A Data Management Framework for Distributed Biomedical Research Environments

Rajkumar Kettimuthu[1,2], Robert Schuler[3], David Keator[4], Martin Feller[1,2], Dingying Wei[4], Michael Link[1,2], John Bresnahan[1,2], Lee Liming[1,2], Joseph Ames[4], Ann Chervenak[3], Ian Foster[1,2], Carl Kesselman[3]
[1]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL
[2]Computation Institute, University of Chicago, Chicago, IL
[3]USC Information Sciences Institute, Marina Del Ray, CA
[4]University of California, Irvine, CA

***Abstract*—Biomedical research increasingly depends on access to and analysis of distributed medical and biomedical data. In biomedical research, datasets are often collected at multiple locations, as it is difficult to recruit required patient populations at one location. The sensitive nature of these datasets leads to a need for secure sharing methods that can work with distributed databases and image archives. The efficient implementation of such methods is complicated by the fact that the researchers who need to access data are often behind firewalls that prohibit inbound connections. This restriction, combined with large data volumes and many small files, leads to significant data transfer challenges. We present a data management framework suitable for such distributed biomedical research environments and describe its application in the context of FBIRN, a distributed data-sharing system for medical researchers. The framework includes security tools, catalogs for managing datasets, and a secure data transfer service.**

Keywords - Data management, Biomedical research, Distributed computing

I. INTRODUCTION

Biomedical research increasingly depends on access to and analysis of multiple, geographically distributed sources of medical and biological data. Data must be stored, processed and made easily available to different biomedical participants: researchers, physicians and healthcare centers.

Healthcare applications impose several requirements on computing infrastructure. The need to record, manage, and use sensitive personal data leads to strict privacy requirements. Healthcare systems must comply with privacy laws, medical information standards, and institutional review board restrictions. Encrypted transmission and storage are not sufficient; automatic de-identification or anonymization may be required to guarantee privacy. Integrity verification is also important.

As health research moves towards personalized medicine, it becomes increasingly difficult to recruit cohorts of patients at a single location. Thus, data are often collected at multiple locations. These data may consist of both images (e.g., MRI scans) and clinical evaluations (both structured and unstructured data). Because these data are sensitive, most centers prefer to store them locally rather than contribute them to central data warehouses. Data are then shared with collaborators over well defined, secure, and robust interfaces. Management of such distributed databases and image archives is therefore critically important in this domain.

Sites hosting individual data repositories are generally behind institutional network firewalls over which data repository operators may not have direct control. Firewall configurations typically do not allow outside services to request connections to servers inside the firewall except for a small number of well-defined service addresses. In addition, researchers and physicians who need to access these distributed datasets are generally behind restrictive network firewalls, which may not allow any inbound connections at all.

The datasets are often large (terabytes) and may contain a large numbers (millions) of small files. These dataset characteristics, combined with strict firewall rules, present significant challenges in transferring these datasets over wide area networks reliably and efficiently.

We present here a data management framework that is suitable for distributed biomedical research environments. This framework includes security tools [1, 2, 3], catalogs for managing data objects [4, 5], and a secure data transfer service [6, 7, 8]. We evaluate this framework in the context of Function BIRN (FBIRN) [9], a distributed data repository that integrates fMRI, structural imaging, and clinical data from multiple sites that the research community uses to develop and test novel hypotheses.

The rest of the paper is as follows. In Section II, we describe the FBIRN collaboration and its requirements for secure, federated data publication and sharing. We present the data management framework in Section III. In Section IV, we evaluate our framework in the FBIRN environment. We discuss related work in Section V and summarize our work in Section VI.

II. FBIRN

The function BIRN (FBIRN) consortium comprises 10 sites across the United States focused on developing fMRI tools and techniques for both federated data management and multi-site image calibration and analysis in the context of schizophrenia research [10]. Each FBIRN site collects and stores both clinical and neuroimaging data and associated metadata in local storage.

A. FBIRN Environment

The FBIRN environment demands a federated system capable of providing robust access to data stored at any site. Typical FBIRN usage suggests that this system must be able to support up to eight concurrent downloads and three concurrent uploads of image data to/from each site.

Original, raw, and analyzed image data sets vary significantly in size. The *original* data from a single subject session comprises ~1.0 GB and 2.9K files, depending on scanning parameters. Normally, consortium sites download multiple subjects/sessions in bulk. These downloads consist of ~20 subjects over two scanning visits, for a total of ~42 GB and 129K files. Users require that datasets transfer with a high success percentage and with verification for consistency at both the source and destination. In addition, FBIRN replicates datasets in the federation so they are available if the originating site's systems are offline.

