University of Victoria
Faculty of Engineering
Summer 2006 Work Term Report

# Greater Demands on Computational Grids: Can Web Services Facilitate Global Grid Expansion?

Department of Physics and Astronomy
University of Victoria
Victoria, Canada

Simon Ramage
0536316
Computer Engineering
sramage@uvic.ca

August 29, 2006

In partial fulfillment of the requirements of the Bachelor of Engineering Degree

| Supervisor's Approval: To be completed by Co-op Employer | | |
|---|---|---|
| I approve the release of this report to the University of Victoria for evaluation purposes only.<br>This report is to be considered: NOT CONFIDENTIAL CONFIDENTIAL | | |
| Signature | Position | Date |
| Name | Email | Fax |

4654 Boulderwood Dr
Victoria, BC
V8Y3G5

Mr. Roel Hurkens
Co-op Coordinator
Faculty of Engineering
University of Victoria
P.O. Box 3055 STN CSC
Victoria, BC
V8W 3P6

August 29, 2006

Dear Mr. Hurkens,

Please accept the accompanying work term report entitled "Greater Demands on Computational Grids: Can Web Services Facilitate Global Grid Expansion?".

This report is the outcome of work completed at the University of Victoria, Department of Physics and Astronomy. Over the course of my first work term in the Computer Engineering program, following my 3A academic term, I was involved in a year long project to evaluate a new implementation of grid computing based on Web services middleware. My specific research tasks included grid system design, deployment, administration, computer networking, programming, and testing, to determine if Web services middleware is capable of supporting the global expansion of the grid.

Throughout the duration of the summer term, I had the opportunity to gain knowledge and experience in grid computing development. I believe this will prove valuable in my future academic and professional ventures.

I would like to extend my thanks to Dr. Randall Sobie, Dr. Ashok Agarwal, Dan Vanderster, Ron Desmarais, Ian Gable, and Patrick Armstrong, for their support and collaboration.

Sincerely,

Simon Ramage

# Contents

# List of Figures

# Executive Summary

Contemporary scientific endeavors such as high energy particle physics demand a level of computational power that cannot be achieved by conventional methods. While computer manufacturers are developing more and more powerful computers at a stunning rate, they are not likely to keep pace with the growth in complexity of compute-intensive applications. This is the driving force behind grid computing, in which the resources of many computers are aggregated to solve a single problem.

There is an increasing need for grids to expand across corporate, institutional, and geographic boundaries, to realize a truly world-wide grid that can meet the goals of the most challenging applications. The solution to this problem presents yet more challenges, as with disparate domains comes disparate ideas when it comes to grid administration, operation, and security. The current de facto standard middleware in the grid community, Globus Toolkit 2, lacks standardization, and is therefore unable to facilitate future wide-spread expansion of the grid. New Web services based middleware designed with a service oriented architecture is highly standards-based, and is not application or platform specific, representing a significant leap forward in the ability to interconnect heterogeneous domains in the grid.

The University of Victoria, in conjunction with the National Research Council, has undertaken the GridX1 Services project, a year long endeavor to evaluate an implementation of the grid using such Web services middleware. This evaluation has involved conceptual research on the use of Web services, and how it would be expected to impact an organization, as well as empirical research in the form of implementing a standalone test grid. The results have shown that migrating an existing grid implementation to Web services is a feasible procedure, has a positive impact on grid performance, scalability, security, and usability, and does indeed allow dissimilar grid clusters to communicate. This leads to the conclusion that Web services can facilitate the growing need for grids to expand on a global scale.

As the GridX1 Services project moves into its final phase in the fourth quarter of 2006, important testing of the Web services implementation will be carried out. Results of these tests will provide further evidence that should be a prominent factor in deciding to deploy Web services middleware in production situations. It is therefore recommended that this testing be completed before further conclusions are drawn.

# Glossary of Terms and Symbols

*API*: Application Programmers Interface

*ATLAS*: A particle physics experiment that will explore the fundamental nature of matter

*BaBar*: A High Energy Physics experiment at the Stanford Linear Accelerator Center

*CANARIE*: The Canadian Network for the Advancement of Research, Industry, and Education

*CERN*: European Particle Physics Laboratory in Geneva, Switzerland

*EGEE*: Enabling Grids for E-sciencE - A European project to develop a service grid infrastructure for scientists

*GASS*: Global Access to Secondary Storage

*GridBus*: A project engaged in the design and development of grid middleware

*GridFTP*: Grid File Transfer Protocol

*GUI*: Graphical User Interface

*HEP*: High Energry Physics

*LGC*: LHC Computing Grid

*LHC*: Large Hadron Collider

*Middleware*: Software that connects two dissimilar applications

*SDK*: Software Development Kit

*OGSA*: Open Grid Services Architecture

*Open Science Grid*: An american grid computing infrastructure

*RFT*: Reliable File Transfer

*SAML*: Security Assertion Markup Language - Framework for exchanging authentication and authorization information

*SOAP*: Simple Object Access Protocol - XML based protocol used to invoke Web services

*SOA*: Service Oriented Architecture

*TeraGrid*: The world's largest distributed cyberinfrastructure for open scientific research

*WSDL*: Web Services Description Language

*WSRF*: Web Services Resource Framework

*XML*: Extensible Markup Language - User defined description language

# 1 Introduction

Grid Computing, at a high level, is when the resources of many computers across a network are aggregated to solve a single problem. Resources may consist of any assets that a computer has that can be made available for use, notably processing power, memory, and data storage. The grid takes its name from the well known electrical power grid, which aggregates electricity from many sources and provides it to the public.

The applications of grids often have different priorities, and as such different flavors exist, of which the most common ones are computational, data, and scavenging grids. Computational grids are concerned with aggregating the computational power of computers for compute-intensive applications, whereas data grids are concerned with combining the storage space of computers to handle data transfers across the grid. Scavenging grids take advantage of under-used resources from desktop computers, often over the internet from the general public. The grid can be thought of as an analogue to the World Wide Web, where the World Wide Web is a network of computers with the goal of sharing information, the grid is a network of computers with the goal of sharing computer resources. By providing ubiquitous access to the combined computing resources of institutions across the globe, grid computing can be tremendously powerful.

The requirements of modern grid applications are ever-increasing, creating a need for the further expansion of computational grids world-wide. In order to realize this expansion across increasingly disparate domains, the problems of standardization, consistency, cross-platform interoperability, and scalability must be resolved. The new Web services based grid middleware, Globus Toolkit 4 (GT4), takes steps to address these problems. This report investigates the implications of migrating a computational grid to this Web services based middleware from the current de facto standard, Globus Toolkit 2 (GT2), and presents results of whether or not Web services answers the challenge of modern grid computing. Initially, important background information will be discussed which will further the readers understanding of grids.

## 1.1 Applications of Grid Computing

The first well known application of grid computing was the SETI@Home project, which was launched in May 1999 at Berkeley University. SETI@Home scavenges the resources of millions of computers around the world to search the skies for evidence of radio transmissions from extraterrestrial intelligence, with the ultimate goal of discovering intelligent life outside earth. Recently, more pragmatic examples of public grid computing have been launched. The World Community Grid also scavenges public resources, and currently has three research projects underway [1]:

*Help Defeat Cancer* examines tissue micro-arrays to improve cancer treatment by developing better diagnostic abilities.

*Human Proteome Folding* attempts to examine human protein structures at higher resolution to better understand how they function.

*FightAIDS@Home* uses computers to help develop drugs that have the ability to block enzymes that cause HIV to mature into AIDS.

Additional applications of grid computing include Earth observational satellite imaging, climate modeling, and high energy particle physics (HEP). In HEP, grids are used to store raw data, generate simulations, and perform analysis. The BaBar experiment at the Stanford Linear Accelerator Center (SLAC), for example, makes use of computational grids. In 2007, the ATLAS experiment to study proton-proton interactions at the LHC at CERN will be underway, and represents a computational challenge far exceeding in scale and complexity any scientific experiment in history. In order to meet this challenge, the LHC Computing grid is being put together, which will store an estimated 15 PetaBytes of data annually, and will require approximately 100,000 CPU's.

