A Proxy Service for the xrootd Data Server

Andrew Hanushevsky, Heinz Stockinger
Stanford Linear Accelerator Center (SLAC), Stanford University
2575 Sand Hill Road, Menlo Park, CA-94025, USA
{abh|heinz}@slac.stanford.edu

Abstract

In data intensive sciences like High Energy Physics, large amounts of data are typically distributed and/or replicated to several sites. Although there exist various ways to store and access this data within a Grid environment, site security policies often prohibit end user access to remote sites. These limitations are typically overcome using a proxy service that requires limited network connections to and from remote sites.

We present a novel proxy server for the xrootd data server that provides access to data stored in an object-oriented data store (typically in ROOT format). Our proxy service operates in a structured peer-to-peer environment and allows for fault tolerant and reliable access to remote data.

1 Introduction

In a Data Grid environment security is one of the main issues. On the one hand, data needs to be securely stored in specific data stores to guarantee local site security as well as data integrity. On the other hand, network access imposes additional security concerns that might not be directly addressed by the storage system. Grid middleware, file servers as well as database specific client-server or peer-to-peer applications need to deal with network security. One of the most commonly applied solutions is to set up firewalls and restrict the number of in- and/or outbound network connections.

Often, client applications in a Data Grid are executed in an Internet Free Zone (IFZ) or run on machines that do not allow for outgoing network connections. However, due to the nature of a Data Grid, data is distributed and/or replicated to several data stores outside the LAN where the client application needs to execute. In addition, a client application may need to access non-local data and therefore requires access to remote sites without compromising the local security infrastructure. One way to overcome this issue is to apply a proxy server that relays client requests to a remote machine outside the local firewall.

A proxy server adds an additional point for retrieving and temporarily caching data, and thus potentially means a slight overhead, appearing as increased latency, in data transfer. However, this overhead is often accepted in cases where security concerns are more important than very high speed data access.

One of the most commonly used data stores for data intensive sciences in the High Energy Physics community is the ROOT framework [9]. It allows for data storage, retrieval, physics specific data analysis, etc. Remote data access is usually done via a ROOT data server called rootd (ROOT daemon). Recently, this data server has been enhanced by xrootd [6], an extended, high performance data server. Xrootd was partly developed to access ROOT files, but it can be considered as a general data server that provides POSIX like file access to any kind of data. Although xrootd is a very reliable and secure server, the problem of outbound network connectivity still needs to be tackled.

In this paper, we describe another extension to the xrootd data server system where proxy servers take care of local client requests and then serve the data from a remote site if required. In this way, application clients only need to have network connectivity to the local proxy server(s) and access to remote data is guaranteed.

This paper is organized as follows. We first give an introduction to the problem domain of data management in a scientific environment in Section 2. In particular, access to distributed, remote data and potential security concerns are discussed. In Section 3 we give a general overview of the xrootd system. This brief introduction to the architecture provides the necessary background for the main discussion of the proxy service in Section 4. Experimental results obtained with our implementation are shown in Section 5. We then give related work on proxy services and data management before we conclude with final remarks and future work.

2 The Problem Domain

Several data intensive sciences like High Energy Physics require access to remote data. However, site administrators do not always allow client applications to

directly access remote files because of network security issues. This is particularly true for Data Grids where client applications are executed on a worker node that is in an Internet free zone.

Several Grid data management solutions in major Grid projects do not directly address this problem, and mainly deal with entire file transfers [15,1] rather than POSIX like read/write file access. Given that a majority of physics data is stored in object stores such as ROOT [9], there is a clear need for a high level file access in order to analyze data. Due to the distribution of both, users and data (potentially all over the world), remote access is of major importance. In addition, partial and random file access is needed in order to reduce network traffic and increase end user access performance.

Physics experiments such as BaBar [12] at SLAC or Large Hadron Collider experiments at CERN generate large quantities of data that are stored both on disks and on tertiary storage devices that include low-latency tape and hierarchical storage systems [5]. Although our proposed system takes care of interaction with such systems, we limit our discussion to disks only.