In summary, FBIRN researchers want a dynamic, robust system that abstracts the users from the underlying complexity of the data federation, making it appear as though all the data exists in a single location.

B. Data Requirements

In the previous sections, we discussed our general observations from the biomedical sciences community and a specific application that motivated our data management framework. Here we discuss in detail the driving requirements that shaped the architecture.

The system must provide to a specified set of project participants access to data located in a set of federated storage resources at several independent sites. Each site contributes commodity hardware and networking resources and limited systems administration resources. Failures at other sites must not prevent users from accessing their own local data, so long as the local resources are online.

The system must store a large number of relatively small files. We anticipate each experiment will produce over a million files ranging in size from hundreds of bytes to several megabytes and will organize them hierarchically in a relatively flat directory tree with degree typically under five.

The system must store files in a manner that is interoperable with other conventional systems. Often, specialized software systems of this nature employ proprietary storage mechanisms, which preclude access by other services such as Network File System [11], Rsync [12], and Secure Shell [13]. Though the system is intended to be robust, in the event of failure, it should never prevent access to locally owned data by keeping it in proprietary storage.

The system must support security policies governing user group management and data access controls including create, read, write, and delete of files and directories. Users can belong to one or more groups, and the system must support multiple user- and group-based data access permissions per file or directory. In addition, the system must allow each storage resource to specify its own set of data access permissions, thus allowing different permissions for local users and users from other organizations in the federation.

Typical workloads are shaped by the typical project lifecycle, which progresses through several phases of activity. In the *data capture* phase, data produced by the instruments are initially stored. In the subsequent *quality assurance* phase, a curator examines project data for defects and abnormalities. Finally, the *data access* phase commences, during which time users across the FBIRN federation may read any data at local or remote sites.

The system must support multiple concurrent projects, which may be at different phases in the project lifecycle. In particular, the system must handle overlapping workloads from both data capture and data access phases. We expect the aggregate workload to consist of about a dozen clients with ~75% readers and ~25% writers.

The workload during the data capture phase typically consists of a small number of writers performing large bulk writes to local storage. A client may write 2 GB or more in tens of thousands of small files. Multiple clients writing to a single dataset are not common. Typically, a single user is responsible for creating a new dataset. Once a dataset is written, it is seldom modified, except in the event of a quality assurance failure, in which case the complete data set is often retracted and republished. Thus, random mutations of data are rare.

The workload during the data access phase typically consists of several local and remote readers. Like the writers in the data capture phase, the readers in the data access phase also perform large bulk operations. We expect that readers will download multiple datasets with over 100,000 small files and a total volume of over 40 GB. We can expect that multiple readers may access the same dataset at the same time, since the data access phase follows data capture for a given project.

We favor reliable data transfer and sustained throughput over latency. Applications tend to access data in large bulk operations, as described, in the range of 2 GB to 40 GB, but in relatively infrequent, bursty patterns. Though the system supports interactive clients, we anticipate mostly batch clients. Also, given that administrative resources are limited, reliable data transfer is favored over performance.

III. Data Management Architecture

We present here the data management framework, including the basic design, the trust model, data access control, group membership service, and file transfer service.

A. Design Overview

The system consists of storage resources, file transfer services, registry services, a credential repository, a group membership service, and clients, as depicted in Figure 1. System components are loosely coupled and can operate independently and without shared state. By keeping the system loosely coupled, we simplify maintenance, reduce

operations costs, and reduce the probability of system-wide failure.

