Globus Toolkit Version 4: Software for Service-Oriented Systems

Ian Foster
(for the Globus Team)

Math & Computer Science Division, Argonne National Lab, Argonne, IL 60439, U.S.A.
Department of Computer Science, University of Chicago, Chicago, IL 60637, U.S.A.

Abstract. The Globus Toolkit (GT) has been developed since the late 1990s to support the development of service-oriented distributed computing applications and infrastructures. Core GT components address, within a common framework, basic issues relating to security, resource access, resource management, data movement, resource discovery, and so forth. These components enable a broader "Globus ecosystem" of tools and components that build on, or interoperate with, core GT functionality to provide a wide range of useful application-level functions. These tools have in turn been used to develop a wide range of both "Grid" infrastructures and distributed applications. I summarize here the principal characteristics of the latest release, the Web services-based GT4, which provides significant improvements over previous releases in terms of robustness, performance, usability, documentation, standards compliance, and functionality.

1 Introduction

Globus is:

- A *community* of users and developers who collaborate on the use and development of open source software, and associated documentation, for distributed computing and resource federation.
- The *software* itself—the **Globus Toolkit**: a set of libraries and programs that address common problems that occur when building distributed system services and applications.
- The *infrastructure* that supports this community—code repositories, email lists, problem tracking system, and so forth: all accessible at **globus.org**.

The **software** itself provides a variety of components and capabilities, including the following:

- A set of *service implementations* focused on infrastructure management.
- Tools for building *new Web services*, in Java, C, and Python.
- A powerful standards-based *security infrastructure*.

– Both *client APIs* (in different languages) and *command line programs* for accessing these various services and capabilities.
– Detailed *documentation* on these various components, their interfaces, and how they can be used to build applications.

These components in turn enable a rich **ecosystem** of components and tools that build on, or interoperate with, GT components—and a wide variety of **applications** in many domains. From our experiences and the experiences of others in developing and using these tools and applications, we identify commonly used design patterns or **solutions**, knowledge of which can facilitate the construction of new applications.

In this article, I review briefly the current status of Globus, focusing in particular on those aspects of the GT4 release that should be of interest to those wishing to work with the software. I provide references to research articles for those desiring more details on the underlying concepts and mechanisms.

2 Motivation and Concepts

Globus software is designed to enable applications that federate distributed resources, whether computers, storage, data, services, networks, or sensors. Initially, work on Globus was motivated by the demands of "virtual organizations" in science. More recently, commercial applications have become increasingly important. Commerce and science often, but not always, have similar concerns.

Federation is typically motivated by a need to access resources or services that cannot easily be replicated locally. For example:

– A scientist (or business analyst) needs to access data located in different databases across a scientific collaboration (or enterprise).
– A business (or physics community) needs to allocate computing, storage, and network resources dynamically to support a time-varying e-commerce (or physics data analysis) workload.
– An engineer needs to design and operate experiments on remote equipment, linking and comparing numerical and physical simulations.
– An astronomy experiment needs to replicate a terabyte of data a day to partner sites around the world.

We find that while every application has unique requirements, a small set of functions frequently recur: for example, we often need to discover available resources, configure a computing resource to run an application, move data reliably from one site to another, monitor system components, control who can do what, and manage user credentials. Good-quality implementations of these functions can reduce development costs. Furthermore, if these implementations are widely adopted and/or implement standards, they can enhance interoperability. Globus software addresses both goals, using an open source model to encourage both contributions and adoption.

GT4 makes extensive use of *Web services mechanisms* to define its interfaces and structure its components. Web services provide flexible, extensible, and widely adopted XML-based mechanisms for describing, discovering, and invoking network

services; in addition, its document-oriented protocols are well suited to the loosely coupled interactions that many argue are preferable for robust distributed systems. These mechanisms facilitate the development of service-oriented architectures—systems and applications structured as communicating services, in which service interfaces are described, operations invoked, access secured, etc., all in uniform ways.

