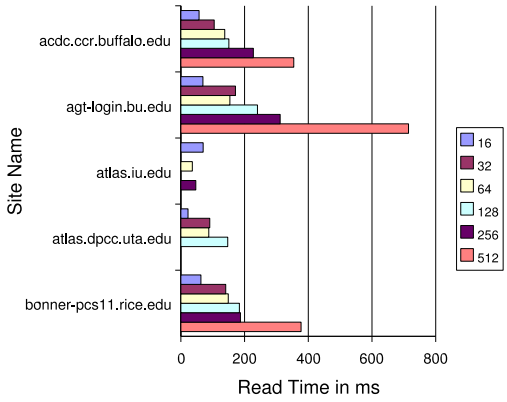
We ran tests to find the execution time for queries, update of GvuTree, creation and deletion time of files.

1) Query Execution: Figure 7(a) shows the execution time of running queries on established trees. The X-axis shows the number of nodes (files) in the GvuTree represented in XML. Four scenarios were run with two types of queries and two types of tree structure: Simple queries that check only one attribute per node, and complex queries that check two attributes. Flat trees had one directory filled with many nodes, and vines had many directories of increasing depth. Ten queries were executed for each data point and the time represents only the time needed to run the query on the server before the results are sent to the application.

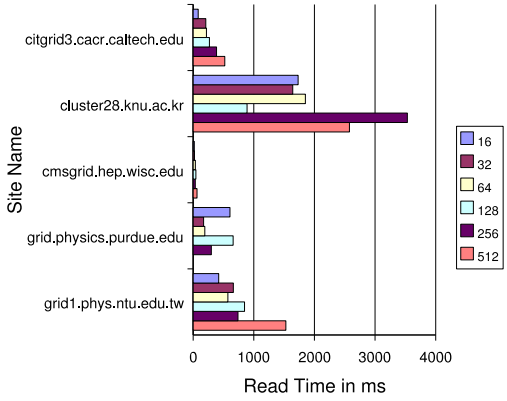
As expected, the search times were linear in the number of nodes because every node must be checked in the current implementation. This could be improved by caching the results and only regenerating the parts of tree that have changed since the last query. The difference between simple and complex queries was negligible. The vines took up twice as long to execute because of the overhead involved in the function calls. The total time to execute a query on 1K nodes is under 60 ms, which is an acceptable cost.

2) Read time for GvuTree: Figure 7(b) shows the read time for GvuTree when the server is initialized. One server is initialized with logical view containing files of sizes 16, 32, . . ., and the time taken to read the GvuTree on a separate server is recorded. The experiment is conducted on three different machines after restarting the first server and Corona. An average of 20 runs is taken on each of the machines.

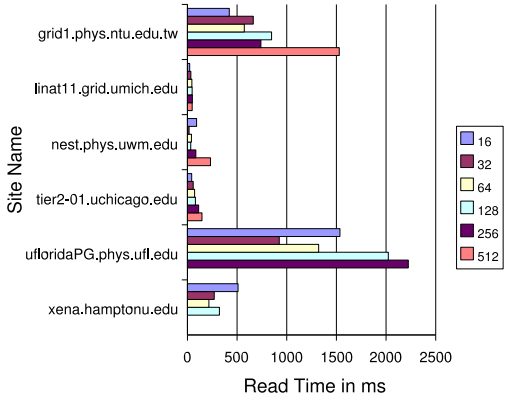
The very first read takes a large amount of time due to Java's initialization and serialization of the GvuTree object. Once this is done, the read time for subsequent GvuTree



(a) Machine set #1

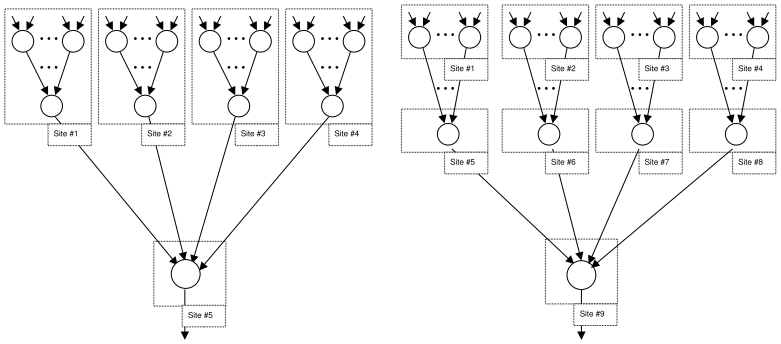


(b) Machine set #2



(c) Machine set #3

Fig. 8. Initial GvuTree read time on Grid3



(a) Workflow-I (b) Workflow-II

Fig. 9. Application workflows

reads is minimal.