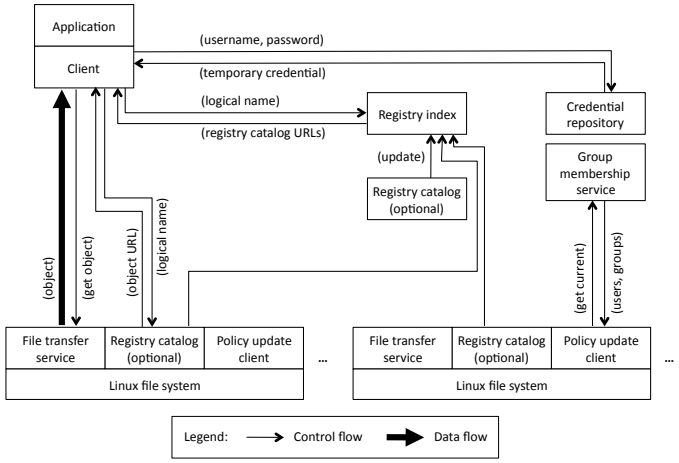

Figure 1: Data Management Architecture

To preserve interoperability with other systems, data are stored conventionally in Linux file systems. Files are organized hierarchically under project directories. At each storage resource, a **file transfer service** is responsible for providing secure remote access to the underlying file system. The data objects at each storage system are identified uniquely by protocol-specific URLs. Currently, we support ftp, sshftp, and gsiftp protocol schemes [14]. The file transfer service supports storage, retrieval, third-party transfer, delete, checksum generation, and checksum validation for individual files and for bulk operations with recursive directory traversal. Additionally, it supports changing ownership, setting access controls, and getting access control lists remotely. For the file transfer service, we used the Globus GridFTP Server [8].

The locations of data objects at each storage site are registered in **registry catalogs** that map those physical locations to logical names. A logical name is a location-independent (i.e., host-independent) name for an object. It may be mapped to one or more instances of the object, usually by mapping to one or more URLs. Though the system supports individual file registration, the large number of files in the system motivates us to register logical name mappings to higher-level directories that serve as data set collections. This approach reduces the communication costs that would be incurred registering and locating many small data objects, and reduces the amount of metadata stored in registry services. Registry catalogs may be installed as a centralized service, or co-located with storage resources, or both. Each registry catalog generates a compressed index of its collection of logical names and updates a **registry index** service whenever its contents change. The registry index is used to quickly redirect clients to the appropriate registry catalog to answer queries for logical names. Together, the registry catalogs and registry index comprise the system's registry services. For the registry services, we used the Globus Replica Location Service [15].

User applications can access the data management services by using a C or Java API. We also provide several client utilities for standalone operations. To retrieve data from the system, clients first refresh a temporary user credential, known as a proxy credential [16], from the **credential repository**, which authenticates the client based on username and password. The client then queries the registry index, which directs the client to the appropriate registry catalog that contains the location metadata for the data object. The client then queries a registry catalog to obtain a listing of one or more URLs to the data object. Finally, the client uses the URL to retrieve the data object using the file transfer service. For the credential repository, we used the MyProxy credential management service [17].

B. Trust Model

Trust between system components is established by use of Public Key Infrastructure (PKI) [18], which depends on a chain of trust rooted by a Certificate Authority (CA). A certificate that traces back to the issuing CA identifies each system component and user. To establish trust between components, the CA root certificate is installed on each server and each service is issued a service or host certificate, which must also be installed on the server. The certificates enable cryptographically strong authentication when any two components communicate. Additional benefits of this trust model are its efficiency and robustness. With the CA root certificate installed on each server, components validate authenticity directly without the need to communicate with a third-party security service for every transaction. This structure reduces the amount of network traffic produced by the system and allows for continuous operation whether or not the central security services are online.

For convenience and usability, the credential repository stores user certificates. When performing a data operation, the user provides his or her username and password to the client, which securely communicates with the credential repository to retrieve a temporary certificate (known as a proxy certificate [16]). The proxy certificate is valid for some duration up to a configurable upper limit, typically up to 12 hours. For the duration of the proxy lifetime, the client reuses the proxy certificate for all communication with the system. Again, the trust model enables efficiency and robustness by reducing unnecessary system interactions to establish trust.

C. Access Control

The system relies on the underlying Linux file systems to enforce data access control. We selected the ext3 Linux file system [19] in part for its support of the POSIX 1003.1e draft specification [20] for extended access control lists (extended ACLs). Using extended ACLs, users can

grant multiple read and write permissions for various users and groups, unlike traditional UNIX file permissions that are limited to granting permissions to a single user and a single group. Though the standards body abandoned the POSIX 1003.1e draft, numerous file systems and utilities support it, thus allowing interoperability between our system and third-party tools.

