IBM

Introduction to Grid Computing

- Learn grid computing basics
- Understand architectural considerations
- Create and demonstrate a grid environment

Bart Jacob
Michael Brown
Kentaro Fukui
Nihar Trivedi

Redbooks

ibm.com/redbooks

International Technical Support Organization

Introduction to Grid Computing

December 2005

SG24-6778-00

Note: Before using this information, read the information in “Notices” on page ix.

First Edition (December 2005)

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AFS®
AIX 5L™
AIX®
Cloudscape™
DB2 Connect™
DB2 Universal Database™
DB2®
developerWorks®
DFS™
Domino®
@server®
eServer™
Everyplace®
ibm.com®
IBM®
iSeries™
LoadLeveler®
Lotus®
OMEGAMON®
Perform™
pSeries®
Rational®
Redbooks (logo) ™
Redbooks™
RS/6000®
ServerGuide™
Summit®
Tivoli Enterprise Console®
Tivoli Enterprise™
Tivoli®
WebSphere®
World Community Grid™
xSeries®
zSeries®

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Preface

In the past several years, grid computing has emerged as a way to harness and take advantage of computing resources across geographies and organizations. In this IBM Redbook, we describe a generalized view of grid computing including concepts, standards, and ways in which grid computing can provide business value to your organization. In a nutshell, grid computing is all about virtualization that enables businesses to take advantage of a variety of IT resources in order to be more responsive to demands of the business and increase availability of applications while reducing both infrastructure and management costs.

There are many products and toolkits available from IBM and other companies that enable different aspects of grid computing. One of the most well known toolkits is the Globus Toolkit. Globus Toolkit 4 provides components and services conforming to existing and evolving standards that can be used as the basis for a grid computing solution. In the second half of this book we provide instructions for installing and configuring a simple Globus environment that can be used to demonstrate various aspects of grid computing and to build a proof-of-concept environment. We also describe, and provide as additional material, a sample grid application that can be used to demonstrate, test, and instruct about the grid computing concepts introduced in this book.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Bart Jacob is a Senior Consulting IT Specialist at IBM Corp - International Technical Support Organization, Austin Center. He has over 25 years of experience providing technical support across a variety of IBM products and technologies, including communications, object-oriented software development, and systems management. He has over 14 years of experience at the ITSO, where he has been writing IBM Redbooks™ and creating and teaching workshops around the world on a variety of topics. He holds a Masters degree in Numerical Analysis from Syracuse University.

Michael Brown is the Technical Project Leader for the Americas and Asia Pacific sites of IBM's Linux® Integration Center, headquartered in Austin, Texas. He leads teams that perform technical support for customers who are evaluating IBM software running on Linux platforms. He is a certified Java™ Programmer, Developer, and Architect and has worked on several previous IBM grid redbooks

and presented on grid computing at the Colorado Software Summit®. He holds HBSc and MSc degrees in Computer Science from the University of Western Ontario, Canada.

Kentaro Fukui is an IT Specialist for IBM and a Red Hat Certified Engineer working in IBM Global Services, Japan. He has more than two years of experience with grid technologies as well as more than eight years of experience with UNIX®-like operating systems, Windows® servers, and Lotus® Domino® servers. He holds a MSc Degree in Information and Computer Science from Keio University, Japan. Currently, he is also working as a PhD candidate student at Keio University. He received the IEEE Computer Society Best Paper Award in 2004.

Nihar Trivedi is a Consultant and a IBM Grid Technical Sales certified professional working for IBM Business Consulting Services in Australia. Nihar has more than eight years of experience in delivering complex e-business applications in Financial Services, Utility, Government, and Telecommunication industries. Nihar is a PhD student affiliated with the University of Sydney and National ICT Australia. Nihar's main research interests include self-adaptive middleware systems and grid computing.

Thanks to the following people for their contributions to this project:

Sean Slevin

Suguru Hamazaki
System Design Center - West, Business Infrastructure Solution, IBM Japan Systems Engineering

Julie Czubik
International Technical Support Organization, Poughkeepsie Center

The team that created a predecessor redbook (*Introduction to Grid Computing with Globus*, SG24-6895) from which we have reused a wide range of material:

Luis Ferreira
Viktors Berstis
Jonathan Armstrong
Mike Kendzierski
Andreas Neukoetter
Masanobu Takagi
Richard Bing-Wo
Adeeb Amir
Ryo Murakawa
Olegario Hernandez
James Magowan

Norbert Bieberstein

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM® Corporation, International Technical Support Organization
  Dept. JN9B Building 905
  11501 Burnet Road
  Austin, Texas 78758-3493

Part 1

Grid fundamentals

1

What grid Computing is

Grid computing can mean different things to different individuals. The grand vision is often presented as an analogy to power grids where users (or electrical appliances) get access to electricity through wall sockets with no care or consideration for where or how the electricity is actually generated. In this view of grid computing, computing becomes pervasive and individual users (or client applications) gain access to computing resources (processors, storage, data, applications, and so on) as needed with little or no knowledge of where those resources are located or what the underlying technologies, hardware, operating system, and so on are.

Though this vision of grid computing can capture one's imagination and may indeed someday become a reality, there are many technical, business, political, and social issues that need to be addressed. If we consider this vision as an ultimate goal, there are many smaller steps that need to be taken to achieve it. These smaller steps each have benefits of their own.

Therefore, grid computing can be seen as a journey along a path of integrating various technologies and solutions that move us closer to the final goal. Its key values are in the underlying distributed computing infrastructure technologies that are evolving in support of cross-organizational application and resource sharing—in a word, virtualization—virtualization across technologies, platforms, and organizations. This kind of virtualization is only achievable through the use of open standards. Open standards help ensure that applications can transparently take advantage of whatever appropriate resources can be made available to

them. An environment that provides the ability to share and transparently access resources across a distributed and heterogeneous environment not only requires the technology to virtualize certain resources, but also technologies and standards in the areas of scheduling, security, accounting, systems management, and so on.

Grid computing could be defined as any of a variety of levels of virtualization along a continuum. Exactly where along that continuum one might say that a particular solution is an implementation of grid computing versus a relatively simple implementation using virtual resources is a matter of opinion. But even at the simplest levels of virtualization, one could say that grid-enabling technologies are being utilized.

This continuum is illustrated in Figure 1-1 on page 5. Starting in the lower left you see single system partitioning. Virtualization starts with being able to carve up a machine into virtual machines. As you move up this spectrum you start to be able to virtualize similar or homogeneous resources. Virtualization applies not only to servers and CPUs, but to storage, networks, and even applications. As you move up this spectrum you start to virtualize unlike resources. The next step is virtualizing the enterprise, not just in a data center or within a department but across a distributed organization, and then, finally, virtualizing outside the enterprise, across the Internet, where you might actually access resources from a set of OEMs and their suppliers or you might integrate information across a network of collaborators.

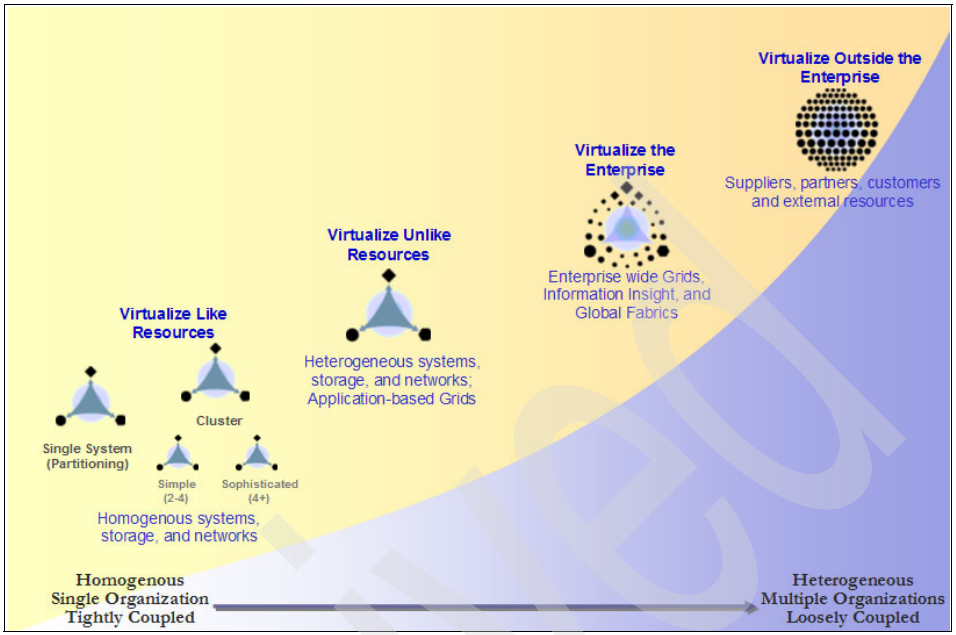

Figure 1-1 Virtualization continuum

Early implementations of grid computing have tended to be internal to a particular company or organization. However, cross-organizational grids are also being implemented and will be an important part of computing and business optimization in the future.

The distinctions between intraorganizational grids and interorganizational grids are not based in technological differences. Instead, they are based on configuration choices given: Security domains, degrees of isolation desired, type of policies and their scope, and contractual obligations between users and providers of the infrastructures. These issues are not fundamentally architectural in nature. It is in the industry's best interest to ensure that there is not an artificial split of distributed computing paradigms and models across organizational boundaries and internal IT infrastructures.

Grid computing involves an evolving set of open standards for Web services and interfaces that make services, or computing resources, available over the Internet.

Very often grid technologies are used on homogeneous clusters, and they can add value on those clusters by assisting, for example, with scheduling or provisioning of the resources in the cluster. The term grid, and its related technologies, applies across this entire spectrum.

If we focus our attention on distributed computing solutions, then we could consider one definition of grid computing to be distributed computing across virtualized resources. The goal is to create the illusion of a simple yet large and powerful virtual computer out of a collection of connected (and possibly heterogeneous) systems sharing various combinations of resources.

2

Benefits of grid computing

When you deploy a grid, it will be to meet a set of business requirements. To better match grid computing capabilities to those requirements, it is useful to keep in mind some common motivations for using grid computing.

2.1 Exploiting under utilized resources

One of the basic uses of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to a peak in activity. The job in question could be run on an idle machine elsewhere on the grid.

There are at least two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application. For example, a batch job that spends a significant amount of time processing a set of input data to produce an output data set is perhaps the most ideal and simple use case for a grid. If the quantities of input and output are large, more thought and planning might be required to efficiently use the grid for such a job. It would usually not make sense to use a word processor remotely on a grid because there would probably be greater delays and more potential points of failure.

In most organizations, there are large amounts of under utilized computing resources. Most desktop machines are busy less than 5 percent of the time over a business day. In some organizations, even the server machines can often be relatively idle. Grid computing provides a framework for exploiting these under utilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

The processing resources are not the only ones that may be under utilized. Often, machines may have enormous unused disk drive capacity. Grid computing (more specifically, a *data grid*) can be used to aggregate this unused storage into a much larger virtual data store, possibly configured to achieve improved performance and reliability over that of any single machine.

If a batch job needs to read a large amount of data, this data could be automatically replicated at various strategic points in the grid. Thus, if the job must be executed on a remote machine in the grid, the data is already there and does not need to be moved to that remote point. This offers clear performance benefits. Also, such copies of data can be used as backups when the primary copies are damaged or unavailable.

Another benefit of a grid is to better balance resource utilization. An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid-enabled, they can be moved to under utilized machines during such peaks. In fact, some grid implementations can migrate partially completed jobs. In general, a grid can provide a consistent way to balance the loads on a wider federation of resources. This applies to CPU, storage, and any other types of resources that may be available on a grid.

2.2 Parallel CPU capacity

The potential for massive parallel CPU capacity is one of the most common visions and attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others.

The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU-intensive grid application can be thought of as many smaller *subjobs*, each executing on a different machine in the grid. To the extent that these subjobs do not need to communicate with each other, the more *scalable* the application becomes. A perfectly scalable application will, for example, finish in one tenth of the time if it uses ten times the number of processors.

Barriers often exist to perfect scalability. The first barrier depends on the algorithms used for splitting the application among many CPUs. If the algorithm can only be split into a limited number of independently running parts, then that forms a scalability barrier. The second barrier appears if the parts are not completely independent; this can cause contention, which can limit scalability. For example, if all of the subjobs need to read and write from one common file or database, the access limits of that file or database will become the limiting factor in the application's scalability. Other sources of inter-job contention in a parallel grid application include message communications latencies among the jobs, network communication capacities, synchronization protocols, input-output bandwidth to storage or other devices, and other delays interfering with real-time requirements.

There are many factors to consider in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. Furthermore, there are no practical tools for transforming arbitrary applications to exploit the parallel capabilities of a grid. There are some practical tools that skilled application designers can use to write a parallel grid application. However, automatic transformation of applications is a science in its infancy. This can be a difficult job and often requires mathematics and programming talents, if it is even possible in a given situation. New computation-intensive applications written today are being designed for parallel execution, and these will be easily grid-enabled, if they do not already follow emerging grid protocols and standards.

2.3 Virtual resources and virtual organizations for collaboration

Another capability enabled by grid computing is to provide an environment for collaboration among a wider audience. In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing can take these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of resources, as illustrated in Figure 2-1 on page 11. The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger grid.

Sharing starts with data in the form of files or databases. A *data grid* can expand data capabilities in several ways. First, files or databases can span many systems and thus have larger capacities than on any single system. Such spanning can improve data transfer rates through the use of striping techniques. Data can be duplicated throughout the grid to serve as a backup and can be hosted on or near the machines most likely to need the data, in conjunction with advanced scheduling techniques.

Sharing is not limited to files, but also includes other resources, such as specialized devices, software, services, licenses, and so on. These resources are *virtualized* to give them a more uniform interoperability among heterogeneous grid participants.

The participants and users of the grid can be members of several real and virtual organizations. The grid can help in enforcing security rules among them and implement policies, which can resolve priorities for both resources and users.

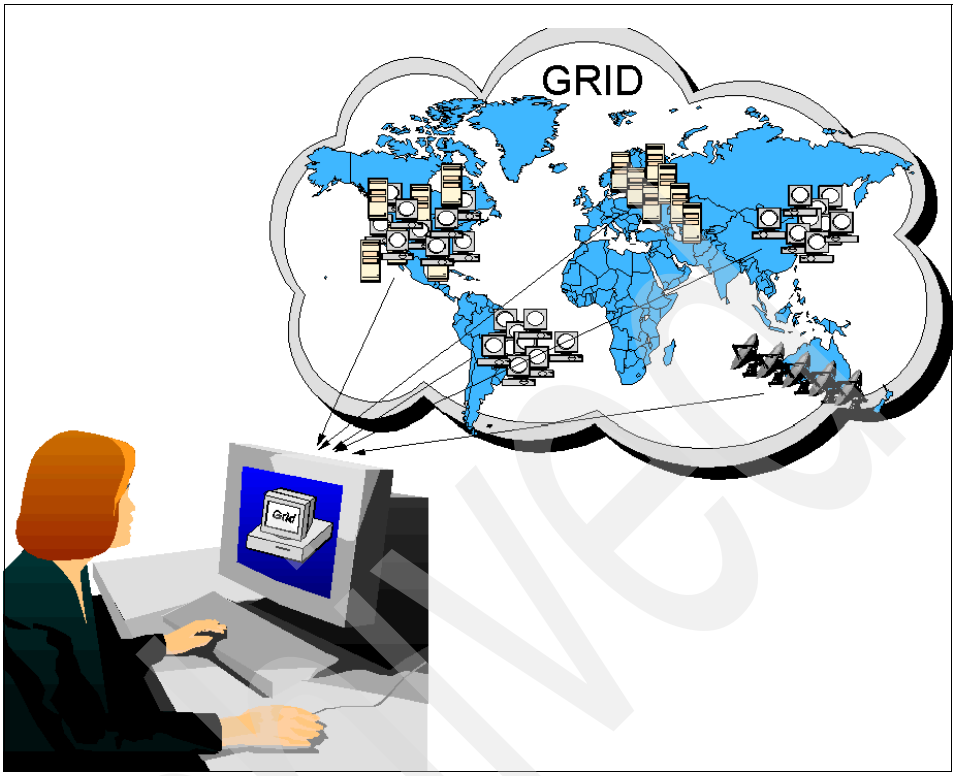

Figure 2-1 The grid virtualizes heterogeneous, geographically disperse resources

2.4 Access to additional resources

As already stated, in addition to CPU and storage resources, a grid can provide access to other resources as well. The additional resources can be provided in additional numbers and/or capacity. For example, if a user needs to increase their total bandwidth to the Internet to implement a data mining search engine, the work can be split among grid machines that have independent connections to the Internet. In this way, total searching capability is multiplied, since each machine has a separate connection to the Internet. If the machines had shared the connection to the Internet, there would not have been an effective increase in bandwidth.

