THE HANDBOOK OF COMPUTER NETWORKS

Distributed Networks, Network Planning, Control, Management, and New Trends and Applications

VOLUME 3

Hossein Bidgoli
Editor-in-Chief

Grid Computing Implementation

Barry Wilkinson, *University of North Carolina at Charlotte*
Clayton Ferner, *University of North Carolina at Wilmington*

INTRODUCTION

This chapter addresses Grid computing implementation that was introduced in Chapter 139. Grid computing allows people to use geographically distributed, networked computers and resources collectively to achieve high-performance computing and resource sharing. The implementation of grids has been evolving since the mid-1990s. Early Grid computing work in the early to mid-1990s focused on customized solutions—that is, projects developed in-house software to achieve the interconnections and Grid computing faculties. The early experiences provided the impetus for later standardized work. One of the most important technical aspects of Grid computing is the use of standard Internet protocols and open standards. The use of standardized technologies is critical to the wide adoption of Grid computing. It has now been recognized that XML and Web services provide a very flexible standard way of implementing Grid computing components. We will begin by briefly outlining Web services, XML, and Web Services Description Language (WSDL). With that basis, we can then describe the components of a Grid computing infrastructure.

GRID COMPUTING INFRASTRUCTURE

Web Services

Web services are software components designed to provide specific operations ("services") that are accessible using standard Internet technology. They are usually addressed by a uniform resource locator (URL). A URL is a string of characters that is used to identify a resource on the World Wide Web. URLs conform to a defined syntax and include the protocol used to access the resource. For example www.cs.uncc.edu is the URL of the main computer science page at UNC-C, to be accessed by hypertext transfer protocol (HTTP). Web services are designed to be accessed over a network by other programs. The Web service itself may be located anywhere that can be reached on the Internet. A Web service can be used as a front end to an application to allow the application to be accessed remotely through the Web service.

Web services build on previous distributed computing concepts. An early form of distributed computing was the remote procedure call (RPC) introduced in the 1980s. This model allows a local program to execute a procedure on a remote computer and then get back results from that procedure. Later forms of distributed computing introduced distributed objects, for example CORBA (Common Object Request Broker Architecture) and Java RMI (remote method invocation). A fundamental requirement of all forms of RPCs is the need for the calling programs to know details of accessing the remote procedure. Procedures have input and output parameters with specific meanings and types. The specific calling details need to be known by the calling program. These details can be described using an interface description language (IDL). Web services use an XML-based IDL, which provides a very flexible and elegant solution, as we shall see.

RPCs introduced an important concept, that of a client-server model with a service registry, which is retained for Web services. In a client/server model, the client accesses the server for a particular operation. The server responds accordingly. For this to happen, the client needs to identify the location of the required service and know how to communicate with the service to get it to provide the actions required. You can achieve these two things by using a service registry, which is usually a third party, in a structure now known as a service-oriented architecture (SOA), as shown in Figure 1. The sequence of events then is as follows: First, the server (service provider) "publishes" its service(s) in a service registry. Second, the client (service requestor) can ask the service registry to locate the service. Third, the client (service requestor) binds with the service provider to invoke the service. An interface description language (IDL) can be used to describe the service interface.

Key aspects of using Web services in a client-registry-service model is the use of XML and Internet protocols to make the arrangement completely platform-independent and nonproprietary. Let us first consider XML and then the XML language used to describe the Web service functionality, called WSDL. Then we will discuss the

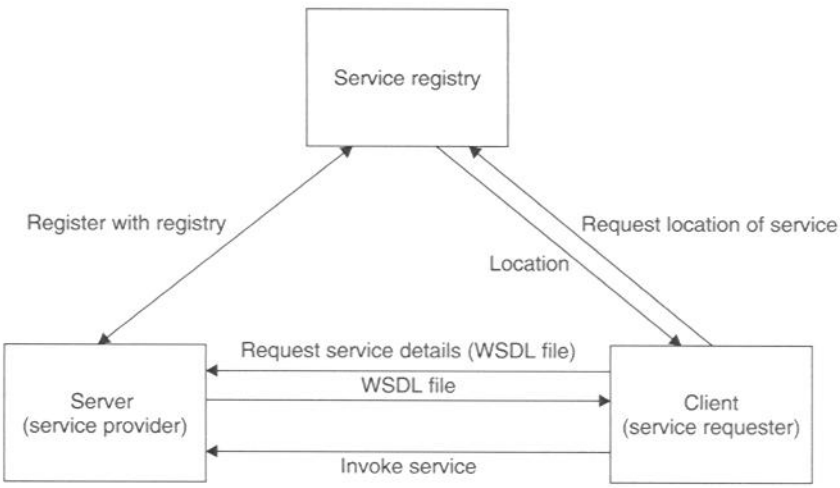

Figure 1: Service-oriented architecture

packaging communication protocols used to transmit the XML documents, known as SOAP.

XML (Extensible Mark-up Language)

Mark-up languages provide a way of describing information in a document such that the information can be recognized and extracted. The Standard Generalized Mark-Up Language (SGML) is a specification for a mark-up language ratified in 1986. A key aspect of SGML is using pairs of named tags that surround information (the body), a begin tag `<tag_name>` and a matching end tag `</tag_name>`, where `tag_name` is the name of the tag. Tags can be nested.

Hypertext Markup Language (HTML) is a subsequent mark-up language using a similar approach but specifically designed for Web pages. "Hypertext" refers to the text's ability to link to other documents. "Markup" refers to providing information that tells the browser how to display the page and other things. The meanings of tags in HTML are predefined—for example, `<B>` to start bold text and `</B>` to end bold text. Certain tags in HTML do not necessarily require end tags, such as `<P>` alone, which indicates the start of a new paragraph. Many tags can have attributes that specify something about the information between tag pair. For example, `<FONT FACE=Times COLOR=red>` specifies that the text in the body will be red and in Times font. The concept of attributes is very important and is used extensively in XML.

XML is very important standard mark-up language; it is a "simplified" SGML, ratified in 1998. XML was developed to represent textual information in a structured manner that could be read and interpreted by a computer. XML is a foundation for Web services, and Web services now form the basis of Grid computing. Two key aspects of XML are:

- Names of tags and attributes are not predefined as they are in HTML. Tags can be defined broadly at will but must be defined somewhere and associated with the document that is using the named tags.
- Tags are always used in pairs delineating information to make it easy to process. (There is an exception to this rule when the body between the tags is empty, in which case the opening and closing tags may be combined into a single tag of the form `<tagName/>`.)

Tag names and meanings can be created to suit the application and become a specific XML language. As such, an infinite number of XML languages could be invented, and many have already been.

Namespace Mechanism

If XML documents are combined, there is the problem of ambiguity if different documents use the same tag names for different purposes. The namespace mechanism provides a way of distinguishing tags with the same name, and is used widely within XML documents. With the namespace mechanism, tag names are given an additional namespace identifier to qualify it. The fully qualified name is given by namespace identifier plus the name used in the document. The namespace identifier used is a uniform resource identifier (URI). A URI is a string of characters used to identify a resource. An extension of URIs to cover international language symbols is called an international resource identifier (IRI). Typically, a namespace uses a URL. (URLs are a subset of URIs and include the protocol used to access the resource.) Even though the namespace may be a URL, the URL is only used as a means of distinguishing tag names and solve any ambiguity. The URL does not need to exist. However, typically one would place a read-me document at the location if a URL is used. This practice has been formalized into providing a Resource Directory Description Language (RDDL) document.

Rather than simply concatenate the namespace identifier with the local name throughout the document, the names in the document are given a prefix separated from the name with a colon. The association of namespace identifier and prefix is done using the `xmls` attribute in the root element, in which the qualifying name is the attribute's value. There can be more than one namespace attribute, each associated with a different prefix, and there will also be a default namespace defined for names without a prefix.

Schemas

So far, we have not said which tags are legal in a document and how the tags are associated with a particular meaning and use. One way to define which tags are legal in a particular XML document is to describe them within the document in the document type definitions (DTD). However, this approach has serious limitations in terms of system integration and flexibility and is not currently used for Web services. In fact, they are not allowed in SOAP messages (the messaging protocol used for XML documents; see below). An alternative and much more powerful approach is to define legal tags in another XML document, called a "schema," and associate that schema document with the content XML document, either explicitly or by common agreement between the parties. XML schemas, also expressed in XML, provide a flexible way of handling legal element names and have the notation of data types. This approach also leads to different XML languages, each with its own schema. Once a schema exists for a particular XML language, it can be associated with XML documents, or *instances* of possible documents using the schema.

Since a schema is also an XML language, there should also be a schema for schemas. The schema for the XML schema is predefined and specified in the XSD (XML schema definition) language. More details on XML schemas can be found at the W3C website (W3C Architecture Domain XML Schema, n.d.).

SOAP

SOAP is a communication protocol for passing XML documents, standardized by the W3C organization (World Wide Web Consortium). Originally, SOAP stood for simple object access protocol. However, the spelled-out version has since been dropped because this name was not accurate; specifically the protocol does not involve object access. The draft W3C specification (Gudgin et al. 2003) describes SOAP as follows:

> SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

SOAP provides mechanisms for defining the communication unit (a SOAP message), the data representation, error handling, and other features. SOAP messages are transported using standard Internet protocols, most likely HTTP.