When the client connects with the file transfer service, the file transfer service uses the proxy credential to authenticate the user and maps her certificate-based identity to her local user account recognized by the Linux file system. The file transfer service then forks (i.e., spawns a new process) and sets its user identity to her local user account. From that point forward, the Linux file system is able to enforce the extended ACLs on file transfer service operations. This approach stands in contrast to the typical approach to security taken by web applications where the web application operates as a privileged user that has near unlimited access to data and system resources. That approach has the disadvantage that relatively new and unproven custom code is used to enforce access control restrictions in the web application. An attacker need only find a way to compromise such a web application to gain access to any and all data on the server. Our approach, on the other hand, relies on kernel and file system implementations, which have been vetted by a large community of users.

D. Group Membership

To support user group based authorization, the system relies on a group membership service. The group membership service manages user and group membership details. We implement the group membership service using caBIG GridGrouper [21]. An authorized user may create groups and add group members. Groups can also be formed by composition or inheritance from other groups.

At storage sites, an automated process periodically updates the local system's users and groups by retrieving the users and groups from the group membership service and merging them into the local system. Since user accounts and group membership change infrequently, periodic synchronization with the group membership service is sufficient to support the scientific applications that use the system. This approach also keeps the system loosely coupled by avoiding communication with the group membership service for every user operation. In the event of network partitions or server failures at the group membership service, storage resources can continue to authorize data accesses.

E. GridFTP Protocol

The file transfer service supports the GridFTP file transfer protocol. The GridFTP protocol is a backward-compatible extension of the legacy RFC959 FTP protocol. It maintains the same command/response semantics introduced by RFC959. It also maintains the two-channel protocol semantics. One channel is for control messaging (the control channel) such as requesting what files to transfer, and the other is for streaming the data payload (the data channel).

Once a client successfully forms a control channel with a server, it can begin sending commands to the server. In order to transfer a file, the client must first establish a data channel. This task involves sending the server a series of commands on the control channel describing attributes of the desired data channel. Once these commands are successfully sent, a client can request a file transfer. At this point a separate data channel connection is formed using all of the agreed-upon attributes, and the requested file is sent across it.

In standard FTP, the data channel can be used to transfer only a single file. Subsequent transfers must repeat the data channel setup process. GridFTP modifies this part of the protocol to allow many files to be transferred across a single data channel. This enhancement is known as *data channel caching*. GridFTP also introduces other enhancements to improve performance over the standard FTP mode. For example, parallelism and striping allow data to be sent over several independent data connections and reassembled on the destination. These enhancements require the use of the extended block mode (MODE E) [6] of GridFTP. In this mode, data channels must go from sender to receiver.

GridFTP servers are typically configured to listen on one port for the control channel, and to use a configurable port range for data channel connections. Firewalls have to be configured accordingly.

F. Constraints on the System

As mentioned above, the biomedical environment imposes specialized requirements on a computing infrastructure. In particular, participating institutions have differing firewall requirements, ranging from no firewall to one or more institutional firewalls.

Some sites do not allow any inbound connections to client machines. Thus, MODE E, which enables advanced GridFTP performance features such as pipelining, parallelism, and striping, cannot be leveraged in transfers on these clients for downloads, because inbound connections are blocked by firewalls. Data has to be downloaded using the standard FTP mode, where a separate TCP data connection has to be formed for each file to be sent. The result is greatly reduced performance.

IV. Experiences and Lessons Learned

We describe our experiences transferring a large FBIRN study, a data set organized as a directory tree with many directories (9292) and files (620791) with an overall size of 69 GB. The dataset is a complete FBIRN study that includes all subjects and all visits collected during a

traveling subjects study from one site. Typically, after data collection is complete, most FBIRN sites will download the entire data set for analysis.

We experimented with transferring this dataset from UCSD to UCI. In initial tests, only one run out of seven attempts completed successfully, and that successful run divided the dataset into 31 sub-datasets and performed 31 data transfers instead of one.

We describe here the problems that prevented these transfers from completing, and the solutions that we deployed to correct these problems.

A. *The many small files problem*

Transferring large directories with many files in standard FTP mode is inefficient, because for each file a data channel connection must be set up and torn down. However, this problem is exacerbated when transfers involve many *small* files, related to the TCP TIME_WAIT status. Upon closing a TCP connection, the connection goes into a TIME_WAIT state for two times the maximum segment lifetime (MSL), before finishing the connection close. This period typically defaults to 1-4 minutes, depending on operating system.

In the case of transfers of many small files, it may happen that the limit for the maximum number of opened connections, defined on the GridFTP server by the TCP port range, is exceeded. If this happens, then all subsequent transfer requests hang.