3) Create and delete time of files: Table V-A.1 shows the creation and deletion time for a single file. The low times are due to the in-memory updates. We also ran macro benchmarks by creating a different number of files in a single directory and results can be seen in Figure 7(c). The delay increases as we create more files, because more time is needed to send the updated (bigger) tree to all the Gvu servers. This can be improved in various ways. For example, one can send only updated parts of the tree to other Gvu servers. Another interesting mechanism would be batching of commands. We have implemented batching for the experiments done on Grid3.

B. Experiments on Grid3

Grid3 project developed under the auspices of iVDGL (International Virtual Data Grid Laboratory) is a data grid consisting of more than 25 sites with thousands of processors. Grid3 is used by various scientific communities including high-energy physics, bio-chemistry, astrophysics and astronomy.

For all the experiments, the Corona server is run on a 3GHz machine with 1GB RAM running Linux 2.6. Information about the various sites used in the experiments can be found on the Grid3 site catalog available at `http://www.ivdgl.org/grid3/`

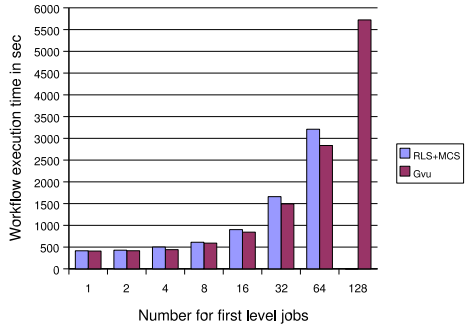
1) Read time for GvuTree: We have run the GvuTree read experiment with different sites for different number of files. Figure 8 shows the read times for various sites. Certain sites were down during a few periods of running the experiments. Note the high latency experienced on the Korean and Taiwanese sites (cluster28.knu.ac.kr, grid1.phys.ntu.edu.tw) and low latencies at the Michigan and Wisconsin sites (linat11.grid.umich.edu, cmsgrid.hep.wisc.edu). The low latency is due to the proximity of these sites to the Corona server in the RTCL. You can see certain anomalies in read times at the Florida site (ufloridapg.phys.ufl.edu). This is due to the high load on the site at the time of our experiments.

2) Real-world Scenarios: To better understand the behavior of Gvu, we have run two workflows that are similar to CMS workflows in various scenarios. The workflows we used for our experiments are shown in Figures 9(a) and 9(b). The circles represent the jobs and the arrows show the data dependencies between the jobs. All the jobs are similar except that they are run with different inputs and produce different outputs. The workflows have three levels of jobs. Experiments are conducted for a different number of first level jobs. The scheduling of the jobs is done using a simple load-balancing mechanism with an equal number of jobs running on each site. The third level job is run on a separate site.

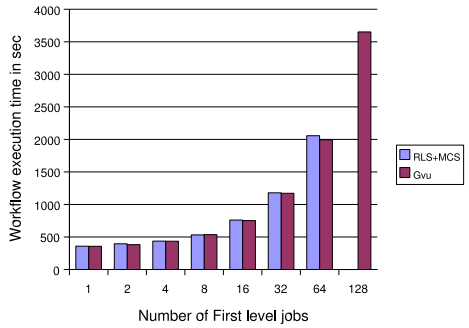
The workflows are run with Gvu and MCS + RLS and the execution time of the workflow is compared. The pseudo-code describing the application is shown in the Figure 11. The application checks with Gvu or MCS for the existence of an input file and if it is available it requests either Gvu or RLS for the location of the file. It produces the output as soon as all the inputs are available.