If no direct access to remote data is allowed, proxies are one way to allow for reading remote data. This implies that a proxy server acts on behalf of a user, contacts a remote data server, reads the requested data and sends the result back to the client. Proxies are very common in the Internet and typically used by web servers. In a Data Grid environment where data security is of major importance, proxies can be used for all possible services that would need data or information from a remote site. In this way, the site firewall can block in- and outgoing network connections on machines where client applications are running but still allow a limited amount of external network activity on dedicated proxy server machines.

Another typical scenario where a proxy is important is when important application metadata is not replicated but located at a single location. This is true for some High Energy Physics experiments.

3 General Overview of xrootd

The following section gives a general overview of the xrootd data server and its environment. We first explain the main components involved in order to have the necessary back ground for the detailed discussion on the proxy service in the next section.

3.1 The Basics of the Data Server

The basic purpose of the xrootd data server is to provide access to files stored on secondary (disk) or tertiary storage devices (tape or hierarchical storage devices). The files do not have to be organized in special formats since the data server provides file I/O similar to a POSIX file system interface. In particular, files that reside in a certain file system are served by the xrootd data server. Using the xrootd, end users that want to access files in a certain domain need to connect to the server which then provides read or write access to a client. Client-sever communication is done via the xrootd protocol [6].

A typical example for such a client-server communication is depicted in Figure 1. The xrootd server runs on a certain machine and serves all files that are located in a specific directory: in our example it is /data. Assume a remote user on the same local area network but without direct access to the disk of the xrootd server. This user now wants to open the file /data/file1 (using a POSIX like *open* command), get the file size (using *stat*) and then read 1,000 bytes of that file starting from offset 3,000 (using the *read* call). In order to achieve all this, a client connects to the server using the xrootd protocol via respective client libraries. All previously mentioned calls are then sent to the server which in turn reads the file from disk and sends the requested bytes back to the client.

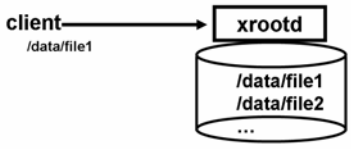

Figure 1. Basic client-server interaction with an xrootd server.

The xrootd request/response protocol contains more file system related calls such as chmod, getfile, putfile and write. For more details we refer the reader to [6].

3.2 Design Principles

xrootd is a high-performance data server where several hundred user requests can be served in parallel, depending on the hardware configuration. However, a single data server is often not resilient to many kinds of server, network or hardware failures. Therefore, the xrootd system is designed to allow for several data servers where client requests get automatically redirected to a different data server for load balancing and fault tolerance reasons. In other words, multiple data servers located on multiple machines can serve client requests potentially for the same file. Based on the load of a single data server and the location of a requested file, a client request gets redirected to one of the servers that is potentially able to serve the request with minimal latency.

Given that the xrootd system is not a single client-server system but requires servers to be coordinated, we can distinguish the following main interactions in the system.

- On the one hand, the system needs to find out where data is located and which server is least loaded to provide the requested data. We refer to this as **control flow**. Part of the control flow can be compared to replica lookups with redirection as we reported in [12].
- Once a suitable data server is selected, the actual data needs to be transferred between the data server and the client. We call this the **data flow.**

The distinction between control and data flow is rather common and is found in many peer-to-peer file sharing systems (BitTorrent, GnuTella), distributed file systems (Google, Lustre, PFS) and data transfer protocols such as FTP. One of the main design principles of the xrootd is a separation of **data flow** and **control flow** when serving data to a client.

In order to allow for separation of concerns, data flow is achieved by data servers (i.e. the **xrootd** discussed in Section 3.1) whereas the control flow is done by servers that are dedicated to that purpose. Since the main purpose of such servers is to locate data and balance the load they are called **open load balancers** (**olb**) in this context. Simply put, data severs and load balancers work together to serve client requests. An end user only needs to know about data servers and their locations in order to retrieve data. Therefore, whenever we talk about the xrootd system in this paper, we refer to the entire system that consists of data servers and load balancers.