While end-user *applications* are typically concerned with domain-specific operations such as pricing a portfolio or analyzing a gene sequence, computing ultimately requires the manipulation and management of *infrastructure*: physical devices such as computers, storage systems, and instrumentation. Thus, GT4 provides a set of *Grid infrastructure services* [12] that implement interfaces for managing computational, storage, and other resources. In many Globus deployments (e.g., TeraGrid, Open Science Grid, LHC Computing Grid, China Grid, APgrid), these services are deployed to support a range of different application communities, each of which then executes their own application-specific code that relies on those services.

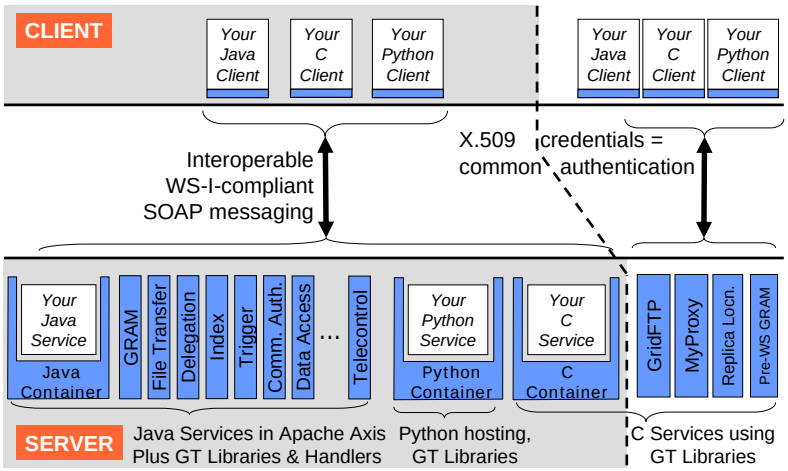

Figure 1: GT4 architecture schematic, showing many (but not all) components. Shared boxes denote GT4 code; white boxes represent user code

3 Globus Architecture

Figure 1 illustrates various aspects of GT4 architecture. I note first of all the following three sets of components:

– A set of *service implementations* (the bottom half of the figure) implement useful infrastructure services. These services address such concerns as execution management (GRAM), data access and movement (GridFTP [2], RFT, OGSA-DAI [4]), replica management (RLS [6], DRS), monitoring and discovery (Index, Trigger, WebMDS), credential management (MyProxy [16], Delegation,

SimpleCA), and instrument management (GTCP). Most are Java Web services but some (bottom right) are implemented in other languages and use other protocols.
- Three *containers* can be used to host user-developed services written in Java, Python, and C, respectively. These containers provide implementations of security, management, discovery, state management, and other mechanisms frequently required when building services. They extend open source service hosting environments with support for a range of useful Web service (WS) specifications, including WS Resource Framework (WSRF), WS-Notification, and WS-Security.
- A set of *client libraries* allow client programs in Java, C, and Python to invoke operations on both GT4 and user-developed services. In many cases, multiple interfaces provide different levels of control: for example, in the case of GridFTP, there is not only a simple command-line client (globus-url-copy) but also control and data channel libraries for use in programs—and the XIO library allowing for the integration of alternative transports.

It is important to note that GT4 is more than just a set of useful services. The use of uniform abstractions and mechanisms means that clients can interact with different services in similar ways, which facilitates the construction of complex, interoperable systems and encourages code reuse. This uniformity occurs at several levels:

- WS-I-compliant *SOAP messaging* among Web services and their clients.
- A common *security and messaging infrastructure* enables interoperability among different applications and services.
- A powerful and extensible *authorization framework* supports a range of different authorization mechanisms.
- The fact that all containers and most services implement common mechanisms for state representation, access, and subscription facilitates *discovery and monitoring*.

4 Globus Software Details: How Do I …?

Figure 2 provides another perspective on GT4 structure, showing the major components provided for basic runtime (on the right) and then (from left to right) security, execution management, data management, and information services. I introduce these components by showing how they are used to perform various tasks.