B. *The many small files solution*

To limit the number of data channel connections used when transferring many small files and to improve performance and reliability, we made use of the popen (pipe open) driver [22] feature of the Globus GridFTP server Globus Toolkit v5.x. This feature allows users to execute arbitrary programs on the server and to communicate with the programs using UNIX pipes from the GridFTP server to the standard I/O of the executed programs. It utilizes the same underlying approach that users of shell scripts employ to compose multiple executables and pipe standard I/O between them.

We leveraged the popen driver to transfer directories as a single large archive file. The following steps illustrate a scenario of a directory download:

1. The client creates a control channel connection to the server and tells the server that the requested data must be archived prior to the transfer.
2. If the server has popen support enabled, the server archives the data with the specified command, and sends the resulting data as a stream over a single data channel, as generated by the archive program (e.g., tar).
3. The client receives the archive file over the data channel and unpacks it as it is received (again using tar), recreating the directory structure in the client file system.

The mechanism is the same for an upload scenario. For the user, the use of the popen driver is opaque.

We note that execution of arbitrary programs as part of a data transfer opens a potential security risk. The Globus GridFTP Server can be configured with a *whitelist* of programs that can be run. A server only permits execution of programs on its popen whitelist. If a client requests a program to be run that is not on the whitelist, the transfer fails.

Table1: Transfer characteristics of tar-enabled GridFTP

Concurrency	Size (GB)	Speed (MB/s)	Success Count	Failure Count	Success Rate(%)
1	2	15.44	88	0	100
1	21	10.41	22	0	100
3	2	9.12	174	0	100
3	21	6.77	27	0	100
5	2	6.43	85	0	100
5	21	5.41	45	0	100
7	2	5.48	84	0	100
7	21	4.59	49	0	100
9	2	3.62	90	0	100
9	21	4.40	77	4	95
11	2	2.96	100	10	91
11	21	2.78	60	17	78

We tested this solution with different dataset sizes (2 GB and 21 GB) and concurrency levels. The 2 GB dataset contained 5.9K files and corresponds to a single subject scanned twice on the fMRI scanner. FBIRN sites typically contribute such datasets during study data collection. The 21 GB dataset contained 64.6K files and is equivalent to 10 subjects over 2 visits. These data sizes are typical of an FBIRN download for data analysis. Table 1 shows that the data transfer success rate goes down as the data transfer time increases. It has been shown in an earlier study [23] that increased concurrency levels improves GridFTP's transfer rate for many small files datasets. Table 1 shows that as the concurrency level increases, the transfer rate decreases. It has to be noted that many small files datasets are tarred up into large archive files and transferred here. Table 1 shows the transfer rates for concurrent transfer of multiple 2 GB files and multiple 21 GB files. For many small files transfers, concurrent transfers improve the performance by overlapping the latency between file transfers on one session with the file transfers on one or more other sessions. Large single file transfers do not have the issue of latency between file transfers, and when multiple large files are transferred concurrently, the load on the server increases, leading to lower transfer rates.

C. *The control channel disconnect problem*

While transferring a nested directory as a single archive stream greatly improved performance and

reliability, it led to another problem. As mentioned above, FTP and thus GridFTP employ a two-channel protocol. To remain compatible with FTP, GridFTP sends no data over the control channel after the connection has been established and while data are being sent. Only after the data have been transferred is the control channel used to complete the transfer. Depending on various parameters like the size of the data to be transferred, network bandwidth, and the disk speed, a directory transfer can take a long time. During this time the control channel is idle.

When clients or servers are behind a NAT proxy or a firewall, connections that are idle for a long time often are disconnected. This behavior is caused by the connection tracking mechanisms implemented in proxies and firewalls, which keep track of all connections that pass through them. These mechanisms can only keep a finite number of connections in their memory. A common policy is to keep the newest and most active connections and to discard old and inactive connections first.

This behavior caused problems with long-running transfers, because the control channel connection was dropped silently. Transfers hung and failed.

