DOI:10.1145/2076450.2076468

Globus Online manages fire-and-forget file transfers for big-data, high-performance scientific collaborations.

BY BRYCE ALLEN, JOHN BRESNAHAN, LISA CHILDERS, IAN FOSTER, GOPI KANDASWAMY, RAJ KETTIMUTHU, JACK KORDAS, MIKE LINK, STUART MARTIN, KARL PICKETT, AND STEVEN TUECKE

Software as a Service for Data Scientists

AS BIG DATA emerges as a force in science,[2,3] so, too, do new, onerous tasks for researchers. Data from specialized instrumentation, numerical simulations, and downstream manipulations must be collected, indexed, archived, shared, replicated, and analyzed. These tasks are not new, but the complexities involved in performing them for terabyte or larger datasets (increasingly common across scientific disciplines) are quite different from those that applied when data volumes were measured in kilobytes. The result is a computational crisis in many laboratories and a growing need for far more powerful data-management tools, yet the typical researcher lacks the resources and expertise to operate these tools.

The answer may be to deliver research data-management capabilities to users as hosted "software as a service," or SaaS,[18] a software-delivery model in which software is hosted centrally and accessed by users using a thin client (such as a Web browser) over the Internet. As demonstrated in many business and consumer tools, SaaS leverages intuitive Web 2.0 in-

» key insights

- **The costs of research data life-cycle management are growing dramatically as data becomes larger and more complex.**
- **SaaS approaches are a promising solution, outsourcing time-consuming research data management tasks to third-party services.**
- **Globus Online demonstrates the potential of SaaS for research data management, simplifying data movement for researchers and research facilities alike.**

terfaces, deep domain knowledge, and economies of scale to deliver capabilities that are easier to use, more capable, and/or more cost-effective than software accessed through other means. The opportunity for continuous improvement via dynamic deployment of new features and bug fixes is also significant, as is the potential for expert operators to intervene and troubleshoot on the user's behalf.

We report here on a research data-management system called Globus Online, or GO, that adopts this approach, focusing on GO's data-movement functions ("GO-Transfer"). We describe how GO leverages modern Web 2.0 technologies to provide intuitive interfaces for fire-and-forget file transfers between GridFTP endpoints[1] while leveraging hosting for automatic fault recovery, high performance, simplified security configuration, and no client software installation. We also describe a novel approach for providing a command-line interface (CLI) to SaaS without distributing client software and Globus Connect to simplify installation of a personal GridFTP server for use with GO. Our experiments show low overhead for small transfers and high performance for large transfers, relative to conventional tools.

Adoption of this new service has been notable. One year after product launch, November 2010, more than 3,000 registered users had in aggregate moved more than two petabytes (2×10^{15}B) and 150 million files; numerous high-performance computing (HPC) facilities and experimental facilities recommend GO to their users; and several "science gateways" are integrating GO as a data upload/download solution. GO has also been adopted as a foundational element of the National Science Foundation's new (as of 2011) Extreme Science and Engineering Discovery Environment (XSEDE) supercomputer network (http://www.xsede.org/).

Data Movement

Researchers often must copy many files with potentially large aggregate size among two or more network-connected locations, or "endpoints," that may or may not include the computer from which the transfer command is issued; that is, third-party transfers may be (indeed, frequently are) involved. Our goal in designing GO is a solution that provides extreme ease-of-use without compromising reliability, speed, or security. A 2008 report by Childers et al.[5] of Argonne National Laboratory makes clear the importance of usability. In particular, failure recovery is often a human-intensive process, as reported by Childers et al.[5]: "The tools that we use to move files typically are the standard Unix tools included with ssh... 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."

In addition, datasets may have nested structures and contain many files of different size (see the sidebar "Other Approaches"). Source and destination may have different security requirements and authentication interfaces. Networks and storage servers may suffer transient failures. Transfers must be tuned to exploit high-speed research networks. Directories may have to be mirrored across multiple sites, but only some files differ between source and destination. Firewalls, Network Address Translation, and other network complexities may have to be addressed. For these and other reasons, it is not unusual to hear of even modest-scale wide-area data transfers requiring days of careful "babysitting" or of being abandoned for high-bandwidth but high-latency (frequently labor-intensive and error-prone) "sneakernet."[11]

A common question about GO is whether data can be moved more effectively through the physical movement of media rather than through communication over networks.

Why Move Data at All?