3.3 Basic Architecture

The overall architecture of the xrootd system consists of several components that are implemented in either the data server or the load balancer. The orchestration and interaction of these main servers are implemented as a **structured peer-to-peer system**, where data servers and load balancers can have different roles to achieve the overall goal to serve data. Before we go into the details of the server interaction, we describe the different roles that servers can have, as depicted in Figure 2.

A data server (xrootd) can act either as a pure data server or as a redirector:

- A **pure data server** knows the xrootd protocol and can only serve data located on a certain file system. Its task is to handle a client request for data and directly transfer data to the client. On the bottom of Figure 2, we have three pure data servers named xrootd, each potentially located on a different machine in the same local area network.
- A **redirector** knows the xrootd protocol and accepts client calls for file access but rather than serving data directly, it interacts with load balancers in order to find a suitable data server that can serve the client request.

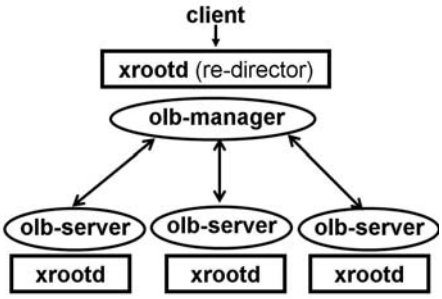

Figure 2. Interaction in the xrootd structured peer-to-peer system. It shows the clear distinction between control and data flow as well as the different roles.

The interaction of load balancers, also referred to as the **control network**, consists of identifying where data is located and which of the pure data servers (xrootds) will serve data to the client. In order to structure the interaction in the control network, a load balancer (olb), can have one of the following roles: manager or server.

- An olb-manager (also referred to as **manager**), has the task to accept data location requests from an xrootd redirector. Next, the manager needs to interact with olb-servers to check the current load and the availability of files.
- An olb-server (also referred to as **server**), subscribes to an olb-manager and thus is committed to provide data location information to the manager. Once a data server is located, the redirector redirects the client to that server.

Note the difference between olb-server and data server. Whereas the olb-server only provides the location information (control flow), the xrootd data server really serves the actual data (data flow). In addition, the interaction between olb-managers and olb-servers is done using the olb protocol rather than the xrootd protocol.

We call this architecture structured peer-to-peer because at any instance has a specific role. This provides critical administrative predictability to the system. On the other hand, it is peer-to-peer since servers act as clients under certain conditions.

3.4 Request Flow

We now extend our example from Section 3.1 in order to explain the request flow (control and data flow) for a typical read request.

1. The end user client contacts an xrootd server that, typically unknown to the client, acts as a redirector.
2. The redirector asks its associated olb-manager, which may or may not be co-located, for the location of the file.
3. The olb-manager interacts with all its olb-servers that have subscribed to the control network. Each olb-server reports if it has an instance of the requested file.
4. The olb-manager collects all responses. If a suitable data server has been reported, the olb-manager returns the hostname of the data server to the redirector.
5. The redirector sends the hostname back to the client, indicating that the client request is redirected to a new xrootd.
6. The client contacts the xrootd server indicated by the redirection response of the redirector. If the current xrootd is configured to be a pure data server, the requested bytes for the read request are sent back to the client. Otherwise, the process is repeated until the client is redirected to a pure data server.

The design and implementation of the system allow that olbs and xrootds can be used with other systems like Objectivity [18] and thus allow for a wide range of interoperability. The current servers can be re-used as long as they speak either the xrootd or the olb protocol.

3.5 Data Security

Up to now, we mainly discussed network security issues related to firewalls and network connections. Within a site, secure access to data is fundamental, in particular for the interaction with the xrootd. The xrootd allows for a secure access to data via authentication of clients as well as authorization policies on the server side. The authorization is handled via ACL like access permissions that are stored in a separate configuration file. By means of a generic security interface, several security protocol implementations can be used, i.e. dynamically loaded at startup time of the xrootd server.

The basic request authentication is done when the client contacts the server to open a file. The xrootd protocol internally requires a login as well as an authentication procedure for this purpose. The flexible security interface allows for several authentication steps, based on the used security protocol. Currently, the system has implementations of Kerberos security, but is also extensible to the more commonly used Grid Security Infrastructure (GSI).