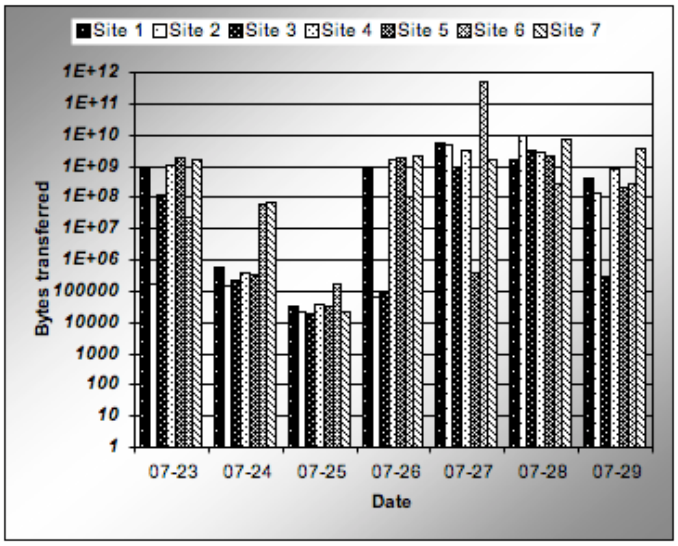

Figure 2: Volumes of data moved using our data management architecture at various FBIRN sites

D. The control channel disconnect solution

To solve this problem, we leverage the TCP keepalive mechanism to avoid connection dropping. With TCP keepalive, "probes" are transmitted to network peers at a configurable time interval, and the probes prevent peers from closing connections by informing them that the sender is still "alive". Linux has built-in support for TCP keepalive; however, the default settings in most Linux distributions exceed the time limit used by proxies and firewalls to close idle TCP connections. Thus, by the time the kernel sends probes to keep the connection open, the proxy or firewall may have already closed the connection. By configuring TCP keepalive on the GridFTP server machines to probe idle connections at a more frequent interval, connection dropping was avoided.

When using the methods just described, all tests listed in Table 1 succeeded, as did the UCSD-to-UCI transfer described at the beginning of this section. Large volumes of data are being transferred between FBIRN sites reliably and efficiently every day using these new capabilities to enable collaborative imaging studies. Figure 2 shows the volumes of data moved to/from various FBIRN sites on a day-to-day basis for a seven-day period (July 23 – 29, 2010).

E. Continuous monitoring

Recall from our requirements discussion (section 2) that in the FBIRN user community, reliable transfer and sustained throughput are preferred over high performance. In non-technical terms, what this means is that the FBIRN scientists need the data transfer services to behave predictably: performance (good or bad) should meet expectations based on past behavior. This requirement led us to two goals: (1) stabilize the system's behavior, and (2) measure the system's behavior and publish results to set reasonable expectations among the users.

Table 2: A sample view of the data from our continuous monitoring tests

		Throughput (MB/s)								
From	To	5/21	5/22	5/23	5/24	5/25	5/26	5/27	5/28	5/29
Site A	Site C	*	X	X	X	X	X	X	*	*
	Site D	**	**	X	X	X	X	X	**	*
	Site E	O	*	O	*	*	*	*	*	O
	Site F	**	**	**	X	**	**	**	**	**
	Site G	**	**	**	X	X	**	**	**	**
	Site B	*	X	X	X	X	X	X	*	*
	Site H	**	**	**	**	**	**	**	**	*
	Site I	**	X	X	X	X	X	X	**	**
	Site J	**	**	**	**	**	**	**	**	**
Site B	Site A	*	*	*	*	*	*	*	*	*
	Site C	*	*	*	*	*	*	*	*	*
	Site D	***	***	***	***	***	***	***	***	***
	Site E	*	*	*	*	*	*	*	*	*
	Site F	**	**	**	**	**	**	**	**	*
	Site G	**	**	**	**	*	**	**	**	**
	Site H	***	***	***	***	***	***	***	***	**
	Site I	***	***	***	***	***	***	**	**	***
	Site J	***	***	***	***	***	***	***	***	***

Legend:
* Complete <25 MB/s
** Complete 25-50 MB/s
*** Complete >50 MB/s
X Failed
O Expired

We accomplished both of these goals by establishing an automated test mechanism. The mechanism performs a set of tests across all of the FBIRN GridFTP servers on a continuous basis and records the results so that they may be reviewed over a period of time to get a sense of the system's stability and performance. Two types of tests are performed. A relatively short data transfer between each pair of data centers is performed frequently (multiple times per day) in order to ensure that the services are operating normally. A relatively long data transfer between pairs of centers is performed infrequently (once per day or every other day) to measure the performance of the system.