Why not just leave data where it is created? Such an option is certainly preferred when possible, and we may hope that over time moving computation to data rather than the other way round will be more common. However, in practice, data scientists often find data is "in the wrong place" and thus must be moved for a variety of reasons. Data may be produced at a location (such as a telescope or sensor array) where large-scale storage cannot be located easily. It may be desirable to collocate data from many sources to facilitate analysis—a common requirement in, say, genomics. Remote copies may be required for disaster recovery. Data analysis may require computing power or specialized computing systems not available locally. Policy or sociology may require replication of data sets in distinct geographical regions; for example, in high-energy physics, all data produced at the Large Hadron Collider, Geneva, Switzerland, must essentially be replicated in the U.S. and elsewhere for independent analysis. It is also frequently the case that the aggregate data-analysis requirements of a community exceed the analysis capacity of a data provider, in which case data must be downloaded for local analysis. This is the case in, for example, the Earth System Grid, which delivers climate simulation output to its 25,000 users worldwide.

Another common question about GO is whether data can be moved more effectively through the physical movement of media rather than through communication over networks. After all, no network can exceed the bandwidth of a FedEx truck. The answer is, again, that while physical shipment has its place (and may be much cheaper if the alternative is to pay for a high-speed network connection), it is not suitable in all situations. Latency is high, and so is the human overhead associated with loading and unloading media, as well as with keeping track of what has been shipped. Nevertheless, it could be feasible to integrate into GO methods for orchestrating physical shipment when it is determined to be faster and/or more cost-effective—as with Cho's and Gupta's Pandora ("People and networks moving data around") system.[6]

Different Interfaces for Different Users

The Computation Institute at the University of Chicago and Argonne National Laboratory operates GO as a highly available service (http://www.globusonline.org/) to which users submit data-movement and synchronization requests. A typical transfer request proceeds as follows: A user authenticates with GO and submits a request. GO records the request into its state database, inspects the request to determine what endpoints are involved, and if necessary prompts the user to provide credentials GO can use to interact with those endpoints on the user's behalf. GO then establishes authenticated GridFTP control channels with each endpoint and issues the appropriate GridFTP commands to transfer the requested files directly between the endpoints. GO monitors the transfer progress and updates transfer state in the state database. This information can be used to restart transfers after faults and re-

Other Approaches

One alternative for data movement involves running tools on the user's computer; for example, Rsync,[20] scp, file transfer program (FTP), secure FTP, and bbftp[13] are all used to move data between a client computer and a remote location. Other software (such as globus-url-copy, Reliable File Transfer, File Transfer Service, and Lightweight Data Replicator) can each manage large numbers of transfers. However, the need to download, install, and run software is a significant barrier to use. Users spend much time configuring, operating, and updating such tools though rarely have the IT and networking knowledge necessary to fix things when they do not "just work," which is all too often.

Some big-science projects have developed specialized solutions to the problem; for example, the PhEDEx high-throughput data-transfer-management system[9] manages data movement among sites participating in the Compact Muon Solenoid experiment at CERN, and the Laser Interferometer Gravitational Wave Observatory (LIGO) project developed the LIGO Data Replicator.[4] These centrally managed systems allow users to hand off data-movement tasks to a third-party service that performs them on their behalf. However, these services require professional operators functioning only among carefully controlled endpoints within these communities.

Managed services (such as YouSendIt and DropBox) also provide data-management solutions but do not address researchers' need for high-performance movement of large quantities of data. BitTorrent[8] and Content Distribution Networks[21] are good at distributing a relatively stable set of large files (such as movies) but do not address data scientists' need for many frequently updated files managed in directory hierarchies. The Integrated Rule-Oriented Data System[17] is often run in hosted configurations, but, though it performs some data-transfer operations (such as for data import), data transfer is not its primary function or focus.

The Kangaroo,[19] Stork,[14] and CATCH[15] systems all manage data movement over wide-area networks using intermediate storage systems where appropriate to optimize end-to-end reliability and/or performance. They are not designed as SaaS data-movement solutions, but their methods could be incorporated into GO.

Web and REST interfaces to centrally operated services are conventional in business, underpinning such services as Salesforce.com (customer relationship management), Google Docs, Facebook, and Twitter—an approach not yet common in science. Two exceptions are the PhEDEx Data Service,[9] with both REST and CLIs, and the National Energy Research Supercomputing Center, or NERSC, Web toolkit called NEWT[7] that enables RESTful operations against HPC center resources.