Some machines may have expensive licensed software installed that users require. Users' jobs can be sent to such machines, more fully exploiting the software licenses.

Some machines on the grid may have special devices. Most of us have used remote printers, perhaps with advanced color capabilities or faster speeds. Similarly, a grid can be used to make use of other special equipment. For example, a machine may have a high speed, self-feeding DVD writer that could be used to publish a quantity of data faster. Some machines on the grid may be connected to scanning electron microscopes that can be operated remotely. In this case, scheduling and reservation are important. A specimen could be sent in advance to the facility hosting the microscope. Then the user can remotely operate the machine, changing perspective views until the desired image is captured.

The grid can enable more elaborate access, potentially to remote medical diagnostic and robotic surgery tools with two-way interaction from a distance. The variations are limited only by one's imagination. Today, we have remote device drivers for printers. Eventually, we will see standards for grid-enabled device drivers to many unusual devices and resources. All of these will make the grid look like a large system with a collection of resources beyond what would be available on just one conventional machine.

2.5 Resource balancing

A grid federates a large number of resources contributed by individual machines into a large single-system image. For applications that are grid-enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization, as illustrated in Figure 2-2 on page 13. This feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization. This can happen in two ways:

- An unexpected peak can be routed to relatively idle machines in the grid.
- If the grid is already fully utilized, the lowest priority work being performed on the grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

Without a grid infrastructure, such balancing decisions are difficult to prioritize and execute.

Occasionally, a project may suddenly rise in importance with a specific deadline. A grid cannot perform a miracle and achieve a deadline when it is already too close. However, if the size of the job is known, if it is a kind of job that can be sufficiently split into subjobs, and if enough resources are available after preempting lower priority work, a grid can bring a very large amount of processing power to solve the problem.

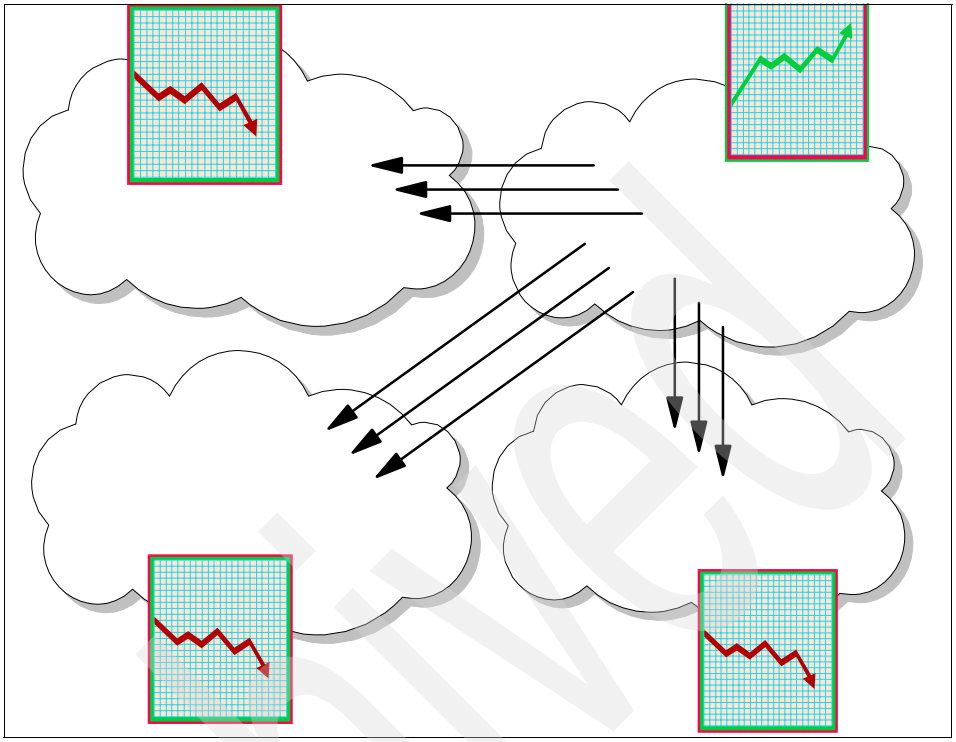

Figure 2-2 Jobs are migrated to less busy parts of the grid to balance loads

Other more subtle benefits can occur using a grid for load balancing. When jobs communicate with each other, the Internet, or with storage resources, an advanced scheduler could schedule them to minimize communications traffic or minimize the distance of the communications. This can potentially reduce communication and other forms of contention in the grid.

Finally, a grid provides excellent infrastructure for brokering resources. Individual resources can be profiled to determine their availability and their capacity, and this can be factored into scheduling on the grid. Depending on the accounting facilities in place, different organizations participating in the grid can build up grid credits and use them at times when they need additional resources. This can form the basis for grid accounting and the ability to more fairly distribute work (and cost) on the grid.

2.6 Reliability

High-end conventional computing systems use expensive hardware to increase reliability. They are built using chips with redundant circuits that vote on results, and contain logic to achieve graceful recovery from an assortment of hardware failures. The machines also use duplicate processors with hot pluggability so that when they fail, one can be replaced without turning the other off. Power supplies and cooling systems are duplicated. The systems are operated on special power sources that can start generators if utility power is interrupted. All of this builds a reliable system, but at a great cost, due to the duplication of expensive components.

In the future, we will see a complementary approach to reliability that relies on software and hardware. A grid is just the beginning of such technology. The systems in a grid can be relatively inexpensive and geographically dispersed. Thus, if there is a power or other kind of failure at one location, the other parts of the grid are not likely to be affected. Grid management software can automatically resubmit jobs to other machines on the grid when a failure is detected. In critical, real-time situations, multiple copies of important jobs can be run on different machines throughout the grid, as illustrated in Figure 2-3 on page 15. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering.

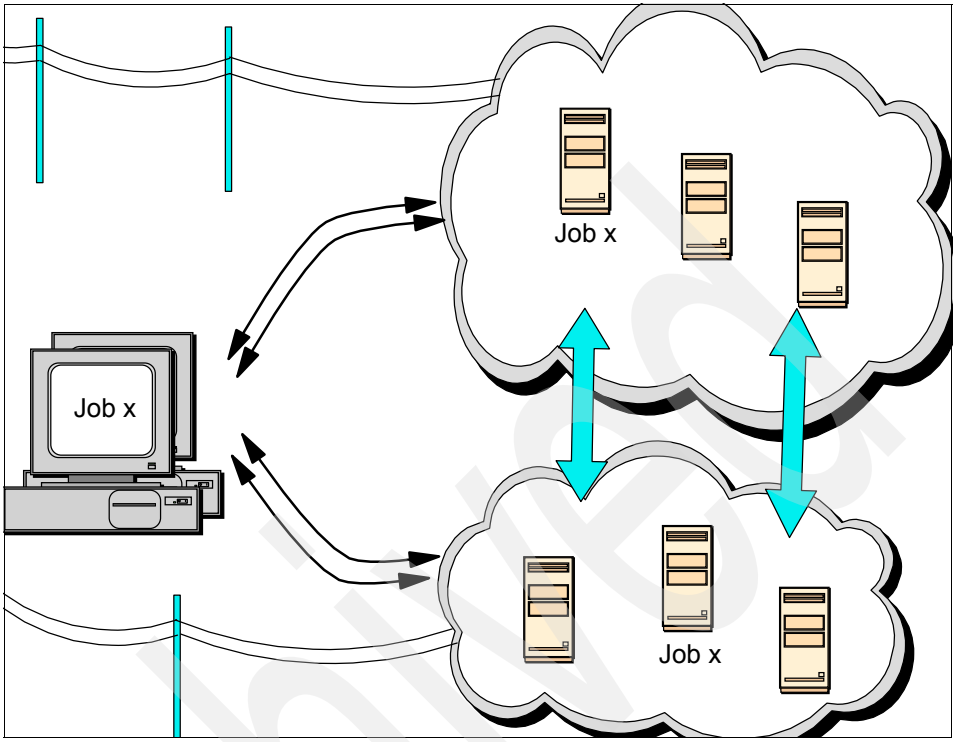

Figure 2-3 Redundant grid configuration

Such grid systems will utilize *autonomic computing*. This is a type of software that automatically heals problems in the grid, perhaps even before an operator or manager is aware of them. In principle, most of the reliability attributes achieved using hardware in today's high availability systems can be achieved using software in a grid setting in the future.

2.7 Management

The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more distributed IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization.

The grid offers management of priorities among different projects. In the past, each project may have been responsible for its own IT resources and the associated expenses. Often these resources might be under utilized while

another project finds itself in trouble, needing more resources due to unexpected events. With the larger view a grid can offer, it becomes easier to control and manage such situations. As illustrated in Figure 2-4, administrators can change any number of policies that affect how the different organizations might share or compete for resources.

Aggregating utilization data over a larger set of projects can enhance an organization's ability to project future upgrade needs. When maintenance is required, grid work can be rerouted to other machines without crippling the projects involved.

Autonomic computing can come into play here too. Various tools may be able to identify important trends throughout the grid, informing management of those that require attention.

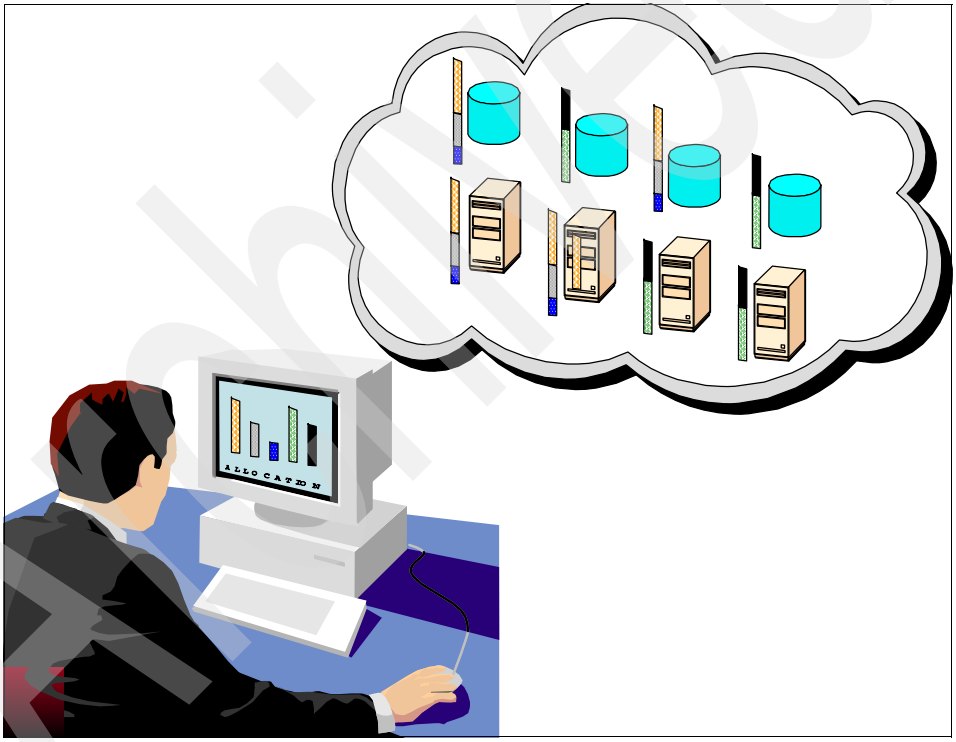

Figure 2-4 Administrators can adjust policies to better allocate resources

2.8 Summary

Grid computing enables organizations (real and virtual) to take advantage of various computing resources in ways not previously possible. They can take advantage of under utilized resources to meet business requirements while minimizing additional costs. The nature of a computing grid allows organizations to take advantage of parallel processing, making many applications financially feasible as well as allowing them to complete sooner.

Grid computing makes more resources available to more people and organizations while allowing those responsible for the IT infrastructure to enhance resource balancing, reliability, and manageability.

3

Grid terms and concepts

In this chapter we introduce a few key grid terms and concepts that we use throughout this book.

3.1 Types of resources

A grid is a collection of machines, sometimes referred to as nodes, resources, members, donors, clients, hosts, engines, and many other such terms. They all contribute any combination of resources to the grid as a whole. Some resources may be used by all users of the grid, while others may have specific restrictions.

3.1.1 Computation

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity. There are three primary ways to exploit the computation resources of a grid.

The first and simplest is to use it to run an existing application on an available machine on the grid rather than locally.

The second is to use an application designed to split its work in such a way that the separate parts can execute in parallel on different processors.

The third is to run an application, that needs to be executed many times, on many different machines in the grid. *Scalability* is a measure of how efficiently the multiple processors on a grid are used. If twice as many processors makes an application complete in one half the time, then it is said to be perfectly scalable. However, there may be limits to scalability when applications can only be split into a limited number of separately running parts or if those parts experience some other interdependencies such as contention for resources of some kind.

3.1.2 Storage

The second most common resource used in a grid is data storage. A grid providing an integrated view of data storage is sometimes called a *data grid*. Each machine on the grid usually provides some quantity of storage for grid use, even if temporary. Storage can be memory attached to the processor or it can be *secondary storage*, using hard disk drives or other permanent storage media. Memory attached to a processor usually has very fast access but is volatile. It would best be used to cache data or to serve as temporary storage for running applications.

Secondary storage in a grid can be used in interesting ways to increase capacity, performance, sharing, and reliability of data. Many grid systems use mountable networked file systems, such as Andrew File System (AFS®), Network File

System (NFS), Distributed File System (DFS™), or General Parallel File System (GPFS). These offer varying degrees of performance, security features, and reliability features.

Capacity can be increased by using the storage on multiple machines with a unifying file system. Any individual file or database can span several storage devices and machines, eliminating maximum size restrictions often imposed by file systems shipped with operating systems. A unifying file system can also provide a single uniform name space for grid storage. This makes it easier for users to reference data residing in the grid, without regard for its exact location. In a similar way, special database software can *federate* an assortment of individual databases and files to form a larger, more comprehensive database, accessible using database query functions.

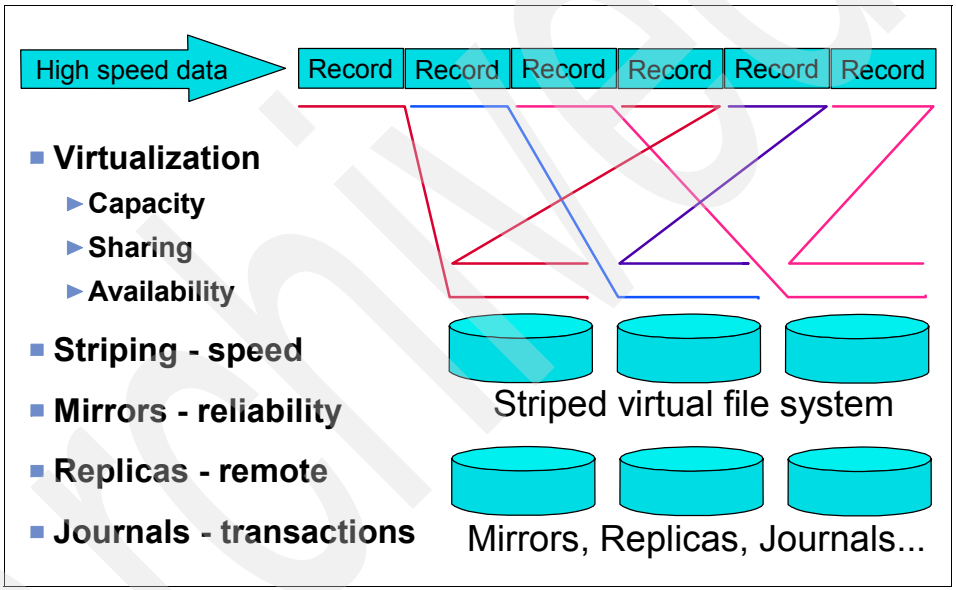

Figure 3-1 Data striping

More advanced file systems on a grid can automatically duplicate sets of data, to provide redundancy for increased reliability and increased performance. An intelligent grid scheduler can help select the appropriate storage devices to hold data, based on usage patterns. Then jobs can be scheduled closer to the data, preferably on the machines directly connected to the storage devices holding the required data.

