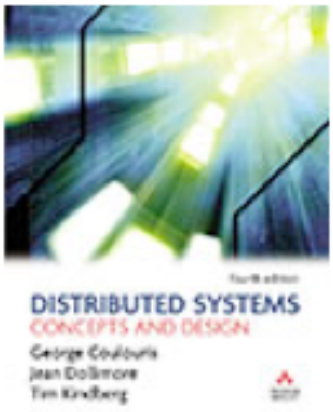


Archive material from *Edition 4* of Distributed Systems: Concepts and Design

Section CDK4–19.7

Originally published as pp. 814- 823 of Coulouris, Dolllimore and Kindberg, Distributed Systems, Edition 4, 2005.

19.7 Case study: the Grid

The name 'Grid' is used to refer to middleware that is designed to enable the sharing of resources such as files, computers, software, data and sensors on a very large scale. The resources are shared typically by groups of users in different organizations who are collaborating on the solution of problems requiring large numbers of computers to solve them, either by the sharing of data or by the sharing of computing power. These resources are necessarily supported by heterogeneous computer hardware, operating systems, programming languages and applications. Management is needed to coordinate the use of resources to ensure that clients get what they need and that services can afford to supply it. In some cases, sophisticated security techniques are required to ensure that the correct use is made of resources in this type of environment.

Section 19.7.1 introduces the World-Wide Telescope, a data-intensive application that is an example of the type of problem solving for which the Grid is designed. It illustrates a typical pattern of sharing and geographic distribution of users from which can be derived the characteristics of a family of scientific applications, which suggest a set of requirements for a Grid (see Section 19.7.2). We use these requirements, to motivate the architecture, which specifies a Grid that runs over web services (see Section 19.7.3). The last section introduces the Globus toolkit, which is an implementation of the grid architecture.

19.7.1 The World-Wide Telescope – a grid application

This project is concerned with deploying the data resources shared by the astronomy community. It is described in the work of Szalay and Gray [2004], Szalay and Gray [2001] and Gray and Szalay [2002]. Astronomy data consists of archives of observations, each of which covers a particular period of time, a part of the electromagnetic spectrum (optical, x-ray, radio) and a particular area of the sky. These observations are made by different instruments deployed at various places throughout the world.

A study of how astronomers share their data is useful for deriving the characteristics of a typical grid application, because astronomers freely share their results with one another and issues of security can be omitted, making this discussion simpler.

Astronomers make studies that need to combine data on the same celestial objects but involve several different periods of time and multiple parts of the spectrum. The ability to use independent observations of data is important to research. Visualization allows astronomers to see the data as 2D or 3D scatter plots.

The teams gathering the data store it in immense archives (currently terabytes) which are managed locally by each team that gathers data. The instruments used in gathering data are subject to Moore's law. Hence the amount of data gathered grows exponentially. As it is gathered, the data is analysed by a pipeline process and stored as derived data for use by astronomers throughout the world. But before data can be used by other researchers, scientists working in a particular field need to agree on a common way of labelling their data.

Szalay and Gray [2004] point out that in the past, scientific research data was included by authors in articles and published in journals which lived in libraries. But nowadays, the quantity of data is too great to be included in a publication. This applies not only to astronomy, but also in the fields of particle physics, genome and biology research. The role of author now belongs to the collaborations, which take 5–10 years to build their experiment before producing the data that is published to the world in web-based archives. Thus, the scientists working on the projects become data publishers and librarians as well as authors.

This additional role requires any project that manages a data archive to make it accessible to other researchers. This implies a considerable overhead in addition to the original task of data analysis. To make such sharing possible, the raw data requires metadata to describe, for example, the time it was collected, the part of the sky and the instrument used. In addition, the derived data needs to be accompanied by metadata describing the parameters of the pipelines through which it was processed.

The calculation of derived data requires heavy computational support. It often has to be recalculated as techniques improve. All of this is a considerable expense for the project that owns the data.

The aim of the World-Wide Telescope is to unify the world's astronomy archives into a giant database containing astronomy literature, images, raw data, derived datasets and simulation data.