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 end-points; support 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 retargeting to GO of scripts using scp for data movement.
Monitor Transfers	status	List transfers initiated by requesting user, along with summary information (such as status, start time, and completion time).
	details	Provide details on a transfer (such as number of files transferred and number of faults).
	events	List events associated with a specified transfer: start, stop, performance, faults.
Control Transfers	cancel	Terminate specified transfer or individual file in a transfer.
	wait	Wait for specified transfer to complete; show progress bar.
		Alter deadline for a transfer.

port progress to the user. GO keeps attempting a failed request periodically until the task deadline is reached or the user cancels the request. When the transfer completes or an unrecoverable fault is encountered the user is notified via email.

GO supports a friendly, intuitive Web GUI for ad hoc and less-technical users; a CLI for use by more advanced users and for scripting; and a Representational State Transfer (REST) application programming interface (API) facilitating integration for system builders that also supports the GO Web interface. The table here lists GO's primary transfer-management functions. Additional endpoint-management functions provide 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).

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, a user issues a POST to https://transfer.api.globusonline.org/v0.10/transfer with a document describing the transfer request, including, for example, source and destination endpoints and file paths and options; to access the status of a task, the user issues a GET request to https://transfer.api.globusonline.org/v0.10/task/<task_id>; the system then returns a document 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 JavaScript Object Notation (JSON) and Extensible Markup Language (XML). Supported security mechanisms include HTTPS mutual authentication with an X.509 client certificate and (for Web browsers) HTTPS server authentication with cookie-based client authentication.

The Web interface builds on the REST interface using standard Asynchronous JavaScript (AJAX) and XML techniques. A GO Web page contains standard HTML, CSS, and JavaScript, interacting with the REST interface through standard-session cookie-based client authentication. The Web GUI supports browsing remote file systems, as well as submitting, monitoring, and cancelling transfer requests.