Data striping can also be implemented by grid file systems, as illustrated in Figure 3-1. When there are sequential or predictable access patterns to data, this technique can create the virtual effect of having storage devices that can

transfer data at a faster rate than any individual disk drive. This can be important for multimedia data streams or when collecting large quantities of data at extremely high rates from CAT scans or particle physics experiments, for example.

A grid file system can also implement journaling so that data can be recovered more reliably after certain kinds of failures. In addition, some file systems implement advanced synchronization mechanisms to reduce contention when data is shared and updated by many users.

3.1.3 Communications

The rapid growth in communication capacity among machines today makes grid computing practical, compared to the limited bandwidth available when distributed computing was first emerging. Therefore, it should not be a surprise that another important resource of a grid is data communication capacity. This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of data to be processed, and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid.

External communication access to the Internet, for example, can be valuable when building search engines. Machines on the grid may have connections to the external Internet in addition to the connectivity among the grid machines. When these connections do not share the same communication path, then they add to the total available bandwidth for accessing the Internet.

Redundant communication paths are sometimes needed to better handle potential network failures and excessive data traffic. In some cases, higher speed networks must be provided to meet the demands of jobs transferring larger amounts of data. A grid management system can better show the topology of the grid and highlight the communication bottlenecks. This information can in turn be used to plan for hardware upgrades.

3.1.4 Software and licenses

The grid may have software installed that may be too expensive to install on every grid machine. Using a grid, the jobs requiring this software are sent to the particular machines on which this software happens to be installed. When the licensing fees are significant, this approach can save significant expenses for an organization.

Some software licensing arrangements permit the software to be installed on all of the machines of a grid but may limit the number of installations that can be simultaneously used at any given instant. License management software keeps track of how many concurrent copies of the software are being used and prevents more than that number from executing at any given time. The grid job schedulers can be configured to take software licenses into account, optionally balancing them against other priorities or policies.

3.1.5 Special equipment, capacities, architectures, and policies

Platforms on the grid will often have different architectures, operating systems, devices, capacities, and equipment. Each of these items represents a different kind of resource that the grid can use as criteria for assigning jobs to machines. While some software may be available on several architectures, for example, PowerPC and x86, such software is often designed to run only on a particular type of hardware and operating system. Such attributes must be considered when assigning jobs to resources in the grid.

In some cases, the administrator of a grid may create a new artificial resource type that is used by schedulers to assign work according to policy rules or other constraints. For example, some machines may be designated to only be used for medical research. These would be identified as having a medical research attribute and the scheduler could be configured to only assign jobs that require machines of the medical research *resource*. Others may participate in the grid only if they are not used for military purposes. In this situation, jobs requiring a *military resource* would not be assigned to such machines. Of course, the administrators would need to impose a classification on each kind of job through some certification procedure to use this kind of approach.

3.2 Jobs and applications

Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing *application* or *job*. Usually we use the term *application* as the highest level of a piece of work on the grid. However, sometimes the term *job* is used equivalently. Applications may be broken down into any number of individual jobs, as illustrated in Figure 3-2 on page 24. Those, in turn, can be further broken down into subjobs. The grid industry uses other terms, such as transaction, work unit, or submission, to mean the same thing as a job.

Jobs are programs that are executed at an appropriate point on the grid. They may compute something, execute one or more system commands, move or collect data, or operate machinery. A grid application that is organized as a

collection of jobs is usually designed to have these jobs execute in parallel on different machines in the grid.

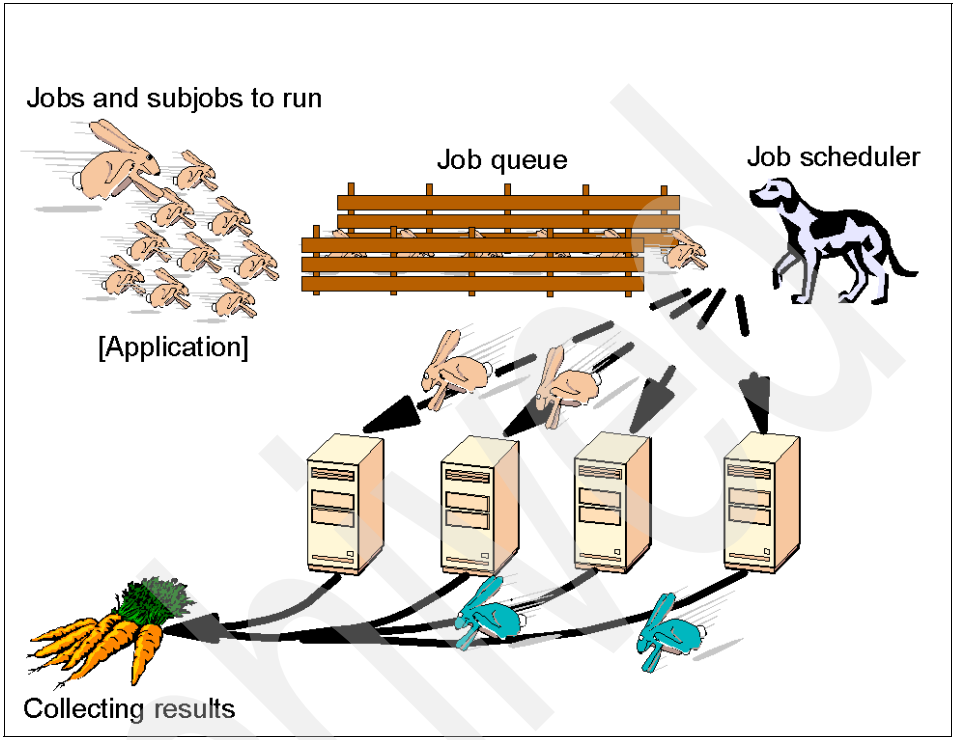

Figure 3-2 An application is one or more jobs that are scheduled to run on grid

The jobs may have specific dependencies that may prevent them from executing in parallel in all cases. For example, they may require some specific input data that must be copied to the machine on which the job is to run. Some jobs may require the output produced by certain other jobs and cannot be executed until those prerequisite jobs have completed executing. Jobs may spawn additional subjobs, depending on the data they process. This work flow can create a hierarchy of jobs and subjobs. Finally, the results of all of the jobs must be collected and appropriately assembled to produce the ultimate output/result for the application.

3.3 Scheduling, reservation, and scavenging

The grid system is responsible for sending a job to a given machine to be executed. In the simplest of grid systems, the user may select a machine suitable for running his job and then execute a grid command that sends the job to the

selected machine. More advanced grid systems would include a job *scheduler* of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid. The term *scheduling* is not to be confused with *reservation* of resources in advance to improve the quality of service. Sometimes the term *resource broker* is used in place of *scheduler*, but this term implies that some sort of bartering capability is factored into scheduling.

In a *scavenging* grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job whose requirements are satisfied by the machine's resources. Scavenging is usually implemented in a way that is unobtrusive to the normal machine user. If the machine becomes busy with local non-grid work, the grid job is usually suspended or delayed. This situation creates somewhat unpredictable completion times for grid jobs, although it is not disruptive to those machines donating resources to the grid.

Grid applications that run in scavenging mode often mark themselves at the operating system's lowest priority level. In this way, they only run when no other work is pending. Due to the performance of modern day processors and operating system scheduling algor, the grid application can run for as short as a few milliseconds, even between a user's keystrokes.

To create more predictable behavior, grid machines are often dedicated to the grid and are not preempted by outside work. This enables schedulers to compute the approximate completion time for a set of jobs, when their running characteristics are known.

As a further step, grid resources can be reserved in advance for a designated set of jobs. Such reservations operate much like a calendaring system used to reserve conference rooms for meetings. This is done to meet deadlines and guarantee quality of service. When policies permit, resources reserved in advance could also be scavenged to run lower priority jobs when they are not busy during a reservation period, yielding to jobs for which they are reserved. Thus, various combinations of scheduling, reservation, and scavenging can be used to more completely utilize the grid.

Scheduling and reservation is fairly straightforward when only one resource type, usually CPU, is involved. However, additional grid optimizations can be achieved by considering more resources in the scheduling and reservation process. For example, it would be desirable to assign executing jobs to machines nearest to the data that these jobs require. This would reduce network traffic and possibly reduce scalability limits. Optimal scheduling, considering multiple resources, is a difficult mathematics problem. Therefore, such schedulers may use heuristics. These heuristics are rules that are designed to improve the probability of finding

the best combination of job schedules and reservations to optimize throughput or any other metric.

3.4 Grid software components

There are many aspects to grid computing that typically are controlled through software. These functions can be handled across a spectrum of very manual procedures to process being handled autonomically through sophisticated software. The software to perform these functions also ranges in capabilities and availability. Over time more sophisticated software will become available, but in many early grids with limited support resources, it makes sense that some of these processes are not implemented completely in software. However, we discus them in the following sections as software that should be considered when designing and deploying a grid environment.

3.4.1 Management components

Any grid system has some management components. First, there is a component that keeps track of the resources available to the grid and which users are members of the grid. This information is used primarily to decide where grid jobs should be assigned.

Second, there are measurement components that determine both the capacities of the nodes on the grid and their current utilization rate at any given time. This information is used to schedule jobs in the grid. Such information is also used to determine the health of the grid, alerting personnel to problems such as outages, congestion, or overcommitment. This information is also used to determine overall usage patterns and statistics, as well as to log and account for usage of grid resources.

Third, advanced grid management software can automatically manage many aspects of the grid. This is known as autonomic computing, or *recovery oriented computing*. This software would automatically recover from various kinds of grid failures and outages, finding alternative ways to get the workload processed.

3.4.2 Distributed grid management

Larger grids may have a hierarchical or other type of organizational topology usually matching the connectivity topology. That is, machines locally connected together with a LAN form a *cluster* of machines. The grid may be organized in a hierarchy consisting of clusters of clusters. The work involved in managing the grid is distributed to increase the scalability of the grid. The collection and grid operation and resource data as well as job scheduling is distributed to match the

topology of the grid. For example, a central job scheduler will not schedule a submitted job directly to the machine that is to execute it. Instead, the job is sent to a lower level scheduler that handles a set of machines (or further clusters). The lower level scheduler handles the assignment to the specific machine. Similarly, the collection of statistical information is distributed. Lower level clusters receive activity information from the individual machines, aggregate it, and send it to higher level management nodes in the hierarchy.

3.4.3 Donor software

Each machine contributing resources typically needs to enroll as a member of the grid and install some software that manages the grid's use of its resources. Usually, some sort of identification and authentication procedure must be performed before a machine can join the grid. Often certificates, such as those available through Certificate Authorities, can be used to establish and ensure the identity of the donor machine as well as the users and the grid itself.

Some grid systems provide their own login to the grid while others depend on the native operating systems for user authentication. In the latter case, a user ID mapping system may be needed to match the user's rights properly on different machines. This typically is manually maintained by a grid administrator. He determines which user ID a given user may possess on each grid machine and enters these IDs in a protected database or registry. In this way, when grid jobs are submitted to different machines for a user, the proper local machine user ID is used for determining the user's rights.

In some grid systems, it is possible to join the grid without any special authentication. And in others, it is possible for any user to submit jobs to the grid. Such systems may be convenient to set up, but should be discouraged in larger deployments due to the serious security problems that they would open up.

The grid system makes information about the newly added resources available throughout the grid. The donor machine will usually have some sort of monitor that determines or measures how busy the machine is and the rate or amount of resources utilized. This information is "bubbled up" to the management software of the grid and used to schedule use of those resources accordingly. In a scavenging system, this information tells the grid management software when the machine is idle and available for work.

Most importantly, the software installed on a given machine can accept an executable job from the grid management system and execute it. A user somewhere on the grid submits a job for execution on the grid. The grid management software must communicate with the grid donor software to send the job there. The donor grid software must be able to receive the executable file or select the proper one from copies pre-installed on the donor machine. The

software is executed and the output is sent back to the requester. More advanced implementations can dynamically adjust the priority of a running job, suspend it, and resume running it later, or checkpoint it with the possibility of resuming its execution on a different machine. These kinds of actions may be necessary to respond to load balancing problems or priority or policy changes in the grid.

3.4.4 Submission software

Usually any member machine of a grid can be used to submit jobs to the grid and initiate grid queries. However, in some grid systems, this function is implemented as a separate component installed on *submission nodes* or *submission clients*. When a grid is built using dedicated resources rather than scavenged resources, separate submission software is usually installed on the user's desktop or workstation.

3.4.5 Schedulers

Most grid systems include some sort of job scheduling software. This software locates a machine on which to run a grid job that has been submitted by a user. In the simplest cases, it may just blindly assign jobs in a round-robin fashion to the next machine matching the resource requirements. However, there are advantages to using a more advanced scheduler.

Some schedulers implement a job priority system. This is sometimes done by using several job queues, each with a different priority. As grid machines become available to execute jobs, the jobs are taken from the highest priority queues first. Policies of various kinds are also implemented using schedulers. Policies can include various kinds of constraints on jobs, users, and resources. For example, there may be a policy that restricts grid jobs from executing at certain times of the day.

Schedulers usually react to the immediate grid load. They use measurement information about the current utilization of machines to determine which ones are not busy before submitting a job. Schedulers can be organized in a hierarchy. For example, a meta-scheduler may submit a job to a cluster scheduler or other lower level scheduler rather than to an individual machine.

More advanced schedulers will monitor the progress of scheduled jobs managing the overall work flow. If the jobs are lost due to system or network outages, a good scheduler will automatically resubmit the job elsewhere. However, if a job appears to be in an infinite loop and reaches a maximum time out, then such jobs should not be rescheduled. Typically, jobs have different kinds of completion codes, some of which are suitable for re-submission and some of which are not.

Reserving resources on the grid in advance is accomplished with a reservation system. It is more than a scheduler. It is first a calendar-based system for reserving resources for specific time periods and preventing any others from reserving the same resource at the same time. It also must be able to remove or suspend jobs that may be running on any machine or resource when the reservation period is reached.

3.4.6 Communications

A grid system may include software to help jobs communicate with each other. For example, an application may split itself into a large number of subjobs. Each of these subjobs is a separate job in the grid. However, the application may implement an algorithm that requires that the subjobs communicate some information among them. The subjobs need to be able to locate other specific subjobs, establish a communications connection with them, and send the appropriate data. The open standard Message Passing Interface (MPI) and any of several variations is often included as part of the grid system for just this kind of communication.

3.4.7 Observation and measurement

We mentioned above that schedulers react to current loads on the grid. Usually, the donor software includes some tools that measure the current load and activity on a given machine using either operating system facilities or by direct measurement. This software is sometimes referred to as a *load sensor*. Some grid systems provide the means for implementing custom load sensors for other than CPU or storage resources.

Such measurement information is useful not only for scheduling, but also for discovering overall usage patterns in the grid. The statistics can show trends that may signal the need for additional hardware. Also, measurement information about specific jobs can be collected and used to better predict the resource requirements of that job the next time it is run. The better the prediction, the more efficiently the grid's workload can be managed.

The measurement information can also be saved for accounting purposes, or to form the basis for grid resource brokering, or to manage priorities more fairly. The information can also be displayed in various forms to better visualize grid activity and utilization.

3.5 Intragrid and intergrid

As already mentioned, the definition of a grid is somewhat subjective. Therefore, the following descriptions of various kinds of grids must be taken loosely.

Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as a hierarchy spanning the world. In this section, we describe some examples in this range of grid system topologies.