4 The Proxy Service

The architecture presented in Section 3 allows for a flexible system of data and load balancing servers distributed to several locations within a local area network or spread around in a Grid environment. The system also allows for access to replicated data in several locations. However, in all cases, a redirection request to a new data server requires that a client (used by an end user application) has external network connectivity to a remote data server. In addition, the load balancer control network requires potential outbound network connectivity if load balancers of remote sites (olb-servers) have subscribed to the local load balancer (olb-manager), i.e. olb-manager and olb-server reside in different sites. In the following section we introduce a proxy service that extends the presented architecture.

4.1 Architectural Overview

The main aim of the proxy service is to act on behalf of a client (in this case also on behalf of the end user). The proxy service retrieves the requested data from a remote data server and sends it back to the client. The basic concept is rather straight forward. However, the challenge is to integrate the proxy service with the existing peer-to-peer system of data servers and load balancers outlined above.

Basically, the xrootd data server is extended to contact a remote xrootd data server. Since the original data server acts as a client to the remote data server, it needs to use the xrootd protocol like any other client. Therefore, in our peer-to-peer system we allow for an additional interaction between a pair of xrootd servers to exchange data.

We recall that the xrootd acting as a pure data server only knows about its local files, and it does not know about any other hosts. Only a redirector knows about an olb-manager and can contact it in order to obtain the location of a file served by a pure data server[1]. Given this, we extend the functionality of a redirector such that an xrootd can serve as a **proxy**. In detail, a proxy-capable xrootd (also referred to as proxy server) first needs to ask its corresponding olb-manager for the location of a file. This lookup is equivalent to the lookup for a potential redirection. However, rather than returning a redirection response to the client, the proxy-capable xrootd directly connects to the remote data server and relays the received client request to the remote server, acting on behalf of the client. Once the proxy server has received the requested data from the remote server, it can send it back to the client. This simplified information flow is depicted in Figure 3. Note that although the proxy has now the feature of a redirector in the sense that it can ask the control network for the location of a remote file, the

[1] We restrict our discussion to this simple arrangement. In fact, the xrootd may redirect a client to a virtual data server, sometimes called aggregator, that will in-turn redirect a client to a pure data server. This scenario is outside the scope of this paper.

proxy is also similar to a pure data server since it also serves data to the user rather than redirecting the client request.

The minimal peer-to-peer topology presented in Figure 3 allows for shielding the client completely from the Internet in case the remote data server is located in a remote site only reachable via a wide area network connection. Therefore, a client has potential access to data distributed all over the globe but only needs to access a local proxy server that then retrieves the requested data.

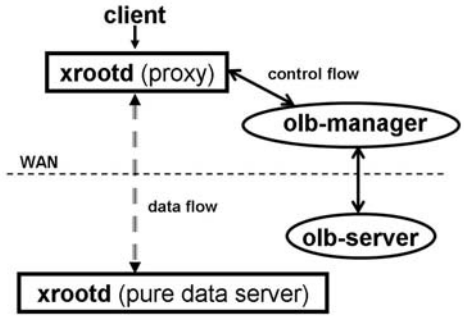

Figure 3. Simplified interaction between client, proxy and remote data server.

Since both the olb-manager as well as the proxy server requires outbound network connectivity to the Internet, only two network ports have to be opened to the "outside" world: the port where the olb-manager talks in order to retrieve data location information, as well as the port where the proxy server sends its data requests to a remote data server. The client itself, typically located in the Internet Free Zone (IFZ), only needs to have access to the proxy server and does not require any further outbound network connections.

The described architecture so far only requires a modification/extension for xrootd data servers whereas load balancers are unchanged. In other words, no modification to the code of the olb was required. This is fine for a simple scenario with a single proxy server. However, if there is a conventional redirector in the local area network, a potential client request might still get redirected to the remote data server since the control network system does not distinguish between servers within the local domain and remote domain when selecting file locations. This apparent problem is readily solvable by a simple configuration options. We can impose the configurable rule that local data servers and remote data servers may not subscribe to the same olb-manager.