A CLI supports client-side scripting; for example, a script that, each evening, transfers new files created during the day to a remote repository or that automatically moves output from an analysis job back to a local machine. A CLI typically requires installation of client-side libraries, though it is counter to the key SaaS tenet of not requiring client software installation to use the service. To obviate having to install software, the GO system provides all GO users with a restricted shell, to which they can ssh to execute commands. Thus, a user, Joe, can write

```
ssh joe@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, or secure copy, command, mirroring 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, Joe can first ssh to http://cli.globusonline.org/, then issue a series of commands directly:

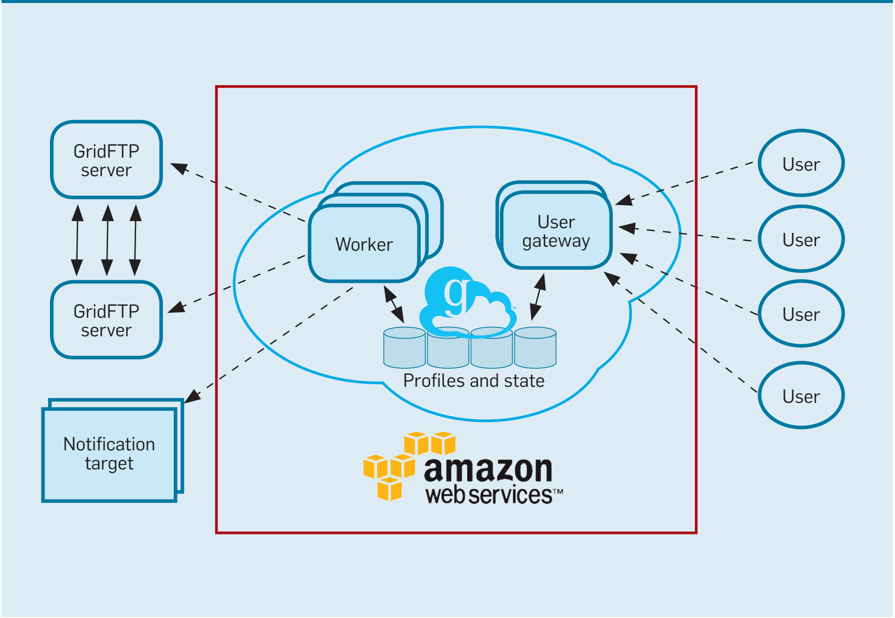
```
joe$ ssh cli.globusonline.org
Welcome to globusonline.org, ian.
$ scp alcf#dtn:~/myfile nersc#dtn:~/
myfile
Contacting 'gs1.intrepid.alcf.anl.gov'...
Enter MyProxy pass phrase: ********
```

This example command 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 (ALCF: http://www.alcf.anl.gov/). Sites can define and publish their own endpoint definitions (such as alcf#dtn, nersc#dtn); users are able to define custom endpoint definitions as well (such as mylaptop, myserver). More than 300 such endpoint definitions have been defined, incorporating many major research computing systems in the U.S. and elsewhere worldwide.

User Profile and Identity Management

An important GO feature is the ability to handle transfers across multiple security domains with multiple user identities. Unlike many systems, including most previous Grid file-transfer services, GO does not require a single, common security credential across all transfer endpoints. Rather, it assumes users have many identities for use with different service providers and that GO's job is to ensure the

Figure 1. Globus Online architecture.



right identities are brought to bear at the right time for any transfer and do so in a way that is easy for users to understand.

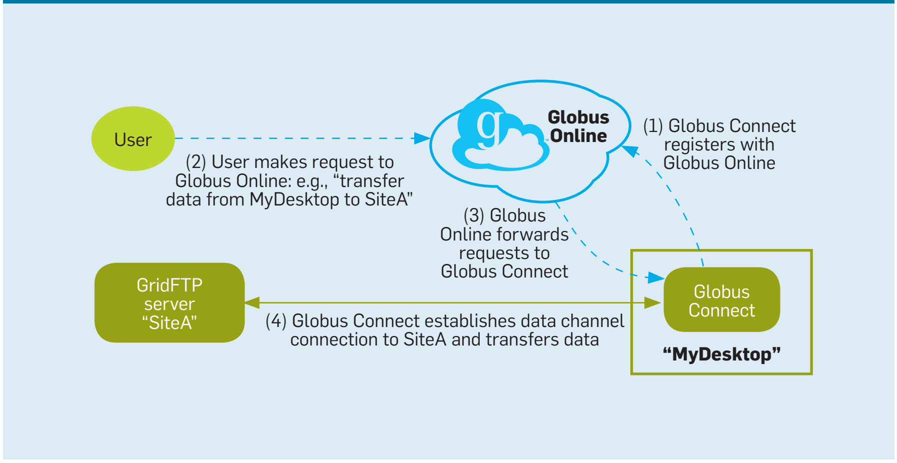
To this end, all users have GO accounts where they easily configure their profile with various identities; for example, they can register their MyProxy Certification Authority (CA)[16] identities (such as for NERSC and other computing centers using the approach), OAuth protocol[12] identities (such as for ALCF, the NSF XSEDE network of supercomputer centers, and Facebook), OpenID identities (such as for Google), Shibboleth[10] identities (such as for campus credentials), and X.509 identities (such as from the U.S. Department of Energy Grids CA and International Grid Trust Federation-certified CA).

Though GO stores identities, it does not store passwords; rather, it knows only the user name so it can prompt for the appropriate information when that identity is needed. In addition, identities can be configured as "federated identities" the user uses to authenticate with GO; for example, users who have already authenticated their browser session with an OpenID, OAuth, or Shibboleth identity can use GO without having to authenticate further, and 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 users may wish to communicate. Then, where possible, it caches information it can use to facilitate access. For example, assume 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. 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[22] 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 prompts the user for them when the user requests the transfer. Alternatively, a user (or script) can proactively push X.509 proxy credentials to GO for use with specific endpoints.

When GO has the needed credentials it proceeds with the transfer,

Figure 2. Globus Connect architecture.


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

caching them until they expire or are explicitly deleted by the user. GO also uses the same user proxy credential for endpoints that have the same default MyProxy server, so users need not enter the same password multiple times. If a credential expires before the transfer completes, GO notifies the user via email that the user must re-authenticate. Until such time as the credential is renewed, the transfer is suspended.

Scalable Cloud-based Implementation

SaaS requires reliability and scalability, continuing to operate despite the failure of individual components and behaves appropriately as usage grows. To this end, the GO team applies methods commonly used by SaaS providers, running GO on a commercial cloud provider, Amazon Web Services (AWS). The GO implementation uses a combination of Amazon Elastic Compute Cloud (EC2), Amazon Elastic Load Balancing, and Amazon Simple Storage Service (S3).