## 1.2 The Basic Components of a Grid

A computational grid environment will typically be made up of a client interface, a resource broker, a data management system, and an implementation of security [2]. This section describes these components.

### 1.2.1 Client Interface

The client interface will present users with some kind of portal to access the grid. This can come in many forms. In some cases a fully transparent GUI may be provided that would not require extensive knowledge of the grid implementation to use, and in some cases a list of function calls for accessing grid services will simply be provided to the user in the form of a Client API.

### 1.2.2 Resource Brokering

Resource brokering is the component responsible for identifying which resources are available and which ones may or may not meet the specific requirements of tasks. Furthermore, given a list of appropriate resources, a task will be *scheduled*, or assigned, to one of those resources. This can be achieved by means of a scheduling algorithm which will make decisions on how to coordinate the job execution. In some applications many jobs will have dependencies on one another, in which case the scheduling will need to handle the work-flow of the jobs and can become quite complex. Some grid environments do not make use of scheduling at all, and simply send jobs to available resources. In HEP applications, jobs are typically referred to as "embarrassingly parallel", which means that a problem can be split up into many parallel tasks with little to no effort, as there is effectively no inter-dependencies between them [3].

### 1.2.3 Data Management

Data management addresses the necessity for having secure and reliable data transfer when a grid application requires that files be sent across the network. Virtually every grid implementation will require data transfer, this component is very important. In many cases a third party will control a file transfer between two other nodes in the grid.

### 1.2.4 Security

Distributed computing makes heavy use of insecure, inter-domain computer networks. At the same time, there is a strong need for the data sent across these insecure networks to be kept confidential, and therefore security is essential in any grid environment. As such, there exist extensive tools providing authentication, authorization, and encryption that are utilized in grids. These are often incorporated into the client interface in the form of a proxy.

# 2 GridX1

The University of Victoria threw it's hat into the grid community in 2001, when the Grid Canada Testbed was developed to study the use of computational grids in high energy particle physics applications. This testbed became what is now known as GridX1, and is one of the first examples of a large Canadian research computing grid. It is currently collaborated on between 7 Institutions across the country. In total GridX1 has 9 clusters of over 2500 CPU's, of which UVic accounts for 2 clusters of over 450 CPU's. Currently GridX1 is processing jobs for the BaBar and ATLAS Particle Physics experiments.

## 2.1 Globus Toolkit 2 Architecture

Because grids are so versatile, and span various domains (corporate, institutional, geographic), even their simplest use can become extremely complex, with each cluster involved in the grid having disparate deployments in terms of security, administration, data management, scheduling, etc. Overcoming this challenge has become one of the major goals in grid computing. In order to provide ease of use to the end user, and thus transparent access to grid resources, the grid community makes heavy use of what is called *middleware* to take care of inter-application communication. Grid middleware essentially provides a set of tools for the deployment of grids and the development of grid applications. This added level of abstraction is portrayed in Figure 1. Middleware acts as a portal for users to access the grid, in the same sense that one can access the electrical power grid through a simple outlet in the wall.

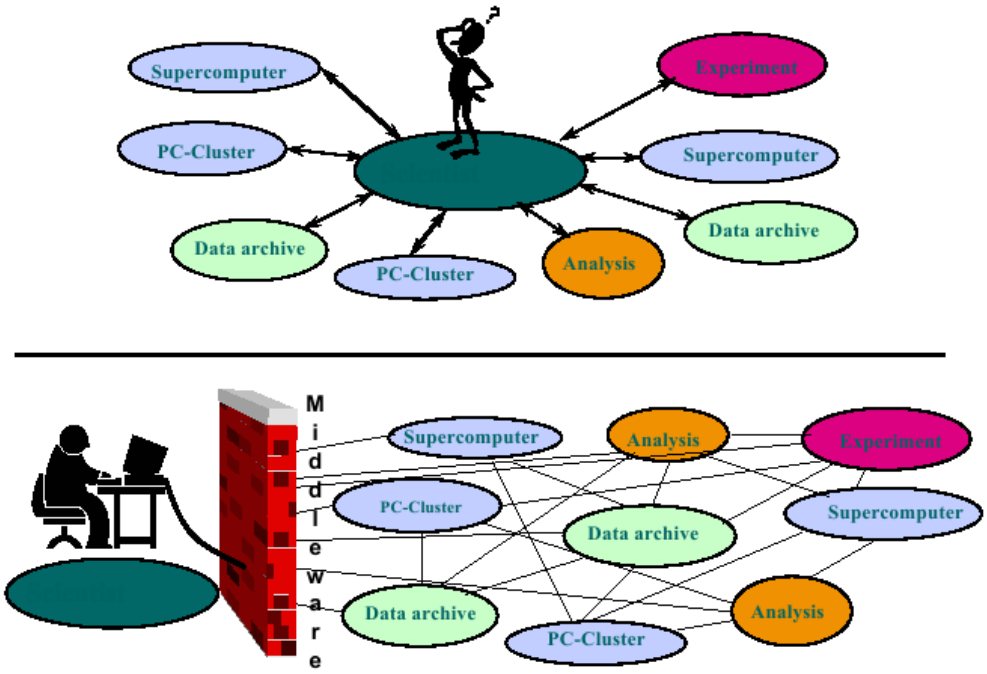


Figure 1: Middleware: Policing inter-application communication [4]

The current de facto standard middleware used by the grid community, (including UVic on GridX1), is Globus Toolkit 2 (GT2), which was developed as part of the Globus Project. Formed in 1996, the Globus Project is a growing community of people, now known as the Globus Alliance, who collaborate on the development of open source software for the purpose of grid computing and resource aggrega-

tion. The Globus Toolkit (GT) represents the collection of software components developed by the Globus Alliance to support the implementation of dynamic, inter-domain computational grids [5]. November 2001 saw the release of GT2, which built upon the loosely structured component-based design of the original GT, however still lacked conformity with a standard architecture. GT2 can be thought of as a set of services that do not follow a common set of standards.

## 2.2 The Three Pillars of GT 2

GT2 is described as being comprised of three primary components, or Pillars, which provide the three ingredients identified by Globus as the primary functions of a grid computing environment. Each of the pillars is implemented in a bundle containing server, client, and SDK packages. The pillars are outlined as follows [6] [7]:

***Resource Management*** deals with grid resource allocation and management (GRAM), which makes up the core functionality of grid computing. Resource management is sometimes referred to as Job Management or Execution Management.

***Information Services*** is comprised primarily of monitoring and discovery services (MDS), which deal with the provision of grid resource information. For example, MDS will keep track of the number of available, and busy processors in a cluster. In GT2 this is achieved by way of querying information from a Lightweight Directory Access Protocol (LDAP) server.

***Data Management*** encompasses data transfer and access protocols, such as grid File Transfer Protocol (GridFTP), which implement *globus data grid services*. GridFTP is used when data files are sent between computers in the grid, which could be the client sending an input file, or executable file, to the metascheduler, or the metascheduler sending back an output file to the client.

## 2.3 Security in GT 2

While not explicitly defined as one of the pillars of GT2, security is still a key component of any grid infrastructure. The Grid Security Infrastructure (GSI) in GT2 allows for the secure communication and identification of users and hosts across insecure networks by providing a certificate based authentication system. RSA encryption is used, meaning the security is based on Public Key Infrastructure (PKI), where both public and private keys are used for mutual authentication. As an additional layer of security, users must delegate credentials to a proxy service which will perform the end-to-end communication with a host.