4.1 How Do I Manage Execution?

Let's say we want to run a task on a computer, or deploy and manage a service that provides some capability to a community. In both cases, we need to acquire access to a computer, configure that computer to meet our needs, stage an executable, initiate execution of a program, and monitor and manage the resulting computation.

The GT4 Grid Resource Allocation and Management (**GRAM**) service addresses these issues, providing a Web services interface for initiating, monitoring, and managing the execution of arbitrary computations on remote computers. Its interface allows a client to express such things as the type and quantity of resources desired,

data to be staged to and from the execution site, the executable and its arguments, credentials to be used, and job persistence requirements. Other operations enable clients to monitor the status of both the computational resource and individual tasks, to subscribe to notifications concerning their status, and control a task's execution.

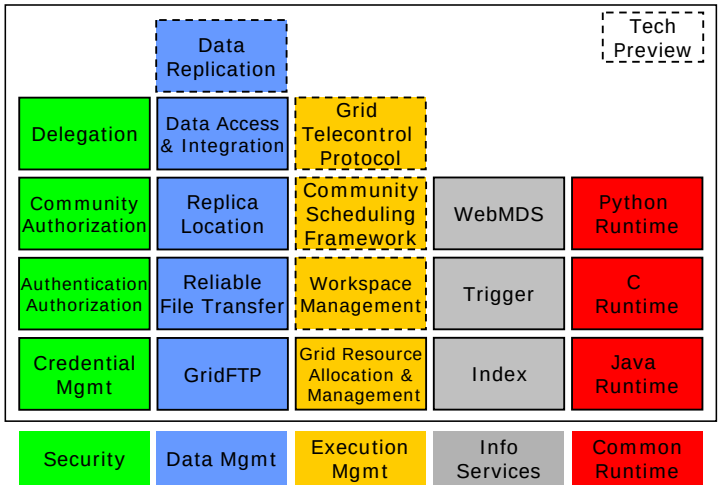

Figure 2: Primary GT4 components (dashed lines represent "tech previews")

A GRAM service can be used for many different purposes. The following are some examples:

- The GriPhyN Virtual Data System (VDS), Ninf-G, and Nimrod-G are all tools that use GRAM interfaces to dispatch (potentially large numbers of) individual tasks to computational clusters. For example, Rodriguez et al.'s GADU service [17] routinely uses VDS to dispatch several million BLAST and BLOCKS runs as it updates its proteomics knowledge base.
- Various applications use GRAM as a service deployment and management service, using a GRAM request first to start the service and then to control its resource consumption and provide for restart in the event of resource or service failure.
- The MPICH-G2 implementation [15] of the Message Passing Interface uses GRAM to coschedule subtasks across multiple computers. Dong et al. [8] have used MPICH-G2 to conduct a complete simulation of the human arterial tree.

The following execution management components are also provided within GT4 as "tech previews," meaning that they are less thoroughly tested than other components and more likely to change in the future:

- A Workspace Management Service (**WMS**) provides for the dynamic allocation of Unix accounts as a simple form of sandbox. (A variant of this service that provides for the dynamic allocation of virtual machines exists in prototype form.)
- The Grid TeleControl Protocol (**GTCP**) service is for managing instrumentation; it has been used for earthquake engineering facilities and microscopes.

4.2 How Do I Access and Move Data?

Globus applications often need to manage, provide access to, and/or integrate large quantities of data at one or many sites. This "data" problem is broad and complex, and no single piece of software can "solve" it in any comprehensive sense. However, several GT4 components implement useful mechanisms that can be used individually and in conjunction with other components to develop interesting solutions. (A recent article [3] reports on these tools and on various success stories.)