Web Service Definition Language

In the Web services approach, there needs to be a way of generally and formally describing a service, what it does, how it is accessed, etc. in an IDL. The World Wide Web Consortium (W3C) has published an XML standard for describing Web services called Web Service Description Language (WSDL). WSDL version 1.1 was introduced in 2001. WSDL version 2 was introduced in 2004 and made a number of clarifying changes.

A WSDL document describes three fundamental properties of a service:

- What it is—operations (methods) it provides
- How it is accessed—data format, protocols
- Where it is located—protocol specific network address

The abstract definition of the operations is contained in the element called `interface` (called a portType in WSDL version 1.1). This element contains elements called `operation` that specify the types of message sent and received (`input` and `output` elements). These operation elements loosely correspond to a method prototype. The messages are further defined in terms of data types being passed in the type element. The `binding` element describes how to access the service—that is, how the elements in abstract interfaces are converted in actual data representations and protocols (e.g. SOAP over HTTP). In WSDL version 2, the `service` element described where to find the service. The `service` element contains the `endpoint` element, which provides the name of the end point and physical address (URL). (The `service` element was defined differently in WSDL version 1.1. The `port` elements describe the location of a service and the `service` element defined a named collection of ports.) More information on WSDL version 2 is given by Booth and Liu (2005).

Putting It Together

The basic parts of a service-oriented architecture are the service provider (server), service requestor (client), and service registry. Web services can use a UDDI (Universal Description, Discovery and Integration) registry, which itself is a Web service. UDDI is a discovery mechanism for Web services and provides a specification for modeling information targeted primarily toward business applications of Web services. For Grid computing applications, other ways of forming service information for discovery are available including using WS-Inspection Language (WSIL) (IBM 2001). After a registry is populated with Web service entries, the client can access the registry to find out whether the desired Web service exists in the registry and if so where it is located. The registry responds with the identification of the server capable of satisfying the needs of the client. Then, the client can access the server for the Web service interface. The server responds with a WSDL document describing the service and how to access it. The client can then send the Web service a request for an operation. The result of the operation is returned in a message from the Web service. All messages in this architecture are SOAP messages. It would be feasible for the registry to return the WDSL interface document and then with this in hand, the client could make a request immediately to the Web service without asking for its WSDL interface document.

Note that the registry itself has to be known both to the client and the service provider. Of course, a registry is

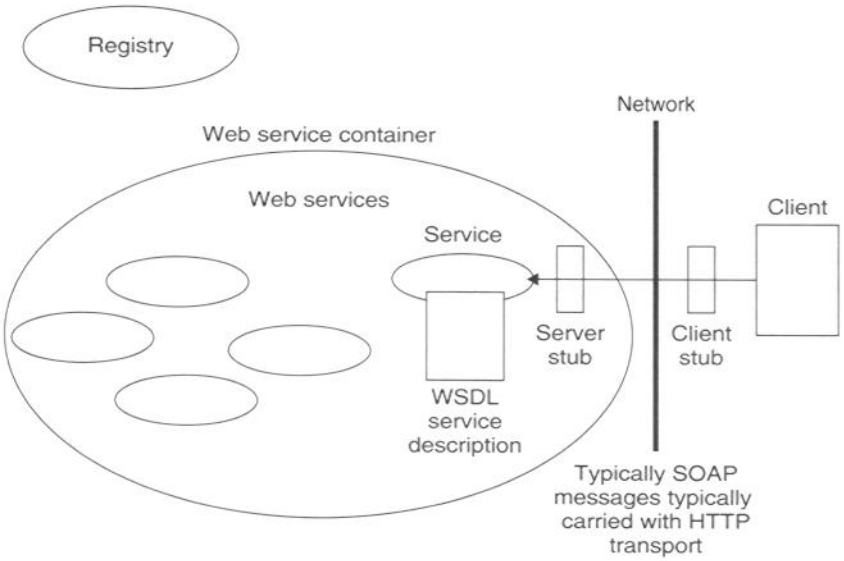

Figure 2: Web services environment

unnecessary if the client knows about the Web service already and the location of the Web service never changes. However, in a dynamic and distributed Grid computing environment, a registry or information service is necessary. Now let us look at how to implement a Web service and client. Web services are generally "hosted" in a Web service container—that is, a software environment that provides the communication mechanisms to and from the Web services. Web services are deployed within such an environment. There are several possible environments that are designed for Web services, notably, Apache Axis (Apache eXtensible Interaction System), IBM Websphere, and Microsoft .NET. The J2EE (Java 2 Enterprise Edition) server container is also a candidate for hosting Web services especially in enterprise (business) applications. Apache Axis requires an application server. It can be installed on top of a servlet engine such as Apache Jakarta Tomcat. However, it could be installed on top of a fully fledged J2EE server.

For the implementation, it is convenient to connect the service code to the client through two Java classes that act as intermediaries, one at the client end called a "client stub" (also called a "client proxy") and one at the service end called a "server stub" (also called a "skeleton"). These stubs provide a structured way to handle the messaging and different client and server implementations. Interaction is between the client and its stub, between the client stub and the server stub, and between the server and its server stub. The interaction between the stubs is across the network using SOAP messages. The client stub is responsible for taking a request from the client and converting the request into a form suitable for transmission, which is called "marshaling." The client stub is also responsible for receiving responses on the network and converting to a suitable form for the client. The server stub is responsible for receiving a SOAP request from the client stub and converting it into a suitable form for the service, called "unmarshaling." The server stub also converts the response from the service into a SOAP message for the client stub. The resulting Web service environment is shown in Figure 2.

Web services deployment descriptor (WSDD) is an XML language used to specify how to deploy a Web service. More details of Web services and their deployment can be found in Graham et al. (2005).

Grid Computing Standards

Early Grid Computing Standards

Although Web services are very attractive as a basis for creating a grid infrastructure, Grid computing requires some way of representing state—that is, values that persist from one invocation of the service to the next. Pure Web services do not have state. Another feature needed in Grid computing is the ability to make a service transient—that is, to be able to create and destroy a service. Typically, transient services are created by specific clients and do not outlive their clients. Web services are usually thought of as nontransient and do not have the concept of service creation and destruction. Hence, some changes are needed to the Web services approach if it is to be adapted to Grid computing.

The Global Grid Forum (GGF), in developing standards for Grid computing, focused on using Web services and originally developed two interrelated standards called Open Grid Services Architecture (OGSA) and Open Grid Services Infrastructure (OGSI). The term *Grid service* was introduced as the extended Web service that conforms to the OGSI standard. OGSA defines standard mechanisms for creating, naming, and discovering Grid services and deals with architectural issues to make interoperable grid services. OGSA is described in the seminal paper "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration" by Foster, Kesselman, Nick, and Tuecke (2002). OGSI specifies the way that clients interact with grid services (that is, service invocation, management of data, security mechanism, etc.). These standards appeared in the early 2000s and

were implemented in the Globus Toolkit version 3 in 2003. In OGSI, the Web service model was extended to enable state to be implemented. An extension of WSDL was invented called Grid Web Service Description Language (GWSDL) to support the extra features in Grid services not present in Web services. However, OGSI had a very short life. The Grid community at large did not embrace OGSI, which was seen by many as too complex. The push was made to align OGSA more closely with Web services so that existing Web service tools could be used. This led to the Web Services Resource Framework, which is discussed below.

Web Services Resource Framework

Web Services Resource Framework (WSRF) presents a way of representing state while still using the basic WSDL for both Web services applications in general and Grid computing in particular. Instead of modifying the WSDL to handle state, the state is embodied in a separate resource. Then, the Web service acts as a front end to that resource as illustrated in Figure 3. The resource still needs to be described in the WSDL file, which is achieved in a resources properties section of the WSDL file. The combination of a Web service and resource in this framework is called a WS-Resource. The resource in this description can mean anything that has state and requires access, such as a database, or variables of a Web service that need to be retained between accesses.

The WSRF specification calls for a set of six Web services specifications:

- WS-ResourceProperties
- WS-ResourceLifetime
- WS-Notification
- WS-RenewableReferences
- WS-ServiceGroup
- WS-BaseFaults

WS-ResourceProperties describes how resources are defined and accessed. Other aspects of WSRF deal with using multiple resources, resource lifetime, and how to obtain notification of changes (WS-Notification).

Web services are traditionally addressed simply by a URL, but this does not provide much functionality. A specification called WS-Addressing has been introduced to provide more than just the URL, and it is particularly relevant to WSRF, as it provides a way of specifying resources as well as the Web service. WS-Addressing introduced a construct called an end-point reference (EPR). An end point is the destination where the service can be accessed. An end-point reference is a structure that provides the end point address as a URI (more recently as an IRI). The structure also provides other, optional information, consisting of reference parameters and metadata. The metadata include information about the behavior, capabilities, and policies of the end point.