The GO implementation involves platform services, which provide user, profile, and group-management functions, and the file-transfer service that implements the data-movement functionality that is the focus of this article. The GO team runs the platform services on a collection of EC2 instances across several availability zones in Amazon's U.S. East region (located in Virginia), including Web server, load balancer, database, and backup. The file-transfer service runs on a collection of EC2 instances hosted in the U.S. East region, including of transaction database, transfer agents, history database, transfer REST API server, CLI server, and backup. In addition, the GO team runs two Nagios servers on EC2 instances, one in the U.S. East region to monitor all other instances, the other in the U.S. West region to monitor the health of the primary Nagios server. The GO team also uses the Chef configuration-management tool for provisioning all servers. The vast majority of GO is programmed in Python, running on Ubuntu Linux servers, with Cassandra and Postgres databases.

Figure 1 is a somewhat abstracted view of the GO implementation, showing the user gateway servers supporting interaction between users and the system via Web GUI, CLI, and REST interfaces; the worker processes orchestrating data transfers and other tasks (such as notifying users of changes in state); and the profiles and state database storing user profiles, request state, and endpoint information.

The authors' current thinking on availability is that that the research community needs between three and four 9s (99.9%–99.99%), corresponding to between one and 10 minutes downtime per week. Longer than 10 minutes lack of availability can be problematic for users employing GO as part of time-critical work processes (such as in astronomy data-processing pipelines). This requirement is a primary reason the GO team hosts GO on AWS rather than on a research computing facility, which, in our experience, provides closer to two-9s availability when occasional maintenance shutdowns are taken into account.

Globus Connect (Multi-User)

GO users 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, as when, for example, transferring data to/from a user's computer.

To address this need, we introduced in early 2011 Globus Connect, a one-click download-and-install application for Linux, MacOS, and Windows. Globus Connect consists of 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 uses only outbound data-channel connections. GO can direct transfer requests to/from a Globus Connect instance via the control-channel tunnel. Thus, to request a transfer to/from the computer on which they have installed Globus Connect, users interact with GO just as they would request any other transfer (see Figure 2). GO relays the request via the tunnel to the Globus Connect server, which then executes the transfer.

Globus Connect establishes only outbound connections and thus can work behind a firewall or other network interface device that does not allow for inbound connections. The Globus Connect server is stateless and thus can be started and stopped at will; all state associated with transfers is maintained by GO. Autoupdate means the user need not maintain the software over time.

The GO team also streamlined the process of standing up a GridFTP server as a shared resource by introducing Globus Connect Multi-User (GCMU), simplifying the process of connecting a shared server or cluster to GO. With GCMU a resource owner can quickly set up a GO endpoint on any server that can then be accessed by multiple users for remote data movement. GCMU packages a GridFTP server, MyProxy server, and MyProxy Online CA pre-configured for GO use, requiring only a few steps to install and use. A growing number of research facilities users (such as the University of Colorado, University of Michigan, Oak Ridge National Laboratory, and Advanced Photon Source) use GCMU to make their resources accessible to remote users.

Globus Connect also incorporates user-friendly methods for automating the process of generating, installing, and configuring the certificate required for a Globus Connect installation. It uses an online private CA incorporated into GO to generate a new service certificate when a user adds a Globus Connect endpoint. Users copy a secret "setup key" from the GO Web site to the Globus Connect setup window to securely pair it with 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 instance and be sure it is talking to the correct one.