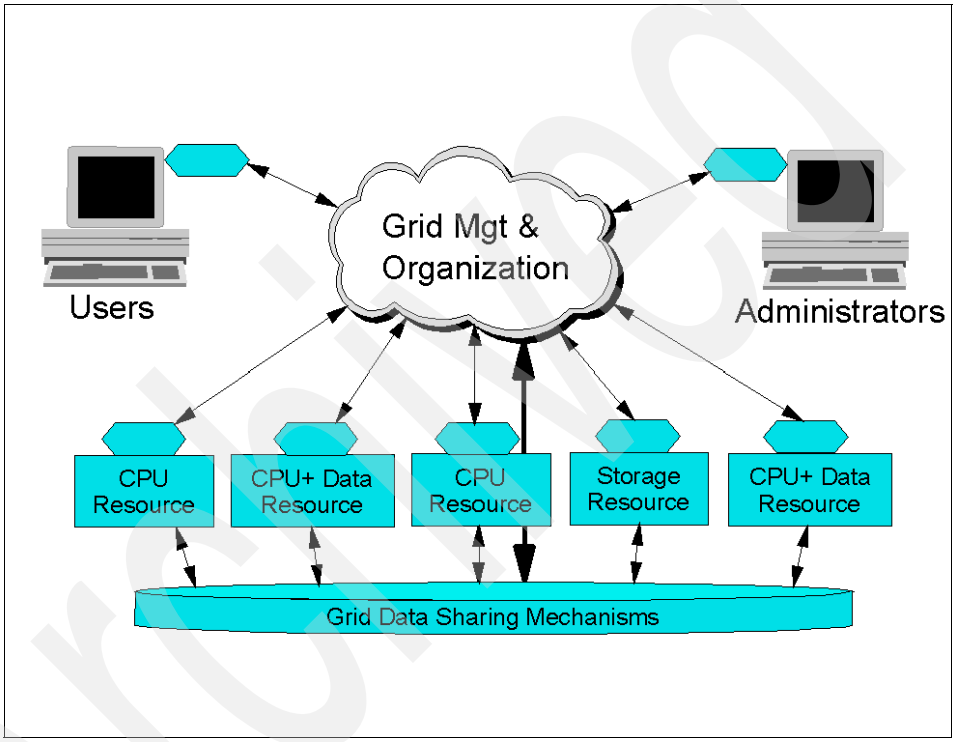

Figure 3-3 A simple grid

As presented in Figure 3-3, the simplest grid consists of just a few machines, all of the same hardware architecture and same operating system, connected on a local network. This kind of grid uses homogeneous systems so there are fewer considerations and may be used for specialized applications. The machines are usually in one department of an organization, and their use as a grid may not require any special policies or security concerns. Because the machines have the same architecture and operating system, choosing application software for these machines is usually simple. Some people would call this a *cluster* implementation rather than a grid.

The next progression would be to include heterogeneous machines. In this configuration, more types of resources are available. The grid system is likely to include some scheduling components. File sharing may still be accomplished using networked file systems. Machines participating in the grid may include systems from multiple departments but within the same organization. Such a grid is also referred to as an *intragrid*.

As the grid expands to many departments, policies may be required for how the grid should be used. For example, there may be policies for what kinds of work is allowed on the grid and at what times. There may be a prioritization by department or by kinds of applications that should have access to grid resources. Also, security becomes more important as more organizations are involved. Sensitive data in one department may need to be protected from access by jobs running for other departments. Dedicated grid machines may be added to increase the quality of service for grid computing, rather than depending entirely on scavenged resources.

The grid may grow geographically in an organization that has facilities in different cities. Dedicated communications' connections may be used among these facilities and the grid. In some cases, VPN tunneling or other technologies may be used over the Internet to connect the different parts of the organization. Security increases in importance once the bounds of any given facility are traversed. The grid may grow to be hierarchically organized to reduce the contention implied by central control, increasing scalability.

Over time, as illustrated in Figure 3-4 on page 32, a grid may grow to cross organization boundaries, and may be used to collaborate on projects of common interest. This is known as an intergrid. The highest levels of security are usually required in this configuration. The intragrid offers the prospect for trading or brokering resources over a much wider audience. Resources may be purchased as a utility from trusted suppliers.

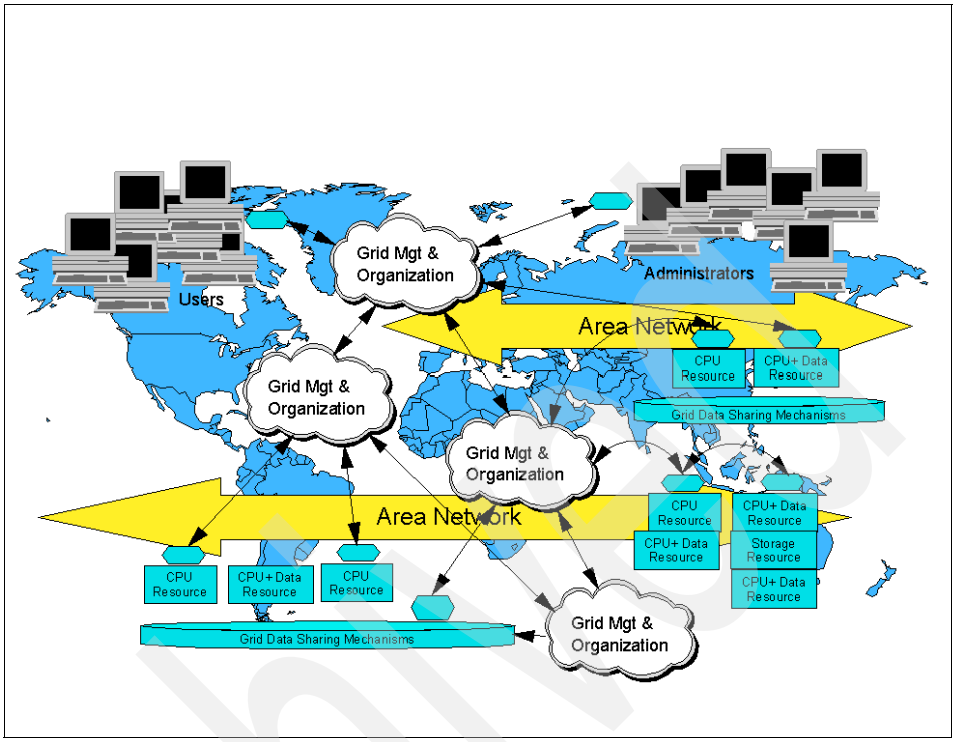

Figure 3-4 A more complex intergrid

3.6 Summary

This chapter provided an overview of some of the key terms and concepts related to grid computing. This information may help as you read this book or other literature on grid computing.

4

Grid user roles

This chapter briefly describes grid computing from the perspectives of the user and the administrator. The architect and application developer are other key roles in a grid environment. Information applicable to those roles are touched on in subsequent chapters.

4.1 Using a grid: A user's perspective

This section describes the typical activities in utilizing a grid from a user's perspective.

4.1.1 Enrolling and installing grid software

A user may first have to enroll in the grid and install the provided grid software on his own machine. He may optionally enroll his machine as a donor on the grid.

Enrolling in the grid may require authentication for security purposes. The user positively establishes his identity with a Certificate Authority. This should not be done solely via the Internet. The Certificate Authority must take steps to assure that the user is in fact who he claims to be. The Certificate Authority makes a special certificate available to software needing to check the true identity of a grid user and his grid requests. Similar steps may be required to identify the donating machine. The user has the responsibility of keeping his grid credentials secure.

Once the user and/or machine are authenticated, the grid software is provided to the user for installing on his machine for the purposes of using the grid as well as donating to the grid. This software may be automatically preconfigured by the grid management system to know the communication address of the management nodes in the grid and user or machine identification information. In this way, the installation may be a one-click operation with a minimum of interaction required on the part of the user. In less automated grid installations, the user may be asked to identify the grid's management node and possibly other configuration information. He may choose to limit the resources donated to the grid, the times that his machine is usable by the grid, and other policy-related constraints. The user may also need to inform the grid administrator which user IDs are his on other machines that exist on the grid.

4.1.2 Logging onto the grid

Most grid systems require the user to log on to a system using an ID that is enrolled in the grid. Other grid systems may have their own grid login ID separate from the one on the operating system. A grid login is usually more convenient for grid users. It eliminates the ID matching problems among different machines. To the user, it makes the grid look more like one large virtual computer rather than a collection of individual machines. Some grid environments may use a proxy login model that keeps the user logged in for a specified amount of time, even if he logs off and back on the operating system and even if the machine is rebooted.

Once logged on, the user can query the grid and submit jobs. Some grid implementations permit some query functions if the user is not logged into the grid or even if the user is not enrolled in the grid.

4.1.3 Queries and submitting jobs

The user will usually perform some queries to check to see how busy the grid is, to see how his submitted jobs are progressing, and to look for resources on the grid. Grid systems usually provide command-line tools as well as graphical user interfaces (GUIs) for queries. Command-line tools are especially useful when the user wants to write a script that automates a sequence of actions. For example, the user might write a script to look for an available resource, submit a job to it, watch the progress of the job, and present the results when the job has finished.

Job submission usually consists of three parts, even if there is only one command required. First, some input data and possibly the executable program or execution script file are sent to the machine to execute the job. Sending the input is called *staging the input data*. Alternatively, the data and program files may be pre-installed on the grid machines or accessible via a mountable networked file system. When the grid consists of heterogeneous machines, there may be multiple executable program files, each compiled for the different machine platforms on the grid. A nice feature provided by some grid systems is to register these multiple versions of the program so that the grid system can automatically choose a correctly matching version to the grid machine that will run the program. Some grid technologies require that the program and input data be first processed or *wrappered* in some way by the grid system. This may be done to add protective execution controls around the application or just to simply collect all of the data files into one.

Second, the job is executed on the grid machine. The grid software running on the donating machine executes the program in a process on the user's behalf. It may use a common user ID on the machine or it may use the user's own user ID, depending on which grid technology is used. Some grid systems implement a protective *sandbox* around the program so that it cannot cause any disruption to the donating machine if it encounters a problem during execution. Rights to access files and other resources on the grid machine may be restricted.

Third, the results of the job are sent back to the submitter. In some implementations, intermediate results can be viewed by the user who submitted the job. In some grid technologies that do not automatically stage the output data back to the user, the results must be explicitly sent to the user, perhaps using a networked file system.

Scripts are also useful for submitting a series of jobs, for a parameter space application, for example. Some computation problems consist of a search for the

desired result based on some input parameters. The goal is to find the input parameters that produce the best desired result. For each input parameter, a separate job is executed to find the result for that value. The whole application consists of many such jobs, which explore the results for a large number of input parameter values. Scripts are usually used to launch the many subjobs, each receiving their own particular parameter values. Parameter inputs can sometimes be more complex than simply a number. Sometimes a different input data set represents the input parameter. Scripts help automate the large variety of more complex parameter space study problems. For simpler parameter space inputs, some grid products provide a GUI to submit the series of subjobs, each with different input parameter values.

When there are a large number of subjobs, the work required to collect the results and produce the final result is usually accomplished by a single program, usually running on the machine at the point of job submission. If there are a very large number subjobs required for an application, the work of collecting the results might be distributed as well. For example, the subjob that submits more subjobs to the grid would be responsible for collecting and aggregating the results of the subjobs it spawned.

4.1.4 Data configuration

The data accessed by the grid jobs may simply be staged in and out by the grid system. However, depending on its size and the number of jobs, this can potentially add up to a large amount of data traffic. For this reason, some thought is usually given on how to arrange to have the minimum of such data movement on the grid.

For example, if there will be a very large number of subjobs running on most of the grid systems for an application that will be repeatedly run, the data they use may be copied to each machine and reside until the next time the application runs. This is preferable to using a networked file system to share this data, because in such a file system, the data would be effectively moved from a central location every time the application is run. This is true unless the file system implements a caching feature or replicates the data automatically.

There are many considerations in efficiently planning the distribution and sharing of data on a grid. This type of analysis is necessary for large jobs to better utilize the grid and not create unnecessary bottlenecks.

4.1.5 Monitoring progress and recovery

The user can query the grid system to see how his application and its subjobs are progressing. When the number of subjobs becomes large, it becomes too difficult to list them all in a graphical window. Instead, there may simply be one

large bar graph showing some averaged progress metric. It becomes more difficult for the user to tell if any particular subjob is not running properly.

A grid system, in conjunction with its job scheduler, often provides some degree of recovery for subjobs that fail. A job may fail due to a:

- Programming error: The job stops part way with some program fault.
- Hardware or power failure: The machine or devices being used stops working in some way.
- Communications interruption: A communication path to the machine has failed or is overloaded with other data traffic.
- Excessive slowness: The job might be in an infinite loop or normal job progress may be limited by another process running at a higher priority or some other form of contention.

It is not always possible to automatically determine if the reason for a job's failure is due to a problem with the design of the application or if it is due to failures of various kinds in the grid system infrastructure. Schedulers are often designed to categorize job failures in some way and automatically resubmit jobs so that they are likely to succeed, running elsewhere on the grid. In some systems, the user is informed about any job failures and the user must decide whether to issue a command to attempt to rerun the failed jobs.

Grid applications can be designed to automate the monitoring and recovery of their own subjobs using functions provided by the grid system software application programming interfaces (APIs).

4.1.6 Reserving resources

To improve the quality of a service, the user may arrange to reserve a set of resources in advance for his exclusive or high-priority use. A calendaring system analogy can be used here. Such a reservation system can also be used in conjunction with planned hardware or software maintenance events, when the affected resource might not be available for grid use.

In a scavenging grid, it may not be possible to reserve specific machines in advance. Instead, the grid management systems may allocate a larger fraction of its capacity for a given reservation to allow for the likelihood of some of the resources becoming unavailable. This must be done in conjunction with tools that have profiled the grid's workload capacity sufficiently to have reliable statistics about the grid's ability to serve the reservation.

4.2 Using a grid: An administrator's perspective

This section describes the typical usage activities in using the grid from an administrator's perspective.

4.2.1 Planning

The administrator should understand the organization's requirements for the grid to better choose the grid technologies that satisfy those requirements. The following sections briefly describe the steps the administrator may take to manage the grid. It is suggested that one should start by deploying a small grid first, to learn about its installation and management, before having to confront more complicated issues involved with a large grid.

The use of a grid is often born from a need for increased resources of some type. One often looks to their neighbor who may have excess capacity in the particular resource. One of the first considerations is the hardware available and how it is connected via a LAN or WAN. Next, an organization may want to add additional hardware to augment the capabilities of the grid. It is important to understand the applications to be used on the grid. Their characteristics can affect the decisions of how to best choose and configure the hardware and its connectivity.

Security

Security is a much more important factor in planning and maintaining a grid than in conventional distributed computing, where data sharing comprises the bulk of the activity. In a grid, the member machines are configured to execute programs rather than just move data. This makes an unsecured grid potentially fertile ground for viruses and trojan horse programs. For this reason, it is important to understand exactly which components of the grid must be rigorously secured to deter any kind of attack. Furthermore, it is important to understand the issues involved in authenticating users and providing proper authorization for specific operations.

Organization

The technology considerations are important in deploying a grid. However, organizational and business issues can be equally important. It is important to understand how the departments in an organization interact, operate, and contribute to the whole. Often, there are barriers built between departments and projects to protect their resources in an effort to increase the probability of timely success. However, by rethinking some of these relationships, one can find that more sharing of resources can sometimes benefit the entire organization. For example, a project that finds itself behind schedule and over budget may not be able to afford the resources required to solve the problem. A grid would give such projects an added measure of safety, providing an extra margin of resource

capacity needed to finish the project. Similarly, a project in its early stages, when computing resources are not being fully utilized, may be able to donate them to other projects in need. A grid also offers the ability for the organization's management to see the bigger picture and react more quickly in shifting resource utilization, priorities, and policies.

4.2.2 Installation

First, the selected grid system must be installed on an appropriately configured set of machines. These machines should be connected using networks with sufficient bandwidth to other machines on the grid. Of prime importance is understanding the fail-over scenarios for the given grid system so that the grid can continue operating even if any of the management machines fail in some way. Machines should be configured and connected to facilitate recovery scenarios. Any critical databases or other data essential for keeping track of the jobs in the grid, members of the grid, and machines on the grid should have suitable backups. Furthermore, public key certificates must be backed up and the private keys must be held in a highly secured place inaccessible by anyone else.