- The Globus implementation of the **GridFTP** specification provides libraries and tools for reliable, secure, high-performance memory-to-memory and disk-to-disk data movement. It has achieved 27 Gbit/s end-to-end over wide area networks, and can interoperate with conventional FTP clients and servers. GridFTP provides the substrate on which are built a wide variety of higher-level tools and applications.
- The Reliable File Transfer (**RFT**) service provides for the reliable management of multiple GridFTP transfers. It has been used, for example, to orchestrate the transfer of one million files from one astronomy archive to another.
- The Replica Location Service (**RLS**) is a scalable system for maintaining and providing access to information about the location of replicated files and datasets. The LIGO experiment uses it to manage more than 40 million file replicas.
- The Data Replication Service (**DRS**: a tech preview) combines RLS and GridFTP to provide for the management of data replication.
- The Globus Data Access and Integration (**OGSA-DAI**) tools developed by the UK eScience program provides access to relational and XML data.

4.3 How Do I Monitor and Discover Services and Resources?

Monitoring and discovery are two vital functions in a distributed system, particularly when that system spans multiple locations, as in that context no one person is likely to have detailed knowledge of all components. Monitoring allows us to detect and diagnose the many problems that can arise in such contexts, while discovery allows us to identify resources or services with desired properties. Both tasks require the ability to collect information from multiple, perhaps distributed, information sources.

In recognition of the importance of these functions, monitoring and discovery mechanisms are built in to GT4 at a fundamental level, as follows (see Figure 3).

- GT4 provides standardized mechanisms for associating XML-based *resource properties* with network entities and for accessing those properties via either pull (query) or push (subscription). These mechanisms—basically implementations of the WSRF and WS-Notification specifications—are built into every GT4 service and container, and can also be incorporated easily into any user-developed service. Services can be configure to register with their container, and containers with other containers, thus enabling the creation of hierarchical (or other) organizations.
- GT4 provides two *aggregator services* that collect recent state information from registered information sources. As not all information sources support WSRF/WS-notification interfaces, these aggregators can be configured to collect data from any

information source, whether XML-based or otherwise. The two aggregators implement a registry (**Index**) and event-driven data filter (**Trigger**), respectively.

– GT4 provides a range of browser-based interfaces, command line tools, and Web service interfaces that allow users to query and access the collected information. In particular, the **WebMDS** service can be configured via XSLT transformations to create specialized views of Index data.

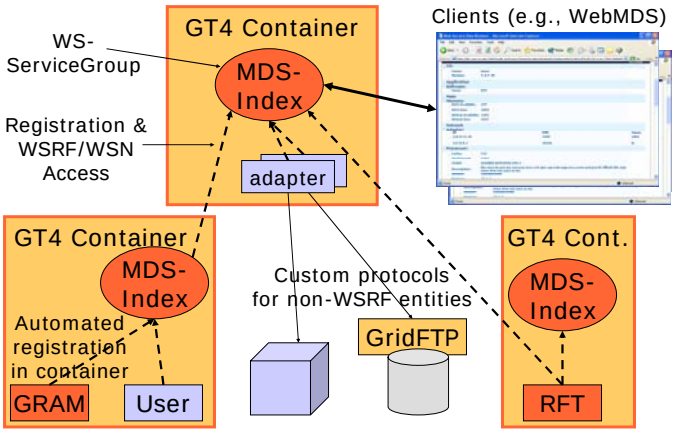

Figure 3: GT4 monitoring and discovery infrastructure

These different mechanisms provide a powerful framework for monitoring diverse collections of distributed components and for obtaining information about components for purposes of discovery. For example, the Earth System Grid (ESG) [5] uses these mechanisms to monitor the status of the various services that it uses to distribute and provide access to more than 100 TB of climate model data.

4.4 How Do I Control Who Can do What?

Security concerns are particularly important and challenging when resources and/or users span multiple locations. A range of players may want to exert control over who can do what, including the owners of individual resources, the users who initiate computations, and the "virtual organizations" established to manage resource sharing. "Exerting control" may include variously enforcing policy and auditing behavior. When designing mechanisms to address these requirements, we must work not only to protect communications but also to limit the impact of breakins at end systems. A complete security "solution" must always be a system that combines components concerned with establishing identity, applying policy, tracking actions, etc., to meet specific security goals. GT4 and related tools provide powerful building blocks that can be used to construct a range of such systems.