Both the workflows are run with two different sets of sites. The first set of sites are connected by a wide-area network with latencies on the order of 60ms. The second set of sites are connected by a wide-area network with latencies on the order of 20ms. The two different sets of sites are chosen to demonstrate how Gvu copes up with the high latencies experienced in wide-area networks.

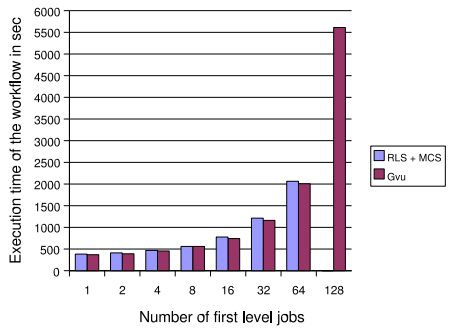
3) Performance of Workflow-I: As one can see in Figure 9(a), workflow I has better data locality, since the level-1 and level-2 jobs are run on the same site. This is the common mechanism for submitting jobs on the grid. The execution time of the workflow with high latency network is shown in Figure 10(a). Note that for 128 jobs, the workflow didn't finish when RLS and MCS are used. This is because of the limit (100) on the number of connections to RLS. This also shows an important aspect of Gvu



(a) Performance of workflow-I with high latency network



(b) Performance of workflow-I with low latency network



(c) Performance of workflow-II

Fig. 10. Grid3 experimental results

with respect to the scalability. Though Gvu servers were also highly-loaded for 128 first-level jobs, the performance degraded smoothly, because of the distributed nature of the Gvu.

Performance of the workflow with a low latency network can be seen in Figure 10(b). As expected, the performance improvement is small with Gvu, because of the fast response times from RLS and MCS (due to the low latency network). However, Gvu still performs better than RLS and MCS, because most of the `stat file` requests are handled locally.

4) Performance of Workflow-II: Workflow-II shown in Figure 9(b) is similar to workflow I except that the level-2

```
boolean checkInputFiles(String inputFiles[])
{
    foreach file in inputFiles {
        /* ask Gvu or MCS whether the file is available */
        stat(file);
    }
    if (all files are available)
        return true;
}

main (String inputfiles[])
{
   /* connect to local Gvu server or
      a centralized MCS server */
    connect(Gvu or MCS);
    while(checkInputFiles(inputFiles) == false) {
        /* sleep for 1000 ms while the inputs
           are not available */
        sleep(1000);
    }
    /* ask Gvu or RLS for location of the files */
    locations(inputFiles);
    /* create output and register with Gvu
       or MCS and RLS */
    createOutput;
    /* register the attributes and location
       of the output file with MCS and RLS
       respectively */
    registerOutput;
}
```

Fig. 11. Pseudo code for the application

jobs are submitted to different sites. This destroys the data locality and yields poor performance. However, Gvu still performs better than RLS and MCS, due to its distributed nature. Note that the performance improvement is less than that with workflow I. Figure 10(c) shows the performance of workflow II with sites connected with an average latency (30ms) network. We did not run workflow II with a low latency network as we couldn't find enough site that are near us.

VI. Concluding Remarks

A. Conclusions

We proposed a view-oriented framework for grid file systems that improves usability and allows distributed collaboration. We developed mechanisms to create, update and maintain views efficiently. The views are synchronized and on-disk attributes are updated automatically. A shell is developed to show various aspects of client interactions with Gvu servers. Experiments are conducted to measure Gvu performance, and results indicate that the overhead of views is minimal and Gvu performs much better than RLS and MCS.

B. Future Work

Gvu framework raises interesting questions for sharing data on a grid. *How do we synchronize views that share*

files? An efficient synchronization algorithm is needed to update all the views that have a file when the file is updated. This is not trivial and requires careful design.

Various optimizations can be done to improve the query execution performance. XML databases may provide clues on how to implement the queries. Caching can be used on the client and local Gvu servers to improve performance. More work is needed to implement access control lists for and views. Policies need to be developed to resolve conflicts in merging views. Real-world scenarios have to be explored to understand the effect of conflicts and when they occur. More work is also needed for achieving better fault-tolerance of Gvu and Corona servers.