After installation, the grid software may need to be configured for the local network address and IDs. The administrator will usually require root access to the machines managing the grid. In some grid systems, he will also need root access to the donor machines required to install the software on those as well. The software to be installed on the donor machines may need to be customized so that it can find the grid management machines automatically and include pre-installed public keys for the grid. This software may be provided to potential donors on an FTP or equivalent server or be made available on physical media.

Once the grid is operational, there may be application software and data that should be installed on donor machines as well. This software may have specific licensing restrictions that should be understood and adhered to. Some grid systems include tools to assist with grid-wide license management. This can both help in following the rules of the licenses and most efficiently exploit those licenses.

4.2.3 Managing enrollment of donors and users

An ongoing task for the grid administrator is to manage the members of the grid, both the machines donating resources and the users. Users may be further organized as project groups. The administrator is responsible for controlling the rights of the users in the grid. Donor machines may have access rights that require management as well. Grid jobs running on donor machines may be executed under a special grid user ID on behalf of the users submitting the jobs.

The rights of these grid user IDs must be properly set so that grid jobs do not allow access to parts of the donor machine to which the users are not entitled.

As users join the grid, their identity must be positively established and entered in the Certificate Authority. The user and his certificate credentials must be added to the user list using the software appropriate for the grid system deployed. In some cases, the administrator must propagate the user information to several or all grid machines. Also, when the grid system depends primarily on the operating system for user login, the administrator may need to add entries to map the grid user to specific operating system user IDs on the donor machines.

Similar enrollment activity is usually required to enroll donor machines into the grid. The machine's identity is established and registered with the Certificate Authority. The administrator of the grid must have an agreement with the administrator of the donor machine about user IDs, software, access rights, and any policy restrictions. The administrator must enter the machine's identification credentials, addresses, and resource characteristics using the appropriate software for enrolling the donor machine into the grid. In some cases, the administrator may need to manually propagate this information to other machines in the grid.

Corresponding procedures for removing users and machines must be executed by the administrator.

4.2.4 Certificate authority

It is critical to ensure the highest levels of security in a grid because the grid is designed to execute code and not just share data. Thus, it can be fertile ground for viruses, trojan horses, and other attacks if the grid system is compromised in any way. The Certificate Authority is one of the most important aspects of maintaining strong grid security. An organization may choose to use an external Certificate Authority or operate one itself. You must be able to trust the Certificate Authority to strictly adhere to its responsibilities.

The primary responsibilities of a Certificate Authority are:

- Positively identifying entities requesting certificates
- Issuing, removing, and archiving certificates
- Protecting the Certificate Authority server
- Maintaining a namespace of unique names for certificate owners
- Serving signed certificates to those needing to authenticate entities
- Logging activity

Briefly, a Certificate Authority is based on the public key encryption system. In this system, keys are generated in pairs, a public key and a private key. Either one can be used to encrypt some data such that the other is needed to decrypt it.

The private key is guarded by the owner and never revealed to anyone. The public one is given to anyone needing it. A Certificate Authority is used to hold these public keys and to guarantee who they belong to. When a user uses his private key to encrypt something, the receiver uses the corresponding public key to decrypt it. The receiver knows that only that user's public key can decrypt the message correctly. However, anyone could intercept this message and decrypt it because anyone can get the originator's public key. If the originator instead doubly encrypts the message with his private key and the intended recipient's public key, a secure communication link is formed. The receiver uses his private key to decrypt the message and then uses the sender's public key for the second decryption. Now the recipient knows that if the message decrypts properly, then only the sender could have sent it and, furthermore, the sender knows that only the intended receiver can decrypt it. The beauty of all of this is that nobody had to securely carry an encryption key from the sender to the receiver, as must be done for conventional encryption systems, and any tampering with the communication is revealed. A similar exchange is used to get anyone's public key from the Certificate Authority, so that the user knows that he has received an unaltered public key for the desired user.

4.2.5 Resource management

Another responsibility of the administrator is to manage the resources of the grid. This includes setting permissions for grid users to use the resources as well as tracking resource usage and implementing a corresponding accounting or billing system. Usage statistics are useful in identifying trends in an organization that may require the acquisition of additional hardware, reduction in excess hardware to reduce costs, and adjustments in priorities and policies to achieve utilization that is fairer or better achieves the overall goals of an organization.

Some grid components, usually job schedulers, have provisions for enforcing priorities and policies of various kinds. It is the responsibility of the administrator to configure these to best meet the goals of the overall organization. Software license managers can be used in a grid setting to control the proper utilization. These may be configured to work with job schedulers to prioritize the use of the limited licenses.

4.2.6 Data sharing

For small grids, the sharing of data can be fairly easy, using existing networked file systems, databases, or standard data transfer protocols. As a grid grows and the users become dependent on any of the data storage repositories, the administrator should consider procedures to maintain backup copies and replicas to improve performance. All of the resource management concerns apply to data on the grid.

4.3 Summary

When considering whether a grid environment is applicable to a particular organization or set of requirements, two key user perspectives must be considered. First, the end users' perspective and how they will access the grid and gain benefits from using it. Second, how will a grid be administered, especially when resources making up the grid may be distributed both geographically as well as organizationally.

This chapter discussed some of the key points to consider for both of these user roles. Once it is decided that a grid may be the right solution, the architect and application developer will need to be involved to ensure the grid and its related applications are designed and implemented to meet the business' requirements.

Part 2

Grid architecture considerations

5

Standards for grid environments

As we described in Chapter 1, “What grid Computing is” on page 3, grid computing consists of many concepts, and can be defined in many ways. But, at its essence, it provides for distributed computing utilizing virtual resources.

Many technologies could be used to implement such an environment. However, to ensure that various resources across a wide variety of hardware and software platforms can peacefully coexist and interoperate, standards need to be defined and widely adopted.

This chapter describes just a few of the key standards and evolving standards that apply to grid computing.

5.1 Overview

As we have discussed, grid computing assumes and/or requires technologies that include:

- Support for executing programs on a variety of platforms
- A secure infrastructure
- Data movement/replication/federation
- Resource discovery
- Resource management

For each of these areas, there are a variety of technologies available that could be used to address them. We will look at just a few of the standards (both proposed and adopted) that could be considered when architecting a grid-based solution.

Standards bodies that are involved in areas related to grid computing include:

- Global Grid Forum (GGF)

  http://www.ggf.org

- Organization for the Advancement of Structured Information Standards (OASIS)

  http://www.oasis-open.org/

- World Wide Web Consortium (W3C)

  http://www.w3.org/

- Distributed Management Task Force (DMTF)

  http://www.dmtf.org/

- Web Services Interoperability Organization (WS-I)

  http://www.ws-i.org/

5.1.1 OGSA

The Global Grid Forum has published the Open Grid Service Architecture (OGSA). To address the requirements of grid computing in an open and standard way, requires a framework for distributed systems that support integration, virtualization, and management. Such a framework requires a core set of interfaces, expected behaviors, resource models, and bindings.

OGSA defines requirements for these core capabilities and thus provides a general reference architecture for grid computing environments. It identifies the components and functions that are useful if not required for a grid environment. Though it does not go to the level of detail such as defining programmatic

interfaces or other aspects that would guarantee interoperabilty between implementations, it can be used to identify the functions that should be included based on the requirements of the specific target environment.

For more information, refer to:

http://www.ggf.org
http://www.globus.org/ogsa/

5.1.2 OGSI

As grid computing has evolved it has become clear that a service-oriented architecture could provide many benefits in the implementation of a grid infrastructure. The Global Grid Forum extended the concepts defined in OGSA to define specific interfaces to various services that would implement the functions defined by OGSA.

More specifically, the Open Grid Services Interface (OGSI) defines mechanisms for creating, managing, and exchanging information among Grid services. A Grid service is a Web service that conforms to a set of interfaces and behaviors that define how a client interacts with a Grid service.

These interfaces and behaviors, along with other OGSI mechanisms associated with Grid service creation and discovery, provide the basis for a robust grid environment. OGSI provides the Web Service Definition Language (WSDL) definitions for these key interfaces.

Globus Toolkit 3 included several of its core functions as Grid services conforming to OGSI.

For more information, refer to:

http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf

5.1.3 OGSA-DAI

The OGSA-DAI (data access and integration) project is concerned with constructing middleware to assist with access and integration of data from separate data sources via the grid. The project was conceived by the UK Database Task Force and is working closely with the Global Grid Forum DAIS-WG and the Globus team.

For more information, refer to:

http://www.ogsadai.org.uk/

5.1.4 GridFTP

GridFTP is a secure and reliable data transfer protocol providing high performance and optimized for wide-area networks that have high bandwidth. As one might guess from its name, it is based upon the Internet FTP protocol and includes extensions that make it a desirable tool in a grid environment. The GridFTP protocol specification is a proposed recommendation document in the Global Grid Forum (GFD-R-P.020).

GridFTP uses basic Grid security on both control (command) and data channels. Features include multiple data channels for parallel transfers, partial file transfers, third-party transfers, and more.

GridFTP can be used to move files (especially large files) across a network efficiently and reliably. These files may include the executables required for an application or data to be consumed or returned by an application. Higher level services, such as data replication services, could be built on top of GridFTP.

For more information, refer to:

http://www.globus.org/grid_software/data/gridftp.php

5.1.5 WSRF

Web Services Resource Framework (WSRF) is described in more detail in Chapter 9, “Web services resource framework” on page 115. WSRF is being promoted and developed through work from a variety of companies, including IBM, and has been submitted to OASIS technical committees. Basically, WSRF defines a set of specifications for defining the relationship between Web services (that are normally stateless) and stateful resources. WSRF is a general term that encompasses several related proposed standards that cover:

- Resources
- Resource lifetime
- Resource properties
- Service groups (collections of resources)
- Faults
- Notifications
- Topics

As the concept of Grid services evolves, the WSRF suite of evolving standards holds great promise for the merging of Web services standards with the stateful resource management requirements of grid computing.

For more information, refer to:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
http://www.globus.org/wsrf/

5.1.6 Web services related standards

Because Grid services are so closely related to Web services, the plethora of standards associated with Web services also apply to Grid services. We do not describe all of these standards in this document, but rather recommend that the reader become familiar with standards commonly associate with Web services, such as:

- XML
- WSDL
- SOAP
- UDDI

In addition, there are many evolving standards related to Web Services Interoperabilty (WS-I) that also can be applied to and bring value to grid environments, standards, and proposed standards.

For more information, refer to:

http://www.w3.org/2002/ws/
http://www.ws-i.org/

6

Application considerations

As we have already discussed, grid computing environments provide a distributed computing environment utilizing virtualized resources. With this in mind, applications that can take advantage of distributed computing capabilities are possible candidates for grid computing environments. Of course, they may need to be adapted to be able to take advantage of virtual resources through the use of open interfaces and standards such as those discussed in Chapter 5, “Standards for grid environments” on page 45.

This chapter provides an overview of various considerations when contemplating the development or modification of an application to take advantage of grid computing.

6.1 General application considerations

Attention: Aside from application-specific criteria described below, it should be obvious that applications that can most easily take advantage of grid computing should be designed to be portable and to utilize virtual resources. Utilizing open standards such as those described in the previous chapter provides a solid foundation for ensuring such portability.

While a grid-based environment may offer many advantages, any given application may not necessarily benefit from a grid. For example, some personal productivity applications are tightly coupled with a user's interface and do not consume a large amount of computing resources. Running them on a grid may not provide significant benefits. However, other applications may be very suited for exploiting a grid.

If we take a parochial view of the grid as an environment that provides access to vast amounts of computing power, one of the simplest concepts for grid utilization is to be able to run an application somewhere else when your own machine is too busy or otherwise does not have the required resources. Almost any kind of application can be executed in a grid environment this way. You may not see spectacular performance gains unless the machine it runs on is much faster than the machine you usually use.

Applications that can be run in a batch mode are the easiest to execute on other resources within the grid. Applications that need interaction through graphical user interfaces are more difficult to run on a grid, but not impossible. For instance, they can use remote graphical terminal support, such as X Windows or other similar capabilities.

In subsequent sections of this chapter, we discuss many considerations for applications that are CPU intensive or have various requirements associated with data access or sharing. These number-crunching types of applications have historically gained efficiencies by running in a cluster environment or more recently in a grid (that some consider a distributed cluster). However, with advances in grid middleware and the economic incentives to run more typical business applications on virtualized resources, there is a trend towards understanding how these business applications can be implemented (or modified) to take advantage of the various resources provided by a grid computing environment.

For this discussion, let us consider a grid environment to be distributed computing on virtualized resources. Distributed computing concepts are well known and most application architects and developers understand what it takes to enable an application to execute and take advantage of a distributed

computing environment. What is new, is the utilization of virtualized resources. This implies that the application developer may not know (and optimally should not need to know) what operating platform or other resources (network, storage, and so on) will be utilized by the application at run time. The more an application can be written to be independent of actual physical resources, the more likely it can take advantage of a grid environment by running on any available resources that provide the required services and conform to any applicable policies.

A key aspect of writing applications that are independent of physical resources is the conformance to widely adopted standards. For instance, applications that are implemented as services and can be deployed to any compliant J2EE container have little or no dependency on the underlying hardware or operating system. Therefore, they could potentially be deployed to and run on any systems within the grid that have a compliant J2EE container.

Likewise, using standardized interfaces for access to storage, databases, and network communications provides the portability required to run on virtualized resources independent of their physical makeup.

Applications specifically designed to use multiple processors or other federated resources of a grid will benefit most. The following discussion is designed to stimulate analysis, which will show how various factors may help decide whether a given application should be deployed on a grid and what modifications, if any, might be considered.

6.2 CPU-intensive application considerations

To determine if existing or planned applications that are CPU intensive can take advantage of a grid environment requires many considerations. This section describes some things to consider related to the possible applicability of a grid to these applications.

Probably the most important step in grid enabling an application is to determine whether the calculations can be done in parallel. While High Performance Computing (HPC) clusters are sometimes used to handle the execution of applications that can utilize parallel processing, grids provide the ability to run these applications across a heterogeneous, geographically dispersed set of clusters. Rather than run the application on a single homogenous cluster, the application can take advantage of the larger set of resources in the grid. If the algorithm is such that each computation depends on the prior calculation, then a new algorithm (if possible) may be beneficial. Not all problems can be converted into parallel calculations. As an oversimplified example, let us take the process of adding up a large list of numbers. The simple serial program may be written to start with the sum of zero and then add each of the numbers, one at a time, until

the final sum is reached. Here each calculation depends on the prior one. However, we can observe that the associative property of arithmetic shows us that we could break the list up into seven pieces, for example, with seven separate programs adding up the numbers in each list, and then a final eighth program adding the 7 sums to form the final answer. This is illustrated by Figure 6-1.