At the lowest level, GT4's highly standards-based security components implement credential formats and protocols that address message protection, authentication, delegation, and authorization. As shown in Figure 4, support is provided for (a) WS-

Security-compliant message-level security with X.509 credentials (slow) and (b) with usernames/passwords (insecure, but WS-I Base Security Profile compliant) and for (c) transport-level security with X.509 credentials (fast and thus the default).

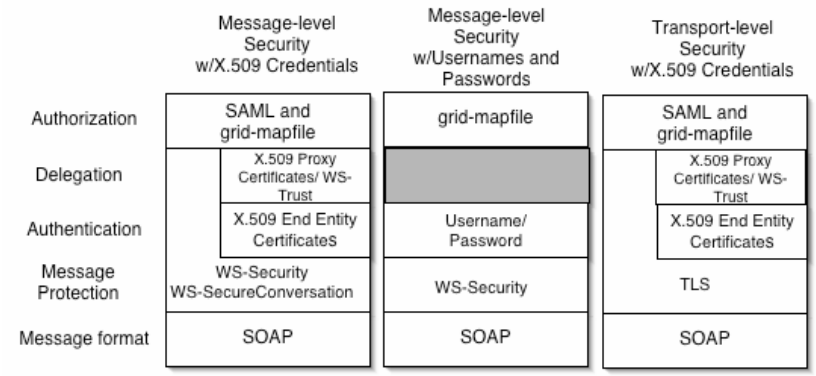

Figure 4: GT4 security protocols (see text for details). From [19].

In GT4's default configuration, each user and resource is assumed to have a X.509 public key credential. Protocols are implemented that allow two entities to validate each other's credentials, to use those credentials to establish a secure channel for purposes of message protection, and to create and transport delegated credentials that allow a remote component to act on a user's behalf for a limited period of time.

Authorization call outs associated with GT4 services can be used to determine whether specific requests should be allowed. In particular, the *authorization framework* component allows chains of authorization modules with well-defined interfaces to be associated with various entities, e.g. services, in the container. It also provides multiple different authorization module implementations, ranging from traditional Globus gridmap-based authorization to a module that uses the SAML protocol to query an external service for an authorization decision.

Supporting tools, some in GT4 and some available from other sources, support the generation, storage, and retrieval of the credentials that GT4 uses for authentication, and address related issues concerning group membership, authorization policy enforcement, and the like. These tools can be configured so that users need never manage their own X.509 credentials.

4.5 How Do I Build New Services?

A wide range of enabling software is included in GT4 to support the development of components that implement Web services interfaces. This software deals with such issues as message handling, resource management, and security, thus allowing the developer to focus their attention on implementing application logic. GT4 also packages additional GT4-specific components to provide *GT4 Web services containers* for deploying and managing services written in Java, C, and Python. As illustrated in Figure 5, these containers can host a variety of different services:

- Implementations of basic WS specifications such as *WSDL, SOAP, and WS-Security* support services that make use of these specifications to implement basic Web services functionality.
- Implementations of other specifications, notably *WS-Addressing, WSRF, and WS-Notification*, support services that want to expose and manage state associated with services, back-end resources, or application activities [11]. (For example, GT4 GRAM and RFT services use these mechanisms to manage state associated with tens of thousands of computational activities and file transfers, respectively.)
- The Java container is used to host the various *GT4 Java Web services* mentioned earlier, such as GRAM, RFT, DRS, Delegation, Index, and Trigger.
- Enhanced *registry and management capabilities*, notably the representation of information about services running in a container as WS-Resources, facilitate the creation of distributed registries and system monitoring tools.