According to the W3C document "Web Service Addressing 1.0 Core" (Gudgin and Hadley 2005), the reference parameters are "namespace qualified element information items that are required to properly interact with the endpoint." For WSRF, the reference parameter feature is used to convey the identity of the resource. (WS-Addressing originally included a parameter named "reference properties" that was used.) The end-point reference is then called "WS-Resource-qualified" (Czajkowski et al. 2004). The resource identity can simply be an assigned number or a character string. It is called a "key" by Ananthakrishnan et al. (2005). The end-point reference typically would be returned when the WS-Resource is created and can be used to identify it for access and destruction.

Software Components for Grid Computing

The Globus project (Foster 2005; The Globus Toolkit 2006) provides reference implementations for Grid computing standards and is a de facto standard itself. The approach taken by Globus is to provide a toolkit of component parts, which can be used separately or, more likely, collectively for creating a grid infrastructure. The Globus Toolkit has gone through four versions from the late 1990s to 2005. Many of the ideas embodied in Globus were present from version 1 of Globus, including job submission and security mechanisms. Globus version 2 describes the basic structural components within a grid security environment as three pillars:

- Resource Management
- Data Management
- Information Services

This division of parts essentially remains in the current release. The principal task of the resource management

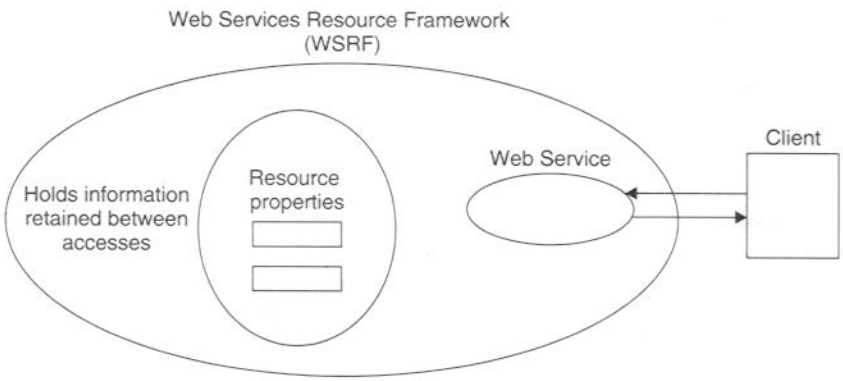

Figure 3: Web Services Resource Framework

component, GRAM (Grid Resource Allocation Manager), is to submit jobs. These jobs are submitted to local resources via a local scheduler. GRAM provides an interface to local schedulers such as load sharing facility (LSF) (Platform 2006), Portable Batch System (PBS) (Altair Grid Technologies 2006), Sun Grid Engine (SGE) (Sun N1 Grid Engine 6 2006), and Condor (Thain, Tannenbaum, and Livny 2003), which then pass the job onto the computer resources. Jobs often need data files, and the data management component provides the ability to transfer files to the required places. The key component is the grid version of FTP called GridFTP. Grid Information Services (GIS), as the name suggests, provide information about the grid infrastructure. The information directory service in Globus version 2 is called Metacomputing Directory Service (MDS), which enables Lightweight Directory Access Protocol (LDAP)–based information structures to be constructed. Globus version 2 Grid Resource Information Service (GRIS) provided information on the computational resources (configuration, capabilities, status). The Grid Index Information Service (GIIS) was also provided for pulling together information.

Globus version 3 retains the basic components of version 2 but is implemented using OGSA/OGSI standards. Globus version 4, introduced in 2005, builds on previous versions and implements the components using the WSRF standard. Note, however, that versions are not backward-compatible, as the standards they followed are not compatible; although some pre-Web-services components remain in later Web-service versions. Also note that even in the WSRF Globus version 4, for efficiency reasons, that some functionalities are not Web services. For example, GridFTP is not a Web service, although there is a Web-service front-end available (reliable file transfer [RFT] service, see below). A common feature of all versions (1, 2, 3, and 4) is the use of the public key infrastructure (PKI) for security. Security is considered in detail in the following section.

GRID SECURITY

Basic Security Concepts

It is clearly necessary in the distributed structure of a Grid infrastructure to secure communication and secure access to resources to stop unauthorized access and tampering. Security is also required in important transactions on the Internet, and hence similar mechanisms that are used there can be used in Grid computing, following the strategy that standard Internet protocols should be used for widespread adoption of Grid computing. There are some special additional security requirements for Grid computing. First, secure communication is needed not only between users but also between users and resources, such as computers, and between the resources themselves. Second, specifically related to communication between resources themselves, it is necessary to delegate the user's authority to programs to act on the user's behalf, including at remote sites, while preferably requiring the user to sign on only once (single sign-on). Before developing the full solution to these factors, let us first consider general security concepts.

Two critically important security concepts are:

- *Authentication*—The process of deciding whether a particular identity is who he, she, or it says he, she, or it is (applies to humans and systems)
- *Authorization*—The process of deciding whether a particular identity can access a particular resource, including whether the specific type of access is allowed. This latter aspect is commonly called Access control.

The traditional way to authenticate users is for each to have a username and password. Users who wish to be authenticated enter their username and password, which are sent to a server through the network. Upon receipt, the server validates the username and password and responds accordingly. There are two aspects to make such password-based authentication workable:

- Information needs to be sent in a form that is unintelligible except to the parties involved, otherwise the username and password could be stolen.
- The identity of the sender has to be proved in some fashion.

The term *data confidentiality* is used to describe the protection of the information exchange from eavesdroppers, and this is done by scrambling the binary patterns in a process called "encryption." There are many algorithms that could be used to encrypt data. Usually, the algorithm is not kept secret. Instead a number used in the algorithm is kept secret. This number is called a "key." To make it difficult to discover the key, the key chosen is a very large number, typically 128 bits or more. In secret key cryptography or symmetric key cryptography, the same key is used to encrypt the data as to decrypt it. In that case, both the sender and the receiver need to know the key, but the key has to be kept secret from everyone else.

It is possible to devise an algorithm that uses two keys, one to encrypt the data and one to decrypt it. This is known as "asymmetric key cryptography." In this form of cryptography, one key, called the "public key," is made known to everyone, while the other key, called the "private key," is known only to the owner. Data encrypted with the receiver's public key can be decrypted only by the receiver using his or her private key. Asymmetric key cryptography is also called "public key cryptography" because of the use of one key being made publicly available. In asymmetric cryptography, there is no known practical way of discovering one key from the other key. However, it is slower to encrypt/decrypt data using public key cryptography than using secret key cryptography. In addition, if you were to encrypt data using a receiver's public key, the receiver cannot be sure of the identity of the sender, as everyone has access to the public key. So mechanisms need to be in place to confirm identities.

Data integrity is the name given to ensuring that data were not modified in transit (either intentionally or by accident). Data confidentiality—that is, protecting the data from eavesdroppers—is obtained by encrypting the data. To achieve data integrity, a binary pattern called a "digest" is attached to the data, computed from the data using a hash function. The digest is different

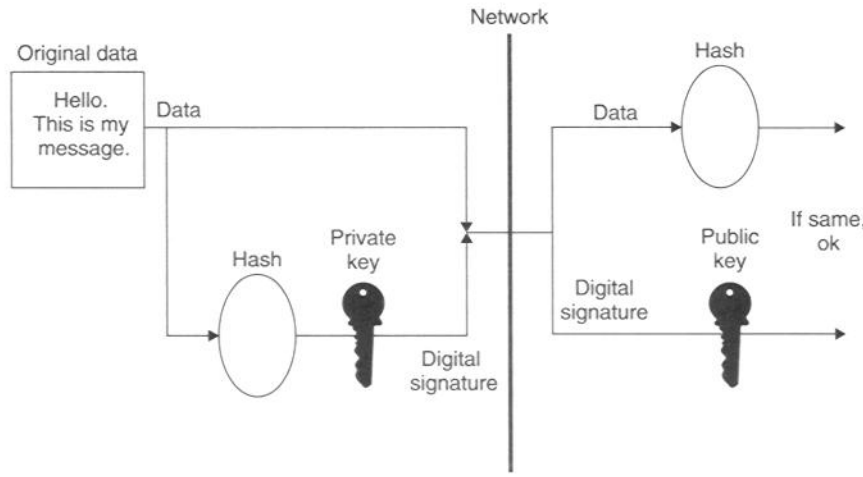

Figure 4: Checking digital signatures

if the data have been altered in transit. To achieve both data integrity and authentication, the digest is encrypted with the sender's private key to create a digital signature, which is attached to the data instead of the digest itself. The receiver can check the digital signature by decrypting the signature (with the sender's public key) and comparing the result with the digest created separately from the received data, as illustrated in Figure 4. Note that if data can be decrypted with the sender's public key, it can only have been encrypted with the sender's private key and hence by the sender.

However, digital signatures alone are not sufficient to ensure that the data are truly from the sender. It is possible that the public key is a fake—i.e. from an attacker instead of from the trusted sender. To cover this possibility, users are issued certificates, which are digital documents listing the user's specific public key. A trusted third party called a "certificate authority" (CA) certifies that the public key does in fact belong to the user named on the certificate.