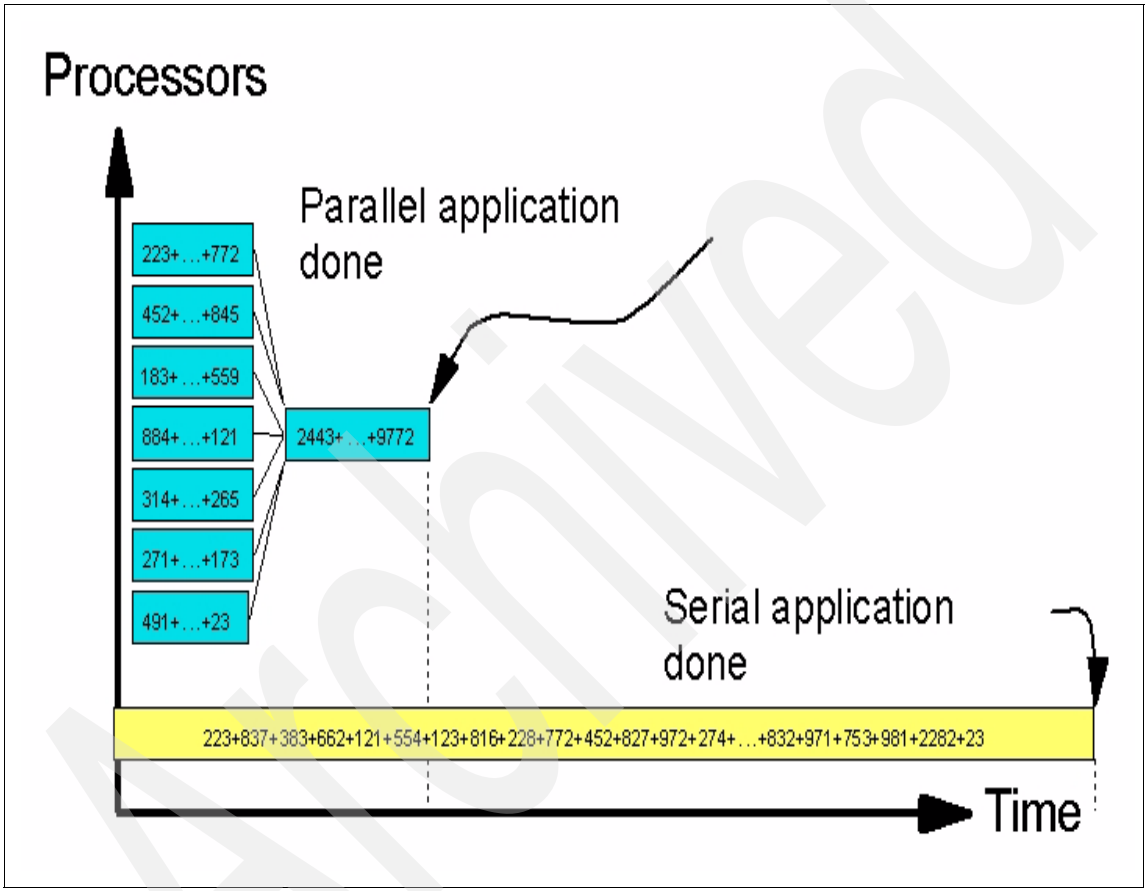

Figure 6-1 Rearranging computations to execute in parallel

On the other hand, some computations cannot be rewritten to execute in parallel. For example, in physics, there are no simple formulas that show where three or more moving bodies in space will be after a specified time when they gravitationally affect each other. These kinds of computations are done by simulating the motions of the bodies, applying Newton's (or Einstein's) laws to small time increments, and computing how the forces and bodies affect each other, given the new position of the objects after each tiny time increment, as illustrated in Figure 6-2 on page 55.

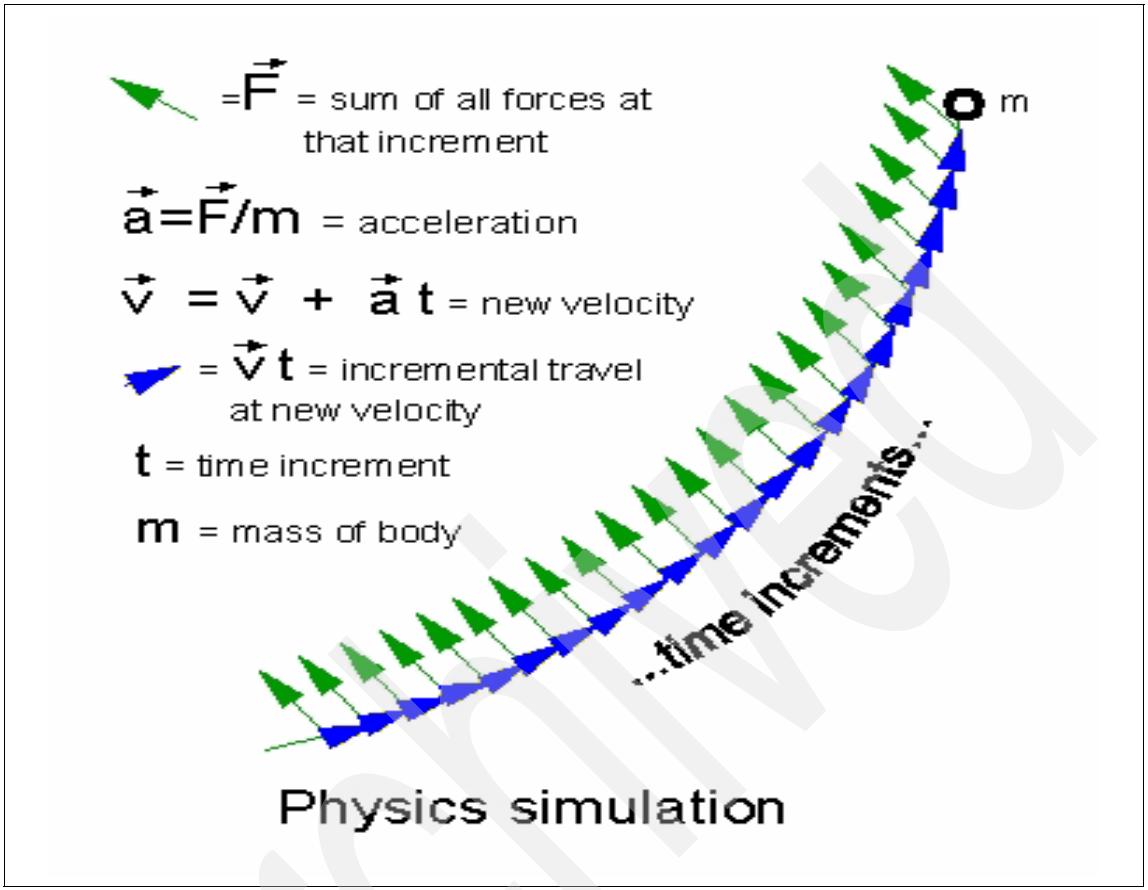

Figure 6-2 Simulation that cannot be made parallel but needs to run many times

This is repeated a great number of times until the desired time is reached. Each computation depends on the prior one. If it did not, then we would have used a different formula or algorithm to begin with. Because the time increments are not infinitely small, after many increments, small errors start adding up. The final computed position of the objects can be in error, perhaps ultimately causing a spacecraft to crash into a planet instead of going into orbit. To improve accuracy in such computations, we make the time increments much shorter. This increases the number of these increments to be computed, and thus the overall computation time. Many simulations suffer from this type of difficulty.

As we saw above, in the list-adding example, such computations can be performed in parallel, while others, such as the 3-body physics problem, cannot. Often, an application may be a mix of independent computations as well as dependent computations. One needs to analyze the application to see if there is

a way to split some subset of the work. Drawing a program flow graph and a data dependency graph can help in analyzing whether and how an application could be separated into independently running parallel parts.

Going back to the space object example, let us say we are trying to find the correct trajectory to aim a rocket so that it loops around Venus, and then Earth, to reach Jupiter more quickly. We might try calculating to see what happens for a large number of different trajectories, pointing the rocket in slightly different directions and firing the engines for different durations. Each trajectory can be thought of as a separate calculation, and then in the end, a program chooses the best one. Here, we are able to perform work in parallel, even though the underlying computation for a single trajectory may be serial. Applications that consist of a large number of independent subjobs are very suitable for exploiting grid CPU resources. These are sometimes called parameter space searches.

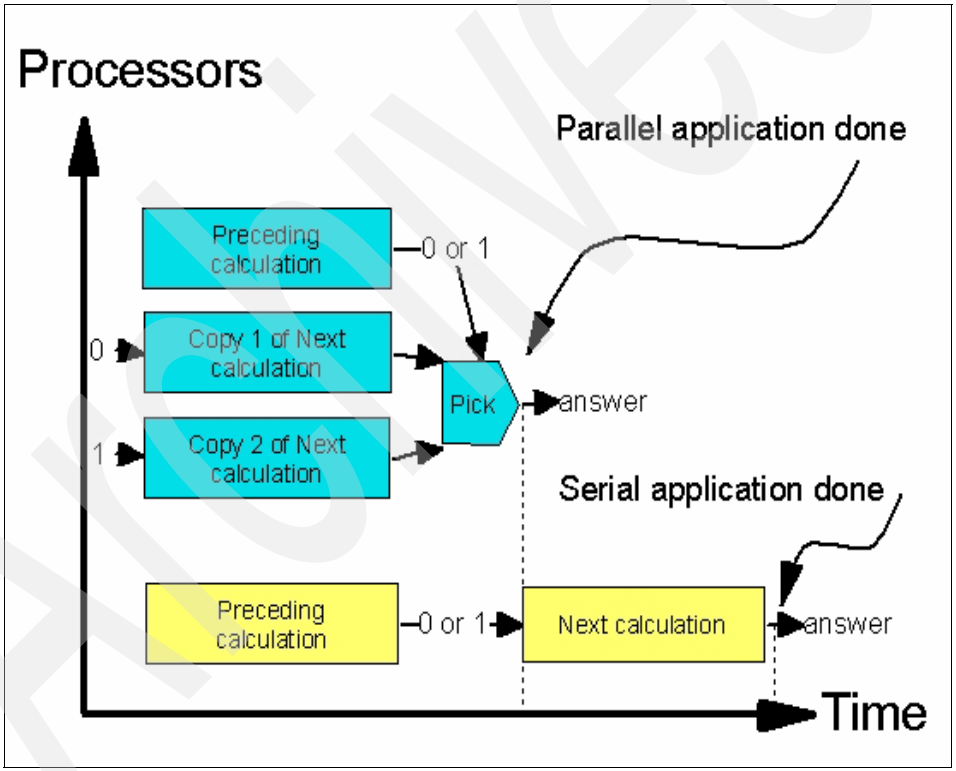

Figure 6-3 Redundant speculative computation to reduce latency

Another approach to reducing data dependency on prior computations is to look for ways to use redundant computations. If the dependency is on a subset of the prior computations, it may be beneficial just to have each successive

computation that needs the results of the prior computation recompute those results instead of waiting for them to arrive from another job. If the dependency is on a computation that has a yes/no answer, perhaps it is better to compute the next calculations for both of the yes and no cases and throw away the wrong choice when the dependency is finally known, as illustrated in Figure 6-3 on page 56. This technique can be taken to extremes in various ways. For example, for two bits of data dependency, we could make four copies of the next computation with all four possible input values. This can proceed to copies of the next calculation for N bits of data dependency. As N gets large, it quickly becomes too costly to compute all possible computations. However, we may speculate and only perform the copies for the values we guess might be more likely to be correct. If we did not guess the correct one, then we simply end up computing it in series, but if we guessed correctly it saves us overall real time. Here heuristics (rules of thumb) could be developed to make the best possible guesses. Furthermore, there may be many points in the application where we could use the speculative approach, and if our guess rate is high enough, there might be an overall improvement in efficiency and parallelism. This same kind of speculative computing is used to improve the efficiency inside CPUs by executing both branches of a condition until the correct one is determined.

Some parameter space problems are finite in nature, and some are infinite or so large that all possible parameter inputs cannot be examined. For these kinds of parameter space problems, it is useful to use additional heuristics to select which parts of the parameter space to try. This may not lead to the absolute best solution, but it may be close enough. The traveling salesman problem can be intractable in this way when there are many cities to be visited. However, various heuristics can be used to get reasonably close to an optimal solution. It may not be worth a month of additional computation to improve the answer from 98 percent to 99 percent efficiency.

It may be acceptable to explore only a small part of the parameter space. One approach is to try a reasonable number of randomly scattered points in the problem's parameter space first. Then one would try small changes in the parameters around the best points that might lead to a better solution. This technique is useful when the parameter space relates relatively smoothly to changes in the result.

By analogy, this can be described as *hill climbing*. To find the highest altitude point in a perpetually fog-shrouded region of land on which to build a television broadcast antenna, you would put a set of people at random on the terrain. Then each would climb to the highest point near them. Whomever reached the highest point would then be declared to have found the highest hill in the land. They may not have found the absolute highest point if nobody started near that point, but they will probably find the nearly highest hill or one that is sufficient for their

antenna tower. This kind of technique is useful when there are too few people and too many hills to visit all of them.

Often, mathematical calculations are commutative, associative, or linear in some way. The simple adding of a list of numbers example illustrates this. By altering some potentially unimportant rules in the computations involved in a calculation, we may be able to break the ordering requirement and thus make it possible to execute more of the application in parallel. For example, in a bank account, deposits and withdrawals are serially calculated and if the account ever goes negative, then the transaction may be rejected, a fine may be imposed, or the account may be frozen. If, however, the bank changes its rules and says that the account must simply be positive at the end of the day, then withdrawals processed before the deposits would not cause a problem and all of these calculations could be broken up into separate, parallel-running jobs.

Many times, an application that was written for a single processor may not be organized or use algorithms or approaches that are suitable for splitting into parallel subcomputations. An application may have been written in a way that makes it most efficient on a single processor machine. However, there may be other methods or algorithms that are not as efficient, yet may be much more amenable to being split into independently running subcomputations. A different algorithm may *scale* better because it can more efficiently use larger and larger numbers of processors. Thus, another approach for grid enabling an application is to revisit the choices made when the application was originally written. Some of the discarded approaches may be better for grid use.

How you go about solving a problem may be quite different, depending on whether it is unique to be solved only once versus being solved repeatedly with different inputs. One might use a less efficient but more straightforward technique if the problem is only to be solved once, reducing debug time and making good use of a grid's ability to absorb momentary peaks of activity. On the other hand, if it is a one- time problem, but is going to take a year of execution, more thought should be put into the problem before proceeding. The following are some additional things to think about.

Is there any part of the computation that would be performed more than once using the same data? If so, and if that computation is a significant portion of the overall work, it may be useful to save the results of such computations.

If we find that an application performs some sets of computations on the same input data every time it is run, produces the same output data, and takes a significant amount of time computing this output, how much output data would need to be saved to avoid the computation the next time? If there is a very large amount of output data, it may be prohibitive to save this data. Perhaps there are a large number of similar computations that might be saved. Even if any one computation's results do not represent a large amount of data, the aggregate for

all of them might. One needs to consider this time-space trade-off for the application. One could presumably save space and time by only saving the results for the most frequently occurring situations. For example, in world class chess playing programs, the opening positions of the game of chess are usually stored in a database containing the best move to take in each such position. This information can be precomputed to a large extent and can save large amounts of computation time during a chess tournament. However, the number of possible chess board positions increases very rapidly with more moves into the game, so only the early move positions of the game or the end-game moves when there are few pieces left, are precomputed and saved.

In a distributed application, partial results or data dependencies may be met by communicating among subjobs. That is, one job may compute some intermediate result and then transmit it to another job in the grid. If possible, one should consider whether it might be more efficient to simply recompute the intermediate result at the point where it is needed rather than waiting for it from another job. One should also consider the transfer time from another job, versus retrieving it from a database of prior computations.

6.3 Data considerations

When considering applications that may be split into multiple parts for execution on a grid, it is important to consider the amounts of data that are needed to be sent to the node performing a calculation and the time required to send it. If the application can be split into small work units requiring little input data and producing small amounts of output data, that would be most ideal. The data in this kind of case is said to be *staged* to the node doing the work. Sending this data along with the executable file to the grid node doing the work is part of the function of most grid systems. However, in many applications, larger amounts of input and/or output data are involved, and this can cause complications and inefficiencies.

When the grid application is split into subjobs, often the input data is a large fixed set of data. This offers the opportunity to share this data rather than staging the entire set with each subjob. However, one must consider that even with a shared mountable file system, the data is being sent over the network. The goal is to locate the shared data closer to the jobs that need the data. If the data is going to be used more than once, it could be replicated to the degree that space permits.

If more than one copy of the data is stored in the grid, it is important to arrange for the subjobs to access the nearest copy per the configuration of the network. This highlights the need for an information service within the grid to track this form of data awareness. Furthermore, one must be careful that the network does not become the bottleneck for such a grid application. If each subjob processes

the data very quickly and is always waiting for more data to arrive, then sharing may not be the best model if the network data transfer speed to each subjob does not at least match disk speeds.