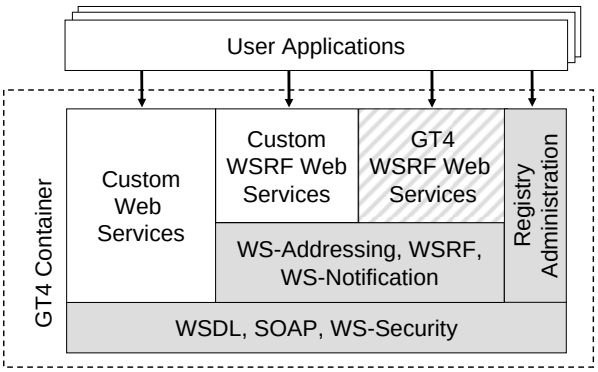

Figure 5: Capabilities of a GT4 container

In general, the Java container provides the most advanced programming environment, the C container the highest performance [14], and (Python enthusiasts would argue) the Python container the nicest language. If developing new services in Java using GT4, see the tutorial text [18] and its accompanying Web site.

Numerous projects are developing exciting services and applications based on GT4 containers. For example, the Belfast eScience Center has 1.5 million lines of GT4 Java code (converted from GT3, a process that required "relatively few changes in service code" [13]), implementing a range of applications including a digital video management system for the BBC, and the China Grid Support Package provides a rich set of services for eScience and education built on the GT4 Java container.

4.6 How Do I Do More Complicated Things?

GT4 services and libraries do not provide complete solutions to many distributed computing problems: to do anything more complex than submit a job or move a file, you must use GT4 software in conjunction with other tools and/or your own code—or access a (GT-based) service that provides the capabilities that you require [10].

In analyzing how people use Globus software, we find that the same patterns tend to reoccur across different projects and application domains. Thus, we have launched an effort to document these **solutions** [1] and how they can be implemented using components of the Globus ecosystem.

5 Processes, Results, and Evaluation

The Globus Alliance's **software engineering processes** have improved steadily over the past five years. These improvements have been made possible by both increased software engineering resources (i.e., dedicated engineers) and more aggressive users available for further testing. These processes now include:

- Extensive *unit test suites* and the use of *test coverage tools* to evaluate coverage.
- Frequent *automated execution of build and test suites* on more than 20 platforms, via both local systems and the NMI GRIDS Center's distributed build/test facility.
- Extensive *performance test suites* used to evaluate various aspects of component performance, including latency, throughput, scalability, and reliability.
- A cross-GT *documentation plan*, managed by a dedicated documentation specialist, to ensure complete coverage and uniform style for all components.
- A well-defined *community testing process*, which in the case of GT4 included a six-month alpha and beta-testing program with close to 200 participants.
- An *issue tracking system* based on bugzilla, used to track both error reports and feature requests, and the work associated with those issues.

GT4 **performance** is summarized in a recent report [9]. This report also provides pointers to more detailed documentation, including reports on the performance of different Web services containers, including GT4's Java, C, and Python [14]; the GT4 implementation of GridFTP [2]; and the GT4 replica location service [7].

The UK eScience program has released an **external evaluation** of GT4 [13]. This detailed report speaks favorably of the overall quality, usability, and performance of the GT4 code and its documentation. It notes, for example, that "GT4 installation was straightforward," "GT4 services demonstrated significant improvements in performance and reliability over their GT3 versions," and "GT4 package descriptions were of a high quality, well structured, and accurate."

6 Contributing

A large and diverse Globus community is working hard to improve the scope and quality of the Globus software. I hope that you, the reader, will feel inspired to contribute also. There are several ways in which you can do so.

Use the software and report your experiences. Simply using the software and reporting back on your experiences, positive or negative, can be immensely helpful. Reports of problems encountered, particularly when well documented, help guide bug

fixes and/or prioritize work on new features. Reports of successful deployments and applications can help justify continued support for the development of the software.

Develop documentation and examples. Despite considerable progress, we remain in desperate need of code examples and associated documentation that can help other users to start work with Globus software or related tools. Take the time to document your successful application, and you will be repaid in gratitude from other users.

Contribute to the development of the software. The list of new features wanted by users is always far greater than Globus developers can handle. You can contribute bug fixes, test cases, new modules, or even entirely new components.