## 2.4 Service Orientation

The focus of research and development on improving grid middleware in recent years, as with much of the IT industry, has been to move towards a *service oriented* software design. By following certain design principles governing communication, architecture, and implementation, the various software components in a grid can be represented as services. Fundamentally, services are intended to be the elements which collectively make up an application environment. Each component can take on the role of service provider, or service requester. This introduces a level of standardization in grid middleware, acting as a *road-map* for the inter-application communication. Service based technology differs from component based technology through a unique set of requirements, an important subset of which is as follows [8]:

- *Re-usability*: Logic is split up between services, helping to meet future requirements without as much development.
- *Loose coupling*: Relationships between services are minimized to the point of only maintaining an awareness of each other, without actual dependence.
- *Abstraction*: To the outside world, a service looks like a black-box, with the underlying logic concealed.
- *Autonomy*: Control of the underlying logic in a service can only be controlled by the service itself.
- *Statelessness*: Information about the current state of a service is not maintained beyond the life of each implementation of that service.
- *Discoverability*: Services are made available to external resources by publishing to a service registry.

## 2.5 Service Orientation in GT2

GT2 is often described as simply providing a *bag of services*, facilitating the use of individual toolkit components by application software [9]. This is a crude but prototypical implementation of service orientation. Packages in GT2 are implemented as standard Unix services, typically located in the /etc/xinetd.d folder (for a standard Linux implementation). For each of these component packages, a C API has been developed. The *Globus Hourglass* in figure 2 is a conceptual depiction of how local OS software communicates through a small number of core globus services, which then communicate with more complex services that interact with applications and utilize grid resources. This creates a layer of transparency at the end user level, and effectively illustrates the concept of middleware.

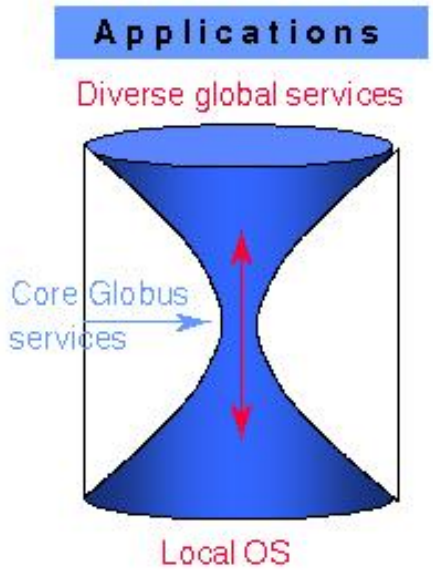


Figure 2: The Globus Hourglass [10]

# 3 GridX1 Services Project

In January 2006 the grid computing team at UVic, in conjunction with the National Research Council in Ottawa, began the GridX1 Services Project (GridX1-SP), funded by CANARIE, to evaluate a new implementation of the grid with *Web services* based middleware. This evaluation involved a significant amount of theoretical research on the concepts of Web services and how they might be applied to solve challenges with the grid, as well as empirical research in implementing a test grid with this new technology, and observing its effectiveness. The latter is still in progress, with the fourth quarter of the project not yet underway. This section will outline the facts, evidence, and testing results of both forms of research.

## 3.1 Web Services

Web services are an emerging technology that can be used to implement Service Orientation. The standard for Web services Architecture has been defined by the World Wide Web Consortium (W3C), as follows:

> A Web service is a software system designed to support interoperable machine-to- machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAPmessages, typically conveyed using HTTP [11].

Web services communicate through an XML format, which is neither platform nor application specific. This means that dissimilar programs, on different operating systems, are able to communicate with each other via a standard Web services protocol. Web services may be thought of as self-contained applications available to users through a Web interface.

Figure 3 depicts how Web services work at a very high level. Two example Web services are shown, one that can provide weather information and one that can perform math operations. Each Web service maintains its own address and a brief description of what it does. This information is registered with a *discovery agent*, which is itself a Web service. The end user might, for instance, require a service that can do math operations. The client interface already knows the location of the discovery Web service, and contacts it to let it know the users requirements. The discovery Web service will then search its own records to discover one or more available Web services that meet these requirements, and will send the client application the necessary information on how to locate them. At this point the client can proceed to make a service request to the appropriate Web service.

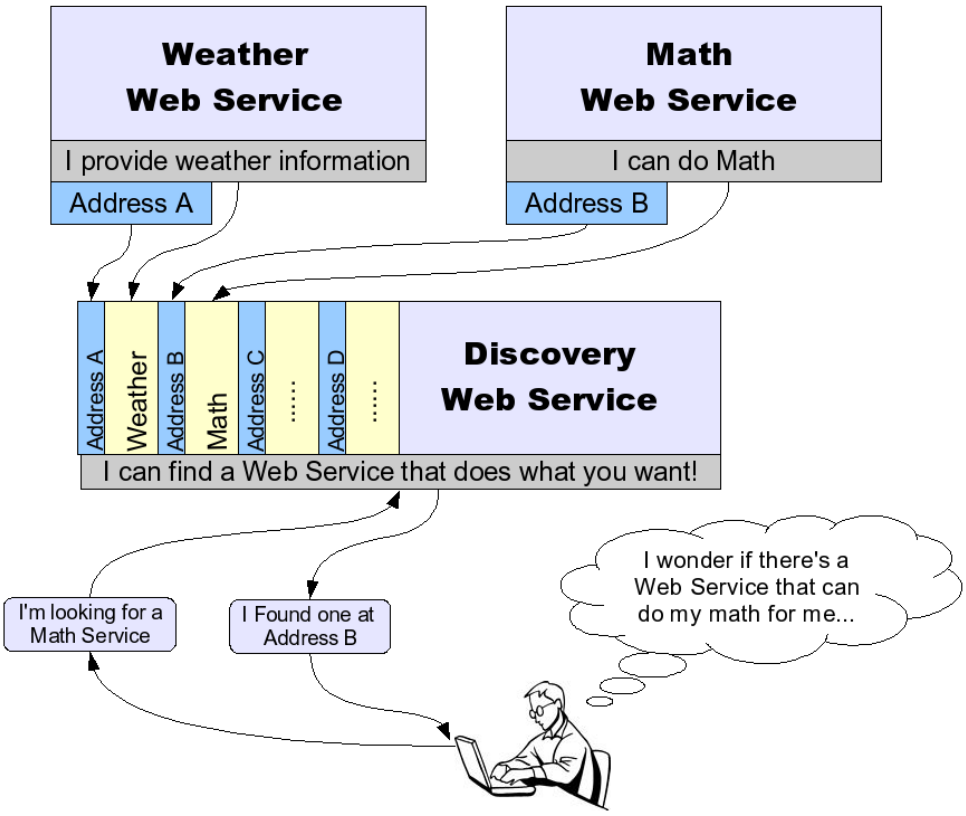


Figure 3: Web service communication

## 3.2 Globus Toolkit 4 Architecture

In the short years following the release of GT2, a wider span of people and organizations became interested and involved in the field of grid computing, and the needs and goals of such a grid computing environment have correspondingly become more well defined. For example, current applications in high energy particle physics, climate modeling, enterprise workload management, and high performance data capture now demand such vast amounts of computing power and storage that they require collaboration on an international level [12]. This collaboration in turn demands a higher level of standardization and documentation. With the release of GT4 in April 2005, these current needs and goals are realized. While GT4 has undergone much more rigorous testing by the development team than was done with GT2 [13], it has not yet seen widespread deployment in the grid community.

### 3.2.1 Service Oriented Architecture

While GT2 made only the first steps towards service orientation, GT4 has been designed with a complete SOA in mind. The various applications of modern grid computing all have a set of unique demands, but at the same time there is typically a large subset of common demands. The concept of a service oriented architecture takes advantage of this commonality by reusing components wherever possible. This SAO is followed by implementing a framework with a set of standard mechanisms and interfaces that realize the common principles of service orientation listed previously [12]. The Open Grid Services Architecture (OGSA) standard has been defined specifically for this purpose. Says the Globus Alliance, "The Open Grid Services Architecture represents an evolution towards a grid system architecture based on Web services concepts and technologies. [14]".