Shared data may be fixed or changing. For example, a database may contain the latest known gene sequences and be constantly growing. However, applications using this data may not need the latest gene sequence data the instant that it is available. This makes it easier and more efficient to share such a database because the updates to it can be batched and processed at off-peak usage times rather than contending with concurrent access by applications. Furthermore, if more than one copy of this data exists, and all of the copies do not need to be simultaneously updated, this improves performance because all applications using the data would not need to be stopped while updating the data. Only those accessing a particular copy would need to be stopped or temporarily paused.

When a file or a database is updated, jobs cannot simultaneously read the portion of the file concurrently being updated by another job. Locking or synchronizing primitives are typically built into the file system or database to automatically prevent this. Otherwise, the application might read partially updated data, perhaps receiving a combination of old and new data.

In some shared data situations, updates must not be delayed. For example, if the subjobs are processing financial transactions, they must be immediately updated in the master balances database. Furthermore, if there are copies of this database elsewhere, they must all be updated with each new item simultaneously. A number of scaling issues come into play here. There can be a large amount of data synchronization communications among jobs and databases. The synchronization primitives can become bottlenecks in overall grid performance. It is important to consider how the database activity can be partitioned so that there is less interference among the parts and thus less potential synchronization contention among those parts.

Applications that access the data they need serially are more predictable, so various techniques can be used to improve their performance on the grid. If each subjob needs to access all of the data, then shared copies might be desirable. Multiple copies of the data should be considered if bringing the data closer to the nodes running the subjobs would help. If each part of the data is examined only once, then copies may not be desirable. However, if the access is serial, some of the retrieval time can be overlapped with processing time. There could be a thread retrieving the data that will be needed next while the data already retrieved is being processed. This can even apply to randomly accessed data, if there is the ability to do some prediction of which portions of data will be needed next.

One of the most difficult problems with duplicating rapidly changing databases is keeping them in synchronization. The first step is to see if rapid synchronization

is really needed. Can the application be modified to work around this? If not, the synchronization mechanisms themselves may need to be changed. If the rapidly changing data is only a subset of the database, memory versions of the database might be considered. Network communication bandwidth into the central database repository could also be increased. Is it possible to rewrite the application so that it uses a data flow approach rather than the central state of a database? Perhaps it can use self-contained transactions that are transmitted to where they are needed. The subjobs could use direct communications between them as the primary flow for data dependency rather than passing this data through a database first.

In some applications, various database records may need to be updated atomically or in concert with others. Locking or synchronization primitives are used to lock all of the related database entries, whether they are in the same database or not, and then are updated while the synchronization primitives keep other subjobs waiting until the update is finished. One should look for ways to minimize the number of records being updated simultaneously to reduce the contention created by the synchronization mechanism. One should exercise caution not to create situations that might cause a synchronization deadlock with two subjobs waiting for each other to unlock a resource the other needs. There are three ways that are usually used to prevent this problem:

- The first is the easiest, but can be the most wasteful. This is to have all waits for resources to include time-outs. If the time-out is reached, then the operation must be undone and started over in an attempt to have better luck at completing the transaction.
- The second is to lock all of the resources in a predefined order ahead of the operation. If all of the locks cannot be obtained, then any locks acquired should be released, and then, after an optional time period, another attempt should be made.
- The third is to use deadlock detection software. A transitive closure of all of the waiters is computed before placing the requesting task into a wait for the resource. If it would cause a deadlock, the task is not put into a wait. The task should release its locks and try again later. If it would not cause a deadlock, the task is set to automatically wait for the desired resource.

It may be necessary to run an application redundantly for reliability reasons, for example. The application may be run simultaneously on geographically distinct parts of the grid to reduce the chances that a failure would prevent the application from completing its work or prevent it from providing a reliable service. If the application updates databases or has other data communications, it would need to be designed to tolerate redundant data activity caused by running multiple copies of the application. Otherwise, computed results may be in error.

6.4 Summary

Portability and the capability to take advantage of virtual resources are key attributes of an application that can take advantage of grid computing. As grid technologies and environments advance, more and more applications will be able to take advantage of the grid.

In general, applications and their requirements should be analyzed to understand how they could be designed and developed to reap the benefits from a grid. However, in many cases today, organizations are looking to identify specific applications that they could adapt quickly to a grid environment to gain immediate benefits and to gain experience and knowledge around grid computing. This chapter has described some of the attributes of applications and data access patterns that more easily lend themselves to grid computing.

7

Security

One of the key questions that usually arises when considering a grid environment is security. This chapter describes security issues, techniques, and solutions needed to provide a robust and secure grid computing environment.

The information presented up to this point in this book has been generic and not specific to a particular grid environment. Many of the security issues and topics described here are also general in nature. However, some of our examples and discussion are made clearer when we can use a specific implementation, so this chapter provides some information that is specific to the Globus Toolkit 4. We introduce Globus Toolkit 4 in more detail in Part 3, “Creating a grid environment with the Globus Toolkit 4” on page 139. Though we have included these specific examples, the general concepts and requirements apply to other environments as well.

7.1 Introduction to grid security

Security requirements are fundamental to the grid design. The basic security components are comprised of mechanisms for authentication, authorization, and confidentiality of communication between grid computers. Without this functionality, the integrity and confidentiality of the data processed within the grid would be at risk. To properly secure your grid environment, there are many different tools and technologies available. This chapter examines some of those technologies.

In order to better understand grid security, it is best to start with some basic grid security requirements and security fundamentals. Grid security builds on well-known security standards. We discuss general security requirements followed by security fundamentals. In this chapter, we discuss the nuts and bolts of grid security and the underlying technologies that allow for grid security to work.

7.1.1 Grid security requirements

A virtual organization is one of the fundamental concepts in a grid environment today. A virtual organization (VO) is defined as a dynamic group of individuals, groups, or organizations who define the conditions and rules (business objectives and policies) for sharing resources.

A grid environment is required to coordinate resource management and sharing within a VO that potentially spans multiple organizations. This implies that a grid application may span multiple administrative domains. Each of these domains would have its own business requirements and policies to adhere to. A grid security infrastructure is required to comply with local domain-level security policies and VO-defined policies. To achieve this requirement the grid security infrastructure requires interoperability amongst various domains while maintaining a clear separation of the security policies and mechanisms deployed by both virtual and real organizations.

The *Security Architecture for Open Grid Services* by Nagaratnam, et. al., 2002; (http://www.cs.virginia.edu/~humphrey/ogsa-sec-wg/OGSA-SecArch-v1-07192002.pdf) summarizes the following security challenges in a grid environment:

- Integration

  The grid security infrastructure is required to integrate with existing security infrastructures across platforms and hosting environments. The overall grid security architecture is required to be implementation agnostic and be extensible to incorporate new security services as they become available.

- Interoperability

The Grid services that traverse multiple domains and hosting environments need to be able to interact with each other to allow domains to exchange messages (for example, via SOAP/HTTP), allow each party to specify security policy applied to a secure conversation, and provide mechanisms to identify a user from one domain in another domain.

- Trust Relationship

  A Grid service request can span multiple security domains. The security domains involved to meet a Grid service request require establishing trust with each other. Due to the dynamic nature of a grid environment, it is unfeasible to establish end-to-end trust prior to execution of an application. The issue of trust establishment becomes complicated with transient Grid services.

At a high level the grid security requirements can be defined as follows:

Authentication	Providing interfaces to plug-in different authentication mechanisms and means to convey the mechanism used.
Delegation	Providing mechanisms to allow delegation of access rights from requesters to services while ensuring that the access rights delegated are restricted to the tasks intended to be performed within policy restrictions.
Single logon	This refers to relieving an authenticated entity from re-authentication for a certain period of time when subsequent access to grid resources are requested while taking multiple security domains and identity mappings into account.
Credential life span and renewal	Ability to refresh requester credentials if a grid application operation takes longer to complete than the life-span of a delegated credential.
Authorization	Ability to control access to grid components based on authorization policies.
Privacy	Allowing both a service requester and a service provider to define and enforce privacy policies.
Confidentiality	Protect confidentiality of underlying transport and message content and

	between OGSA-compliant components in either point-to-point or store and forward mechanisms.
Message integrity	Ensuring unauthorized changes made to message content or data can be detected at the recipient end.
Policy exchange	Allows security context negotiation mechanism between service requesters and service providers based on security policy information.
Secure logging	Provides a foundation for non-repudiation and auditing that enables all services to time-stamp and log various types of information without interruption or information alteration by adverse agents.
Assurance	Provides means to qualify the security assurance level that can be expected of a hosting environment. The security assurance level indicates the types of security services provided by an environment. This information is useful in deciding whether to deploy a service in the environment.
Manageability	This requirement mainly deals with various security service management issues such as identity management, policy management, and so on.
Firewall traversal	Ability to traverse firewalls without compromising local control of firewall policy to enable cross-domain grid computing environment.
Securing the OGSA infrastructure	This refers to securing core OGSA components.

The diagram below gives a high-level view of various components of a grid security model that addresses the requirements described above.

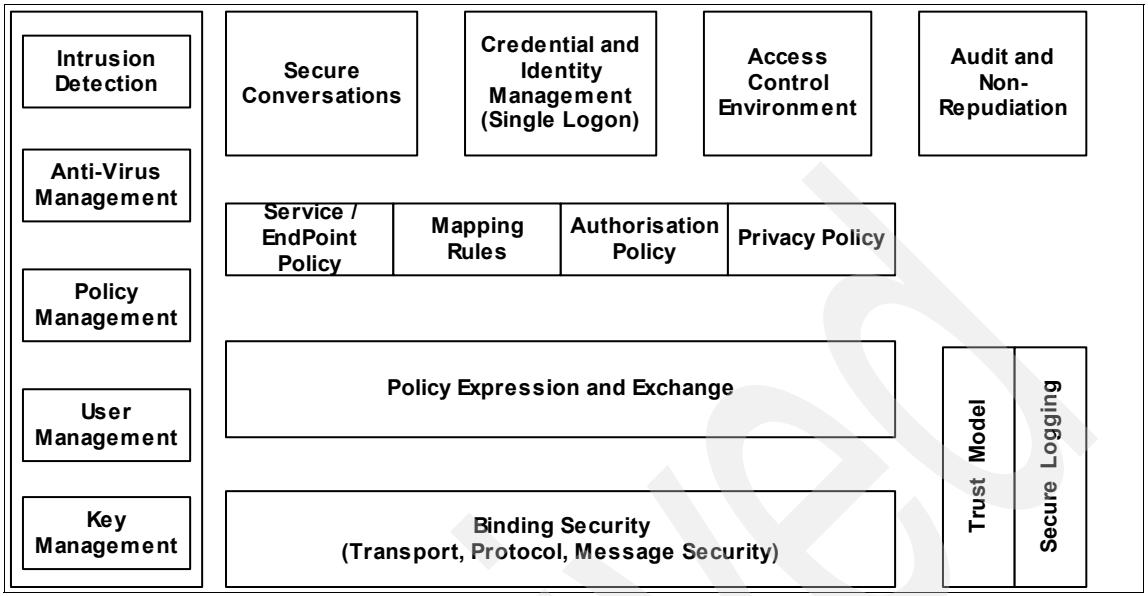

Figure 7-1 Grid security model

This grid security model abstracts enterprise security services as a single model to enable organizations to utilize their existing security infrastructure to communicate with other enterprises that uses different technology.

Please refer to the *The Security Architecture for Open Grid Services* by Nagaratnam, et. al., for a detailed discussion of each of the components shown in the figure above.

7.1.2 Security fundamentals

Security requires three fundamental services: Authentication, authorization, and encryption. A grid resource must be authenticated before any checks can be done as to whether any requested access or operation is allowed within the grid. Once the user has been authenticated within the grid, the grid user can be granted certain rights to access a grid resource. This, however, does not prevent data in transit between grid resources from being captured, spoofed, or altered. The security service to insure that this does not happen is encryption.

The world of security has its own set of terminology. The International Organization for Standardization (ISO) has defined the common security services found in modern IT systems. The list was first put in ISO 7498-2 (OSI Security Architecture) and later updated in ISO 10181 (OSI Security

Frameworks). To have a better understanding of security systems and services, some security terms with explanations are listed below:

Authentication Authentication is the process of verifying the validity of a claimed individual and identifying who he or she is. Authentication is not limited to human beings; services, applications, and other entities may be required to authenticate also.

Access control Assurance that each user or computer that uses the service is permitted to do what he or she asks for. The process of authorization is often used as a synonym for access control, but it also includes granting the access or rights to perform some actions based on access rights.

Data integrity Data integrity assures that the data is not altered or destroyed in an unauthorized manner.

Data confidentiality Sensitive information must not be revealed to parties that it was not meant for. Data confidentiality is often also referred to as privacy.

Key management Key management deals with the secure generation, distribution, authentication, and storage of keys used in cryptography.

The Grid Security Infrastructure (GSI) provided as part of the Globus Toolkit and a Public Key Infrastructure (PKI) provide the technical framework (including protocols, services, and standards) to support grid computing with five security capabilities: User authentication, data confidentiality, data integrity, non-repudiation, and key management.

7.1.3 Important grid security terms

During the course of this chapter, we go over many important security terms. While some of the terms covered within this section provide the background as to how grid security works, there are some important concepts that should be highlighted. This is due to the fact that some areas within grid security require a precise understanding of the security concepts. Also, some security components may work slightly differently within a grid environment as opposed to a standard network. Below are some important security concepts that you should be aware of when reading this chapter. These concepts are described in greater detail throughout the chapter.

- Symmetric encryption: Using the same secret key to provide encryption and decryption of data.

- Asymmetric encryption: Using two different keys for encryption and decryption. The public key encryption technique is the primary example of this using a *public key* and a *private key* pair.
- Secure Socket Layer/Transport Layer Security (SSL/TLS): These are essentially the same protocol. TLS has been renamed by the IETF, but they are based on the same RFC.
- Public Key Infrastructure (PKI): The different components, technologies, and protocols that make up a popular asymmetric encryption solution.
- Mutual Authentication: Instead of using an LDAP repository to hold the public key (PKI), two parties who want to communicate with one another use their public key stored in their digital certificate to authenticate with one another. This topic is covered in 7.2.2, “Grid secure communication” on page 82.

These are all important concepts to remember and will give you a head start in understanding how grid security works.

7.1.4 Symmetric key encryption

Symmetric key encryption is based on the use of one shared secret key to perform both the encryption and decryption of data. To ensure that the data is only read by the two parties (sender and receiver), the key has to be distributed securely between the two parties and no others. If someone should gain access to the secret key that is used to encrypt the data, they would be able to decrypt the information. This form of encryption has performance benefits over asymmetric encryption, but requires additional care and administration in the handling of the shared key. As we mention in the next section, asymmetric key encryption may be used to help manage the keys when using symmetric encryption.

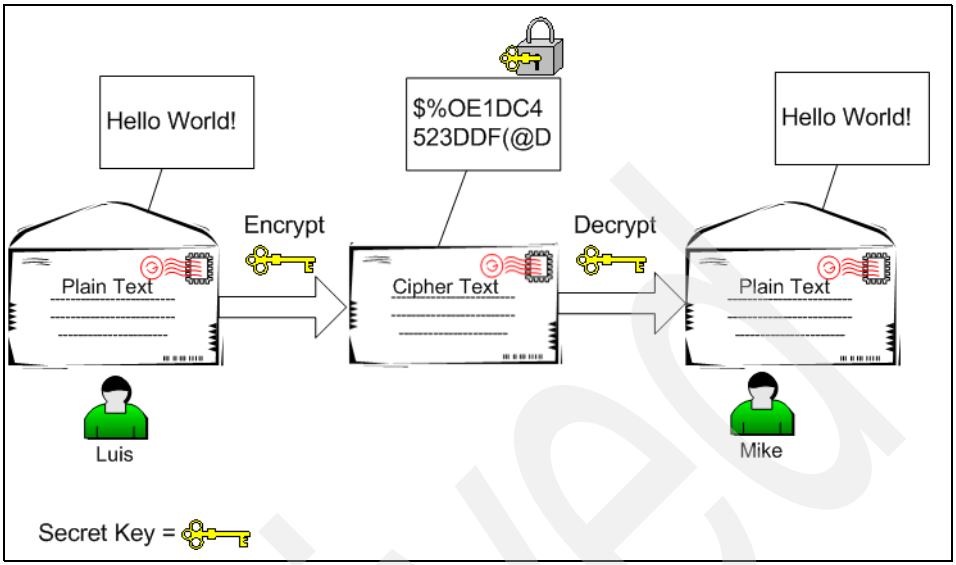

Figure 7-2 Symmetric key encryption using a shared secret key

Here are some commonly used examples of a symmetric key cryptosystem:

- Data Encryption Standard (DES): 56-bit key plus 8 parity bits, developed by IBM in the mid-1970s
- Advanced Encryption Standard (AES): Cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits
- Triple-DES: 112-bit key plus 16 parity bits or 168-bit key plus 24 parity bits (that is, two to three DES keys)
- RC2 and RC4: Variable-sized key, often 40 to 128 bits long

To summarize, secret key cryptography is fast for both the encryption and decryption processes. However, secure distribution and management of keys is difficult to guarantee.