Our data from automated, continuous testing confirmed what we suspected: that the majority of FBIRN's servers were stable and performed well. The data allowed us to easily identify and tend to several servers that were unreliable (either erratic or consistently failing). In each case, the issue was caused by a unique combination of GridFTP software mis-configuration, operating system mis-configuration, or institutional firewall configuration. The data also allowed us to provide the FBIRN user community with a sense of the performance they can expect under normal circumstances, given the limitations placed on the system by firewalls, network connections, and hardware configuration. This has helped to give FBIRN users a sense that they can rely on the system to behave predictably, allowing them to focus their attention on scientific challenges.

A continuous view of data from the monitoring system for the period 5/21/2010 to 5/29/2010 is shown in table 2. Site A exhibits consistent failures to several other sites through 5/27. The site administrator corrected a configuration problem, and as of 5/28 the errors cleared up. Site B exhibits consistently successful behavior, so it did not require any attention. When moving data from Site B, users at sites D, H, I, and J could expect high performance, sites F and G could expect moderate performance, and sites A, C, and E could expect relatively low performance.

V. Related Work

Distributed file systems [24, 25, 26] can be used to enable access to remote data while maintaining file system semantics. These systems can replicate data transparently across sites and provide a level of fault tolerance. Our system is intended for use in loosely coupled environments where file system semantics is neither achievable nor desirable.

Before it adopted the framework described in this paper, FBIRN used Storage Resource Broker (SRB) [27, 28] for data publication and sharing. SRB uses a logically centralized catalog (MCAT) to manage data publication, metadata and replica creation and consistency. In contrast, our framework supports more relaxed consistency among replicas and offers distributed file registry services that eliminate the bottleneck of the centralized catalog approach of MCAT. An important advantage for FBIRN users is that our framework maintains data in file systems, allowing access using conventional clients. FBIRN's transition from SRB to the presented system was motivated by unacceptably slow and unreliable data transfers, bottlenecks due to the reliance on a centralized metadata catalog, and lack of access to local data when central services were offline.

Another approach to management of images and metadata is the XNAT (eXtensible Neuroimaging Archive Toolkit) system [29] from Washington University, which has some analogous capabilities to the present system. XNAT is positioned as a research imaging archive and offers support for medical imaging protocols, annotations, extensible metadata schema, and file storage. Unlike the system we describe, XNAT does not attempt to federate storage systems for large collaborative research groups, and it allows access to images via medical imaging protocols, a custom web service protocol, and its Web user interface. In cases where a scientific application wishes to keep data storage centralized, a system such as XNAT may provide the preferred approach.

Pipelining [30] and concurrency [22] approaches in GridFTP improve the transfer performance of datasets with many small files. In order for pipelining to be effective, data channel caching in GridFTP is required, which is available only in MODE E protocol [6] in GridFTP. But MODE E protocol often cannot be used in this environment due to the fact that inbound connections are prohibited on the user (data receiver) end. In the absence of MODE E, concurrency requires that many ports be opened on the server (data source), which is also not possible. The tar functionality in GridFTP (developed for this framework) improves performance for many small files without the need for MODE E protocol and/or more open ports at the server.

VI. Summary

We have developed a data management framework for distributed biomedical research applications. The framework was motivated by the need of FBIRN researchers to securely and reliably transfer data stored at multiple sites nationwide. To this end, we adopted a loosely coupled approach comprising a registry catalog, a credential repository, a group membership service, and a file transfer service and relying on an underlying Linux file system for access control. To facilitate the transfer of many small files, we implemented a tar-enabled feature in GridFTP. To avoid connection dropping, we use TCP keepalive to probe idle connections frequently. We also developed a continuous monitoring framework to measure the system's stability and performance. With these new capabilities, we are reliably and efficiently transferring large volumes of data among FBIRN sites to enable collaborative imaging studies that produce and share millions of files over wide area networks between secure biomedical research facilities.

Acknowledgment

This work was supported in part by the Biomedical Informatics Research Network (1 U24 RR025736-01). This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

REFERENCES

[1] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist, "X.509 Proxy Certificates for Dynamic Delegation," 3rd Annual PKI R&D Workshop, 2004.

[2] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, "A Security Architecture for Computational Grids," in *5th ACM Conference on Computer and Communications Security Conference*, 1998, pp. 83-92.

[3] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch, "A National-Scale Authentication Infrastructure," IEEE Computer, vol. 33, no. 12, 2000, pp. 60-66, 2000.