4.2 Configuration/Deployment Options

Both, xrootds and olbs are configurable and allow for several different deployment scenarios. For simplicity, we previously mentioned only the minimal number of servers in the structured peer-to-peer system to explain the basic interaction. However, in a real deployment environment where thousands of users and potentially millions of files are distributed to several sites in a Grid environment, a single proxy and/or a single data server per site is not sufficient. One can also have several redirectors or several proxies per site. Furthermore, while we always coupled xrootds with olbs (manager or server) on the same machine; one can use any number of xrootds and olbs on any number of machines. Further, as mentioned previously, olbs can act as aggregators for other olbs. Thus, one olb might actually report file information for several xrootd data servers.

The xrootd data server can either serve data directly from a disk location or contact a mass storage system such as HPSS [5] to stage the file. Once the file is staged, the xrootd can serve it to the client. Since a data server acts as a proxy, we still provide this configurable option that files can also be staged. In addition, a proxy first tries to find a file on a potential local disk and only if it fails, it requests the file from remote data server. In this way, potential transfer latencies are reduced if the file can be served locally. This also allows administrators to configure pure data servers to act as proxies as a matter of last resort.

4.3 Implementation

The entire xrootd system has been implemented in C++ and ported to several platforms (Sun Solaris 8, 9, Linux RedHat 7.x[2], Linux RedHat Enterprise edition 3.0) using native Solaris, GNU, and Intel compilers.

The standard xrootd system as presented in Section 3 has been fully implemented and is used in production in the BaBar experiment. Although the data servers are implemented using the xrootd protocol, the actual servers allow for other protocols by dynamically loading a protocol implementation compatible with a client's first interaction with the server.

For the proxy server, the most important xrootd methods have been implemented, i.e. login, open, read, stat, auth etc. in a synchronous, blocking environment. The xrootd dynamically loads a proxy object, which implements the xrootd client code, when the olb-manager returns the hostname and port information of a remote data server.

4.4 Additional Security Considerations

One of the key elements in a scalable architecture is to keep the number of interaction combinations to a

[2] Note that RedHat 7.x is currently the most commonly used Linux platform in the High Energy Physics community.

minimum. One reason that proxy servers are an important critical element in providing scalable security is that they automatically place a fixed bound on the number of combinations of client-server authentication requests that is independent of the number of clients. This simplifies key management because fewer keys need to be distributed. However, our proxy architecture provides for an even more significant simplification in required key-exchanges.

If we configure the system to prevent cross-domain data server subscriptions, then we are forced to have the local olb-manager communicate requests, as an olb proxy, to the remote olb-manager. This immediately provides for a key-synchronization control point. Only the olb-managers need to develop a session key. Then they, in-turn, can distribute the session key to their respective clients. By the definition of the architecture, those clients must be data servers and are the only entities that will need to do cross-domain authentication. This elegantly simplifies cross-domain authentication and key distribution while maintaining robust security.

5 Experimental Results

The xrootd system as described in Section 3 has been in active use in the High Energy Physics experiment BaBar for more than half a year, and physicists have access to data served by the system. We now give detailed experimental results for our proxy server and the interaction with the entire system.

5.1 Testbed Setup

The testbed we used for our experimental results corresponds to a typical Data Grid setup where an end-user client application gets executed on a worker node of a computing element [15]. For simplicity, we only assume one worker node that is behind a firewall and requires access to local data on the same local area network as well as remote data accessible via a proxy server. We used four different sites as indicated below. The client was deployed at SLAC with servers at each of the four sites. The numbers in brackets correspond to the Round Trip Time (RTT) of TCP packets from SLAC to the given site:

- SLAC: Menlo Park, California (RTT: local)
- Caltech: Pasadena, California (RTT: 13ms)
- CERN: Geneva, Switzerland (RTT:165ms)
- Pisa: Pisa, Italy (RTT:170ms)

Note that both links between SLAC and CERN as well as SLAC and Pisa represent high-latency, wide area network links. We used the following hardware for the performance tests of the xrootd system where each machine had a 100Mbps Fast Ethernet card:

- **Client** (SLAC): Sun Fire V.240, 1 GHz, 2 GB RAM
- **Proxy** (SLAC): Sun Fire V.240, 1 GHz, 2 GB RAM
- **Server** (SLAC): Dual Pentium III, 1.4 GHz, 2 GB RAM, 512 KB cache
- **Server** (Caltech): Intel Xeon, 4 CPUs, 2.8 GHz, 1GB RAM, 512KB cache,
- **Server** (CERN): Dual Intel Pentium III, 800 MHz, 256 KB cache, 512 MB RAM
- **Server** (Pisa): Intel Xeon, 1.7 GHz, 1 GB RAM, 512 KB cache

All Intel-based machines ran either RedHat Linux 7.2 or 7.3 with gcc 2.95.2, 2.95.3 or gcc 3.2 whereas the Sun based machines used Sun Solaris 9 and the native Sun compiler. All server machines above ran both the xrootd data server as well as a corresponding olb-server. The olb-servers were all subscribed to the olb-manager on the Proxy machine (Sun Fire). In addition, the proxy machine also hosted the xrootd proxy which was the main contact (proxy) for the client application residing on a second Sun Fire machine on the same LAN. This corresponds to the scenario depicted in Figure 3.

An important consideration when setting up the servers was which network port can be used. Since SLAC does not allow incoming network connections at certain ports, we selected a high port range for both olbs. Similar restrictions also applied to other sites like Pisa where the IANA assigned ROOT port (1094) was blocked.

5.2 Performance Results

In order to measure the performance of the newly introduced proxy server for read requests, we first isolate a few latency parameters. Note that the system has a certain start-up time because all peers need to talk to each other. Once an olb-server subscribes to an olb-manager, a certain interaction takes places where the manager collects information about the server. We call this the *start-up latency*. Once this is done, the manager knows about all its servers and which name spaces they are serving, i.e. which file path on disk is addressed. Next, each time a client requests a file for the first time (i.e. an open request is sent to the xrootd proxy), the olb-manager checks for the file location at the site of the olb-server. We call this the *look-up latency*: this time is a configurable parameter and is set to 5 seconds by default at the server startup. The olb-manager caches all the file locations it has looked up and keeps them for a default of 8 hours in its local cache.

We are mainly interested in the performance of the proxy server and the potential overhead it implies as compared to a potential direct data transfer between the

client and a remote server. We therefore compare the time of a read request on an open file. In the first experiment, the client requested to read all bytes of a 9.5 MB file issuing a read starting from offset 0. The results are depicted in Figure 4. ”proxy” corresponds to reading data via a proxy and “direct access” corresponds to reading the file directly from the remote data server where the file is located. In both cases, the client was located at SLAC.

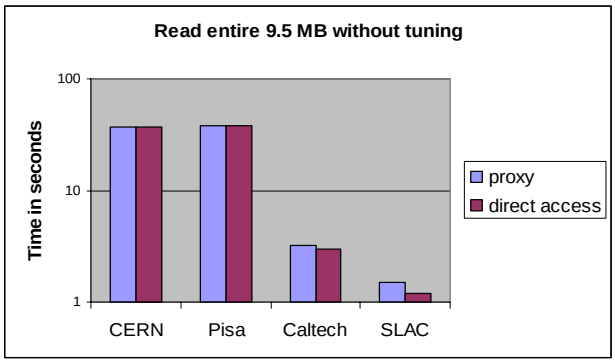

Figure 4. Read 9.5MB file via the proxy server from a remote site compared to read the file directly from the remote data server.

From the results in Figure 4 we see that for high-latency links like California (SLAC) to Europe (CERN, Pisa), the proxy server did not impose any additional overhead whereas for fast links between SLAC and Caltech, a small overhead can be seen. Note the logarithmic scale on the time axis. This overhead is also true for a local read.

We compared the read time to the transfer time of the same file via scp as well as bbcp[3] [17] in order to see how much we can potentially improve the access time over the wide area network. Transferring the same 9.5 MB with scp (single stream, no window size tuning) between SLAC and CERN resulted in a transfer rate of about 370KB/s. The transfer rate of xrootd is ~277 KB/s.

Next, we studied the effects of tuning the network transfer with bbcp and experimented with different window sizes and various parallel streams. We reached the optimum using 20 parallel streams and a window size of 1 MB. We gained an effective transfer rate of 1128 KB/s, which also includes the security handshake. The raw transfer rate, which corresponded to only reading the file, was 5232 KB/s. Consequently, we see that on the WAN link we needed to optimize the transfer rate by window size tuning and the usage of parallel streams.

In our current implementation of xrootd we only allow for window size tuning and thus all subsequent tests were run with a TCP window size of 1 MB. We therefore redid the pervious test and gained a transfer rate of ~355KB/s for the SLAC-CERN link, i.e. almost equivalent to scp. There is clearly more improvement possible once we apply parallel stream transfer as used in bbcp.

In the next test depicted in Figure 5 we tested partial file reads. We used the same 9.5MB file but only read 100 KB starting from offset 50,000**.** We observe that the raw transfer rate is higher with tuned window sizes.

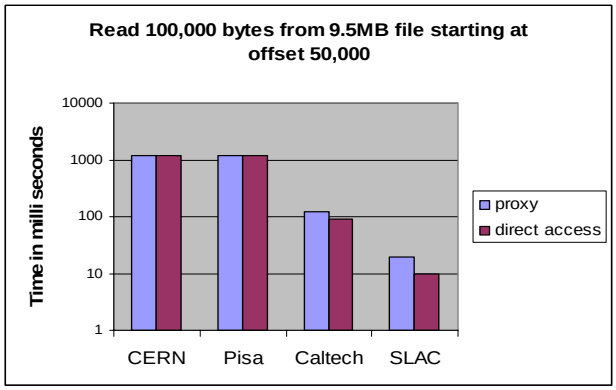

Figure 5. Read parts of a file with tuned TCP window set to 1 MB.

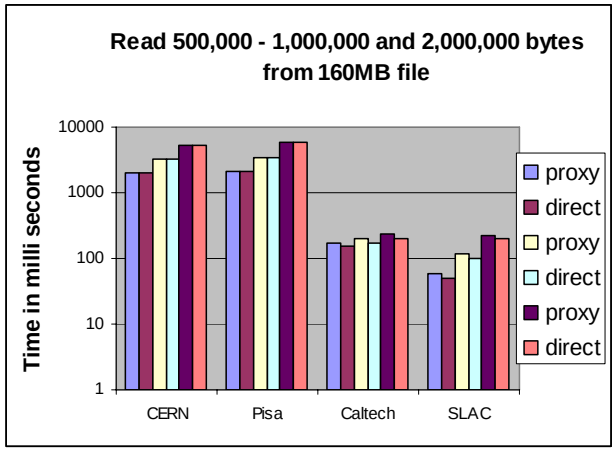

Figure 6. Read bytes with tuned TCP window size, single stream. On each machine, a set of 3 read operations (one set through the proxy, one with direct access) with different numbers of bytes are performed.

In a final experiment we tested partial file reads of a large file. We used a 160MB file and started reading at offset 3,000,000. We compared this again with a proxy as well as directly reading 500 KB, 1 MB and 2MB bytes of

[3] bbcp[b1] represents one of the many multi-parallel stream data transfer programs currently available. In addition to the myriad of tuning options designed to optimize data transfer operations, bbcp also incorporates a unique algorithm to deal with network congestion and high-latency links. We chose bbcp because, even with marginal tuning, it can deliver a transfer rate that is essentially limited by the hardware being used.

the same file**.** We have chosen these numbers since they clearly demonstrate the read latencies. If small numbers of bytes are read, the time measurements (in milli seconds) are not very significant for performance comparisons. The results in Figure 6 show that the proxy only has a small percentage of overhead. In addition, the read performance is increasing relative to the number of bytes read.

We also tested for fault tolerance and resilience to server failures. The proxy can dynamically respond to failures of data or olb servers. In addition, new olb and xrootd data servers can be added/deleted dynamically without any negative influence on the entire system.

6 Related Work

There has been considerable work done on proxy servers for web-based content with [7][8] spanning a decade of research. Proxies now exist for practically all protocols in many contexts; from protocol conversion [14] to dealing with firewall issues [16]. Considerable work in caching methodologies in distributed file systems, peripherally related to proxy services, represents a closely related area [2][3][10][11]. Perhaps the closest work in terms of what xrootd tries to do, though in many different ways, is the Google File System [4]. However, we know of no on-going work that uses a proxy service, let alone a structured network of control and data proxies, to address a typical quandary that develops in peer-to-peer file services: what happens when a file that was expected to be found in the local domain can only be found in a remote domain? While clients in unstructured, non-fire-walled, P2P systems can easily solve this problem by simply getting the file, structured systems are rarely so lucky**.** We feel that our approach, while not theoretically new, is technically novel and represents a significantly effective solution to a relatively recent problem in data management.

7 Conclusions and Future Work

We have presented a proxy server for the xrootd system that can be used to address network security issues related to in- and outbound network connections. This is an important solution to data intensive science applications like High Energy Physics.

Our current server only imposes a small, expected overhead for read access over wide area networks in a Data Grid. In order to reduce the overhead further, we might also add parallel streams for the actual data transfer. There is also the future potential to call external file transfer protocols like bbcp or GridFTP in case nearly the entire file needs to be read by the client. Other plans are to implement more proxy functionalities such as write access or to allow for asynchronous communication with the server.

Overall, our proposed proxy server shows promising results. We plan to do further acceptance and performance tests in the High Energy Physics community. However, our system is flexible enough to be adopted also in other domains.

Acknowledgements

We thank Flavia Donno and Kurt Stockinger for useful comments on the paper. This work was supported by Stanford Linear Accelerator Center and Stanford University under contract DE-AC03-76-SF00515 with the US Department of Energy.

References

[1] B. Allcock et al. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. *18th IEEE Mass Storage Conference*, San Diego, Apr. 2001.

[2] M. Blaze, Caching in Large Scale Distributed File Systems, PhD thesis, Princeton University, Nov. 1992.

[3] J. Howard, et. al., Scale and performance in a distributed file system. *ACM Transactions in Computer Systems*, 6(1):231-244, 1988.

[4] S. Ghemawat, H. Gobioff, S. Leung, File and storage systems: The Google File System, *19th ACM Symposium on Operating Systems Principles*, Bolton Landing, Oct. 2003.

[5] A. Hanushevsky, M. Nowak. Pursuit of a Scalable High Performance Multi-Petabyte Database. *IEEE Symposium on Mass Storage Systems*, San Diego, Mar. 1999.

[6] A. Hanushevsky. eXtended ROOT Daemon (xrootd), http://www.slac.stanford.edu/xrootd

[7] A. Luotonen and K. Altis, World-wide web proxies, Computer Networks and ISDN systems, *Int. Conference on the World-Wide Web*, Elsevier Science BV, 1994.

[8] V. S. Pai, L. Wang, et. al., The dark side of the Web: an open proxy's view, *ACM SIGCOMM Computer Communication Review*, 34(1): 57-62, Jan. 2004.

[9] ROOT Framework, http://root.cern.ch

[10] M. Satyanarayanan, The evolution of Coda, *ACM Trans. on Computer Systems*, 20(2):85–124, May 2002.

[11] F. Schmuck, R. Haskin, GPFS A Shared-Disk File System for Large Computing Clusters. *Int.Conf. on File and Storage Technologies (FAST)*, USENIX, Monterey, Jan. 2002.

[12] H. Stockinger, A. Hanushevsky. HTTP Redirection for Replica Catalogue Lookups in Data Grids. *ACM Symposium on Applied Computing*, Madrid, Mar. 2002.

[13] H. Stockinger, A. Samar, S. Muzaffar, F. Donno. Grid Data Mirroring Package (GDMP). *Scientific Programming Journal - Special Issue: Grid Computing*, 10(2):121-134, 2002.

[14] http://de.samba.org/samba/samba.html

[15] http://www.edg.org

[16] http://www.ftpproxy.org/

[17] http://www-iepm.slac.stanford.edu/monitoring/bulk/bbcp.html

[18] http://www.objectivity.com