Certificates are comparable to driver's licenses and passports as a form of identity. The most widely used certificate format is the X.509. Version 1 of this format was defined in 1988 by International Telecommunications Union (ITU). Versions 2 and 3 added some fields. Information provided in the certificate includes the name and digital signature of the issuer (certificate authority), algorithm used for signature, name of the subject (user), subject's public key, public key algorithm used, and validity period. Names have to be unique and recognizable. To this end, the X.500 distinguished name format is used. The X.500 naming format is a hierarchical list of attributes that is intended to make the name globally unique. An example of a distinguished name is: `/C=us/O=University of North Carolina at Charlotte/OU=Computer Science/CN=Barry Wilkinson`, where `C` indicates country, `O` the organization, `OU` the organizational unit, and `CN` the common name. Another possible attribute is `/L` for location. How names are constructed must be agreed on by all parties and is part of written certificate policies. There are obviously many possibilities, even within the constraints of the X.500 distinguished name format, from highly hierarchical to almost flat.

To obtain a signed certificate for themselves, users first contact a certificate authority. Generally, the user generates his or her own public/private key pair and sends the public key to the certificate authority, keeping the private key in a very secure place. The certificate authority returns a signed certificate. The certificate authority has to be given some means of proving the identity of the user, which may require some off-line procedure in which humans communicate—i.e. the user communicates with the manager of the certificate authority. Once the user has a signed certificate, the certificate can be sent with the data to other parties. Alternatively, the data can be sent without the sender's certificate and the receiver retrieves the sender's certificate from a public place. Either way, the receiver of a user certificate can verify the certificate by verifying the CA's signature. Then, it can trust the sender's public key contained in the certificate. It is necessary for the receiver to have the certificate authority's public key, which is obtained from the certificate authority's own certificate. (The certificate authority signs its own certificate.) This process works if one can trust the certificate authority and its public key.

There are several protocols that use the above PKI, the most notable being SSL (secure sockets layer), which can be added on top of protocols such as HTTP and FTP. The SSL protocol involves exchanging a sequence of messages that includes randomly generated numbers, and provides for mutual authentication, although in many Web applications the user is not authenticated, only the server. Commercial certificate authorities exist, such as Verisign and Entrust Technologies. Web browsers have built-in recognition for such trusted certificate authorities to allow SSL and other secure connections.

Certificates have a valid time period specified on the certificate defined by "not before" and "not after" parameters. Typically, the validity period set is quite long, say one year or five years. It is critically important to maintain the private key very securely, usually on your local computer encrypted using a password. The private key will be used to encrypt data that are to be decrypted with your public key and to decrypt data that was encrypted with your public key.

At the beginning of this section, we differentiated between authentication (the process of deciding whether a particular identity is who he says he is), and authorization (the process of deciding whether a particular identity can access a particular resource). A simple authorization mechanism is the access control list found in UNIX/Linux based systems and also in Windows systems. An access control list is a table listing the users and groups of users allowed to access particular resources and what type of access is allowed. In a UNIX/Linux-based system, for example, access to files and directories are controlled by an access control list. The access control list concept can be carried over to Grid computing systems, although the distributed nature and different administrative domains of resources of a grid make it desirable to have further mechanisms, which we shall discuss in the next section.

Grid Security Infrastructure

Grid Security Infrastructure (GSI) is the Grid computing security that is implemented in Globus; it is a GGF standard. GSI uses PKI and the SSL protocol for mutual authentication. More recently, WS-Security can be used, which is an extension to SOAP messaging for security. Grid computing projects usually form their own certificate authorities. The Globus Toolkit provides an implementation of a certificate authority called "simpleCA" that can be used for small projects. We shall discuss CAs for larger projects later. First let us consider additional security features required for Grid computing.

Delegation

Delegation is the process of giving authority to another identity (usually a computer or process) to act on one's behalf. Implicit in delegation is the concept of single sign-on, which enables users and its agents to acquire additional resources without repeated authentication. Usually, the user's private key is password-protected (encrypted with a password) and each time a user has to access the private key to perform a security-related operation, the user would need to type in the password. Single sign-on avoids this practice. Delegation is achieved by the use of additional certificates called "proxy certificates" (loosely called "proxies"). Proxy certificates are signed by the user (or the proxy entities themselves in a chain of trust) rather than by the certificate authority. Proxy certificates are created with new public and private keys and the user's name with the attribute `/CN=proxy` added to the name. The private key of the proxy is not password-protected, only file-system-protected in the user's file system. (If it were password-protected, it would defeat the purpose of reducing the need for typing passwords). To combat this insecurity, the validity period of proxy certificates is set to be short, say 12 hours, and, of course, not past the validity of the original user certificate from which it is based.

Suppose the user wants to delegate his authority to a third party to act on his behalf. This third party requires a proxy certificate signed by the user and she makes a request to the user for a signed proxy certificate, just as the user asked a certificate authority for a certificate for himself. Acting as his own proxy certificate authority, the user returns a proxy certificate signed by himself. The user also sends her own certificate. The third party requires the user's certificate to be able to validate to the proxy certificate, just as the CA's certificate is required to validate the certificate. As described by Ferreira et al. (2004), both the certificate and the proxy certificate are checked for the user's name (the proxy with the added proxy attribute). Once all validation is done, the third party can issue requests to others on the user's behalf, using the proxy's private key to encrypt messages.

The delegation process can also be chained—that is, the third party having the proxy can generate a proxy to another party using the same procedure as described for the user giving its proxy. In that case, the proxy issuing its proxy certificate signs it. Proxy certificates are part of the Globus GSI and are proposed as a standard (GT 4.0 Security: Key Concepts, n.d.). Proxies are used for delegation locally as well as remotely. Creating a proxy is one of the first actions a user must do when using Globus with the command `grid-proxy-init`.

Credential Management

A certificate and corresponding private key are collectively called the "credentials" (i.e. user credentials, proxy credentials, host credentials, etc.). However, the private key is available only to the owner, and the word *credentials* is used quite loosely as not including the private key, which should not be transferred under any circumstances. It is convenient to have a central credential repository to store proxy credentials, which then can be accessed as required rather than having to maintain the credentials in several places. Globus version 4 provides a proxy credential repository called MyProxy (MyProxy Credential Management Service 2006). MyProxy is accessed by other components to store, retrieve, and renew proxy credentials. These components include grid portals and job managers, as described later.

Authorization

To connect to a remote site, you need permission (authorization) and an account. The simplest authorization mechanism is a form of access control list called a "grid-map file." The grid-map file is a file maintained at the site and holds list of user's distinguished name (as given on their certificates) and their corresponding local account name. An entry might be: `"C=us/O=University of North Carolina at Charlotte/OU=Computer Science/CN=Barry Wilkinson" abw` where `abw` is the local username. The user's local access rights apply. It is allowable to map more than one user to a single local account if desired—i.e., to have a group working with a single account.

The grid-map file approach, although straightforward and implemented in GT 4.0, does not scale well. It requires each resource in the grid to maintain a separate grid-map file containing entries for all those expected to use the site. For a system with more than two or three sites, this becomes unmanageable.

Large-Scale Grid Computing Security Infrastructure

There are many issues in creating a large-scale Grid computing infrastructure. In the previous section, we saw

that the traditional access control list, although feasible for a small system, is unworkable for a large system. Grid computing is all about large-scale geographically distributed infrastructures. Many of the problems for large-scale structures are still open research problems with many potential solutions being suggested. Building a large-scale grid system around the basic tools of GSI will require additional scalable tools.

Authentication

Rather than have a single certificate authority, one might choose to use multiple certificate authorities, perhaps one at each major site in the Grid. This was done for an undergraduate Grid computing course taught across North Carolina (Wilkinson and Ferner 2005). Each user receives a certificate issued by one of the certificate authorities. When the certificate is submitted to another site for authentication, it is necessary for that site to trust the issuing certificate authority as illustrated in Figure 5. Globus provides for the configuration of trusting multiple certificate authorities. The certificate of the certificate authority and a configuration file defining the distinguished names of the certificates signed by the certificate authority are simply loaded in the trusted certificate directory of Globus. Each Globus installation would need these two files for each certificate authority to be trusted. It is clear that this approach is not scalable or flexible for a grid that might grow or change in configuration. However, maintaining a list of trusted certificate authorities is the way a Web browser handles trusted certificate authorities.

PKI trust can be chained. Suppose A receives B's certificate for validation, and B's certificate is signed by a certificate authority C, whom A does not immediately trust. If A is also given C's certificate, which is signed by certificate authority D, whom A does trust, then A can obtain the public key of B's certificate authority and trust it. One could construct a hierarchical, tree-like certificate authority structure, with a single root certificate authority that everyone trusts. This root certificate authority signs certificates of certificate authorities below it, which themselves can sign certificates of certificate authorities below and so on until a certificate authority is reached that signs users' certificates. If A wishes to accept B's certificate, it needs to work back from B's certificate to the root certificate authority that it trusts. A certification path is established that has to be navigated. A single-root certificate authority has the disadvantage that if the root is compromised, the whole system is compromised. However, the tree structure may be convenient in some organizations having a similar physical hierarchical structure.

There are other certificate authority configurations using the feature that certificate authorities can be cross-certified. Here, a pair of certificate authorities sign each other's certificates. Usually, a user trusts the certificate authority that signs its own certificate. If a group of certificate authorities are cross-certified in pairs and there is a path between the certificate authority of A and the certificate authority of B, one can establish trust between A and B. This avoids the single trust point of a tree structure and also enables arbitrary PKIs to be formed.

If more than one PKI already exists, either a tree or some arbitrary network, a *bridge certificate authority* may be attractive, which forms a trust between one certificate authority in one PKI and one certificate authority in another PKI. Each such certificate authority is cross-certificated with the bridge. The advantages of a bridge configuration is that it reduces the number of cross certificates from $O(N^2)$ to $O(N)$, where N is the number of CAs in the configuration, and the whole configuration is not compromised if a single CA is compromised. An example of a grid using bridge cross-certification is the SURA-Grid (SURA NMI Testbed Grid PKI Bridge Certification Authority, n.d.).

Authorization

Scalable authorization structures are also needed in larger grids. The Communication Authorization Service (CAS) is a component, available in Globus versions 3 and 4, designed to handle the authorization of many distributed users and many resources. CAS maintains information on what rights the grid community grants to users.

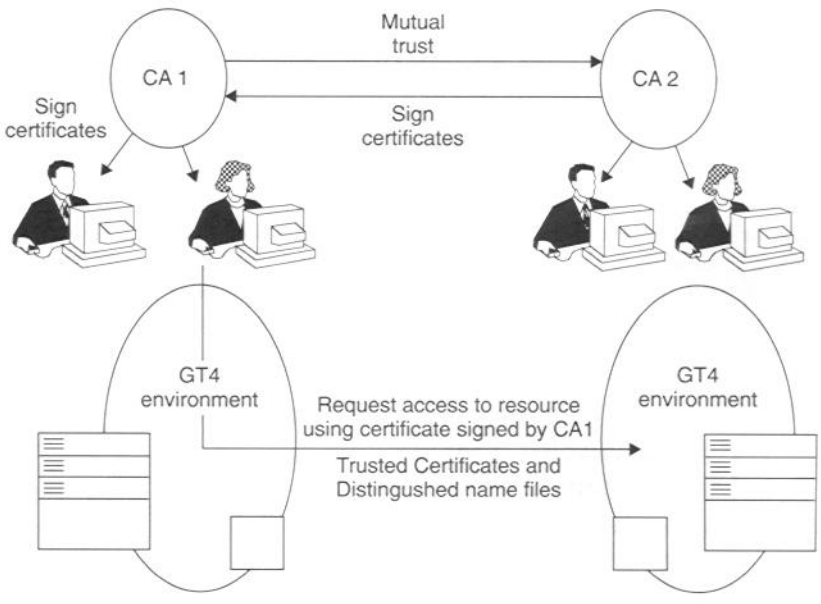

Figure 5: Certificate authorities with mutual trust

The user first asks CAS for permission to use a resource. CAS gives the user proxy credentials with specific rights—that is, a policy statement signed by CAS. The user then presents these credentials to the resource. The resource trusts credentials signed by CAS and accepts the request if the policy statement contained in the credentials authorizes the user for the specific request.

VOMS (Virtual Organization Membership Service) is another system for granting authorization to access resources in a virtual organization. In the VOMS system, the user is assigned membership of groups with roles and capabilities. This information is contained in the proxy certificates returned to the user. Interestingly, this project started by facing the difficulties of using grid-map files but does provide an automatic way of creating grid-map files. Another system that can create grid-map files locally is GUMS (Grid User Management System 2006). In GUMS, the grid-map file can be created statically or dynamically to jobs by the GUMS server.

Organizations will already have a local security system in place for their networks and computers, with which a grid security system may have to interface. One such system, created by MIT, is Kerberos, which provides a network authentication protocol. Kerberos uses tickets, which are electronic credentials used to verify your identity and provide specific access rights. Users first receive a ticket-granting ticket encrypted with their password. If they can decrypt this ticket, they have proved their identity and can obtain additional tickets for specific access rights. Kerberos has the single sign-on feature. For more information, see the Kerberos documentation (Kerberos: The Network Authentication Protocol, n.d.) The KX.509 project (KX.509: X.509 Certificates via Kerberos 2005) provides a means of creating X.509 certificates that are Kerberos-authenticated. These certificates are issued by a Kerberos-authenticated server (KCA). PKINIT is an Internet Engineering Task Force (IETF) Internet draft for authentication in Kerberos that allows a Kerberos ticket to be obtained using GSI credentials rather than a Kerberos password.

SAML (Security Assertions Markup Language) is an Organization for the Advancement of Structured Information Standards (OASIS) standard XML language for communicating user authentication and authorization information. SAML provides assertions that contain statements. These statements are grouped into authentication statements that indicate that the user has been authenticated, attribute statements used for making access control decisions, and authorization decision statements. SAML is a language for use in single sign-on authentication and authorization systems but is not an implementation. An example of a system using SAML is Shibboleth. Detailed information on SAML is given by Hughes and Maler (2005). More information on Shibboleth can be found at Shibboleth Project Web site (Shibboleth Project, n.d.).

RESOURCE MANAGEMENT

Data Management

One of the most challenging issues that has come up in recent years is the management of the large amounts of data being collected by scientists. The challenges of managing large volumes of data are where to store it, how to classify it, what metadata does it need, how to transfer it, where to find data of interest, how to deal with replication, how to deal with distribution, etc. There are several tools included with the Globus Toolkit that can assist with dealing with data in particular for transferring files.

One component is GridFTP (Allcock et al. 2005). GridFTP is actually a protocol, although the Globus Toolkit has an implementation of GridFTP. The protocol builds on the FTP protocol. The extension includes security, reliability, striping, third-party transfers, partial file access, and parallel transfers. GSI security is used, which is PKI-based. Reliability is achieved by having the receiving server periodically send restart markers to the sending server. These markers indicate which bytes have been received so far. This allows the sending server to restart the transfer as necessary. Striping is when either the sending side, receiving side, or both are clusters sharing a parallel file system. Striping allows for each node in the cluster to send part of the file in parallel. GridFTP also supports parallel data transfer through the use of multiple TCP streams between pairs of hosts.

Third-party transfers are a very useful feature, in which a user can initiate from one machine a file transfer that takes place between two other machines. For example, a scientist may want to run a program on a supercomputer with data that currently resides in a data store on another server, but the scientist is currently using a laptop for access. Third-party transfer will allow the scientist to stage the file on the supercomputer without the need to transfer it to the local laptop, which may be impractical or even infeasible if the file is large. Partial file access allows one to transfer only part of a file by providing an offset and total number of bytes desired. This is useful for file caching.

The GridFTP protocol is a low-level file transport protocol. It has very nice and useful features, but should be viewed as a tool on which higher-level tools should be developed. Reliable file transfer (RFT) (Chervenak et al., 2004) is a WSRF-compliant Web service for the transfer of files. Since it makes use of GridFTP, many of the features of GridFTP are available through RFT, such as GSI security, reliability, third-party transfers, and parallel streams. RFT also allows for file deletion and third-party file and directory transfers. RFT extends the reliability of GridFTP by using a PostgresSQL database to store restart markers. These restart markers are used as checkpoints and will allow RFT to continue the file transfer after server, network, container, or file system failures. The RFT Resource is implemented as a Persistent-Resource, which allows RFT to restart file transfers even after the Web services container is restarted. RFT is a very useful Web service for data management. Its primary contribution is its reliability. Although it is still fairly low-level, most higher-level tools should make use of RFT to take advantage of the security and reliability.

Once a file is transferred between hosts (either FTP, SFTP, GridFTP, or RFT) then there is an automatic duplication of that file. Often, the new copy is a temporary file and will be destroyed in short order. Other times, the file persists. One might view this as wasted space. However, duplication can also be useful. If a scientist transfers

a file from a data store to a supercomputer and another scientist needs the same file for another computation on the same machine, then the file would not need to be transferred twice. More likely, there may be a copy of a desired file on a file system that is more local than the file's source. Transferring the file from the local file system will likely be faster.

Replica location service (RLS) (Cai, Chervenak, and Frank 2004) attempts to make these scenarios possible by creating a distributed registry of file replicas. RLS introduces the terms *logical file name*, which is a unique identifier associated with the file contents, and *physical file name*, which refers to an actual file on an actual file system. An RLS deployment consists of at least one local replica catalog (LRC), which stores the logical to physical file name mappings. Clients can query this catalog as well as publish newly created files.

The RLS registry may be distributed, which allows for scalability as well as reliability against a single server crash. For a distributed configuration, RLS uses one or more replica location index (RLI) nodes, which collect logical to physical file name mappings from one or more LRCs. Figure 6 shows such a configuration. When a client queries the RLI, the RLI provides it with a list of LRCs where it believes the desired mappings exist. The client would then need to query those LRCs for the physical locations of the desire file replica. Mappings are sent from the LRCs to the RLIs using a soft-state update protocol. This allows for the mappings to be transferred only periodically. Since the mappings have timeouts associated with them, a RLI that is brought back online after a failure will have its mapping restored.

The data replication service (DRS) (Chervenak et al. 2005) is a WSRF-compliant Web service built on top of RFT and RLS. It attempts to make sure that a set of files is available on a particular file system. It will query RLS to find the desired files. Then it will use RFT to carry out the transfer. After the files have been transferred, DRS will register the duplicates with RLS. DRS is a very powerful tool that provides a nice abstraction of data management to the user. There does not seem to be much use of this level of abstraction in Grid applications yet. However, tools like DRS will be necessary components as the Grid resources and number of users grow.

Job Schedulers and Resource Managers

The Globus Toolkit does not have schedulers built in (with the exception of a "fork" scheduler, which is hardly a scheduler). Schedulers are provided by third-party vendors. There are two levels of scheduling in the Globus environment: local scheduling, which Globus uses to carry out the execution of a job on a local resource, and metascheduling, which is a higher level than the grid middleware.

The basic scheduling mechanism that ships with Globus as the default scheduler is Fork. Fork simply forks a new process in which to run the job. Fork does not provide many features other than the ability to start a job and monitor its status. It is meant only as a tool to allow a server with Globus to function out of the box. Users desiring more functionality should use one of the many other third-party schedulers that provide a much richer set of features.

There are many scheduling packages available, and most of them provide similar functionality. A few of the more popular or well-known scheduling systems are Portable Batch System (PBS) (Altair Grid Technologies 2006), Sun Grid Engine (SGE) (Sun N1 Grid Engine 6 2006), Platform Load Sharing Facility (LSF) (Platform 2006), Maui, and Torque, all of which provide the basics mechanics of scheduling and executing batch jobs on a variety of execution servers. They have similar functionality, such as allowing the user to specify scheduling algorithms, job priority, job interdependency, and automatic file staging.

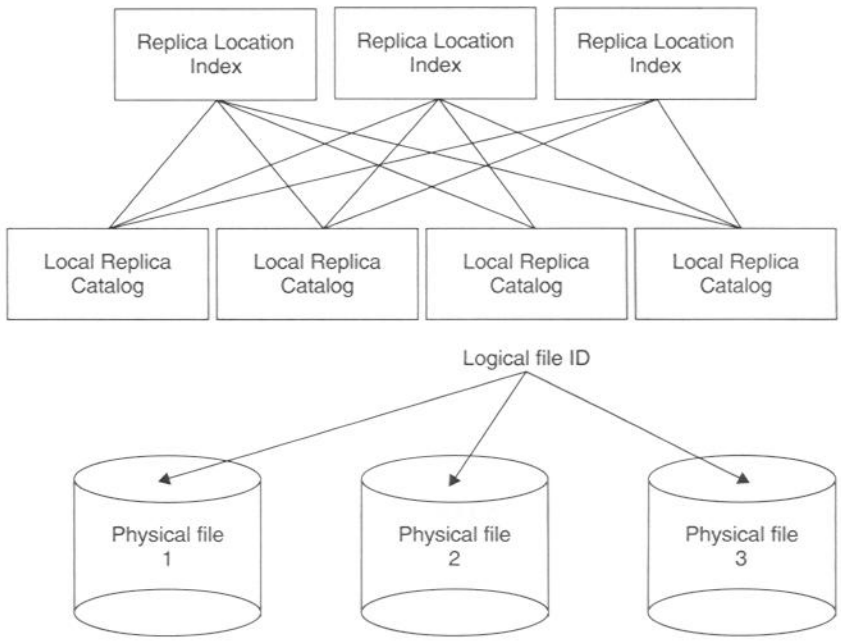

Figure 6: Replica location service configuration

The first implementation of PBS is the OpenPBS by NASA. The PBSPro (by Altair) provides some additional features such as a graphical user interface (GUI), security and access control lists, job accounting, desktop idle-cycle harvesting, automatic load-leveling, username mapping, and parallel job support by supporting libraries such as the Message Passing Interface (MPI), Parallel Virtual Machine (PVM), and High Performance Fortran (HPF). SGE also provides a GUI, checkpointing, and support for parallel jobs.

Condor (Thain, Tannenbaum, and Livny 2003) is a scheduler that is designed to allow users to make use of idle machines. One of the benefits of Condor is that the source code of the job does not need to be modified. However, if the source code can be recompiled and linked with the Condor libraries, then the job can have the benefit of being checkpointed. Checkpointing allows a job to be restarted in mid-execution. The purpose of this is to allow a job to recover after a server failure. Condor also intercepts the system calls of the job and executes them on the local machine (where the job was submitted). What this accomplishes is to allow data to reside on a separate machine, which is less intrusive on the remote machine. Furthermore, the user does not need to have an account on the remote machine.

Nimrod (Abramson et al. 1995) is a tool that controls the distributed execution of parameter-sweep applications. In parameter-sweep applications, the same program will be executed many times on a slightly difference set of input parameters each time. These types of applications are very well suited for distributed computing, as each execution can be run independently. Nimrod is a tool that makes it easy to set up and control the execution of parameter-sweep applications.

Metaschedulers

Metaschedulers are systems that are designed to work at a very high level, on top of existing grid and scheduling middleware, scheduling jobs between sites. Since grids often include systems with heterogeneous resources, they will also likely have a variety of local schedulers. Currently available metaschedulers include Community Scheduler Framework (CSF 2005), Nimrod/G (Buyya, Abramson, and Giddy 2001), and Condor-G (Thain, Tannenbaum, and Livny 2003). These metaschedulers provide similar functionality, such as the ability to submit jobs, define scheduling policies, and make reservations for resources. The resources themselves may use different local schedulers, such as Fork, PBS, LSF, SGE, Nimrod, or Condor. Metaschedulers make use of the Globus Toolkit and can take advantage of the many features provided through the toolkit, such as authentication through GSI, resource discovery through GDS (grid data service), job execution and scheduling through GRAM, and data management through RLS and DRS.

USER INTERFACE AND WORKFLOW MANAGEMENT

Most examples of Grids use a batch-mode processing model. Unfortunately, this has been the mode of operation since the beginning of the computer age. For Grids to gain widespread acceptance and use, they will eventually need to become as easy to use as, say, the World Wide Web. As the World Wide Web and the browser propelled the use of the Internet by scientists and the general population and make the Internet a household name, a similar user-friendly environment will be necessary to propel the use of grids by the general population. There are two main thrusts in the area of graphical and intuitive user interfaces for grids: portals and workflow editors, which are discussed below.

Portals

Grid portals are browser-based dynamic content environments that provide a single sign-on interface to grids. Portals are similar to servlets, in that they are Java-based components that generate dynamic content, usually in the form of HTML, extensible HTML (XHTML), or wireless markup language (WML), which is displayed in a browser. Portals are made up of Java Web-based components called "portlets," which generate the content and are managed in a portal container. Portals also use the client/server relationship as with servlets.

Of course, the purpose of portals is to provide an interface to grids. Although the architecture of portals is similar to servlets, there are a number of differences, which necessitates a separate designation for portals. The most salient of these differences include: portlets only need to generate content fragments as opposed to complete documents, portlets are not associated with a URL, there can be many instances of a portlet within a portal page, portlets can maintain persistent data, and portlets have access to and require access to the user's profile data.

The JSR 168 Portlet Specification (JSR-000168 Portlet Specification 2003) defines a standard Java API for creating portlets. This is an important specification that will ensure the interoperability of portal containers produced by different vendors. Two well-known portal toolkits that will help users created portals that are JSR 168–compliant are open grid computing environments (OGCEs) (Gannon et al. 2003) and GridSphere (Novotny, Russell, and Wehrens 2004).

Workflow Editors

Workflow editors are another form of graphical user interfaces to Grids. Whereas portals allow the user to start, monitor, terminate, etc. a Grid application, workflow editors also allow users to run several interdependent applications and describe dependencies between the applications, coordinating the inputs and outputs of various Grid applications.

OGSA-DAI (Antonioletti, Atkinson, Baxter, et al. 2004; Antonioletti, Atkinson, Borley, et al. 2004) is an initiative to create middleware that makes accessing and integrating databases much easier. OGSA-DAI provides the mechanism through Web services to access a variety of database types, including relational databases, XML files, and flat files. The data that are extracted from databases can be transformed using extensible stylesheet language transformation (XSLT) or compressed using ZIP or GZIP,

then delivered to a variety of sinks. The sinks can be other OGSA-DAI services, URLs, FTP servers, GridFTP services, or simply files. Although OGSA-DAI comes with a GUI, it is intended to be middleware. OGSA-DAI is a very powerful tool that can be very helpful in building other tools, especially workflow editors.

GridNexus (Brown et al. 2006) is a workflow editor, which uses Ptolemy (Lee 2003) as its GUI to develop workflows that are described in an XML scripting language called JXPL. The GUI allows users to create actors, or modules that can function as generic clients to Web services and grid services. GridNexus also has an actor that functions as a generic client to GRAM for the submission of a batch job.

Figure 7 shows a workflow in GridNexus that uses a Web service and a grid service. The actors are configured by providing the WSDL of the Web service or `AddressingLocator` class of the grid service. Once the actor is configured, its input and output ports are created to match the inputs and outputs of the service. The user is then able to call these services simply by providing input to one of the ports.

Although the functionality of this workflow is very simple, it demonstrates how the output of one actor becomes the input of another. Given that these actors represent calls to remote services, chaining together these services without the use of a workflow editor is a rather tedious process. Consider the workflow shown in Figure 8. This workflow implements a computational chemistry application in which a user wishes to use a local molecule file as input, convert it to an intermediate format, manipulate it using an interactive program, manipulate it again using a local program, run it through a Grid service that performs analysis, then save the result of that to the local machine. The steps necessary to perform this work without the use of a workflow editor is very tedious and prone to error.

The GRAM client (GridExec actor) allows the user to run a job on a remote machine. The user can also perform file staging with this actor. If the file staging is not a viable option, then there are also actors to perform these

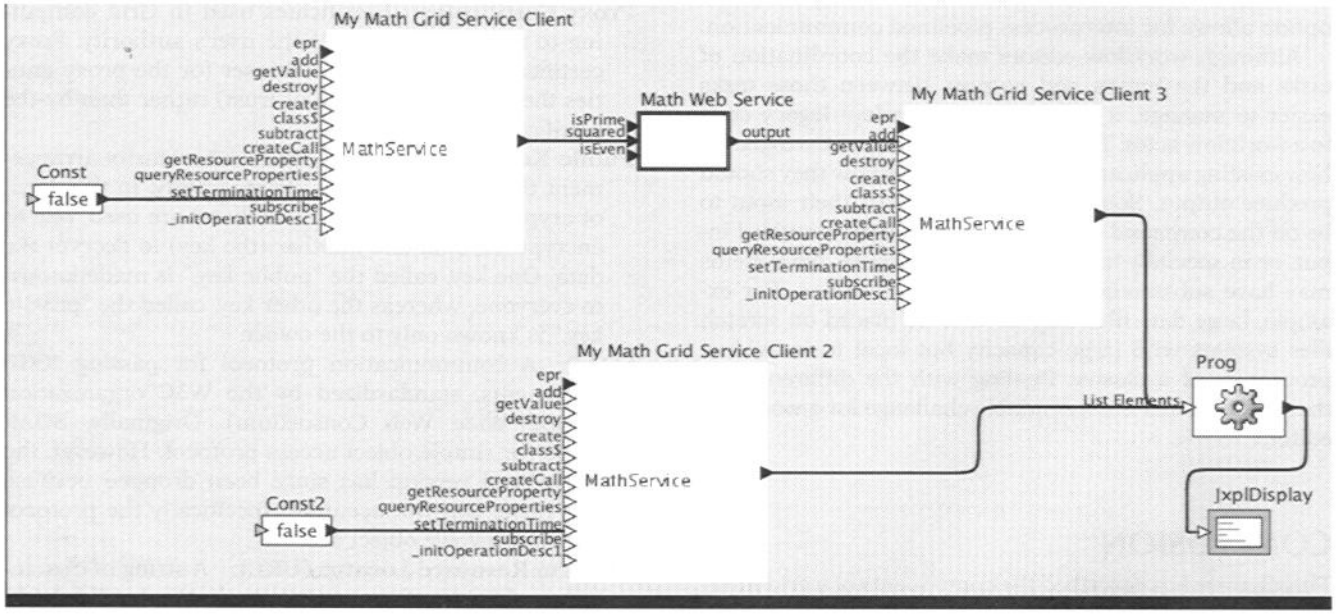

Figure 7: Simple GridNexus workflow

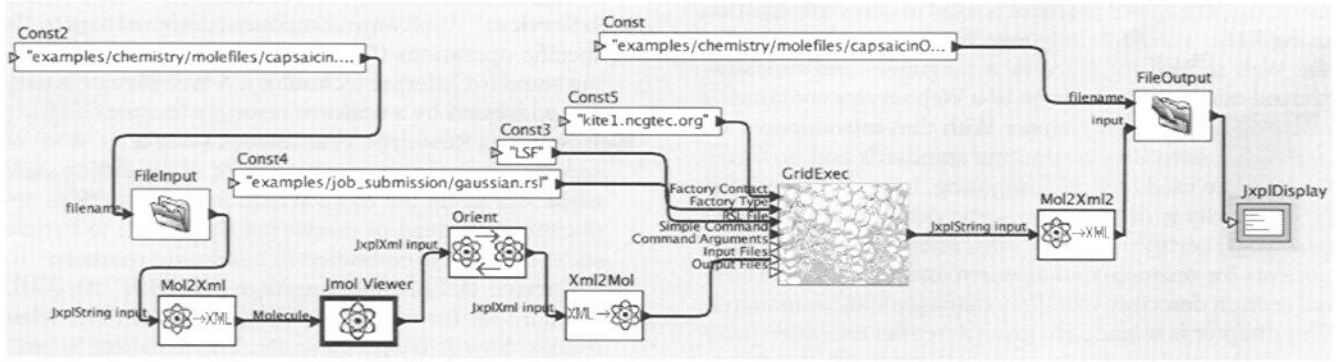

Figure 8: Chemistry application using GridNexus

functions through the use of GridFTP or SFTP (SSH file transfer protocol). GridNexus also has a set of actors to interface with OGSA-DAI.

Kepler (Altintas et al. 2005) is another workflow editor that is based on Ptolemy. One significant difference between Kepler and GridNexus is that Kepler is an extension of the Ptolemy system, whereas GridNexus uses Ptolemy to produce scripts in JXPL, and the JXPL interpreter provides the functionality. The Kepler approach has the advantage of leveraging the functionality of the already mature Ptolemy implementation. The GridNexus approach requires an additional language and corresponding interpreter. However, the advantage of using a separate language is that it separates the execution from the GUI, allowing various parts of the workflow execution to migrate to other processors.

Kepler has another component that provides a useful advantage. The library GriddleS provides a rich set of interprocess communication facilities. The GriddleS library traps input/output system calls and redirects them to services that can perform the desired operation on a local file, remote file, replicated file, or shared buffer. Access to remote files is available through GridFTP, SRB (storage resource broker), or SCP (secure copy). The shared buffer option allows for interprocess pipelined communication.

Although workflow editors make the coordination of tasks and the inputs and outputs between those tasks easier to manage, it is a challenge to adapt legacy code to a workflow actor. There are no standards with respect to how existing applications expect input or how they should produce output. Some applications expect their input to be on the command line, in input files, from standard input, or in specially named files. Furthermore, file systems may have substantially different configurations. For example, large data files may need to be placed on scratch files systems with large capacity but local to individual processors of a cluster. Dealing with the differences of these file systems is a formidable challenge for a workflow editor.

CONCLUSION

This chapter has described the components of a grid infrastructure, starting with Web services, which is an underlying technology used in grid software such as Globus. Web service technology is an attractive technology because of its language- and machine-neutral interface. Web services use XML. The SOAP protocol is used to carry information using XML. The XML language WSDL is used to describe the Web service interface in a language- and machine-neutral manner. The concept of a Web service container is also described in this chapter. With that introduction, we then move onto Grid computing standards and how Web services are used in Grid computing, in particular WSRF. Grid security is described in some detail, including ways to arrange certificate authorities. Then, we describe components for resource management and schedulers. The final section describes workflow management components. This chapter is intended to give the reader an understanding of the various fundamental components needed to create a Grid computing infrastructure.

GLOSSARY

Credentials: A certificate and corresponding private key collectively. There can be user credentials, proxy credentials, host credentials, etc. However, the private key is available only to the owner, and the word *credentials* can be used quite loosely to not include the private key.

Globus: A project that provided reference implementations for Grid computing standards. Provides a toolkit of component parts, which can be used separately or, more likely, collectively for creating a Grid infrastructure.

Grid-Map File: A form of access control list for authorization. The grid-map file is a file maintained at the site that holds the list of users' distinguished names (as given on their certificates) and their corresponding local account names.

Grid Portals: Browser-based dynamic-content environments that provide a single sign-on interface to grids.

Open Grid Services Architecture (OGSA): Defines standard mechanisms for creating, naming, and discovering services in a grid computing environment. Deals with architectural issues to make interoperable grid services.

Proxy Certificates: Certificates used in Grid computing to allow delegation of the user's authority. Proxy certificates are signed by the user (or the proxy entities themselves in a chain of trust) rather than by the certificate authority.

Public Key Infrastructure (PKI): A security arrangement that uses public key cryptography. In this form of cryptography, two keys (numbers) are used, one to encrypt the data and another (the key) to decrypt the data. One key, called the "public key," is made known to everyone, whereas the other key, called the "private key," is known only to the owner.

SOAP: A communication protocol for passing XML documents, standardized by the W3C organization (World Wide Web Consortium). Originally, SOAP stood for simple object access protocol. However, the spelled-out version has since been dropped because this name was not accurate; specifically the protocol does not involve object access.

Uniform Resource Locator (URL): A string of characters used to identify a resource on the World Wide Web; it includes the protocol used to access the resource. For example, www.cs.uncc.edu is the URL of the main computer science page at UNC-C, to be accessed by hypertext transfer protocol (HTTP).

Web Service: A software component designed to provide specific operations ("services") that are accessible using standard Internet technology. A Web service is usually addressed by a uniform resource locator (URL).

Web Services Resource Framework (WSRF): A set of specifications that present a way of representing state while still using the basic WSDL for Web services applications. Instead of modifying the WSDL to handle state, the state is embodied in a separate resource.

Web Service Definition Language (WSDL): An XML standard for formally describing a Web service—what it does, how it is accessed, etc. The standard is published by the World Wide Web Consortium (W3C).

Workflow Editor: A graphical user interface to grids that allow the user to run several interdependent applications and describe dependencies between the applications, coordinating the inputs and outputs of various grid applications.

WS-Resource: The combination of a Web service and resource in the Web Services Resource Framework (WSRF).

XML (Extensible Mark-up Language): A standard mark-up language developed to represent textual information in a structured manner that could be read and interpreted by a computer. XML is a foundation for Web services, and Web services now forms the basis of Grid computing.

CROSS REFERENCES

See *Cluster Computing Fundamentals; Grid Computing Fundamentals; Next Generation Cluster Networks; Utility Computing on Global Grids.*

REFERENCES

Abramson D., R. Sosic, J. Giddy, and B. Hall. 1995. Nimrod: A tool for performing parametised simulations using distributed workstations. *The 4th IEEE Symposium on High Performance Distributed Computing*. http://www.csse.monash.edu.au/%7Edavida/papers/nimrod.pdf (accessed April 21, 2007).

Allcock, W., J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, et al. 2005. The Globus Striped GridFTP Framework and Server. *Proceedings of Super Computing 2005 (SC05)*, Seattle.

Altair Grid Technologies. 2006. Portable Batch System Home Page. www.openpbs.org (accessed January 20, 2006).

Altintas, I., A. Birnbaum, K. Baldridge, W. Sudholt, M. Miller, C. Amoreira, et al. 2005. A framework for the design and reuse of Grid workflows. *International Workshop on Scientific Applications on Grid Computing (SAG'04), Lecture Notes in Computer Science* 3458.

Ananthakrishnan, R., C. Bacon, L. Childers, J. Gawor, J. Insley, and B. Clifford 2005. *How to build a service using GT4*. http://www-unix.mcs.anl.gov/~childers/tutorials/BAS/SDSC/GT4BuildAServiceV15.pdf (accessed December 29, 2005).

Antonioletti, M., M. P. Atkinson, R. Baxter, A. Borley, N. P. Chue Hong, B. Collins, B., et al. 2004. OGSA-DAI status report and future directions. *Proceedings of the UK e-Science All Hands Meeting*. Nottingham, United Kingdom.

Antonioletti, M., M. P. Atkinson, A. Borley, N. P. Chue Hong, B. Collins, J. Davies, et al. 2004. OGSA-DAI usage scenarios and behaviour: Determining good practice. *Proceedings of the UK e-Science All Hands Meeting*.

Booth, D., & C. K. Liu (2005). *Web Services Description Language (WSDL) version 2.0 part 0: Primer*. www.w3.org/2002/ws/desc/wsdl20-primer(accessed January 4, 2006).

Brown, J., Ferner, C., Hudson, T., Stapleton, A., Vetter, R., Carland, T., et al. (2006). GridNexus: A Grid Services Scientific Workflow System. *International Journal of Computer & Information Science* 6(2):72–82.

Buyya, R., D. Abramson, and J. Giddy. 2001. Nimrod-G resource broker for service-oriented grid computing. *IEEE Distributed Systems Online*, 2(7).

Cai, M., A. Chervenak, and M. Frank. 2004. A peer-to-peer replica location service based on a distributed hash table. *Proceedings of the SC2004 Conference*, Pittsburgh.

Chervenak, A. L., N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. 2004. Performance and scalability of a replica location service. *Proceedings of the International IEEE Symposium on High Performance Distributed Computing*, HPDC-13.

Chervenak, A., R. Schuler, C. Kesselman, S. Koranda, and B. Moe. 2005. Wide area data replication for scientific collaborations. *Proceedings of 6th IEEE/ACM International Workshop on Grid Computing (Grid2005)*, Cardiff, Wales.

CSF: Community Scheduler Framework. 2005. www.globus.org/toolkit/docs/4.0/contributions/csf/CSF_Release_Notes.html (accessed January 20, 2006).

Czajkowski, K., D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, et al. 2004. The WS-Resource Framework, version 1.0, 03/05/2004. www.globus.org/wsrf/specs/ws-wsrf.pdf#search='WSResource%20Framework%2003%2F05%2F2004 (accessed December 29, 2005).

Ferreira, L., V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, M., et al. 2004. *Introduction to Grid Computing with Globus*. IBM RedBooks. www.redbooks.ibm.com/abstracts/sg246895.html (accessed December 28, 2005).

Foster, I. (2005). Globus Toolkit version 4: Software for service-oriented systems. IFIP International Conference on Network and Parallel Computing. *Lecture Notes in Computer Science*, 3779:2–13.

Foster, I., C. Kesselman, J. M. Nick, and S. Tuecke. 2002. The physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Global Grid Forum. Also reprinted as "Chapter 8: The Physiology of the Grid" in eds. 2003. *Grid computing: Making the global infrastructure a reality*, edited by F. Berman, G. Fox, and T. Hey, 217–49. Chichester, England: Wiley.

Gannon, D., G. Fox, M. Pierce, B. Plale, G. von Laszewski, C. Severance, et al. 2003. Grid portals: A scientist's accesspointforgridservices(draft1)September19,2003. http://www.extreme.indiana.edu/groc/ggf-portals-draft.pdf (accessed May 11, 2007).

The Globus Toolkit. 2006. www.globus.org/toolkit (accessed May 25, 2006).

Graham, S., D. Davis, S. Simeonov, G. Daniels, P. Brittenham, Y. Nakamura, et al. 2005. *Building Web services with Java: Making sense of XML, SOAP, WSDL, and UDDI*. 2nd ed. Indianapolis: SAMS.

Grid User Management System. 2006. http://grid.racf.bnl.gov/GUMS (accessed January 2, 2006).

GT 4.0 Security: Key Concepts. n.d. www.globus.org/toolkit/docs/4.0/security/key-index.html (accessed December 29, 2005).

Gudgin, M., M. Hadley, and T. Rogers. (2006). Web services addressing 1.0 – W3C recommendation May 2006.

http://www.w3.org/TR/ws-addr-core/ (accessed May 11, 2007).

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J-J., & Nielsen, H. F., eds. 2003. SOAP Version 1.2 Part 1: messaging framework W3C recommendation 24 June 2003. www.w3.org/TR/2003/REC-soap12-part1-20030624/#intro (accessed December 29, 2005).

Hughes, J., and E. Maler, eds. 2005. Security Assertion Markup Language (SAML) 2.0 Technical Overview. http://xml.coverpages.org/SAML-TechOverview20v03-11511.pdf (accessed May 25, 2006).

IBM. 2001. Web Service Inspection Language. http://www-128.ibm.com/developerworks/library/specification/ws-wsilspec (accessed May 25, 2006).

JSR-000168 Portlet Specification, Final Release. 2003. www.jcp.org/aboutJava/communityprocess/final/jsr168 (accessed January 20, 2006).

Kerberos: The Network Authentication Protocol. n.d. http://Web.mit.edu/kerberos (accessed January 2, 2006).

KX.509: X.509 Certificates via Kerberos. 2005. http://kx509.org (accessed January 2, 2006).

Lee, E. 2003. Overview of the Ptolemy Project Technical Memorandum UCB/ERL M03/25. University of California, Berkeley, CA, 94720. http://ptolemy.eecs.berkeley.edu (accessed January 20, 2006).

MyProxy Credential Management Service. 2006. http://grid.ncsa.uiuc.edu/myproxy (accessed May 25, 2006).

Novotny, J., M. Russell, and O. Wehrens. 2004. GridSphere: An advanced portal framework. *30th EUROMICRO Conference (EUROMICRO'04)*, Rennes, France.

Platform. 2006. Platform LSF Family of Products Home Page. www.platform.com (accessed January 20, 2006).

Shibboleth Project. n.d. http://shibboleth.internet2.edu (accessed January 2, 2006).

Sun N1 Grid Engine 6. 2006. www.sun.com/software/gridware/index.xml (accessed January 20, 2006).

SURAgrid User Management and PKI Bridge Certification Authority. n.d. Available at https://www.pki.Virginia.edu/nmi-bridge (accessed May 11, 2007).

Thain, D., T. Tannenbaum, and M. Livny. 2003. Condor and the Grid. In *Grid computing: Making the global infrastructure a reality*, edited by F. Berman, A. J. K. Hey, and G. Fox 299–335. Chichester, England: Wiley.

W3C Architecture Domain XML Schema. n.d. www.w3.org/XML/Schema. (accessed January 4, 2006).

Wilkinson, B., and C. Ferner. 2005. *Grid Computing Fall 2005 Course Home Page*. www.cs.uncc.edu/~abw/ITCS4010F05/index.html (accessed December 28, 2005).