Acknowledgments

In the early days of this project, Michael Nettleman and Paul Deluca helped us in designing and implementing a few parts of the Gvu.

References

[1] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a Future Computing Infrastructure*. San Francisco, California: Morgan Kaufmann Publishers, 1999.

[2] I. Foster and C. Kesselman, "The Globus project: A status report," in *Proceedings of the Heterogeneous Computing Workshop*. IEEE Computer Society Press, 1998, pp. 4–18.

[3] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high-performance computational grid environments," *Parallel Computing*, vol. 28, no. 5, pp. 749–771, May 2002.

[4] A. L. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, "Giggle: A framework for constructing scalable replica location services," in *SC'2002 Conference CD*. Baltimore, MD: IEEE/ACM SIGARCH, Nov. 2002, pap239.

[5] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman, "A metadata catalog service for data intensive applications," in *SC2003: Igniting Innovation. Phoenix, AZ, November 15–21, 2003*, ACM, Ed. New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA: ACM Press and IEEE Computer Society Press, 2003, pp. ??–??

[6] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny, "Pipeline and batch sharing in grid workloads," in *Proceedings of the Twelfth IEEE Symposium on High Performance Distributed Computing*, June 2003.

[7] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid datafarm architecture for petascale data intensive computing," in *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002)*, H. E. Bal, K.-P. Löhr, and A. Reinefeld, Eds., IEEE. Berlin, Germany: IEEE Computer Society, 2002, pp. 102–110.

[8] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The sdsc storage resource broker," in *Proceedings of IBM Centers for Advanced Studies Conference*. IBM, 1998.

[9] B. S. White, M. Walker, M. Humphrey, and A. S. Grimshaw, "LegionFS: A secure and scalable file system supporting cross-domain high-performance applications," in *SC'2001 Conference CD*. Denver: ACM SIGARCH/IEEE, Nov. 2001, u. of VA.

[10] I. Sun Microsystems, "NFS: Network file system protocol specification," *Internet Request for Comments*, no. 1094, Mar. 1989.

[11] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying and automating data derivation," in *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.

[12] J. H. Howard, "An overview of the andrew file system," in *Proceedings of the USENIX Winter Conference*. Berkeley, CA, USA: USENIX Association, Jan. 1988, pp. 23–26.

[13] M. Satyanarayanan, "Coda: a highly available file system for a distributed workstation environment," in *Proceedings of the Second Workshop on Workstation Operating Systems (WWOS-II), September 27–29, 1989*, IEEE, Ed. IEEE Computer Society Press, 1989, pp. 114–116.

[14] S. R. Soltis, T. M. Ruwart, E. Erickson, K. W. Preslan, and O. O'Keefe, "The global file system," in *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, H. Jin, T. Cortes, and R. Buyya, Eds. New York: IEEE/Wiley Press, 2001, chap. 23.

[15] P. Padala, "A survey of the grid file systems," Oct. 2003.

[16] P. Pazandak and V. Vesudevan, "Semantic file systems," pp. 16–25, Jan. 1997. [Online]. Available: http://www.objs.com/survey/OFSExt.htm

[17] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O. Jr., "Semantic file systems," *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pp. 16–25, October 1991.

[18] D. R. Hardy and M. F. Schwartz, "Essence: A resource discovery system based on semantic file indexing," in *Proceedings of the Winter USENIX, January 25–29, 1993*, 1993, pp. 361–374.

[19] P. D. Ezhilchelvan, R. Macdo, and S. Shrivastava, "Newtop: A fault-tolerant group communication protocol," ESPRIT Basic Research Project BROADCAST, Technical Report BROADCAST#TR94-48, Oct. 1994.

[20] K. Shima and M. Takizawa, "Fault-tolearnt intra-group communication," in *Proceedings of the 10th International Conference on Information Networking*, Jan. 1996, pp. 467–473.

[21] R. Litiu and A. Prakash, "Stateful group communication services," in *Proceedings of the International Conference on Distributed Computing Systems*, June 1999, pp. 82–89.

[22] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids," in *ACM Conference on Computers and Security*. ACM Press, 1998, pp. 83–91.