19.7.2 The characteristics of a family of data-intensive scientific applications

In this section we refer to the study of the World-Wide Telescope in presenting the defining characteristics of a family of similar scientific applications, and then go on to outline the requirements for supporting them. The characteristics are as follows:

- data is collected by means of scientific instruments;
- the data is stored in archives on a set of separate sites whose locations can be in different sites anywhere in the world;
- the data is managed by teams of scientists belonging to separate organizations;
- an immense and increasing quantity (terabytes or petabytes) of raw data is generated from the instruments;
- computer programs will be used to analyse and make summaries of the raw data, for example, to classify, calibrate and catalogue the raw data representing celestial objects.

The Internet makes all of these data archives potentially available to scientists throughout the world. They will be able to get data from different instruments taken at different times and at different sites. However, a particular scientist using this data for their own research will be interested in just a subset of the objects in the archives.

The immense quantity of data in an archive makes it infeasible to transfer it to the location of the user before processing it to extract the objects of interest, due to considerations of the transmission time and the local disk space required. Therefore, it is not appropriate to use ftp or web access in this context. The processing of the raw data should take place at the location where it is collected and stored in a database. Then when a scientist makes a query about particular objects, the information in each database should be analysed and if necessary, visualizations produced before returning the results to the remote query.

The fact that data is processed at many different sites provides an inbuilt parallelism that effectively divides the immense task being undertaken.

From the above characteristics, the following requirements are derived:

R1: Remote access to resources – that is, to the required information in the archives.

R2: Processing of data at the site where it is stored and managed, either when it is gathered or in response to a request. A typical query might result in a visualisation based on data collected for one region of sky recorded by different instruments at different times. It will involve selecting a small quantity of data from each massive data archive.

R3: The resource manager of a data archive should be able to create service instances dynamically to deal with the particular section of data required, just as in the distributed object model, where servants are created whenever they are needed to handle different resources managed by a service.

R4: Metadata to describe:

- characteristics of the data in an archive, for example, for astronomy: the area of the sky, the date and time collected and the instruments used;
- the characteristics of a service managing that data, for example, its cost, its geographic location, its publisher or its load or space available.

R5: Directory services based on the above metadata.

R6: Software to manage queries, data transfers and advance reservation of resources, taking into account that the resources are generally managed by the projects that generate the data and that access to them may need to be rationed.

Web services can deal with the first two requirements by providing a convenient way for allowing scientists to access operations on data in remote archives. This will require that each particular application provides a service description that includes a set of methods for accessing its data. The grid middleware deals with the remaining requirements.

Although the World-Wide Telescope is a typical data-intensive application, Grids are also used for computationally-intensive applications such as image analysis and other examples discussed in Section 19.7.4. Where computationally-intensive applications are deployed on a Grid, resource management will be concerned with allocating computing resources and balancing loads.

Finally, security will be needed for many grid applications. For example, the Grid is in use for medical research and for business applications. Even when the privacy of data is not an issue, it will be important to establish the identity of the people who created the data.

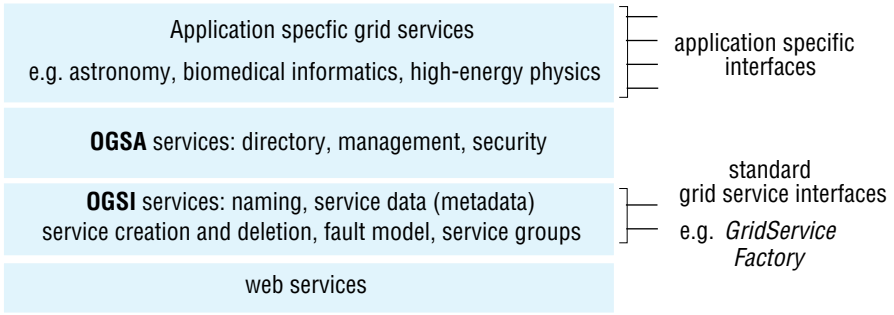
19.7.3 Open grid services architecture

The open grid services architecture (OGSA) is a standard for grid-based applications [Foster *et al.* 2002 and 2001]. It provides a framework within which the above requirements can be met. It is based on web services. Resources are managed by application-specific grid services. The Globus toolkit, which implements the architecture is discussed in Section 19.7.5.

Figure 19.19 shows the main components of the grid architecture. It illustrates two important aspects of *application-level grid services*:

1. They are web services that implement standard grid-service interfaces in addition to their application-specific interfaces. Specifically, they implement the following grid-service interfaces and additional functionality:
   - an interface to a set of data (called *service data*) that contains metadata about the service. Metadata includes, for example, the time to be terminated, but can

Figure 19.19 Open grid services architecture



include any of the items mentioned in requirement R4 above, as well as application values such as sets of recent results or average values;

- the context in which a service runs must provide a *factory* with the ability to create new service instances and to stop them when their time runs out, as suggested by requirement R3. The factory relies on the naming facilities of the OGSI layer to manage the names of the service instances it creates.

2. They make use of the standard grid services which are provided as two separate layers, called OGSI and OGSA, above web services. The OGSA services layer includes:
   - directory service – allowing client software to select service instances suitable for its needs, based on the metadata collected from service instances that have registered with the directory service. This deals with requirement R5;
   - management service – to monitor services from information in their service data, to deal with failures; and to control the lifetimes of service instances by setting values in the service data. This deals with requirement R6;
   - security services – providing single logins and delegation as well as authentication and data encryption.

The open grid services infrastructure (OGSI) layer includes the following:

- the implementation of a scheme for the naming of service instances;
- the definition of standard *service data* elements which must be implemented by every application-level grid service instance, together with operations to set and get their values. These elements include, for example, names of the interfaces supported, a reference to the factory that created the instance or the termination time;
- definitions of the interface to the factory for creating new service instances and operations to set their termination times or to destroy them;
- a fault model for use by all grid services;

– notification services – to enable services to become publishers of information about service data; and other services to become subscribers;

– service groups – operations for adding and removing members, for use by cooperating groups providing a service.

The above definitions and services are provided in the *GridService*, *Factory* and other interfaces of the OGSI layer.

Open grid services infrastructure ◊ We now introduce the two-level naming scheme that relates the high-level names of service instances to low-level names and then present further details of the other services in the OGSI layer.

Two-level naming scheme: Grid services can be created dynamically. Therefore to fulfil the need to distinguish one instance from another, the infrastructure includes a naming scheme that provides each instance with a long-lived and globally unique identifier called a grid service handle (GSH). When an instance of a service is restarted with the same state as before, the same GSH may be used.

A GSH is represented by a URI, which can be mapped onto a short-lived name called a grid service reference (GSR), which is used to refer to the destination of an invocation. The infrastructure includes a handle-resolution service to perform this mapping. The GSR is a name whose structure depends on the request-reply mechanism in use, for example, in the case of SOAP, the GSR will refer to a WSDL document containing <*service*> and <*binding*> elements. A GSR becomes invalid when the service instance it refers to fails.

If a URN is used for the GSH, it may be resolved to different service instances at different times, allowing the latter to be migrated or even replicated at different resources.

Service Data (metadata): The idea is that clients can request an instance of a service to return information about its current state, for example, its capacity, free space, load or current errors or even its recent results. This requires that each instance of an application-level grid service must implement storage for service data and a set of operations to access it.

A standard set of operations is provided in the *GridService* interface – these include operations to access the names of the interfaces supported by the service, the names of the service data elements, the identity of the factory that created it, its GSH and GSR and its termination time. Every grid service must support the *GridService* interface, but may also include OGSI interfaces for notification or for service groups.

Service creation and deletion: This part of the infrastructure defines a standard *Factory* interface that specifies operations for creating transient service instances on demand with new names in the form of a GSH and GSR pair. The *Factory* interface must be implemented by any container for running application-level grid services.

A typical request to create a new service instance is made by an application which specifies as arguments its service description and the end of its lifetime. In response, the factory creates a new service instance and registers it with a handle resolution service, finally returning the GSH and the current GSR to the client.

A service instance will end, either when its task is completed or on the request of the client that created it. Each service instance is given a lifetime to prevent it from surviving for ever, in the event that messages are lost. But the creator can request

extensions of the lifetime by sending 'keep alive' messages. In the event that the client fails, there will be no more keep alive messages and the service instance will eventually terminate. To provide service autonomy, an instance can decide to change its own lifetime. All the resources used by a service instance are recovered when its lifetime expires. Service instances may be shared by clients.

Fault model: The infrastructure defines a common approach to the reporting of faults, which is used by all the OGSI-level services and is recommended for use by the application-level grid services. It is defined as an XML schema which requires at least two elements to specify the originating service and a timestamp. It offers optional elements for describing a fault in plain language, its cause and a fault code. There is scope for extending the fault report to include information specific to an OGSI or application service.

WSDL interface extensions: Since standard WSDL does not include mechanisms for defining service data, OGSI has provided an extension to WSDL for associating definitions of the names and types of service data elements with a particular interface in a service description. The data elements might, for example, hold its capacity, free space, load, or current error indications.

OGSA services ◊ Higher level services are built over the OGSI services. According to Foster *et al.* [2004] different developers will provide a variety of different OGSA services to meet the specific requirements of application-level services. They list those OGSA components that are sufficiently widely-applicable to be included in any grid system, for example, directory services, management and monitoring; and security. The availability of standard interfaces for these services is a basis for ensuring that different implementations can work with one another. We defer a discussion of directory and security services until Section 19.7.5, in which we present them in the context of the Globus toolkit.

Management services: Management in the Grid is concerned with any kind of resource that can be shared or exploited. This amounts to the management of services, which is concerned with arranging for them to be used and monitoring their state. Issues that it addresses include task submission, agreement to provide some particular quality of service and advanced reservation of resources. These can be provided with the help of *service-level agreement*s (SLAs) [Hauch and Reiser 2000] which give the client of a resource a guarantee as to the type of service being provided, and allow the owner of the resource to maintain control as to how it is used and how much information is exposed to the client.

Three different sorts of SLA are proposed: task level, which is used to agree on the performance of an activity; resource level, which is used to agree on the right to consume a resource, for example by advance reservation; and binding, which is used to agree on the use of a resource in a task.

19.7.4 Some examples of grid applications

Figure 19.20 shows some examples of applications that are using grid technology. The first three of these have characteristics similar to the World-Wide Telescope in that data is collected by scientific instruments and stored at the site where it is collected. In

Figure 19.20 A selection of the grid projects presented in Foster and Kesselman [2004]

Description of the project	*Reference*
1. Aircraft engine maintenance using fault histories and sensors for predictive diagnostics	www.cs.york.ac.uk/dame
2. Telepresence for predicting the effects of earthquakes on buildings, using simulations and test sites	www.neesgrid.org
3. Bio-medical informatics network providing researchers with access to experiments and visualizations of results	nbcr.sdsc.edu
4. Analysis of data from the CMS high energy particle detector at CERN by physicists world-wide over 15 years	www.uscms.org
5. Testing the effects of candidate drug molecules for their effect on the activity of a protein, by performing parallel computations using idle desktop computers	[Taufer *et al.* 2003] [Chien 2004]
6. Use of the Sun Grid Engine to enhance aerial photographs by using spare capacity on a cluster of web servers	www.globexplorer.com
7. The butterfly Grid supports multiplayer games for very large numbers of players on the internet over the Globus toolkit	www.butterfly.net
8. The Access Grid supports the needs of small group collaboration, for example by providing shared workspaces	www.accessgrid.org

example 1, vibration data is collected by means of a sensor on an aircraft engine. In example 2, data is collected from test structures that are being subjected to violent shaking to simulate an earthquake. In example 3, instruments such as MRI or CT are used to collect brain images. In all cases, the quantity of raw data grows over time, as more measurements are made. In addition, the quantity of processed data grows as analyses are carried out.

In these three examples, the data is managed by teams of scientists or engineers belonging to separate organizations. Analyses of the raw data and simulations are made locally and are available to co-workers throughout the world.

These examples confirm that the World-Wide Telescope is a typical data-intensive application. However, grid applications are also used for computationally-intensive applications such as:

- Example 4 refers to the new detector at CERN that will become operational in 2007. Prior to this, teams of physicists are carrying out simulations of the expected results from the detector. This is a computationally intensive task which is being carried out by multiple cooperating computers.
- Example 5 is about virtual screening – the testing of a database of millions of drug molecules to see whether they block the activity of each of a large number of potential proteins. Spare computing power is used on a set of desktop computers running grid software. The task is performed in parallel by each of the computers

testing the effect of a particular drug molecule on a particular protein. It uses spare capacity on idle desktop computers.

- Example 6 is about image analysis. The company GlobeExplorer provides good quality satellite images and aerial photographs, derived from an archive of thousands of raw images, each of which requires enhancement. It uses spare capacity on a cluster of web servers.

A common feature of examples 5 and 6 is the use of spare computing capacity and the partitioning of the task to be performed. They may be compared to the SETI@home project (see Section 10.1), whose task is to search for extra-terrestrial intelligence in radio telescope data. SETI@home uses peer-to-peer software to solve a computationally intensive problem by using spare computing power on end-user computers. The large database is partitioned so that the task may be carried out in parallel.

Examples 7 and 8 are included because they illustrate two different uses of the Grid that are outside of the realms of science and engineering. Each of them requires the management of distributed state; in the first case, the state of the game and the second a shared workspace.

19.7.5 The Globus toolkit

The Globus Project started in 1994 with a view to providing software that integrates and standardizes the functions required by a family of scientific applications. These functions include directory services, security and resource management. The first Globus toolkit appeared in 1997. The OGSA evolved from the second version of the toolkit (called GT2), which is described in Foster and Kesselman [2004].

The third version appeared in 2002. It is called GT3 and is based on OGSA and is therefore built on web services. It was developed by the Globus Alliance (www.globus.org) and others and is available as open-source software.

The core of GT3 is described by Sandholm and Gawor [2003]. It includes all of the interfaces in the OGSI layer as Java classes which implement interfaces such as *GridService* and *Factory* and are compatible with JAX-RPC (see Section 19.2.3). The simplest way to add this functionality to existing web services is to make their classes subclasses of the OGSI classes. In fact, application-level grid services may be configured to include whatever grid service functionality is required. For example, notification services or service group operations may be added as desired.

Grid service instances and factories are deployed in a runtime environment called a grid-service *container*. A grid-service container is similar in some respects to a CORBA POA (Section 20.2.2) in that it can do dispatching, but it also deals with:

- the dynamic creation and management of service instances with global names;
- simple access to the state of service instances by means of the *FindServiceData* operation of the *GridService* interface, or alternatively by means of notification;
- security, including delegation of credentials, signing of messages, encryption and authorization.

A service instance may be created inside the same container as its factory or it may be deployed elsewhere. A container may, for example, be based on a servlet container (see

page 796). When a container is started up, the service instances do not become active until the first time they are used. Containers may deactivate idle service instances such as those in use for notification or service groups.

Security service ◊ The security service in GT3 provides for the protection of SOAP messages. It is based on WS-Security [Kaler 2002], XML-Signature, and XML-Encryption (see Section 19.5). Authentication uses X.509 certificates as credentials, which are supplied in the usual manner by a trusted certification authority. It uses an extension to X.509 to provide for proxy certificates that may be used by services acting on behalf of users.

Directory service ◊ GT3 does not use the UDDI because its data structures and hence its search criteria contain the wrong sort of information, for example, technical details about protocols (see Figure 19.15), whereas clients of grid services need to find out about the particular characteristics of service instances. For example, management clients may be interested in the load on the service and astronomers may be interested in information about when, where and how the data was collected.

Instead, GT3 provides an *index service* which is intended to be used for finding out which instance of a service matches a set of particular requirements. The index service collects information based on the service data of the group of service instances that registers with it. It could use the *FindServiceData* operation to collect information, but for variable service data, it would be more appropriate to use the notification interface so as to be informed whenever data values change.

The index service can respond to queries by carrying out algorithms to decide on the service instance most suitable for a particular client. For example, it might base this on the type of processor used by a service instance, its speed, the clients it supports or the cost of using it. Index services support notification interfaces in addition to responding to queries.

Index services can be combined together in various ways in order to provide an information service for use within a large community.

An implementation of the index service needs to include fault-tolerance measures to deal with situations such as the case when it is left holding stale data in its indexes when a service instance stops running without notifying it.

Management and reliable file transfer service ◊ The management service in GT3 is concerned with monitoring and managing the container and the service instances inside it, for example, monitoring the current state or load and activating or deactivating instances.

GT3 also provides a reliable file transfer service to allow for bulk transfer of data between grid services.

Future relation with web services ◊ There has been some discussion of the possibility of integrating OGSI features into web services – under the heading of the WS-Resource framework [Globus 2004]. The Globus Alliance suggests that the ability to create service instances and to make use of service state could be used more widely than in the context of grid applications. However, its distributed object approach may not be successful for many of the sort of loosely-coupled applications commonly running as web services.