7 Futures

We are entering an exciting time for Globus, due to the confluence of the following factors:

- The completion of GT4 means that Globus now has a solid Web services base on which to build additional services and capabilities.
- Sustained funding for eScience support will allow us to accelerate efforts aimed at meeting demands for ever-greater scalability, functionality, usability, and so forth.
- The creation of organizations dedicated to the support needs of industry means that commercial adoption (and contributions) will accelerate.
- A rapidly growing user community is increasing the quantity and quality of user feedback, code contributions, and components within the larger Globus ecosystem.
- Revisions to the Globus infrastructure and governance processes will make it easier for us to engage additional contributors to the software and documentation.

Acknowledgements

I report here on the work of many talented colleagues and collaborators (see www.globus.org). The core team is based primarily at Argonne National Lab, U. Chicago, the USC Information Sciences Institute, U. Edinburgh, the Royal Institute of Technology, the National Center for Supercomputing Applications, and Univa Corporation. Many others in both academia and industry have contributed to code, documentation, and testing, or made our work worthwhile by using the code.

Work on Globus has been supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by the National Science Foundation (NSF) under its NSF Middleware Initiative and other programs, and by IBM, DARPA, NASA, Microsoft, the UK Engineering and Physical Sciences Research Council and Department of Trade and Industry, and the Swedish Research Council.

Foster is also co-founder and Chief Open Source Strategist at Univa Corporation.

References

1. Grid Solutions, 2005. www.globus.org/solutions.
2. 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.
3. Allcock, W., Chervenak, A., Foster, I., Kesselman, C. and Livny, M., Data Grid Tools: Enabling Science on Big Distributed Data. *SciDAC Conference*, 2005.
4. Atkinson, M., Chervenak, A., Kunszt, P., Narang, I., Paton, N., Pearson, D., Shoshani, A. and Watson, P. Data Access, Integration, and Management. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 2004.
5. 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.
6. Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunst, P., Ripenu, M., Schwartzkopf, B., Stockinger, H., Stockinger, K. and Tierney, B., Giggle: A Framework for Constructing Scalable Replica Location Services. *SC'02: High Performance Networking and Computing*, 2002.
7. Chervenak, A.L., Palavalli, N., Bharathi, S., Kesselman, C. and Schwartzkopf, R., Performance and Scalability of a Replica Location Service. *IEEE International Symposium on High Performance Distributed Computing*, 2004.
8. Dong, S., G, K. and Karonis, N. Cross-site computations on the TeraGrid. *Computing in Science & Engineering, 7* (5). 14-23. 2005.
9. Foster, I. Performance of Globus Toolkit Version 4. Globus Alliance, 2005. www.globus.org/alliance/publications.
10. Foster, I. Service-Oriented Science. *Science*, *308*. 814-817. 2005.
11. Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D. and Tuecke, S. Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF. *Proceedings of the IEEE, 93* (3). 604-612. 2005.
12. Foster, I. and Tuecke, S. Describing the Elephant: The Different Faces of IT as Service. *ACM Queue*, *3* (6). 2005.
13. Harmer, T., Stell, A. and McBride, D. UK Engineering Task Force Globus Toolkit Version 4 Middleware Evaluation, UK Technical Report UKeS_2005-03, 2005.
14. Humphrey, M., Wasson, G., Jackson, K., Boverhof, J., Meder, S., Gawor, J., Lang, S., Pickles, S., McKeown, M. and Foster, I. A Comparison of WSRF and WS-Notification Implementations: Globus Toolkit V4, pyGridWare, WSRF:Lite, and WSRF.NET. *14th International Symposium on High Performance Distributed Computing*. 2005.
15. Karonis, N., Toonen, B. and Foster, I. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, *63* (5). 551-563. 2003.
16. 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.
17. Rodriguez, A., Sulakhe, D., Marland, E., Nefedova, V., Maltsev, N., Wilde, M. and Foster, I., A Grid-Enabled Service for High-Throughput Genome Analysis. *Workshop on Case Studies on Grid Applications*, Berlin, Germany, 2004.
18. Sotomayor, B. and Childers, L. *Globus Toolkit 4: Programming Java Services*. Morgan Kaufmann, 2005.
19. Welch, V. Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective, 2004. http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf.