Globus Online: Radical Simplification of Data Movement via SaaS

Bryce Allen[*], John Bresnahan[*], Lisa Childers[*], Ian Foster[*¶], Gopi Kandaswamy[§], Raj Kettimuthu[*], Jack Kordas[*], Mike Link[*], Stuart Martin[*], Karl Pickett[*], Steven Tuecke[*]

[*]Computation Institute
Argonne National Lab & U. Chicago
Chicago, IL 60637, USA

[¶]Department of Computer Science
The University of Chicago
Chicago, IL 60637, USA

[§]Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292, USA

ABSTRACT

We propose that research data management capabilities be delivered to users as hosted "software as a service" (SaaS). We report on a system, Globus Online, that adopts this approach, initially for a small set of data movement functions. Globus Online leverages modern Web 2.0 technologies to provide fire-and-forget file transfers, automatic fault recovery, and high performance, while also simplifying security configuration and requiring no client software installation. SaaS means that new features are rapidly available, and provides for consolidated support and troubleshooting. We describe Globus Online's design and implementation. We also describe a novel approach for providing a command line interface to SaaS without distributing client software, and Globus Connect, which simplifies installation of a GridFTP server for use with Globus Online. Experiments show low overhead for small transfers and high performance for large transfers, relative to conventional tools, and demonstrate scalability to many concurrent requests and files.

General Terms

Management, Measurement, Performance, Design, Economics, Reliability, Experimentation, Security, Human Factors.

Keywords

Globus, Software-as-a-Service, SaaS, file transfer, data management, cloud, grid, Globus Online

1. INTRODUCTION

As big data emerges as a force in science [5], so too do new and onerous tasks for researchers. Data generated by specialized instrumentation, numerical simulations, and downstream analyses must be collected, indexed, archived, shared, replicated, and mined—and often manipulated in many other ways besides. These tasks are not new, but the complexities inherent in performing them for terabyte data sets, which are increasingly common across scientific disciplines, are quite different from those that applied when data volumes were measured in kilobytes or megabytes. The result is a crisis of sort in many research laboratories and a growing need for far more powerful data management tools.

We focus in this paper on a small subset of the overall research data management problem, namely the orchestration of data movement among two or more locations. This task may sound trivial, but in practice it is often tedious and difficult. Datasets may have complex nested structures, containing many files of varying sizes. Source and destination may have different security requirements and authentication interfaces. Network and storage server failures must be recovered from. Transfers must be carefully tuned to exploit high-speed research networks. Perhaps only some files differ between source and destination. Firewalls, Network Address Translation (NAT), and other network complexities may need to be dealt with. For these and other reasons, it is not unusual to hear stories of even modest-scale wide area data transfers taking days or even weeks, with frequent user intervention, or of being abandoned for high bandwidth but also high latency (and frequently also labor-intensive and error-prone) "sneakernet" [15].

Current research data movement solutions fall into three main classes. (1) *Client software* allows the user to move data between their own computer and a single remote location. Examples are secure copy (scp) and rsync, both of which provide ease of use and access to most facilities, but often only modest performance, and leave fault recovery to users. In effect, these systems are a lowest common denominator. (2) *Third-party systems* allow for the movement of data among two or more remote sites. Examples include globus-url-copy, Reliable File Transfer (RFT) [16], CERN's File Transfer Service (FTS), and LIGO's Data Replicator (LDR) [7]. These systems' support for third-party transfer is valuable for many users, and they often have better performance and fault behaviors than client systems, but the need to install and operate complex software can be a barrier to use. They are also often custom to specific data types and network configurations, and so can only be used by specific communities. (3) *Hosted approaches* introduce an intermediate system to which data is copied for subsequent download by its intended recipient(s). This approach, exemplified by systems such as email, Dropbox and YouSendIt, provides simplicity and a degree of delivery guarantee, but is not suitable for the extremely large datasets that are common in science.

Globus Online (GO) adopts a hybrid of the third-party and hosted approaches. It implements management functionality similar to (but more sophisticated than) that provided by systems such as RFT, FTS, and LDR, but delivers this functionality not as downloadable software but as hosted "software as a service" (SaaS). It is a cloud-hosted, managed service, meaning that you ask it to move data; it does its best to make that happen, and tells you if it fails.

SaaS allows GO to exploit economies of scale, as a single hosted service serves many individuals and communities. SaaS means that new features are immediately available to users, and allows experts to intervene and troubleshoot on the user's behalf to deal with more complex faults. It leverages widely available resource access services (e.g., GridFTP, and in the future HTTP) to integrate with storage resources, and operates across multiple security domains and technologies, so that it can be used across a wide range of facilities. GO's support for GridFTP delivers the benefits of grid technology, including support for heterogeneity and local control of access control and resource allocation policies at individual endpoints [14].

GO further leverages modern Web 2.0 technologies to provide extreme ease of use while requiring no client software installation. It can be accessed via different interfaces depending on the user and their application. A simple Web GUI serves the needs of ad hoc and less technical users. A command line interface via ssh exposes more advanced capabilities and enables scripting for use in automated workflows. A REST application programming interface (API) facilitates integration for system builders who do not want to re-engineer file transfer solutions for their end users; this REST interface also supports the GO Web interface.

The Web GUI allows a client to establish and update a user profile, and to specify desired authentication method(s). All access methods enable the client to specify the method(s) to be used to authenticate to the facilities to/from which data will be transferred; authenticate using various common methods, such as username and password, Google OpenID, OAuth [16], Shibboleth [13], or MyProxy [21] providers; characterize endpoints to/from which transfers may be performed; request transfers; monitor transfer progress; get detailed information about the transfer; and cancel active transfers. Having authenticated and requested a transfer, a client can disconnect, and return later to find out what happened. GO tells you which transfer(s) succeeded and which, if any, failed. It notifies you when a transfer completes, or if a critical fault has occurred such as a deadline not met, or a transfer requires additional credentials to proceed. In addition, GO can handle transfers across multiple security domains with multiple user identities.