### 3.2.2 Web Services in GT4

The Web service design model was selected as the underlying middleware in GT4 on which to base the OGSA. Key in the implementation of an OGSA, however, is the need for stateful services. For a service to be stateful, it must have the ability to maintain information about its previous state, and not be wiped clean after each use. In order to have this capability with Web services, which are *not* stateful, the Web Services Resource Framework (WSRF) is incorporated as the infrastructure for the OGSA. WSRF provides the necessary groundwork to persist service state information within a resource.

The autonomous nature of Web services, and the XML platform on which they are based makes them a very powerful tool. The communication between applications reaches new heights of standardization, making the expansion of grids across increasingly disparate domains more realistic, as there is no notion of a common op-

erating environment. In Figure 4, the middleware diagram is shown in a yet simpler form involving Web services.

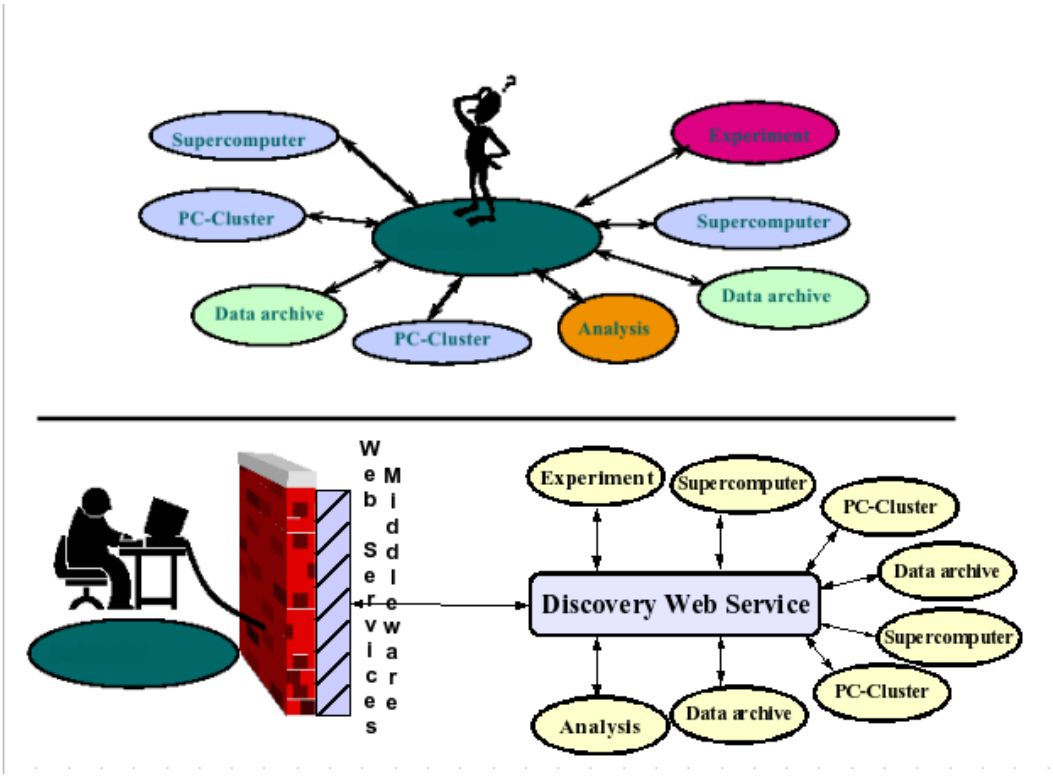


Figure 4: Web services based middleware [4]

### 3.2.3 GT4 Components

Analogous to the pillars described in GT2, the primary components of GT4 are split into the following four categories:

***Execution Management*** In GT4, both Web services (WS) and pre-Web services (pre-WS) interfaces are provided for the submission, monitoring, and termination of remote computational tasks (jobs). These interfaces are known as Web services Grid Resource Allocation and Management (WS-GRAM), and correspondingly pre-WS-GRAM. GRAM addresses the concerns of reliability, credential management, stateful monitoring, and file staging, which are all integral to the execution of jobs on the grid. The core of WSGRAM is made up of Web services developed for use in the WSRF environment. Of particular note are the ManagedExecutableJob and ManagedJobFactory Web services. Every time a job is submitted, the job is represented as a resource in the form of a ManagedExecutableJob Web service, which acts as an interface for the user to monitor or terminate the job. Similarly, every distinct compute element,

or execution host, is represented as a resource in the form of a ManagedJobFactory Web service, which acts as the interface by which job resources (or ManagedExecutableJob Web services) are created [15].

WS-GRAM makes use of several other components of GT4. File transfer reliability is achieved through the invocation of the Reliable File Transfer (RFT) Web service during file staging operations. Credential delegation into the desired hosting environment is achieved through invoking the Delegation Service. WS-GRAM includes client API's for C, Java, and Python, as well as a command line client.

In addition to GRAM, the execution management component of GT4 also includes the Globus Teleoperations Control Protocol (GTCP), which provides telecontrol in earthquake simulation experiments, and a Workspace Management Service (WMS), which allows clients to run a Unix-type workspace at remote locations [16],[17],[18].

***Data Management*** Data Management involves the transfer, replication, and management of data in a distributed environment. GT4 primarily makes use of two key components in this respect which will be described in this document: GridFTP, and RFT. GridFTP is a protocol that facilitates the transfer of distributed data via an open socket connection. Both client and server side installations are required for this transfer to take place. It is important to note that a Web services based implementation of GridFTP has yet to be released. Regardless, GT4 does introduce a new and very different version of GridFTP, primarily on the server side. This reimplementation of the server is built using the high performance Globus Extensible Input/Output (XIO) libraries. XIO is a packet-based system bus that connects high performance devices to a computer's memory. This replaces the previous implementation of GridFTP server in GT2, which was a modification of the Washington University FTP server (wuftpd server). Noteworthy is that this reimplementation eliminates a licensing problem and a security hole that existed with the distribution of the wuftpd server. Data striping with multiple network endpoints or destinations is now supported in GridFTP, whereby the workload of a file transfer is shared between nodes, each taking care of a piece of the file. Clusters with parallel shared file systems can take advantage of this. The server in GT4 also supports anonymous access if the administrator wishes to deploy it. GridFTP in GT4 is robust on the server side, however does have a local point of failure in the client workstation [15].

RFT goes one step further, by holding GridFTP file transfer state in third party reliable storage. This means that both remote and local failures (client and

server) can be recovered from. Furthermore, RFT is a Web service, representing the migration of the Data Management component of GT4 towards the aforementioned service architecture. Similar to the way GRAM acts in response to job submissions, an RFT resource is created when a user makes a third-party GridFTP transfer request. The state of the transfer is persisted in this resource, allowing the user to monitor the status or manage the transfer [15],[19].

***Monitoring and Discovery*** The Monitoring and Discovery System (MDS), sometimes referred to as Information Services, is concerned with the acquisition, monitoring, and discovery of resources in a grid environment. GT4 includes the Web services MDS (WS MDS) suite, also known as MDS4, which contains Web services that provide monitoring and discovery. GT4 also maintains the pre-WS MDS (MDS2) suite for use with older grid implementations.

In MDS4, an interface is provided for users to obtain resource information by querying and subscribing to WSRF services. While MDS4 can suffice as a standard cluster monitoring system, it is not intended as such, and includes an extensible interface which is optimized for use when used with more detailed grid monitoring systems, such as Ganglia [15]. At a high level, MDS4 is structured in an aggregator framework, which acts as the underlying mechanism for building WSRF compliant Aggregator Services, outlined as follows [12]:

- *Index Service*: responsible for collecting data and making it available to users.
- *Trigger Service*: responsible for collecting data and allowing users to define specific actions to be triggered based on sets of conditions in the data.
- *Archiver Service*: responsible for maintaining old data.

These Aggregator Services collect information from Aggregator Sources (also known as Information Sources). MDS4 has three Aggregator Sources, listed as follows [15]:

- Query Aggregator Source
- Subscription Aggregator Source
- Execution Aggregator Source

In order to build a registry in MDS4 with the goal of aggregating information from multiple sources, the Index Service on one host can be registered to the Index Service on another, forming a hierarchy. As well, both Aggregator Services and Sources have the ability to use external software to access

information. Such external software components are commonly referred to as *Information Providers*.

***Security*** The GSI in GT4 has been migrated to a Web services based system of authentication and authorization, while still maintaining the pre-WS components from GT2. Foremost in the new GSI is a more defined set of standards and motivations. Services are now designed to use different functional layers of security (ie: transport level, message level) [20]. The Globus Alliance outlines the primary motivations of GSI as follows [6]:

- The need for secure communication between elements of a computational grid.
- The need to support security across organizational boundaries, thus prohibiting a centrally managed security system.
- The need to support "single sign-on" for users of the grid, including delegation of credentials for computations that involve multiple resources and/or sites.

An important advancement in regards to GT4's security capabilities is the new authorization framework. Administrators have more configuration options, including access control lists through the **grid-mapfile** or specific definitions by a service, customizable authorization schemes, and access control through the Security Association Markup Language (SAML) protocol. SAML is a framework for sending and receiving authorization and authentication credentials in an XML format [12],[20].

GT4 also provides additional forms of proxy certificates, one of which is partially compliant with RFC 3820, the specification for the X.509 Public Key Infrastructure Proxy Certificate Profile, and another of which is fully RFC 3820 compliant. Legacy support is maintained for old proxy certificates as well. Furthermore, GT4 gives clients the option of delegating a proxy certificate to a Web service.

Through this design model, GT4 has been made into a standards-based middleware, streamlining the interoperability between heterogeneous systems. Figure 5 depicts a functional comparison of the components of GT2 and GT4.

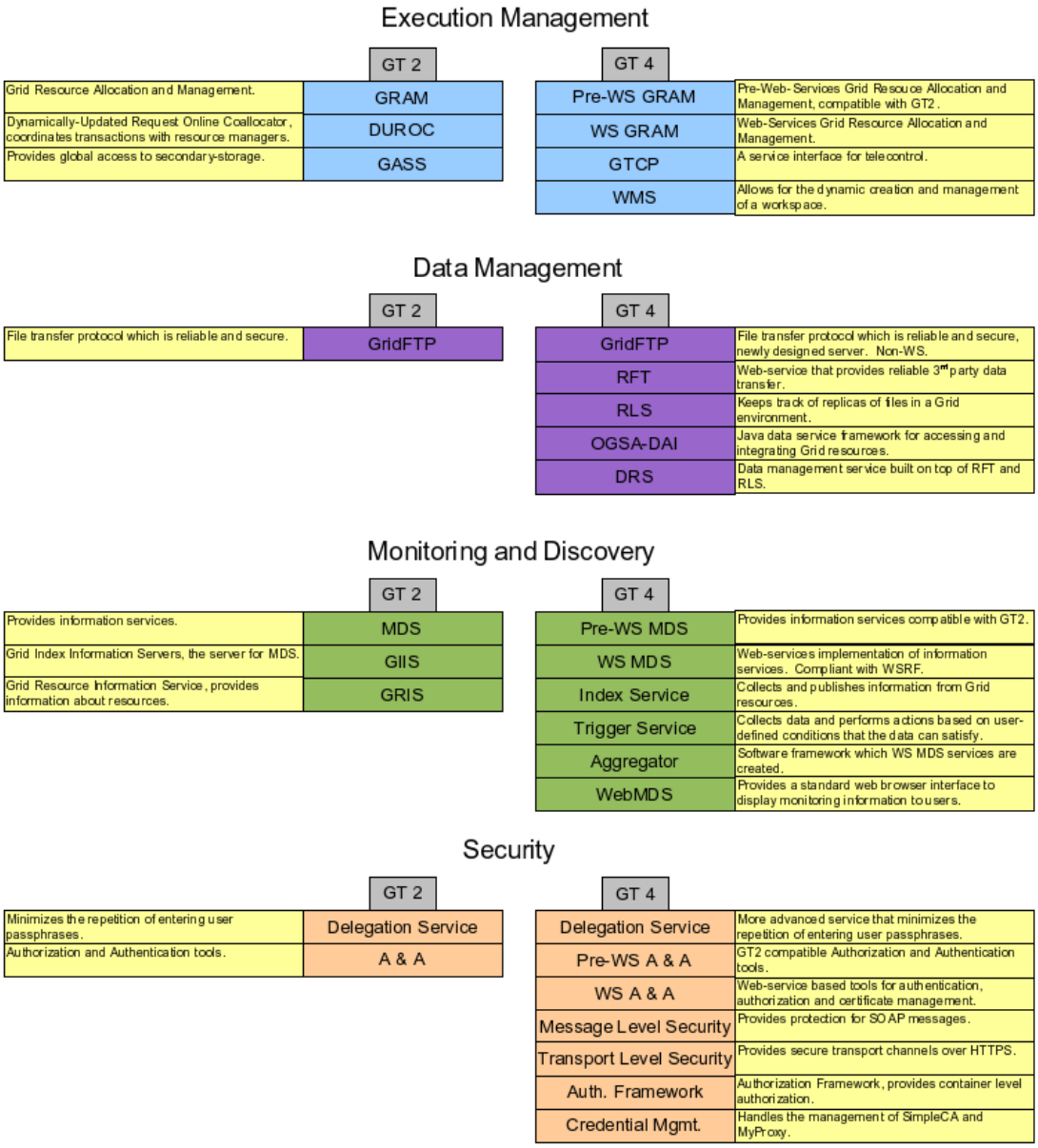


Figure 5: Functional comparison of GT2 and GT4

### 3.2.4 Adoption of GT4 in the Grid Community

During the first year of its release, the adoption of GT4 in the grid community was very limited, however, it has recently started to gain strong momentum. TeraGrid's newest production release of the Common TeraGrid Software Stack (CTSS 3) is based on GT4 [21]. Furthermore, the Open Science Grid, and the Gridbus project are using GT4 middleware. Another major grid project, EGEE, has migrated to Web services middleware called gLite, which is used by the LCG.

## 3.3 Test Grid Deployment

The test grid currently consists of ten computers. There are two GT4 metaschedulers, each of which has access to the same three clusters. This provides a layer of redundancy in the case that one metascheduler fails. The head node, or execution host, of each of the clusters acts as a single element representing the combined resources of the cluster of worker nodes attached to it, which will be the actual machines that execute jobs. The GT4 head nodes **Gridhn** and **Gridsn** each have two GT4 worker node computers in their respective clusters. The **silicon** GT4 head node, however, has been setup to use an existing GridX1 production cluster of 22 worker node computers which are using GT2. This has been implemented successfully, and provides significant evidence that non-Web services clusters can operate concurrently with Web services clusters, making the migration of one to the other a realistic process. The remaining computer in the test grid, **Ugdev05**, has been setup as a GT4 Web services based *registry*, which is running MDS4, and maintains information about the state of the resources in the grid. By pulling information from the registry, the metaschedulers can determine which resources are available, and which is best suited to run a certain job. Figure 6 gives a graphical view of this layout.

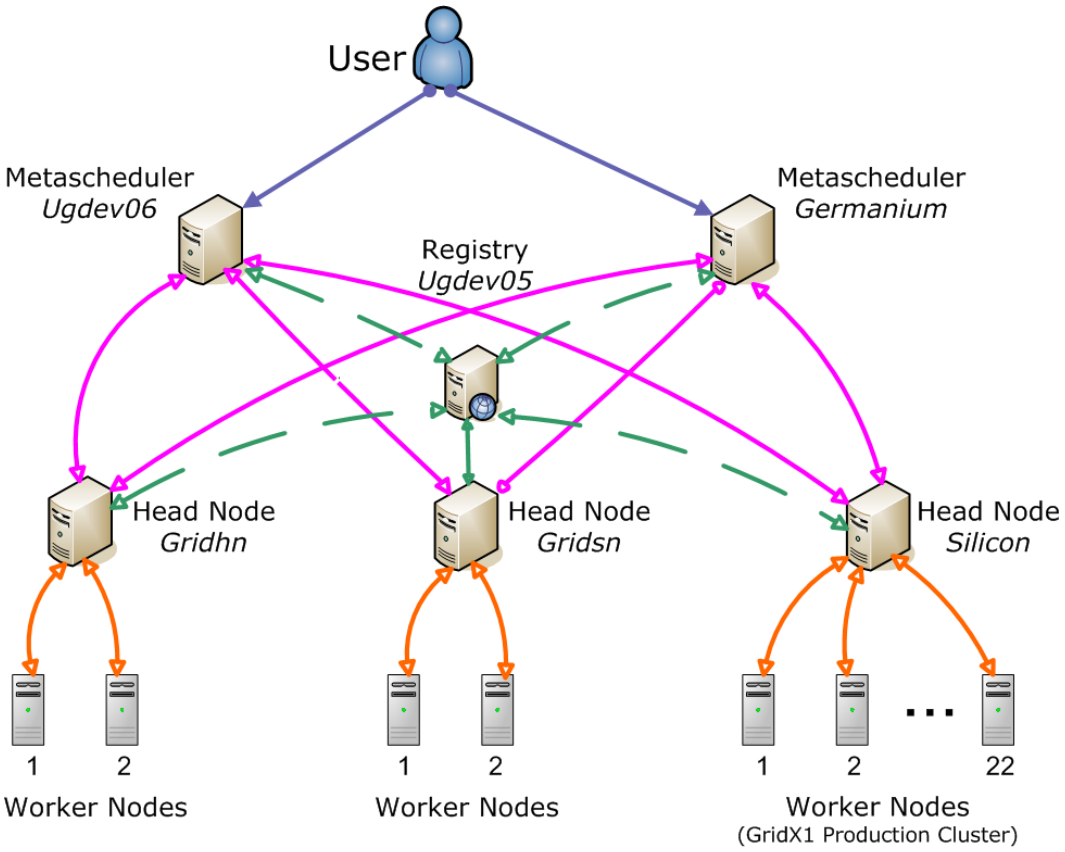


Figure 6: GridX1 Services Project Test Grid

# 4 Impact of Migration from GT2 to GT4

In this section the compatibility between GT2 and GT4 is studied, based on information gathered about the design of the toolkits, as well as results from the test grid that was deployed. The evident impacts of migrating a computational grid from GT2 to GT4 on performance, integration and scalability, security, and usability, is then outlined.

## 4.1 Compatibility

The compatibility of the core components of the Globus Toolkits is summarized as follows:

***GridFTP*** from release 2.4 and above of GT2 is entirely compatible with GT4. There are as yet no deprecated components from this release. Since nearly all current grid deployments are using at least GT2.4, the migration to GT4 should be

trivial. As previously mentioned, GT4 does add new functionality, primarily to the GridFTP server, but does not in any way break compatibility with GT2.4. Any GT2.4 client can interact with any GT4 server, and vise versa. It should be noted that implementations older than the GT2.4 release do not share this compatibility and would need to perform an upgrade [22],[13]. Experimental results from the current project verify this claim. The original GridX1 runs GT-2.4.3, and performing GridFTP file transfers with any of the GridX1-SP test grid machines running GT-4.0.2 works automatically. This is easily demonstrated in the following example:

> First, the user is authenticated to the system by initiating a proxy through the command line:
>
> **grid-proxy-init**
>
> Now, the *globus-url-copy* command is invoked, which is a simple way to perform a GridFTP file transfer from one machine to another across a network:
>
> **globus-url-copy gsiftp://ugdev06.phys.uvic.ca/home/sramage/testfile gsiftp://gcgate01.phys.uvic.ca/home/sramage/testfile2**
>
> Where *ugdev06.phys.uvic.ca* is the hostname of the source computer running GT-4.0.2, *testfile* is the name of the source file to be copied, *gcgate01.phys.uvic.ca* is the hostname of the destination computer running GT-2.4.3, and *testfile2* is the name of the destination file after the transfer is complete.
>
> The file *tester2* is successfully transferred to the destination machine.

***WS-GRAM*** is not compatible with the the older, pre-WS-GRAM, however both versions are supported in GT4. The two implementations are fundamentally different. For example, pre-WS GRAM and WS GRAM interact with their schedulers in different ways, due to the porting of functionality previously provided by Perl scripts into Web services. The format of job description files in GT4 has changed from RSL to XML, which are not compatible. At this time all pre-WS-GRAM components are maintained in GT4, with equal or better capabilities [12],[22],[13]. For grid developers who wish to run old-style GT2 jobs on a new GT4 host, the GT2 gatekeeper can be run, which will authenticate job submissions and translate them to the desired Job Manager. This compatibility was tested on the head node, *Gridhn*, which runs GT-4.0.2.

> Assuming a grid proxy has been initiated, the user can submit a GT2 (non Web services) job to a GT4 Host running a gatekeeper, by issuing at the

command line:

**globusrun -o -f date.rsl**

Where date.rsl is the GT2 style (non-XML) job submission file, which specifies that the job is to be run on *Gridhn.phys.uvic.ca*, and the executable to be run is simply *date*, which will output the current time and date.

The job is successfully executed on the head node, and the output is streamed into a file where the current time and date is verified.

An example GT2 job description file, date.rsl, is shown below:

```
+( \&
   (resourceManagerContact="gridhn.phys.uvic.ca")
   (executable = "/bin/date")
   (count = 2)
   (stdout = "/tmp/date.out")
   (stderr = "/tmp/date.err")
)
```

***WS-MDS*** , having been completely redesigned in GT4, is also fundamentally different from the MDS2 implementation in GT2. In general, while the basic concepts, goals, and functionality remains the same, the dissimilarities in standards, architecture, and configuration must be investigated by a developer planning on migrating to WS MDS. For example, the MDS2 LDAP querying system has been completely replaced with the XML Path Language (Xpath), which can query an XML database [23]. While the two versions of MDS are not functionally compatible at all, GT4 again provides full support for MDS2 to make the transition to GT4 as smooth as possible [15],[22].

***Security*** features in GT4 have been implemented with Web services, and the most important aspects, such as GSI-OpenSSH, authorization and authentication, and MyProxy are fully backwards compatible with the older versions in GT2.

***Client Applications*** written with the C common libraries in GT2 have a clearly defined migration to the new Web services enabled API in GT4. Equivalent function calls exist for most of those implemented in GT2 [22]. However, the API's are not source code compatible, and careful consideration of the differences between API's should be made when migrating.

## 4.2 Impact on Performance

Improvement in the performance of the Globus Toolkit was given high priority in the development of GT4 [13]. The important impacts on speed and reliability are outlined as follows:

***Reliability*** : The new RFT Web service is one of the most effective additions to GT4, as it provides redundancy for file transfers in the case of failure on the server and the client side, which was not the case in GT2. This gives the administrator control over the choice of secure third party host that will store the state information about file transfers. This is especially key when performing multiple file transfers simultaneously, when the impacts of failure are compounded.

MDS4 is much more robust than MDS2, with fewer components and a more simplified, automated implementation [12]. The use of XML in MDS4 means information providers are not required to have any predetermined schema, which lends to more reliable, portable grid expansion.

***Speed*** : GridFTP's server implementation in GT4 has been bench-marked with iperf, which is an effective tool for measuring TCP/UDP network bandwidth performance [24]. Results have been very positive, with the server performing at around 80% of the raw iperf on a consistent basis. This means that GridFTP is achieving 80% of the maximum theoretical limit for bandwidth performance. At this high level of performance, most of the limitations can be attributed to the disk subsystem, where Globus does not play a role [13].

Similar experiments were conducted by TeraGrid to test the throughput of a striped data transfer. Transfer to secondary storage using 64 nodes in parallel was bench-marked at 57% of the theoretical limit, attributed again solely to the limitations of the hardware. Transfer to primary storage (memory) using 32 nodes in parallel was benchmarked at a remarkable 90% of the theoretical limit [13].

Closer integration of WS-GRAM with GridFTP has deprecated the use of GASS, which in turn eliminates considerable redundant software in the core GRAM code base. This improves performance when these components are interacting [15].

The use of the Xpath query language in MDS4 is expected to have a positive impact on performance, as LDAP has more overhead due to its non-modular use of transactions [23]. Both Xpath and LDAP have high space utilization.

## 4.3 Impact on Integration and Scalability

Among the primary motivations of grid computing is the desire to facilitate the widespread integration and collaboration of resources. With the heavily open standards based implementation of GT4, the dominance of proprietary software in grid

computing is prevented, which is vital to the scalability of the grid over heterogeneous domains [9].

The abundant use of XML in the development of web services in GT4 is effective in the cross-platform deployment of a computational grid. Furthermore, the platform independent structure is ideal for the expansion of virtual organizations, as the previous obstacle of incongruity between administrative domains is overcome.

While the future of the grid looks to be driving towards a Web services standard, there remains a portion of the grid community that use pre-Web services, and indeed some of the new components of GT4 are still not Web services based. Fortunately, all such legacy services (from GT 2.4+) are included and supported. This lends to the integrability of GT4 with legacy installations. Clearly, the platform and application independent design of GT4 makes it a scalable grid middleware, realizing the needs of the expanding grid community. In contrast, the scattered, non-standards based design of GT2 does not accommodate such scalability and integrability.

## 4.4 Impact on Security

As grid deployments continue to grow, the potential for security holes and exploitations is proportionally increased. This fact is well known by the Globus Community, and it is no surprise that GT4 includes many improvements in this regard. Many of the emerging standards that GSI adheres to in GT4 are being integrated not only in grid computing, but in the Web services community as a whole [20].

Development of the new GT4 security authorization framework improves modularity in grid management, allowing for interaction between disparate platforms and applications without compromising security. The XML based SAML protocol option for credential handling plays a major role in this respect by continuing with the common theme of XML standardization seen throughout GT4.

The addition of message level security in GT4 expands the overall security capabilities of a grid. This allows administrators to have secure SOAP messages when necessary, while still providing the default transport-level security when the tradeoff of security for speed is desired.

## 4.5 Impact on Usability

***For Administrators*** : To make it easier for system administrators, Globus now uses the standard Linux makefile procedure for installing GT4 components. Previously GT2 required knowledge of the Grid Packaging Toolkit (GPT) for its

installation.

The advancements in GT4 have increased the number of services that an administrator needs to configure and maintain. For example, setting up RFT requires that the administrator manually create a database, configure it to accept TCP connections and then populate it with an RFT schema [25]. This configuration was not required in GT2. The RFT Web service does provide flexibility in administration by allowing the number of concurrent staging operations to be specified. This option was not available with GASS (GT2's analogue to RFT) [13].

Fortunately, most GT4 services require little to no configuration to function properly in a default installation. In addition, nearly all services function as a part of the Globus container (GridFTP being the notable exception), which simplifies their management.

In the administration of GT2, the lack of documentation available was often frustrating. Among the prominent advantages GT4 has over GT2 is that there is a comprehensive set of documentation is available, including separate developer, system administrator, and user guides, as well as an efficient bug-tracking system. This assists GT4 administrators in the configuration of options that are not part of the default installation, such as deploying services into non-default containers.

The higher level of standardization and documentation in GT4 has a positive impact on administration, by streamlining the installation, configuration, and maintenance of a computational grid.

***For Clients*** : The principal advantage for clients is that all of the grid services are now available as Web services, thus using a uniform paradigm to for accessing them.

The expanded documentation has a strong impact on clients. Detailed documentation is now available for creating Globus client applications, and using client tools. This means increased comprehensibility for users, facilitating the troubleshooting process, and helping to avoid many problems in the first place.

The new XML format for Job Description Files in GT4 impacts users by providing a more ubiquitous language than the previous RSL v1.0 used in GT2 [26].

Many of the attribute names between the two formats are similar. However, while RSL uses the formating style:

- `attribute_name = value`

XML uses the formatting style:

- `<attributeName>value</attributeName>`

Most of the XML equivalents to RSL attributes simply remove the underscores and distinguish between words using the lower/upper-case style seen above. For an experienced GT2 user, the transition from RSL to XML Job Descriptions should be relatively simple.

An example of a GT4 XML job description file is shown below:

```
<job>
    <executable>simonTest2</executable>
    <directory>\${GLOBUS\_USER\_HOME}</directory>
    <stdout>\${GLOBUS\_USER\_HOME}/stdout</stdout>
    <stderr>\${GLOBUS\_USER\_HOME}/stderr</stderr>
    <fileStageIn>
        <transfer>
            <sourceUrl>
                gsiftp://heplw12.phys.uvic.ca:2811 \
                /hepuser/sramage/jobdescfiles/globus/simonTest
            </sourceUrl>
            <destinationUrl>
file:///\${GLOBUS\_USER\_HOME}/simonTest2
            </destinationUrl>
        </transfer>
    </fileStageIn>

    <fileStageOut>
        <transfer>
            <sourceUrl>
                file:///\${GLOBUS\_USER\_HOME}/stdout
            </sourceUrl>
            <destinationUrl>
                gsiftp://heplw12.phys.uvic.ca:2811 \
                /hepuser/sramage/jobdescfiles/globus/globusTest.out
            </destinationUrl>
        </transfer>
    </fileStageOut>

    <fileCleanUp>
      <deletion>
        <file>file:///\${GLOBUS\_USER\_HOME}/simonTest2</file>
      </deletion>
      <deletion>
        <file>file:///\${GLOBUS\_USER\_HOME}/stdout</file>
      </deletion>
      <deletion>
        <file>file:///\${GLOBUS\_USER\_HOME}/stderr</file>
      </deletion>
</fileCleanUp>
</job>
```

The improved client comprehensibility and flexibility through the standardization and documentation in GT4 has a significant positive impact on usability.

## 4.6 GridX1-SP Test Grid Experimental Results

A test run of the GridX1-SP grid was run on August 24th, whereby a series of jobs were submitted to the in order to determine the success of the Web services implementation. Bundled in a script, this suite of jobs tested the full functionality of the end-to-end system, by running a specific set of 8 tasks:

1. Submitting a simple job using the metascheduler as the execution host
2. Submitting a simple job using a specified cluster head node as the execution host
3. Submitting a simple job allowing the metascheduler to choose a remote execution host
4. Submitting a simple job using the metascheduler as the execution host, and staging back the standard output to a file on the client machine
5. Submitting a simple job using a specified cluster head node as the execution host, and staging back the standard output to a file on the client machine
6. Submitting a job that stages in a new executable file, uses the metascheduler as the execution host, and stages back the standard output to a file on the client machine
7. Submitting a job that stages in a new executable file, uses a specified cluster head node as the execution host, and stages back the standard output to a file on the client machine
8. Submitting a job that stages in a new executable file, as well as an input file which the executable will use, uses a specified cluster head node as the execution host, and stages back output that the executable generates to a file on the client machine.

The results of running 5 tials of the script (for a total of 40 jobs), are summarized in figure 7:

| *Trial* | *Job #1* | *Job #2* | *Job #3* | *Job #4* | *Job #5* | *Job #6* | *Job #7* | *Job #8* |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Figure 7: Results of Job Submission Tests

Each of the 40 jobs ran successfully. Each fastest trial run completed in 12 minutes, the slowest in 16 minutes. The success of each job is verified in the script by checking for the output of the job on the client machine, which might for example return the hostname of an execute node, such as **gsn-wn1**. The results of one trial run of the script is shown here:

```
Please choose a metascheduler:
1 - ugdev03.phys.UVic.CA - Condor-G
2 - ugdev06.phys.UVic.CA - Condor-G
3 - ugdev07.phys.UVic.CA - Condor-G
4 - ugdev08.phys.UVic.CA - Gridway
5 - germanium.sao.nrc.ca - Condor-G
2
Please choose an execution host:
1 - gridhn.phys.UVic.CA
2 - gridsn.phys.UVic.CA
3 - silicon.sao.nrc.ca
2

Delegating Globus Credential...

Starting Globus job submission tests...
---------------------------------------
Test 01: Submitting a simple job using the MS as a resource
Waiting for output from test...
Passed Test
Output: ugdev06.phys.UVic.CA

Test 02: Submitting a simple job using the remote resource
(gridsn.phys.UVic.CA)
Waiting for output from test...
Passed Test
Output: gsn-wn1

Test 03: Test 03: Submitting a simple job allowing the MS
to choose a gt4 resource
Waiting for output from test...
```

```
Passed Test
Output: ugdev06.phys.UVic.CA

Test 04: Submitting a simple job using the MS as a resource,
staging back the stdout to a file on the client machine
Waiting for output from test...
Passed Test
Output: ugdev06.phys.UVic.CA

Test 05: Submitting a simple job using the remote resource
gridsn.phys.UVic.CA, staging back the stdout to a file on
the client machine
Waiting for output from test...
Passed Test
Output: gsn-wn1

Test 06: Submitting a job that stages in a new executable
file, and uses the MS as a resource, and stages back the
output to a file on the client machine
Waiting for output from test...
Passed Test
Output: Thu Aug 24 15:36:56 PDT 2006
ugdev06.phys.UVic.CA
Linux ugdev06.phys.UVic.CA 2.6.9-34.0.2.EL \#1 Fri Jul
7 09:57:49 CDT 2006 i686 athlon i386 GNU/Linux

Test 07: Submitting a job that stages in a new executable
file, and uses the remote resource gridsn.phys.UVic.CA,
and stages back the output to a file on the client machine
Waiting for output from test...
Passed Test
Output: Thu Aug 24 15:38:19 PDT 2006
gsn-wn1
Linux gsn-wn1 2.6.9-5.0.5.EL \#1 Tue Apr 19 14:33:20 CDT
2005 i686 athlon i386 GNU/Linux

Test 08: Submitting a job that stages in an executable
file, and an input file which it will use, then executing
on the remote resource gridsn.phys.UVic.CA, and staging
back the output to a file on the client machine
Waiting for output from test...
Passed Test
Output: gsn-wn1
```

# 5 Conclusions

Research has shown that the Web services based middleware, GT4, provides a model that satisfies the growing need for cross-platform interoperability in grids. By placing a strong focus on standards and compatibility, the Globus Alliance has created the groundwork for expanding the grid globally.

It has been shown that while the Web services components in GT4 are not directly compatible with their equivalents from GT2, GT4 has been designed with strong support for these legacy components in the interest of streamlining migration. In this document it has been made apparent that GT4 performs and scales significantly better, is more secure, and has improved usability for administrators and clients, than GT2.

Results of preliminary testing have confirmed the feasibility and stability of Web services, and it has been observed that GT2 and GT4 can coexist in a grid implementation.

This research leads to the conclusion that Web services will better enable grids to meet the challenges that cutting edge applications present.

# 6 Recommendations

To date the GridX1 Services Project has yielded very positive results on the use of Web services in a computational grid. The project is however just now maturing from the development stage into the testing stage, and the results of this final stage will provide more concrete evidence as to just how well Web services can solve the problems of modern grid computing. It is recommended that decisions on implementing Web services in production situations await these results, which should include testing the grid under heavy loads of jobs, benchmarking the performance versus a non-WS implementation, and further observation of the interoperability between GT2 and GT4.

# References


[1] World Community Grid: Active Research (2006).
http://www.worldcommunitygrid.org

[2] Bart Jacob, Grid computing: What are the key components? June 1, 2003 (Updated June 27, 2006).
http://www-128.ibm.com/developerworks/grid/library/gr-overview/

[3] Wikipedia, Parallel Computing: Embarrassingly parallel.
http://en.wikipedia.org/wiki/Embarrassingly_parallel

[4] R. Sobie, Scientific Research and the Grid: Strategies for Public Sector Transformation 2003, Victoria, September 2003.

[5] Globus Toolkit 1.1.3 System Administration Guide: The Globus Alliance, December 2000.
http://www.globus.org/toolkit/docs/1.1.3/globus_sag1.1.3.pdf

[6] Globus Toolkit TM 2.2 Overview: The Globus Alliance.
http://www.globus.org/toolkit/docs/2.2/overview.html

[7] National Research Council of Canada: Getting Started with the Globus Toolkit Overview of the Globus 2 Toolkit. Gabriel Mateescu, Aug 19, 2006.
http://rcsg- gsir.imsb-dsgi.nrc-cnrc.gc.ca/globus_tutorial/index.html#gt2_prologue

[8] Introduction to Web Services Technologies: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services, Thomas Erl. Apr 16, 2004.

[9] On the Grid: Standardizing the Grid. Lee Liming, Tom Garritano, Steve Tuecke. Clusterworld, April 2004.

[10] The Globus Alliance: Globus Toolkit, A User-Level Tutorial to Grid Programming: The Globus Hourglass.
http://www.hpcvc.lboro.ac.uk/GridWorkshop/Tuecke/img21.htm

[11] Web Services Architecture. W3C, Working Draft. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. , 2003.
http://www.w3.org/TR/2003/WD-ws-arch-20030808/

[12] A Globus Primer (Draft Version): Describing Globus Toolkit Version 4, Ian Foster.

[13] GT4: What's in it for you? Lee Liming, Ian Foster. Submitted for Publication, May 2005.

[14] The Globus Alliance: Open Grid Services Architecture.
http://www.globus.org/ogsa/

[15] Globus Toolkit 4.0 Release Manuals: The Globus Alliance.
http://www.globus.org/toolkit/docs/4.0/

[16] The Globus Alliance: GT4 Community Scheduler Framework.
http://www.globus.org/toolkit/docs/4.0/contributions/csf/

[17] The Globus Alliance: GT4 Globus Teleoperations Control Protocol.
http://www.globus.org/toolkit/docs/4.0/techpreview/gtcp/

[18] The Globus Alliance: GT4 Workspace Management Service.
http://www.globus.org/toolkit/docs/4.0/techpreview/wms/

[19] Reliable File Transfer: Lessons Learned. Bill Allcock, Ravi Madduri.

[20] Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective.
The Globus Security Team. September 12, 2005.

[21] Globus 4.0 Software and Service Testing: TeraGrid.
http://www.teragrid.org/userinfo/software/gt4.php

[22] Migrating from GT2 to GT4. The Globus Alliance.
http://www.globus.org/toolkit/docs/4.0/migration_guide_gt2.html

[23] Deciding How to Store Provenance: Kiran-Kumar Muniswamy-Reddy, Harvard University.
http://www.eecs.harvard.edu/kiran/pubs/TR-03-06.pdf

[24] End-to-End Performance: Iperf ,NLANR Distributed Applications Support Team.
http://dast.ncsa.uiuc.edu/Guides/GettingStarted/Performance.html , May 3 2005

[25] The Globus Alliance: GT4 Admin Guide.
http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch10.html

[26] The Globus Alliance: The Globus Resource Specification Language RSL v1.0
http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html