Optimized File Transfers for All

GridFTP client interfaces allow users to optimize transfer performance by setting parameters (such as TCP buffer size, number of outstanding requests, or "pipelining," number of concurrent control channel connections, or "concurrency," and number of TCP channels used for data movement, or "parallelism"). However, few users have the experience and time needed to apply these settings effectively.

GO obviates the need for user tuning by applying heuristics to set parameters based on the number and size of files in a transfer. Upon arrival of a recursive transfer request, GO crawls the directory to find the files to transfer, determining file size in the process. It then sorts them by size

Figure 3. Data-transfer performance between ALCF and NERSC.

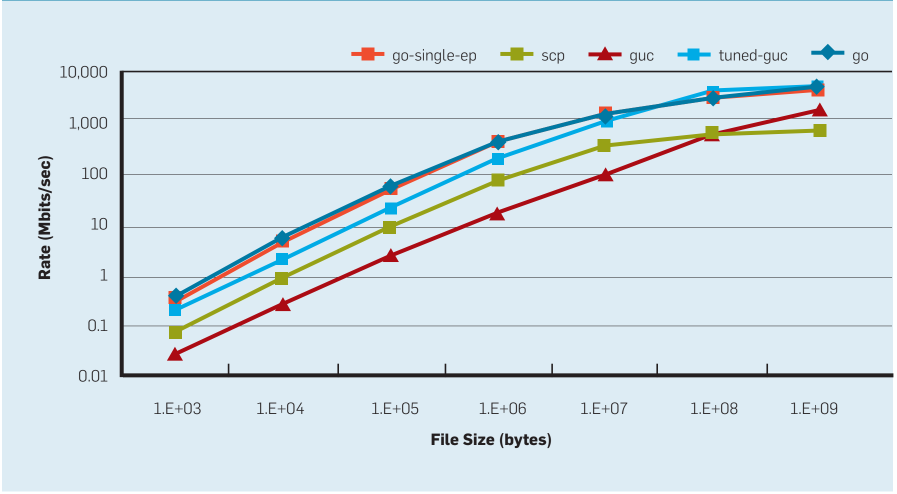
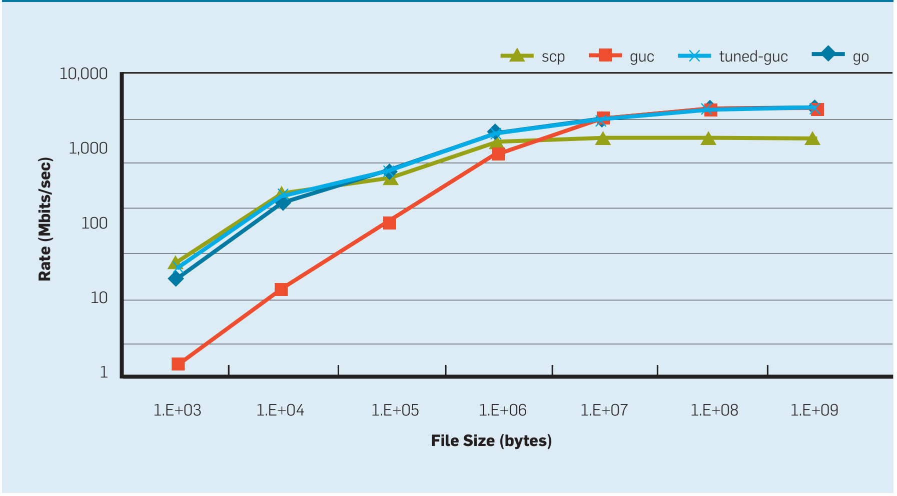


Figure 4. Data-transfer performance between two EC2 instances.



and performs the transfer in chunks, setting parameters for each chunk according to the average size of its files. If a chunk has more than 100 files and an average file size smaller than 50MB, GO applies small settings, making use of pipelining: Specifically, concurrency=2 files in transit at once, parallelism=2 sockets per file, and pipelining=20 requests outstanding at once. If all files in a chunk are larger then 250MB, GO applies large settings that use more parallelism and moderate pipelining: concurrency=2, parallelism=8, and pipelining=5. In all other cases, the default setting is used: concurrency=2, parallelism=4, and pipelining=10. When a Globus Connect or GCMU endpoint is used as the destination in a GO transfer, then stream mode (not mode E, which allows for out-of-order transmission) must be used, and concurrency is the only optimization that can be applied. When using steam mode, then for small file chunks, GO sets concurrency=8.

These simple heuristics have proved effective but can surely be improved; for example, GO could be extended to manipulate the TCP buffer size (such as on the basis of round-trip-time measurements), select alternative transport protocols, or reserve the network.

Though GO stores identities, it does not store passwords; rather, it knows only the user name and how to use it, so it can prompt for the appropriate information when that identity is needed.

Performance and Scalability

Dispatching requests to a hosted service rather than executing them directly on a user's computer introduces temporal overhead due to the need to communicate the request to the GO user gateway operating on a remote computer. To evaluate this overhead, we conducted tests (in 2011), issuing 100 consecutive requests to transfer a 1B file between two locations, using scp first, 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. We concluded that the request-setup cost associated with the use of GO is ~1.8 seconds. This overhead is acceptable for many data-transfer applications, though certainly not for all. Note that users who want to request many transfers will normally do so with a single request. If users want to perform consecutive small requests, they may choose to log into the GO CLI gateway and issue the commands directly, thus avoiding the per-request ssh cost.

To evaluate GO's performance-optimization logic in practical situations, we also conducted tests (in 2011) to compare GO performance when transferring large quantities of data between pairs of endpoints with that achieved using scp and the globus-url-copy (GUC) client. As scp is known to perform poorly, particularly over wide-area networks, we included this option in the test 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 represents a fairer comparison. However, in its default configuration (which, Globus developers tell us, many users use unchanged) GUC does not employ optimizations used by GO; for example, GUC does not enable concurrency, parallelism, pipelining, or data channel caching. This comparison thus permitted evaluation of the performance gains many users expect from GO. We also compared GO against GUC with parameters tuned by an expert to maximize performance—tuned-guc in the results.