We describe the GO design and its implementation, which leverages Amazon Web Services (AWS) infrastructure-as-a-service ("cloud") resources for reliability and scalability. We also report on experiments designed to evaluate the performance, scalability, and cost of the system. These experiments show that overhead for small transfers is modest relative to client-side tools and that performance for large transfers is usually not only far better than that of scp, but also significantly better than that achieved with out-of-the-box configurations of the widely used globus-url-copy, due to GO's ability to automatically tune transfers for high performance. We also demonstrate scalability to many concurrent requests and files.

This work is part of a project that aims to deliver research data management functionality as hosted "software as a service" (SaaS), rather than downloadable software that must be installed and operated by the user. In this way, we believe, we will be able to exploit economies of scale and the benefits of expert management for many individuals and communities, while alleviating research users of the burden of installing and operating complex software.

We believe that this paper makes two useful contributions to our understanding of research data management. First, we describe and evaluates an innovative new data management solution: the GO SaaS-based data movement service, which has its roots in "Grid" techniques but goes far beyond traditional Grid approaches. Second, we make the case for the use of SaaS methods for research data management more generally, arguing for the advantages of thus making sophisticated capabilities available to all researchers and facilities, in an easy-to-use and cost-effective manner.

2. REQUIREMENTS & RELATED WORK

We first define our problem and review previous approaches.

2.1 General Problem Statement

Our broad interest is in accelerating discovery by outsourcing challenging research management problems—that is, delivering sophisticated functionality to researchers in a manner that minimizes their IT burden and cost. The focus of the work presented here is on the specific problem of file transfer. Our target users need to copy a number of files, with potentially large aggregate size, among two or more network-connected locations ("endpoints")—reliably, rapidly, securely, and easily. These endpoints may or may not include the computer from which the data movement command is issued: in other words, third-party transfers may be (and indeed frequently are) involved. Our goal is a solution that provides extreme ease-of-use without compromising reliability, speed, or security.

A 2008 report [9] makes clear the importance of usability. In particular, failure recovery has often been a human-intensive process:

> "The tools that we use to move files typically are the standard Unix tools included with ssh. And for that we don't need more information—it's just painful. Painful in the sense of having to manage the transfers by hand, restarting transfers when they fail—all of this is done by hand. Then of course there are the hardware problems: dealing with the sluggishness on some of the networks like the ESNet, which we've had some problems with recently. The file transmission rates are painfully slow, errors occur and then we have to retransmit."

The same user bemoans the need to determine manually which transfers have succeeded and, perhaps more importantly, failed:

> "If I have a list of a few hundred files, and I want to get from point A to point B, and I start the process and something happens (it breaks or dies in the middle) I have to retransmit. But I don't want to retransmit all of it. The process of sorting through which one succeeded and which ones did not, and restarting. It's a time-consuming and annoying process and is something that slows down work."

One approach to data movement involves running tools on the user's computer. For example, rsync [26], scp, FTP, SFTP, and bbftp [17] are often used to move data between a client computer and a remote location. Other software, such as globus-url-copy, RFT [19], FTS, and LDR, can manage many transfers. However, the need to download, install, and run software is a significant barrier to use. Users spend much time installing, configuring, operating, and updating such tools, particularly when dealing with large data volumes, many files, high-speed networks, and/or frequent failures. Additionally, users rarely have the IT and networking knowledge necessary to fix things when it does not "just work", which as suggested by the user quoted above happens all too often.

Various big science projects have developed specialized solutions to this problem. For example, PhEDEx [23] manages data movement among sites participating in the CMS experiment, and LIGO developed the LIGO Data Replicator [8]. These centrally managed systems allow users to hand off data movement tasks to a third party that performs them on their behalf. However, these services require professional operators and only operate amongst carefully controlled endpoints within those communities: they cannot easily be installed, operated, or used by a smaller project amongst its endpoints.

Managed services such as YouSendIt and DropBox also provide data management solutions, but do not address our target users' need for high-performance movement of large quantities of data. BitTorrent [11] and Content Distribution Networks [28] such as Akamai, are good for distributing a relatively stable set of large files (e.g., movies), but do not address our target users' need for many, frequently updated files managed in directory hierarchies. Storage Resource Broker (SRB [4]) and its iRODS [22] successor are storage managers that are often run in hosted configurations. However, while these systems can perform some data transfer operations (e.g., for data import), data transfer is not their primary function or focus.

Kangaroo [25], Stork [18], and CATCH [20] manage data movement over wide area networks, using intermediate storage systems where appropriate to optimize end-to-end reliability and/or performance.

These systems are not intended as SaaS data movement solutions, but we could incorporate their methods within GO.

2.2 More Detailed User Requirements

Limited space does not permit a comprehensive presentation of user requirements, but we point out some important issues identified during our use case development process.

We assume for now that data is accessible via FTP or GridFTP [1], though as we describe below our Globus Connect agent makes it easy to make data accessible from other resources including end-user laptops, desktops and servers. (Support for HTTP, WebDAV, SRM [24], and other access methods is planned for the future.) Different endpoints may require different security credentials and configurations. Some may have firewalls that prevent incoming connections. The networks over which transfers are to be performed may vary greatly in their performance and reliability. Nevertheless, users desire fire-and-forget data transfer capabilities.

We find it useful to distinguish among multiple user types. *Ad-hoc users* perform mostly one-off transfers, as opposed to performing similar transfers repeatedly. They want to move data in the same way as they perform other tasks on their computer: via a convenient and easy graphical user interface. For them, a Web interface is essential.

Script writers create automated workflows to assist with their work: for example, a script that, each evening, transfers new files created during the day's work to a remote repository, or which automatically moves output from an analysis job back to their local machine. For them, a command line interface (CLI) is often desirable, as it allows such workflows to be specified using shell scripts or other simple solutions. One challenge to which we describe a solution below is how to deliver a CLI to users without violating the Software-as-a-Service tenant of no client software installation.

Finally, *systems builders* develop or adapt domain-specific tools for various communities, such that the file transfer is integrated into a larger system. A REST API is desirable, to facilitate integration without requiring re-engineering of the file transfer solutions that they deliver to end-users. System builders often want to customize the user experience, so that their users experience a consistent look and feel, even when they are interacting directly with GO.

Web and REST interfaces to centrally operated services have become conventional in business, where it underpins services such as Salesforce.com (customer relationship management), Google Docs, Facebook, and Twitter. It is not yet common in science. Two exceptions are PhEDEx Data Service [12], which provides both REST and CLI interfaces to CMS data operations, and the NERSC Web Toolkit (NEWT) [10], which enables RESTful operations against HPC center resources.

3. DESIGN AND IMPLEMENTATION

We present the system design from the perspective of the user and also describe important elements of its implementation.

3.1 Architecture Overview

As illustrated in Figure 1, GO comprises one or more *user gateway* servers, which support interaction between users and the system via the Web GUI, CLI and REST interfaces; one or more *workers*, which orchestrate data transfers and perform other tasks, such as notify users of changes in state; and a *profiles and state database* for user profiles, request state, and endpoint information. Web, command-line, and REST interfaces provide similar capabilities, although not all functionality is available through the Web GUI as of June 2011.

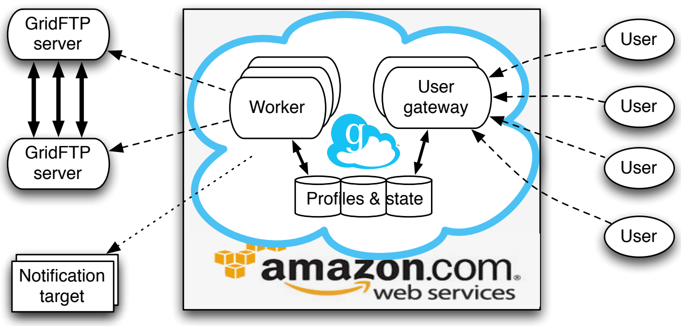

Figure 1: Schematic of the Globus Online architecture

The **REST interface** uses HTTP GET, PUT, POST, and DELETE operations against a defined set of URLs representing GO resources. Thus, to create a transfer task, we issue a POST to https://transfer.api.globusonline.org/v0.10/transfer with a document describing the transfer request (source and destination endpoints and file paths, options, etc.). To access the status of a task with a specified <task_id>, we issue a GET request to https://transfer.api.globusonline.org/v0.10/task/<task_id>; a document is returned with the status information. The REST interface is versioned, so GO can evolve its REST interface without breaking existing clients. Documents passed to and from HTTP requests can be formatted using JSON or XML. Supported security mechanisms include HTTPS mutual authentication with an X.509 client certificate, and HTTPS server authentication with cookie-based client authentication (for Web browsers).

The GO **Web interface** builds on the REST interface, using standard AJAX techniques. A GO Web page contains standard HTML, CSS, and Javascript, and interacts with the REST interface using standard session cookie-based client authentication. The Web GUI supports browsing remote file systems, and submitting, monitoring, and cancelling transfer requests.

A **command line interface** (CLI) supports client-side scripting. However, a CLI typically requires installation of client-side libraries, which is counter to a key SaaS tenet of not requiring client software installation to use the service. To provide SaaS CLI, GO has employed a novel "CLI 2.0" approach that eliminates the need to install client software (for those machines already configured with ssh). It provides all GO users with a restricted shell, to which any user can ssh to execute commands. Thus, I can write

```
ssh ian@cli.globusonline.org \
  scp alcf#dtn:~/myfile nersc#dtn:~/myfile
```

to copy myfile from source alcf#dtn to destination nersc#dtn. The boldface text invokes the GO scp command, which mirrors the syntax of the popular scp. It supports many regular scp options, plus some additional features—and is much faster because it invokes GridFTP transfers. Alternatively, I can first ssh to cli.globusonline.org and then issue a series of commands directly:

```
ian$ ssh cli.globusonline.org
Welcome to globusonline.org, ian.
$ scp alcf#dtn:~/myfile nersc#dtn:~/myfile
Contacting 'gs1.intrepid.alcf.anl.gov'...
Enter MyProxy pass phrase: ********
```

This example also illustrates how endpoints can define logical names for physical nodes. For example, alcf#dtn denotes the data transfer nodes running GridFTP servers at the Argonne Leadership Computing Facility. Sites can define and publish their own endpoint definitions (e.g., alcf#dtn, nersc#drn); users have the ability to define custom endpoint definitions as well (e.g., mylaptop, myserver).

Table 1 lists the primary transfer management functions supported by the GO interfaces. Additional endpoint management functions provides for the creation, deletion, configuration, activation, and deactivation of logical endpoints. Other functions support administrative tasks, such as listing available commands, reviewing command history, and obtaining command help.

GO maintains a database of active transfer requests ("tasks") and, for each task, its current state. The state transition diagram associated with a task is simple. A request is submitted and (subject to limits on the number of active requests allow for one user at one time) becomes active. The task then remains active until it is suspended because of an expired credential; completes successfully because all files in the task have completed; or fails, either due to the deadline defined with the task being reached, or the user cancelling the transfer. When executing a task, GO attempts to execute each file transfer in turn; multiple concurrent transfers may be used to improve performance when dealing with small files. If a transfer fails, GO will keep attempting it periodically until either the task deadline is reached or the user cancels the request.

3.2 User Profile and Identity Management

An important feature of GO is its ability to handle transfers across multiple security domains with multiple user identities. Unlike most other systems, including most previous Grid file transfer services, GO does not require the use of a single, common security credential across all endpoints involved in a transfer. Rather, GO assumes from the start that users have many identities for use with many different service providers, and that it is GO's job to make sure that the right identities are brought to bear at the right time for any transfer; and to do so in a way that is easy for users to understand and use.

Table 1: Principal Globus Online data transfer commands

Class	Name	Description
Create transfer	`ls`	List files and directories on an endpoint.
	`transfer`	Request data transfer of one or more files or directories between endpoints; supports recursive directory transfer and rsync-like synchronization.
	`scp`	Request data transfer of a single file or directory; syntax and semantics based on secure copy utility, to facilitate application porting
Monitor transfers	`status`	List transfers initiated by requesting user, along with summary information such as status, start time, completion time, etc.
	`details`	Provide details on a transfer, such as number of file transferred, number of faults, etc.
	`events`	List events associated with a specified transfer (start, stop, performance, faults)
Control transfers	`cancel`	Terminate the specified transfer or individual file in a transfer
	`wait`	Wait for the specified transfer to complete; shows a progress bar
	`modify`	Alter the deadline for a transfer

To this end, each GO user has a GO account, in which that user can easily configure their profile with their various identities. For example, a user can easily add to their profile their MyProxy CA [21] identities (e.g., for NERSC, ALCF, and other computing centers that use this approach), OAuth [16] identities (e.g., for TeraGrid or Facebook), OpenID identities (e.g., for Google), Shibboleth [13] identities (e.g., for campus credentials), and X.509 identities (e.g., from DOE Grids CA or another IGTF-certified CA).

While GO can store identities, it does *not* store passwords. Rather, it know just the identity and how to use it, so that it can prompt the user with the appropriate information when that identity is needed. In addition, some or all identities can be configured as "federated identities" that the user can use to authenticate with GO. For example, if I have already authenticated my browser session with an OpenID, OAuth, or Shibboleth identity, I can use the GO Web site without having to authenticate further; X.509 (proxy) identities can be used to authenticate with the GO Web site, CLI, and REST API.

GO keeps track of what security credentials are required by the different endpoints with which a user may wish to communicate. Then, where possible, it caches information that it can use to facilitate access. For example, assume that user U must provide X.509 credential U-A to access endpoint A, and X.509 credential U-B to access endpoints B1 and B2. In order for GO to perform a file transfer from A to B1 as requested by the user, GO requires short-term (typically 12 hour) X.509 proxy credentials [27] that it can use to authenticate the user request to the GridFTP servers running at endpoints A and B1. If GO does not have such credentials, it will prompt the user for them when the user requests the transfer. For example, if endpoints A and B1 each use their own MyProxy CA servers, GO will prompt the user for their site logins at each endpoint, in order to retrieve the necessary X.509 proxy credentials from the MyProxy CA server. Alternatively, a user (or script) can proactively push X.509 proxy credentials to GO for use with specific endpoints.

Once GO has needed credentials it can proceed with the transfer. GO will then cache those credentials until they expire or are explicitly deleted by the user. If user U subsequently requests another transfer that requires existing credentials (e.g., A→B1), GO will use these cached credentials. GO will also use the same user proxy credential for endpoints that have the same default MyProxy server, so a user does not have to enter the same password multiple times. For example, it will recognize that it already has U-B from the first transfer to B1, and that this same credential can be used with B2. Of course, each user has a distinct credential cache. If a credential expires before the transfer completes, GO will notify the user via email (and SMS, Instant Messenger, and Twitter in the future) that the user need to re-authenticate to get a fresh credential. Until such time as the credential is renewed, the transfer will be suspended.

3.3 A Scalable Cloud-based Implementation

Successful SaaS requires reliability and scalability: the service must continue operating despite the failure of individual system components, and behave appropriately as usage grows. To this end, we apply methods commonly used by SaaS providers. We run GO on Amazon Web Services. User and transfer profile information are maintained in a database that is replicated, for reliability, across multiple geographical regions. Transfers are serviced by nodes in Amazon's Elastic Compute Cloud (EC2), which makes it possible to scale the number of servers used to manage data transfers as service demands increase. User gateways are also run across multiple servers, with load balancing, to achieve reliability and scalability. All of this infrastructure runs within Amazon Security Groups, so that only appropriate HTTP and SSH ports are exposed to the Internet.

GO workers initially used the globus-url-copy client program to manage transfers. However, globus-url-copy is a complex program comprising more than 40,000 lines of code with many features not needed by GO, and that furthermore does not include important optimizations and fault handling that are possible in the hosted GO context. Thus, we developed fxp, a simple GridFTP client that uses the same Globus control channel library that underlies globus-url-copy, but with GO-specific optimizations. For example, when fxp gets a transfer request consisting of multiple source-destination URL pairs, it knows that all source URLs correspond to a single logical source endpoint and all destination URLs correspond to a single logical destination endpoint. Thus, it can do better load balancing. Fxp also provides good error attributions, and improves on globus-url-copy's pipelining performance for many small files.

3.4 Globus Connect

The GO user need not install software to request transfers between remote GridFTP servers. However, software installation is required if a source or destination computer does not have GridFTP installed: for example, when transferring data to/from a user's computer.

To address this need, we introduce Globus Connect, a one-click download-and-install application for MacOS, Linux, and Windows. In brief, Globus Connect comprises a GridFTP server that runs as the user (rather than root from inetd like a typical GridFTP server), and a GSI-OpenSSH client configured to establish an authenticated connection to a GO relay server, so as to tunnel GridFTP control channel requests from GO. This Globus Connect GridFTP server only uses outbound data channel connections. GO can direct transfer requests to/from a Globus Connect instance via the control channel tunnel. Thus, to request a data transfer to/from the computer on which they have installed Globus Connect, the user interacts with GO just as they would to request any other transfer (Figure 2). GO relays the request via the tunnel to the Globus Connect server, which then engages in the transfer. This approach has several advantages. First, the Globus Connect server only establishes outbound connections and thus can work behind a firewall or other network interface device that does not allow for inbound connections. Second, the Globus Connect server is stateless and thus can be started and stopped at will: all state associated with transfers is maintained by GO.

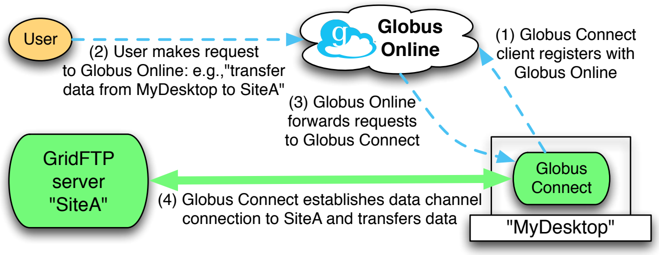
User
(2) User makes request to Globus Online: e.g.,"transfer data from MyDesktop to SiteA"
Globus Online
(1) Globus Connect client registers with Globus Online
(3) Globus Online forwards requests to Globus Connect
GridFTP server "SiteA"
(4) Globus Connect establishes data channel connection to SiteA and transfers data
Globus Connect
"MyDesktop"

Figure 2: A schematic of the Globus Connect architecture

Note that Globus Connect does not currently allow for transfers between two Globus Connect endpoints, as both would attempt to establish an outgoing connection to the other. We expect to introduce support for Globus Connect-to-Globus Connect transfers, for example via the use of intermediate data channel relay servers.

3.5 Globus Connect Security Configuration

Globus Connect also incorporates user-friendly methods for automating security configuration. GridFTP's use of public key authentication protocols means that each GridFTP server must possess an X.509 certificate issued by a certificate authority (CA), which the client (in our case, a GO worker) uses to verify that it is talking to the correct GridFTP server. Thus, to configure a conventional GridFTP server, a user must either obtain an X.509 certificate from a well-known certificate authority (CA)—or, alternatively, set up their own CA and issue a certificate, which they must then ensure is accepted by other sites. Likewise, the client must possess an X.509 certificate issued by a CA that is trusted by the GridFTP server. In addition, in order to access the files on a GridFTP server, the subject of the user's X.509 certificate must be mapped to a local user account. These tasks can be much more tedious than building/installing the GridFTP software.

We overcome these problems in two ways. First, we automate the process of generating, installing, and configuring the certificate required for a Globus Connect installation. We use an online, private CA incorporated in GO to generate a new service certificate when a user adds a Globus Connect endpoint. The user copies a secret "Setup Key" from the GO web site to the Globus Connect setup window in order to securely pair it to their new endpoint definition. Globus Connect uses the setup key as a one-time-use token to download the certificate, private key, and GridFTP gridmap configuration over a secure GSI-OpenSSH connection. GO can then authenticate to the Globus Connect and be sure it is talking to the correct one.

Further problems relating to data channel authentication (DCAU) can arise when a user requests a transfer involving the Globus Connect endpoint. The GridFTP protocol [2] defines DCAU for a third-party transfer to involve mutual validation of the X.509 credentials that the requesting user provides to the two endpoints involved in the transfer. Thus, if endpoint A requires that the user supply a credential A (e.g., an X.509 proxy certificate issued by CA-A) and endpoint B requires that the user supply a credential B (e.g., an X.509 proxy certificate issued by CA-B), then endpoint A must, as part of data channel setup, receive and validate credential B, and endpoint B must receive and validate credential A. However, this process fails if CA-A is unknown to endpoint B, or vice versa: see the upper part of Figure 3.

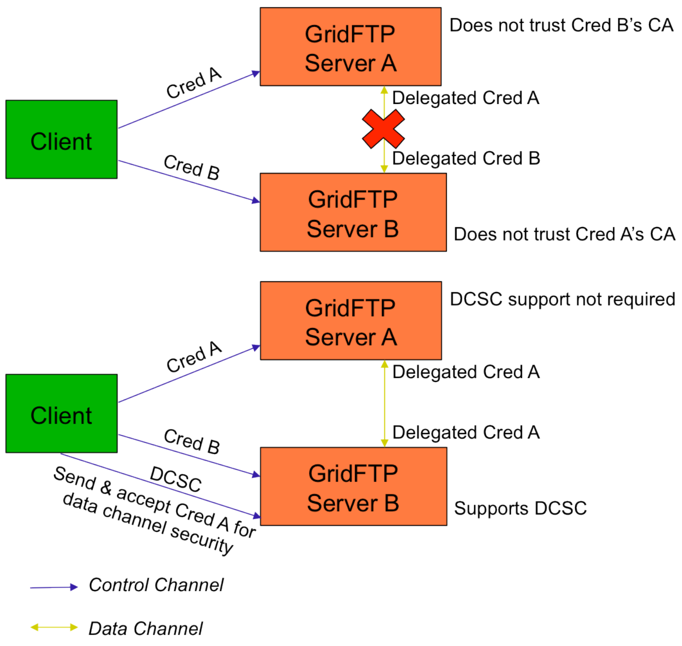

Figure 3: Above: the data channel authentication problem. Below: using DCSC to overcome the problem.

This problem is not unique to Globus Connect, but is particularly likely to arise when using Globus Connect because Globus Connect endpoints use credentials from the GO private CA, which few other sites trust. In order to address this limitation, we introduced a new Data Channel Security Context (DCSC) command in GridFTP. A GridFTP client (e.g., GO) can use this command to tell a DCSC-enabled GridFTP endpoint to both accept and present to the other endpoint a different credential than was used to authenticate the control channel. For example (see Figure 3, below), it can use DCSC to pass credential A to site B, for subsequent presentation to site A. Note that this command enables DCAU even if one endpoint is a legacy GridFTP server that knows nothing about DCSC.

4. EXPERIMENTAL EVALUATION

We summarize studies of GO performance and scalability.

4.1 Request dispatch overhead

We first evaluate the costs associated with dispatching requests to a hosted service rather than executing them directly on the user's computer. We may expect that this dispatch will introduce some overhead, due to need to communicate the request to the GO user gateway that is operating on a remote computer. The GO implementation has been designed and implemented to minimize these costs, but we are still concerned to evaluate them.

We issued 100 consecutive requests to transfer a one-byte file between two locations, using first scp and then GO scp dispatched to GO via SSH. We measured total times of 93 and 273 seconds, respectively—an average per-request cost of 0.93 seconds for SCP and 2.73 seconds for GO scp. Thus, we conclude that the request set up cost associated with the use of GO is ~1.8 seconds. We view this overhead as acceptable. Note that if a user wants to request many transfers, they will normally do this with a single request. If a user wants to perform many consecutive small requests, then they may choose to log into the GO CLI gateway and issue the commands directly, and thus avoid the per-request SSH cost.

4.2 Concurrent request overheads

We may also be concerned that if GO is overloaded, the time required to respond to a request may increase. Thus, we measure transfer time as a function of the number of requests that a single EC2 instance is processing. For these experiments, all transfers are performed entirely between GridFTP servers hosted in the Amazon cloud, so that we can easily scale up and down the number of "users" that generate requests concurrently to a user gateway server.

Figure 4 shows the mean transfer rate (in bytes/sec) achieved when GO is processing between 1 and 63 requests at the same time, where each request involves many transfers of the same size, ranging from 0.001 MByte to 1000 MB. In this experiment, all transfer management operations occur within a single Amazon EC2 c1.xlarge instance, which has eight virtual cores and 7 GB of memory; this demonstrates the scalability of a single instance, though GO can scale larger by using multiple instances to handle the requests. Transfers are between GridFTP endpoints instantiated on different EC2 servers within different Availability Zones in the same Region. (From aws.amazon.com/ec2: "Availability Zones are distinct locations that are engineered to be insulated from failures in other Availability Zones and provide inexpensive, low latency network connectivity to other Availability Zones in the same Region." We map transfer endpoints to Availability Zones within the same region to approximate data transfer within a single campus.)

We see that, as expected, average transfer rate is lower for smaller files. Furthermore, there is a decline in transfer rate as the number of concurrent users increases. However, this decline is modest except for small files. This result makes sense, as transferring smaller files put a larger load on GO than larger files, due to the increased GridFTP control channel traffic, and the resultant increased load on the GO database that tracks the transfer progress.

4.3 Data transfer performance

We next evaluate performance when transferring large quantities of data between pairs of endpoints. To evaluate GO's performance optimization logic in practical situations, we compare GO performance with that achieved when using scp and the globus-url-copy (GUC) client. As scp is known to perform badly, particularly over wide area networks, we include it primarily as a sanity check: if GO is not better than scp, then something is wrong. GUC, on the other hand, drives GridFTP transfers, and so is a fairer comparison. However, in its default configuration (which many people use unchanged) it does not employ optimizations used by GO. For example, it does not enable concurrency, parallelism, pipelining, or data channel caching. Thus this comparison allows us to evaluate the performance gains that many users can expect to gain from GO. We also compare GO against GUC with parameters hand tuned by an expert to maximize performance: "tuned-guc" in the results.

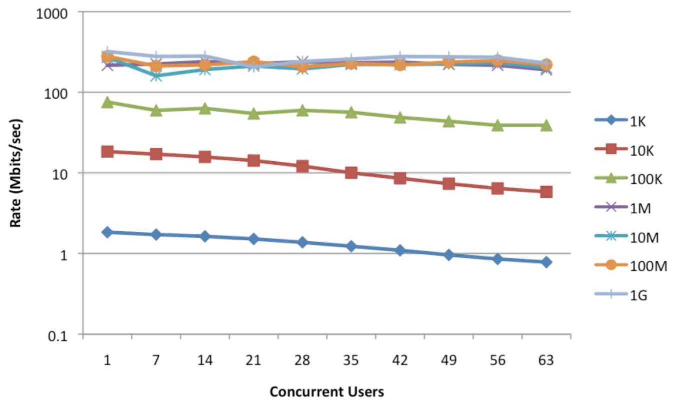

Figure 4: Average transfer performance as a function of number of concurrent requests, for different files sizes

We present results in Figure 5 over a high-speed wide area network (ESNet) between two high-performance parallel storage systems, and Figure 6 between local instance storage of two EC2 instances within different Amazon Availability Zones within a single Region, to approximate a transfer over a campus network. In Figure 5, we give results both between a single data transfer node (DTN) at ALCF and NERSC ("go-single-ep"), and (the default configuration) using the two DTNs that are supported by ALCF and NERSC ("go"). Each DTN is a fast server with a 10 Gb/s network to ESnet, and a fast local connect to each site's GPFS parallel file system.

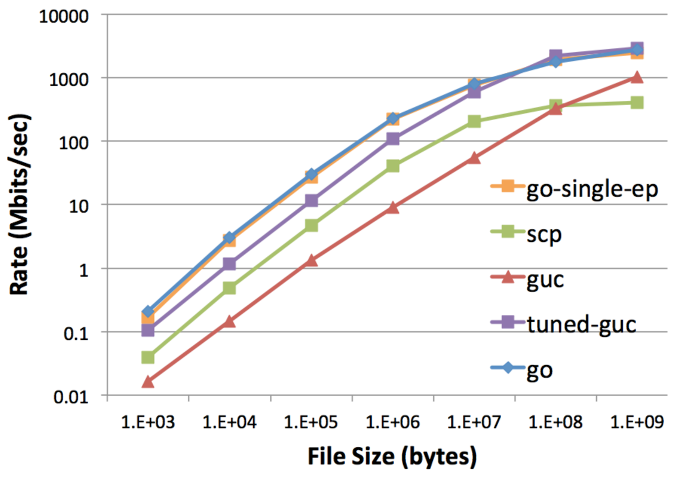

Figure 5: Data transfer performance between ALCF and NERSC

Scp performs badly in all cases. GUC with its default configuration performs badly for all sizes over the wide area, and for small files in the local area. (Clearly, the default configuration requires improvements.) Tuned-guc performs better than GUC in almost all cases. In the wide area case, it does less well than GO for smaller files—probably because GO drives GridFTP pipelining more aggressively, due to the improved pipelining support in GO's fxp GridFTP client. Tuned-guc does better than GO for large files; there remain opportunities to tune GO performance further. Note that the GO transfers to a two DTNs vs. a single DTN are not substantially different except for the largest transfer. We conclude the bottleneck is not the DTNs, but rather either the network or local storage.

5. EXPERIENCES

We assess early user, operator, and site experiences with GO.

5.1 User Experiences

Account creation and new user training requests initially made up the bulk of GO support mail. As account management functions were automated and documentation matured, such requests declined. Current emails are mostly feature requests, problem reports for new code (e.g., Globus Connect,) and endpoint-related problems (e.g., firewall troubles, GridFTP configuration issues). GO staff work with network engineers and system administrators around the world to help fix their problems.

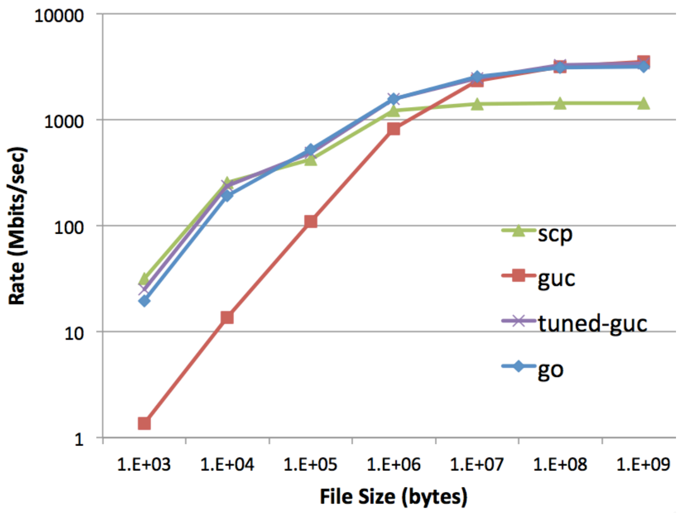

Figure 6: Data transfer performance between two EC2 instances

Though inevitably anecdotal, initial feedback from the GO user community has been positive. We briefly describe one ad hoc interaction and two domain-specific projects. The ad hoc example involved a 300,000 file, 586-terabyte transfer from Argonne, half to the Oak Ridge Leadership Computing Facility (OLCF) and half to NERSC. In Figure 7, the X-axis gives the file numbers, ordered by completion time, and the Y-axis the completion time. During a first phase, extending until the dashed vertical line, transfers proceeded to both remote sites, with a total average transfer speed of 5.7 gigabit/sec. At the dashed line, all transfers to NERSC had completed, and all subsequent transfers are to OLCF.

The extended pause was due to an error relating to Amazon not correctly handling reverse DNS for OLCF hosts. (Sites other than OLCF resolved correctly inside Amazon; non-Amazon sites found OLCF hosts.) GO operators could not pinpoint the source of the problem, and thus installed a workaround that enabled the GO server to resolve the OLCF hosts. Eventually, reverse DNS started working properly for OLCF hosts within Amazon, allowing the OLCF-specific workaround to be removed by the GO administrators. OLCF administrators speculate that these errors occurred because DNS servers used by Amazon had bad data cached after ORNL made some DNS changes, and that it took time for those caches to update.

GO administrators took several days to correct the DNS lookup problem because it took the various system administrators time to understand the details. GO operators provided OLCF system administrators with additional data from internal tests, but in the end the group could not pinpoint the problem. Once the workaround was deployed the transfers resumed immediately.

The Earth System Grid (ESG) [6] community has integrated GO into its gateway nodes. GO has also been integrated, in less than half a day, into a testbed version of the LIGO [3] ARchival Service (LARS). A LIGO user with a LARS identifier and some metadata about analysis files can click a button that submits a transfer job to cli.globusonline.org via GSISSH to have the files replicated to LIGO's archival space. In both cases, use of GO provides users with higher performance and eliminates the need for the application system to monitor data movement and manage fault recovery.

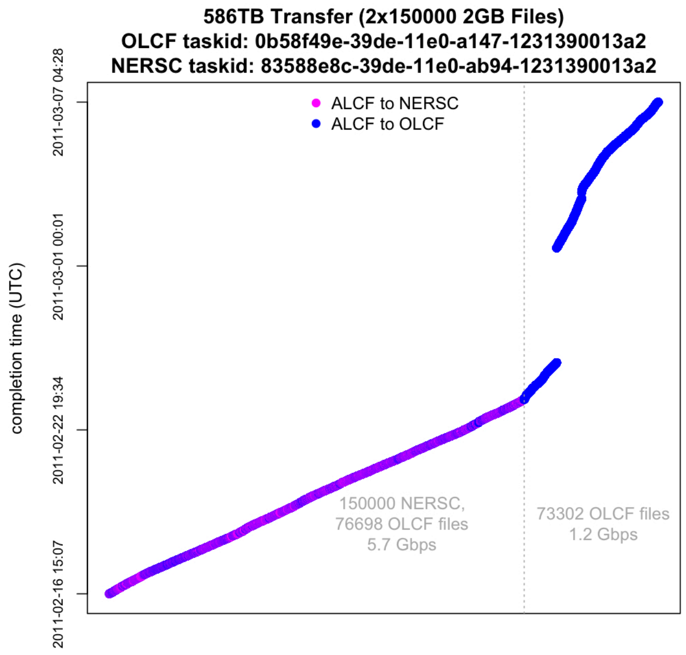

Figure 7: Log for 586 TB transfer (see text for details)

5.2 Globus Online Operator Experiences

The GO team provides professional systems administration, operation, and help desk. All GO servers are monitored using Nagios, which allows us to respond quickly to failures. Our most commonly observed failure (several times in the past three months) is loss of an Amazon EC2 instance running one or more GO components. This failure has occurred far more often than we expected, indicating that we must make improvements to GO's automatic fail-over capabilities to ensure that the service continues to operate through these node crashes without human intervention or user impact.

The SaaS approach also facilitates troubleshooting. For example, one user employed GO to move data to 11 sites. All completed successfully, but monitoring showed that three sites had high transfer retry rates—more than 50 retries per file in one case. Investigation revealed misconfigured systems that had gone unnoticed for months. We worked with the site admins to get them fixed within days.

5.3 Site Operator Experiences

A third important class of GO participants are the sites that host endpoints involved in data transfers. We have seen positive responses from both high performance computing centers (e.g., DOE supercomputer centers, TeraGrid [7] sites) and operators of other major facilities such as DOE light sources—two groups that have a strong interest in providing their users with reliable, secure, and performance data movement services. NERSC (the National Energy Research Scientific Computing Center) at Lawrence Berkeley National Laboratory has adopted GO as a recommended method for its 4000+ users to transfer files to and from NERSC.

6. CONCLUSIONS AND NEXT STEPS

The Globus Online (GO) hosted data movement service leverages software-as-a-service (SaaS) methods to provide fire-and-forget file transfer, automatic fault recovery, and high performance, and to simplify the use of multiple security domains while requiring no client software installation to submit and monitor requests. Globus Connect provides for easy installation of a GridFTP server on a user's machine, so that it can participate in GO transfers. The result is a system that provides automatic fault recovery, high performance, and easy-to-use security, to virtually all researchers and facilities.

Experimental studies show that GO can achieve excellent performance in a wide variety of settings, with low per-transfer overhead and overall bandwidth better than the popular globus-url-copy tool in its default configuration, and only exceeded by a human-tuned globus-url-copy in a few large-file settings. Feedback from users and sites is also positive. For example, the NERSC supercomputer center recommends GO to its users for moving data in and out of NERSC storage systems.

We continue to refine GO to further improve ease of use and to support other protocols and other data-related functions. Recognizing that a frequent reason for moving data is to share it with others, we are adding data sharing. To simplify the specification of sharing policies, we are integrating group management. These mechanisms provide in turn a foundation on which can be built a range of other capabilities such as support for collaborative tools.

7. ACKNOWLEDGMENTS

We thank Vijay Anand, Rachana Ananthakrishnan, Joshua Boverhof, Kyle Chard, Ann Chervenak, Paul Dave', Martin Feller, Daniel Gunter, Thomas Howe, Lukasz Lacinski, Steven Link, Ravi Madduri, Daniel Morgan, Michael Russell, Eugene Sadhu, Mei-Hui Su, Vas Vasiliadis, Vanamala Venkataswamy, and Andrew Zich for work on Globus Online design, implementation and testing, and to many users for helpful suggestions (and encouragement). This work is supported by U.S. Department of Energy, Contract No. DE-AC02-06CH11357; National Science Foundation, contract OCI-534113; and National Institutes of Health, NCRR grant 1 U24 RR025736-01.

8. REFERENCES

1. Allcock, B., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I. and Foster, I., The Globus Striped GridFTP Framework and Server. SC'2005, 2005.
2. Allcock, W. GridFTP: Protocol Extensions to FTP for the Grid. GFD-R-P.020, Global Grid Forum, 2003.
3. Barish, B.C. and Weiss, R. LIGO and the Detection of Gravitational Waves. *Physics Today*, 52(10):44, 1999.
4. Baru, C., Moore, R., Rajasekar, A. and Wan, M., The SDSC Storage Resource Broker. 8th Annual IBM Centers for Advanced Studies Conference, Toronto, Canada, 1998.
5. Bell, G., Hey, T. and Szalay, A. Beyond the Data Deluge. *Science*, 323(5919):1297-1298, 2009.
6. Bernholdt, D., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A., Cinquini, L., Drach, B., Foster, I., Fox, P., Garcia, J., Kesselman, C., Markel, R., Middleton, D., Nefedova, V., Pouchard, L., Shoshani, A., Sim, A., Strand, G. and Williams, D. The Earth System Grid: Supporting the Next Generation of Climate Modeling Research. *Proceedings of the IEEE*, 93(3):485-495, 2005.
7. Catlett, C. and others. TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications. High Performance Computing and Grids in Action, 2007.
8. Chervenak, A., Schuler, R., Kesselman, C., Koranda, S. and Moe, B., Wide Area Data Replication for Scientific Collaborations. 6th IEEE/ACM Int'l Workshop on Grid Computing 2005.
9. Childers, L., Liming, L. and Foster, I. Perspectives on Distributed Computing: Thirty People, Four User Types, and the Distributed Computing User Experience. Argonne National Laboratory Technical Report ANL/MCS/CI-31, 2008.
10. Cholia, S., Skinner, D. and Boverhof, J., NEWT: A RESTful service for building High Performance Computing web applications. Gateway Computing Environments Workshop, 2010, 1-11.
11. Cohen, B. Incentives Build Robustness in BitTorrent, http://bittorrent.com/bittorrentecon.pdf, 2003.
12. Egeland, R., Wildishb, T. and Huang, C.-H. PhEDEx Data Service. *Journal of Physics: Conference Series*, 219(062010), 2010.
13. Erdos, M. and Cantor, S. Shibboleth Architecture. Internet 2, http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v05.pdf, 2002.
14. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3):200-222, 2001.
15. Gray, J., Chong, W., Barclay, T., Szalay, A. and Vandenberg, J. TeraScale SneakerNet: Using Inexpensive Disks for Backup, Archiving, and Data Exchange. Technical report MSR-TR-2002-54, 2002.
16. Hammer-Lahav, E. The OAuth 1.0 Protocol. Internet Engineering Task Force (IETF) RFC 5849, 2010.
17. Hanushevsky, A., Trunov, A. and Cottrell, L., Peer-to-Peer Computing for Secure High Performance Data Copying. 2001 International Conference on Computing in High Energy and Nuclear Physics, Beijing, China, 2001.
18. Kosar, T. and Livny, M. A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 65(10):1146-1157 2005.
19. Madduri, R., Hood, C. and Allcock, W. Reliable File Transfer in Grid Environments. *LCN*:737-738, 2002.
20. Monti, H., Butt, A.R. and Vazhkudai, S.S., CATCH: A Cloud-based Adaptive Data Transfer Service for HPC. 25th IEEE International Parallel & Distributed Processing Symposium, Anchorage, Alaska, 2011.
21. Novotny, J., Tuecke, S. and Welch, V., An Online Credential Repository for the Grid: MyProxy. 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, 2001, IEEE Computer Society Press.
22. Rajasekar, A., Moore, R., Hou, C.-Y., Lee, C.A., Marciano, R., de Torcy, A., Wan, M., Schroeder, W., Chen, S.-Y., Gilbert, L., Tooby, P. and Zhu, B. *iRODS Primer: Integrated Rule-Oriented Data System*. Morgan and Claypool Publishers, 2010.
23. Rehn, J., Barrass, T., Bonacorsi, D., Hernandez, J., Semeniouk, I., Tuura, L. and Wu, Y., PhEDEx high-throughput data transfer management system. Computing in High Energy Physics (CHEP), Mumbai, India, 2006.
24. Shoshani, A., Sim, A. and Junmin Gu, Storage Resource Managers: Middleware Components for Grid Storage. Nineteenth IEEE Symposium on Mass Storage Systems, 2002.
25. Thain, D., Basney, J., Son, S.-C. and Livny, M., The Kangaroo Approach to Data Movement on the Grid. 10th IEEE International Symposium on High Performance Distributed Computing, 2001, IEEE Computer Society Press, 7-9.
26. Tridgell, A. and Mackerras, P. The rsync algorithm. TR-CS-96-05, Department of Computer Science. Australian National University. Canberra, ACT 0200, Australia, 1994.
27. Tuecke, S., Welch, V., Engert, D., Pearlman, L. and Thompson, M. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. Internet Engineering Task Force, 2004.
28. Wang, L., Park, K.S., Pang, R., Pai, V. and Peterson, L., Reliability and security in the CoDeeN content distribution network. USENIX Annual Technical Conference, Boston, MA, 2004, USENIX Association, 14-14.