[4] Chervenak, A. L., Palavalli, N., Bharathi, S., Kesselman, C., and Schwartzkopf, R. Performance and Scalability of a Replica Location Service. 13th IEEE international Symposium on High Performance Distributed Computing, 2004

[5] A. Chervenak, R. Schuler, M. Ripeanu, M. Amer, S. Bharathi, I. Foster, A. Iamnitchi, and C. Kesselman, "The Globus Replica Location Service: Design and Experience," *IEEE Transactions on Parallel and Distributed Systems,* vol. 20 no. 9, pp. 1260-1272, 2009.

[6] Allcock, W. GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-R-P.020, 2003.

[7] Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S., and Foster, I. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. 18th IEEE Symposium on Mass Storage Systems and Technologies, 2001

[8] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus Striped GridFTP Framework and Server," SC'05, ACM Press, 2005

[9] "Function BIRN," http://www.birncommunity.org/current-users/function-birn/.

[10] Keator DB, Grethe JS, Marcus D, Ozyurt B, Gadde S, Murphy S, Pieper S, Greve D, Notestine R, Bockholt HJ, Papadopoulos P; BIRN Function; BIRN Morphometry; BIRN-Coordinating. A national human neuroimaging collaboratory enabled by the Biomedical Informatics Research Network (BIRN). IEEE Trans Inf Technol Biomed. 2008 Mar;12(2):162-72.

[11] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network File System (NFS) version 4 Protocol," IETF, RFC 3530, 2003

[12] "Rsync," http://www.samba.org/rsync/, 2009.

[13] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," IETF, RFC 4251, 2006

[14] "GridFTP URL Schemes," http://www.globus.org/toolkit/docs/latest-stable/data/gridftp/user/#globus-url-sync-syntax, 2010

[15] A. Chervenak, R. Schuler, M. Ripeanu, M. Amer, S. Bharathi, I. Foster, A. Iamnitchi, C. Kesselman, "The Globus Replica Location Service: Design and Experience," IEEE Transactions on Parallel and Distributed Systems, pp. 1260-1272, 2009

[16] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson, "Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile," IETF, RFC 3820, 2004

[17] J. Basney, M. Humphrey, and V. Welch. "The MyProxy Online Credential Repository," Software: Practice and Experience, Volume 35, Issue 9 pages 801-816, 2005

[18] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," IETF, RFC 2459, 1999

[19] S. Tweedie. "Journaling the Linux ext2fs Filesystem" Proceedings of the 4th Annual LinuxExpo, Durham, NC, 1998

[20] POSIX 1003.1e draft specification "http://www.suse.de/~agruen/acl/posix/posix_1003.1e-990310.pdf

[21] "Grid Grouper" http://cagrid.org/display/gridgrouper/ 2009

[22] "Globus XIO popen driver," http://www.mcs.anl.gov/~bresnaha/Stretch/

[23] R. Kettimuthu, A. Sim, D. Gunter, B. Allcock, P. Bremer, J. Bresnahan, A. Cherry, L. Childers, E. Dart, I. Foster, K. Harms, J. Hick, J. Lee, M. Link, J. Long, K. Miller, V. Natarajan, V. Pascucci, K. Raffenetti, D. Ressman, D. Williams, L. Wilson, L. Winkler, *"Lessons learned from moving Earth System Grid data sets over a 20 Gbps wide-area network"*, 19th ACM International Symposium on High Performance Distributed Computing (HPDC), 2010

[24] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N. and West, M.J. Scale and Performance in a Distributed File System.*ACM Transactions on Computer Systems*, *6* (1). 51-81. 1988.

[25] M. Satyanarayanan, J. Kistler, E. Siegel, "Coda: A Resilient Distributed File System," IEEE Workshop on Workstation Operating Systems, Cambridge, MA, 1987

[26] S. Weil, S. Brandt, E. Miller, D. Long, C. Maltzahn, Ceph: A Scalable, High-Performance Distributed File System, 7th Conference on Operating Systems Design and Implementation (OSDI), November 2006.

[27] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S. Y. Chen, and R. Olschanowsky, "Storage Resource Broker-Managing Distributed Data in a Grid," *Computer Society of India Journal, Special Issue on SAN,* vol. 33(4), pp. 42-54, 2003.

[28] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in *1998 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'98)*, Toronto, Ontario, Canada, 1998, p. 5.

[29] "XNAT," http://www.xnat.org/.

[30] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster, "GridFTP Pipelining," in *Teragrid 2007 Conference* Madison, WI, 2007.