Figure 3 charts results of GO-based data transfer in 2011 over a high-speed wide-area network—ESNet, the Energy Sciences Network, http://www.es.net/—between two high-performance parallel storage systems, and Figure 4 between local-instance storage of two EC2 instances within different Amazon Availability Zones in a single geographic region to approximate a transfer over a campus network. Figure 3 gives results both between a single data-transfer node (DTN) at ALCF and NERSC ("go-single-ep") and (the default configuration) using the two DTNs supported by ALCF and NERSC ("go"). Each DTN is a fast server with a 10Gb/s network to ESnet and a fast local connect to each site's GPFS parallel file system. Meanwhile, scp performs poorly for all data transfers, and GUC, with its default configuration, performs poorly for all data transfer sizes over the wide area, as well as for small files in the local area. (The default configuration clearly requires improvement.)

Fortunately, tuned-guc performs better than untuned 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 GridFTP client. However, tuned-guc does better than GO for large files, though GO performance can be tuned further. Note, GO transfers to a two-DTNs vs. a single-DTN are not substantially different, except for the largest transfer. We conclude that the bottleneck is not the DTNs but either the network or local storage.

Conclusion

Exploding data volumes are transforming many researchers into data scientists, with urgent need for more capable, efficient data-management tools. SaaS may represent the means by which such tools are provided cost-effectively, with GO as a first step in that direction. A hosted data-movement service with intuitive interfaces, automatic fault recovery, high performance, and easy-to-use security, it has already (since its introduction, late 2010) won enthusiastic adoption in the world of big-data science applications. Many operators of scientific facilities worldwide recommend GO to their users. Our experiments show GO can achieve high performance and exceptional reliability in a variety of settings, with low per-transfer overhead and bandwidth rarely exceeded in human-tuned transfers.

These positive results have encouraged us to expand GO to address other research data-management problems. Recognizing that a notable reason for moving data is to share it with other scientists, the GO development team is adding data-sharing support like that provided by DropBox. To simplify the specification of sharing policies, GO developers have integrated group management. In turn, these mechanisms provide a foundation on which can be built a range of other capabilities, notably support for collaborative tools.

Acknowledgments

We thank Vijay Anand, Rachana Ananthakrishnan, Joshua Boverhof, Kyle Chard, Ann Chervenak, Paul Davé, 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 their work on GO, as well as many users for helpful suggestions. This work was supported in part by DOE DE-AC02-06CH11357; NSF OCI-534113; and NIH NCRR 1 U24 RR025736-01.

References

1. Allcock, B., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing* (Seattle, Nov. 12–18). ACM Press, New York, 2005.
2. Bell, G., Hey, T., and Szalay, A. Beyond the data deluge. *Science 323*, 5919 (Mar. 2009), 1297–1298.
3. Berriman, G.B. and Groom, S. How will astronomy archives survive the data tsunami? *Commun. ACM 54*, 12 (Dec. 2011), 52–56.
4. Chervenak, A., Schuler, R., Kesselman, C., Koranda, S. and Moe, B. Wide-area data replication for scientific collaborations. In *Proceedings of the Sixth IEEE/ACM International Workshop on Grid Computing* (Seattle, Nov. 13). IEEE Computer Society, Washington, D.C., 2005.
5. Childers, L., Liming, L., and Foster, I. *Perspectives on Distributed Computing: 30 People, Four User Types, and the Distributed Computing User Experience, Technical Report ANL/MCS/CI-31.* Argonne National Laboratory, Argonne, IL, 2008.
6. Cho, B. and Gupta, I., Budget-constrained bulk data transfer via Internet and shipping networks. In *Proceedings of the Eighth ACM international conference on Autonomic Computing* (Karlsruhe, Germany, June 14–16). ACM Press, New York, 2011, 71–80.
7. Cholia, S., Skinner, D., and Boverhof, J. NEWT: A RESTful service for building high-performance computing Web applications. In *Proceedings of the 2010 Gateway Computing Environments Workshop* (New Orleans, Nov. 14). IEEE Computer Society Press, 2010, 1–11.
8. Cohen, B. Incentives build robustness in BitTorrent. In *Proceedings of the First International Workshop on Economics of P2P Systems* (Berkeley, CA, June 5–6, 2003).
9. Egeland, R., Wildishb, T., and Huang, C.-H. PhEDEx data service. *Journal of Physics: Conference Series 219* (2010).
10. Erdos, M. and Cantor, S. *Shibboleth Architecture.* Internet 2, May 2, 2002; http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v05.pdf
11. 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.* Microsoft Research, Redmond, WA, 2002.
12. Hammer-Lahav, E. *The OAuth 1.0 Protocol.* Internet Engineering Task Force RFC 5849, 2010; http://tools.ietf.org/html/rfc5849
13. Hanushevsky, A., Trunov, A., and Cottrell, L. Peer-to-peer computing for secure high-performance data copying. In *Proceedings of the 2001 International Conference on Computing in High Energy and Nuclear Physics* (Beijing, Sept. 3–7, 2001).
14. 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 (Oct. 2005), 1146–1157.
15. Monti, H., Butt, A.R., and Vazhkudai, S.S. CATCH: A cloud-based adaptive data-transfer service for HPC. In *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium* (Anchorage, Alaska, May 16–20). IEEE Computer Society, 2011, 1242–1253.
16. Novotny, J., Tuecke, S., and Welch, V. An online credential repository for the grid: MyProxy. In *Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing* (San Francisco, Aug. 7–9). IEEE Computer Society Press, Washington, D.C., 2001, 104–111.
17. 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.
18. Sun, W., Zhang, K., Chen, S.-K., Zhang, X., and Liang, H. Software as a service: An integration perspective. In *Proceedings of the Fifth International Conference on Service-Oriented Computing*, B. Krämer, K.-J. Lin, and P. Narasimhan, Eds. (Vienna, Austria, Sept. 17–20). Springer, Berlin/Heidelberg, 2007, 558–569.
19. Thain, D., Basney, J., Son, S.-C., and Livny, M. The Kangaroo approach to data movement on the grid. In *Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing* (San Francisco, Aug. 7–9). IEEE Computer Society Press, Washington, D.C., 2001, 325–333.
20. Tridgell, A. and Mackerras, P. *The Rsync Algorithm TR-CS-96-05.* Department of Computer Science, Australian National University, Canberra, 1994.
21. Wang, L., Park, K.S., Pang, R., Pai, V., and Peterson, L. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the USENIX Annual Technical Conference* (Boston, June 27–July 2). USENIX Association, Berkeley, CA, 2004, 171–184.
22. Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Tuecke, S., Gawor, J., Meder, S., and Siebenlist, F. X.509 proxy certificates for dynamic delegation. In *Proceedings of the Third Annual Public Key Infrastructure R&D Workshop* (Gaithersburg, MD, Apr. 12–14), National Institute of Standards and Technology, Gaithersburg, MD, 2004.

Bryce Allen (ballen@ci.uchicago.edu) is a software developer in the Computation Institute of the University of Chicago and Argonne National Laboratory, Argonne, IL.

John Bresnahan (bresnaha@mcs.anl.gov) is a senior software developer in the Mathematics and Computer Science Division of Argonne National Laboratory, Argonne, IL.

Lisa C. Childers (childers@mcs.anl.gov) is a staff member of the Mathematics and Computing Science Division of Argonne National Laboratory, Argonne, IL, and of the Computation Institute of the University of Chicago.

Ian T. Foster (foster@anl.gov) is an Argonne Distinguished Fellow, Director of the Computation Institute, and the Arthur Holly Compton Distinguished Service Professor of Computer Science at Argonne National Laboratory, Argonne, IL, and at the University of Chicago.

Gopi Kandaswamy (gopikandaswamy@gmail.com) is an associate consultant in Tata Consultancy Services and former senior research systems developer in the Information Sciences Institute, Los Angeles.

Rajkumar Kettimuthu (kettimut@mcs.anl.gov) is a fellow in the Computation Institute and principal software development specialist in the Mathematics and Computer Science Division of Argonne National Laboratory, Argonne, IL, and the University of Chicago.

Jack Kordas (kordas@ci.uchicago.edu) is a senior architect and developer in the Computation Institute of the University of Chicago and Argonne National Laboratory, Argonne, IL.

Michael Link (mlink@mcs.anl.gov) is a software developer in the Computation Institute of the University of Chicago and Argonne National Laboratory, Argonne, IL.

Stuart Martin (smartin@mcs.anl.gov) is a manager of software development in the Computation Institute of the University of Chicago and Argonne National Laboratory, Argonne, IL, managing software development for the Globus Toolkit and portions of Globus Online.

Karl Pickett (kjp@ci.uchicago.edu) is a programmer in the Computation Institute of the University of Chicago and Argonne National Laboratory, Argonne, IL.

Steven Tuecke (tuecke@ci.uchicago.edu) is deputy director of the Computation Institute of the University of Chicago and Argonne National Laboratory, Argonne, IL, leading the Globus Online project.