7.1.5 Asymmetric key encryption

Another commonly used cryptography method is called public key cryptography. The RSA public key cryptography system is a prime example of this. In public key cryptography, an asymmetric key pair (a so-called public key and a private key) is used. The key used for encryption is different from the one used for decryption. Public key cryptography requires the key owners to protect their private keys while their public keys are not secret at all and can be made

available to the public. Normally, the public key is present in the digital certificate that is issued by the Certificate Authority.

The computation algorithm relating the public key and the private key is designed in such a way that an encrypted message can only be decrypted with the corresponding key of that key pair, and an encrypted message cannot be decrypted with the encryption key (the key that was used for encryption). Whichever (public/private) key encrypts your data, the other key is required to decrypt the data. A message encoded with the public key, for instance, can only be decoded with the private key. One of the keys is designated as the public key because it is made available, publicly, via a trusted Certificate Authority, which guarantees the ownership of each of the public keys. The corresponding private keys are secured by the owner and never revealed to the public.

The public key system is used twice to completely secure a message between the parties. The sender first encrypts the message using his private key and then encrypts it again using the receiver's public key. The receiver decrypts the message, first using his private key and then the public key of the sender. In this way, an intercepted message cannot be read by anyone else. Furthermore, any tampering with the message will make it not decrypt properly, revealing the tampering.

The asymmetric key pair is generated by a computation that starts by finding two vary large prime numbers. Even though the public key is widely distributed, it is practically impossible for computers to calculate the private key from the public key. The security is derived from the fact that it is very difficult to factor numbers exceeding hundreds of digits.

This mathematical algorithm improves security, but requires a long encryption time, especially for large amounts of data. For this reason, public key encryption is often used to securely transmit a symmetric encryption key between the two parties, and all further encryption is performed using this symmetric key.

7.1.6 The Certificate Authority

A properly implemented Certificate Authority (CA) has many responsibilities. These should be followed diligently to achieve good security. The primary responsibilities are:

- Positively identifying entities requesting certificates
- Issuing, removing, and archiving certificates
- Protecting the Certificate Authority server
- Maintaining a namespace of unique names for certificate owners
- Serving signed certificates to those needing to authenticate entities
- Logging activity

Within some PKI environments, a Registrant Authority (RA) works in conjunction with the CA to help perform some of these duties. The RA is responsible for approving or rejecting requests for the certificate of public keys and forwarding the user information to the CA. The RA normally has the responsibility of validating that the user's information is correct before the signed digital certificate is sent back to the user. Simple CAs, such as those provided with the Globus Toolkit, can be installed for testing purposes. Within this scenario, the simple CA handles the job of both the CA and RA within the grid environment. As the number of certificates expands, these two jobs are normally separated.

One of the critical issues within a grid PKI environment is guaranteeing the system's trustworthiness. Before a CA can sign and issue certificates for others, it has to do the same thing to itself so that its identity can be represented by its own certificate. That means a CA has to do the following:

1. The CA randomly generates its own key pair.
2. The CA protects its private key.
3. The CA creates its own certificate.
4. The CA signs its certificate with its private key.

If a grid resource needs to securely communicate with another grid resource, it needs a certificate signed by a CA. The grid resource has to enroll with the CA by generating an unsigned digital certificate specifying his or her own information. The information submitted will be used by the CA to identify whether this grid resource is real and should be granted a certificate. The CA will then sign the digital certificate if the grid resource is eligible to receive the certificate. This certificate, after the CA signs the certificate, will be passed back to the requesting grid resource. So, one basic function of a CA is to create and issue certificates for a grid resource.

The CA's private key

The CA's private key is one of the most important parts in the whole public key infrastructure. It is used, for example, by the CA to sign every issued digital certificate within the grid network. Thus, it is especially susceptible to attacks from hackers. If someone were to gain access to the CA's private key, they would be able to impersonate anyone within the environment. Therefore, it is very important to protect this key. Knowing how sensitive the private key is to the rest of your grid environment, it is important to provide your CA server with any available security measures. This includes restricting physical and remote access and monitoring and auditing the server.

CA cross certification

Generally within a single grid environment, a CA will provide certificates to a fixed group of users. If two companies or virtual organizations (VOs) need to communicate with and trust one another, this may require that both CAs trust

one another or participate in cross certification. For example, Alice, an employee belonging to an organization with its own CA, may want to run a job on grid computer Mike, who is outside the organization, and who belongs to a different CA. In order to do so, the following should be considered:

- Alice and Mike need a way to obtain each other's public key certificates.
- Mike needs to be sure that he can trust Alice's CA. Alice needs to be sure that she can trust Mike's CA.

Grid computers from different security domains or VOs will need to trust each others' certificates, so the roles and relationships between CAs have to be defined. The purpose of creating such trust relationships is to eventually achieve a global, interoperable PKI and enlarge the grid infrastructure. Once the relationship is established, both of the CAs can be configured to work with the grid system.

Managing your own CA

It is important to note that the simple CA provided with the Globus Toolkit is a fully functioning CA for a PKI environment, but it is only recommended for testing or demo purposes. For a production grid environment, it is recommended that you evaluate commercial PKI solutions that may better suit your needs and remove the responsibility of managing your own CA.

7.1.7 Digital certificates

Digital certificates are digital documents that associate a grid resource with its specific public key. A certificate is a data structure containing a public key and pertinent details about the key owner. A certificate is considered to be a tamper-proof electronic ID when it signed by the Certification Authority for the grid environment.

Digital certificates, also called X.509 certificates, act very much like passports: They provide a means of identifying grid resources. Unlike passports, digital certificates are used to identify grid resources. Another difference between a digital certificate and a passport is that a certificate can (and should) be distributed and copied without restriction, while people are normally very concerned about handing their passports to someone else. Certificates do not normally contain any confidential information, and their free distribution does not create a security risk.

The important fact to know and understand about digital certificates is that the CA certifies that the enclosed public key belong to the entity listed in the certificate. The technical implementation is such that it is considered extremely difficult to alter any part of a certificate without easy detection. The signature of the CA provides an integrity check for the digital certificate.

When a grid client wants to start a session with a grid recipient, he or she does not attach the public key to the message, but the certificate instead. The recipient receives the communication with the certificate and then checks the signature of the Certificate Authority within the certificate. If the signature was signed by a certifier that he or she trusts, the recipient can safely accept that the public key contained in the certificate is really from the sender. This prevents someone from using a fraudulent public key to impersonate the public key owner.

Contained in your digital certificate is the information about you and your public key. When you communicate with another party on the grid, the recipient will use your public key (contained in your digital certificate) to decrypt the SSL session ID, which is used to encrypt all data transferred between grid computers.

A digital certificate is made up of a unique distinguished name (DN) and certificate extensions that contain the information about the individual or host that is being certified. Some information in this section may contain the subject's e-mail address, organizational unit, or location.

Figure 7-3 is a graphical depiction of the digital certificate.

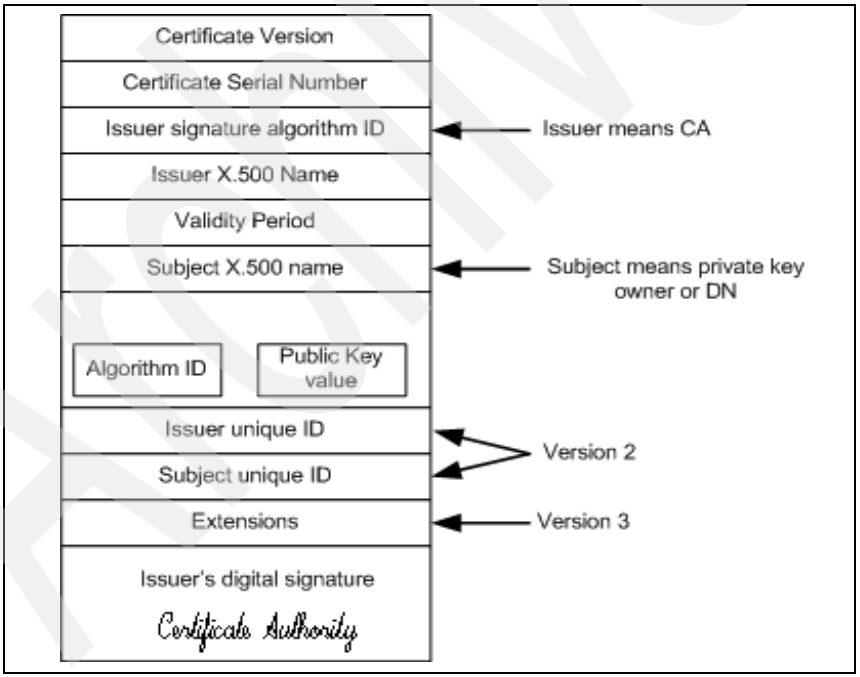

Figure 7-3 Digital certificate

Obtaining a client or a server certificate from a CA involves the following steps:

1. The grid user requiring certification generates a key pair (private key and certificate request containing the public key).
2. The user signs its own public key and any other information required by the CA. Signing the public key demonstrates that the user does, in fact, hold the private key corresponding to the public key.
3. The signed information is communicated to the CA. The private key remains with the client and should be stored securely. For instance, the private key could be stored in an encrypted form on a Smartcard, or on the user's computer.
4. The CA verifies the that the user does own the private key of the public key presented.
5. The CA (or optionally an RA) needs to verify the user's identity. This can be done using out-of-band methods, for example, through the use of e-mail, telephone, or face-to-face communication. A CA (or RA) can use its own record system or another organization's record system to verify the user's identity.
6. Upon a positive identity check, the CA creates a certificate by signing the public key of the user, thereby associating a user to a public key. The certificate will be forwarded to the RA for distribution to the user.

Verification of the user

The authentication described above is a one-time authentication for the purpose of certificate issuance. This can be compared to the process when a government authority issues a passport to an individual. The passport then serves as an authentication mechanism when this individual travels to foreign countries. Just like passports, digital certificates can subsequently be used in daily operations for authenticating subjects to other parties that require authentication.

Different types of certificates

There are two different types of certificates that are used within a grid environment. The first type of certificate is a user certificate that will identify different users on the grid. The second type of certificate is issued to grid servers.

User

As a grid user, you will need a user certificate to identify yourself within the grid. This certificate will identify your user name within the grid, not your server or workstation name. For a user named John Doe, the digital certificate might have the distinguished name:

```
"/O=Grid/O=GridTest/OU=test.domain.com/CN=John Doe"
```

Server

If you plan on running PKI-enabled programs on your server, you will need to register a server certificate. This server certificate will register the fully qualified domain name of your server to your certificate. For your certificate to work, your fully qualified DNS name will have to match your digital certificate. For example, if your a server name was goban.<companyname>.com, your server certificate would read:

```
/CN=Service/goban.<companyname>.com
```

PKI directory services

Within some PKI environments, the signed keys are published to a public directory for easy retrieval. Instead of having the clients handle the mutual authentication, an external server is responsible for handling the authentication process. A good example of this process is the MyProxy server, which works as a grid Web proxy for Web portals. In this example, the user would authenticate to the Web portal, which would request the user's online credentials that are stored in the directory. Upon this authentication, the proxy would extract the DN within their digital certificate and match their credentials with the public key stored within the directory. If they two keys matched up, the user would be given access to resources within the grid.

7.2 Grid security infrastructure

Now that we have gone over some security fundamentals, explaining how the different grid security components interact will be much easier. In this section of the chapter, we choose to summarize the basic mechanisms used by the Grid Security Infrastructure (GSI) provided by the Globus Toolkit. This is just one example of an implementation of a grid security infrastructure. We describe how the different security components within the Globus Toolkit provide security services. We examine different scenarios and walk through the various functions of the GSI.

7.2.1 Getting access to the grid

In order to build a grid environment using the GSI components, you have to create a set of keys for public key cryptography and request your certificate from the Certificate Authority and a copy of the public key of the CA. Figure 7-4 on page 77 and the following procedure describe the steps to establish the GSI communication:

1. Copy the Certificate Authority's public key to your grid host with which you set up GSI.

2. Create your private key and a certificate request.
3. Send your certificate request to CA by e-mail or another more secure way if you are running a production system and need to positively identify the sender.
4. CA signs your request to make your certificate and sends it back to you.

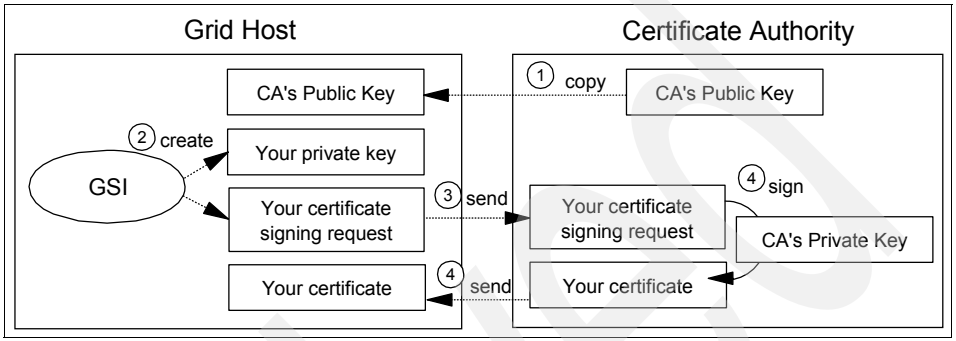

Figure 7-4 Preparation procedure for GSI

When that procedure has been completed and you have received your signed digital certificate, you will have three important files on your grid host. They are:

- The CA's public key
- The grid host's private key
- The grid host's digital certificate

In order to provide secure authentication and communication for your grid computer, you should not let others have access to your private key. An extra layer of security was added to the private key, which includes a secret passphrase that must be used when using your private key along with your digital certificate. This is to prevent someone from stealing your digital certificate and private key and being able to automatically use them to access grid resources. The host key is protected by the local operating system privileges within the grid server.

Authentication and authorization

Imagine a scenario where you need to communicate with another grid computer's application and you want to ensure that the data from the host is really from the host. Besides making sure that you can trust the grid host, you want to make sure the grid host that you want to communicate with trusts your grid computer. In these cases, you can use the authentication function of GSI, as shown in Figure 7-5 on page 79. After you have authenticated with the remote grid resource, you then have the option of having the grid resource give you

access to resources on your behalf. In this case, you can use the authorization function of GSI.

Through the steps described below, grid host A (or a user on grid host A) is authenticated and authorized by grid host B. Almost all steps are for authentication, except the last authorization step:

1. A user or application on A sends its certificate to the host B.
2. Host B will get the public key for A and will use it to extract the subject from the certificate.
3. Host B creates a random number and sends it to host A.
4. Host A receives the number, encrypts it with its private key, and sends the encrypted number to host B.
5. Host B will decrypt the number and check that the decrypted number is really the one that it sent before. Then host B authenticates that the certificate is really that from the user on host A, because only that user on host A can encrypt the number with its private key.
6. The certificate is authenticated by host B, and the subject in the certificate is mapped to a local user name. The subject is in the form of Distinguished Name (DN) like "`O=Grid/O=Globus/OU=itso.grid.com/CN=your name`", and it is the name that is used by LDAP to distinguish the entries in the directory service. The subject is used to specify the user identity in a grid environment. The user defined by the Distinguished Name is authorized by host B to act as a local user on host B.

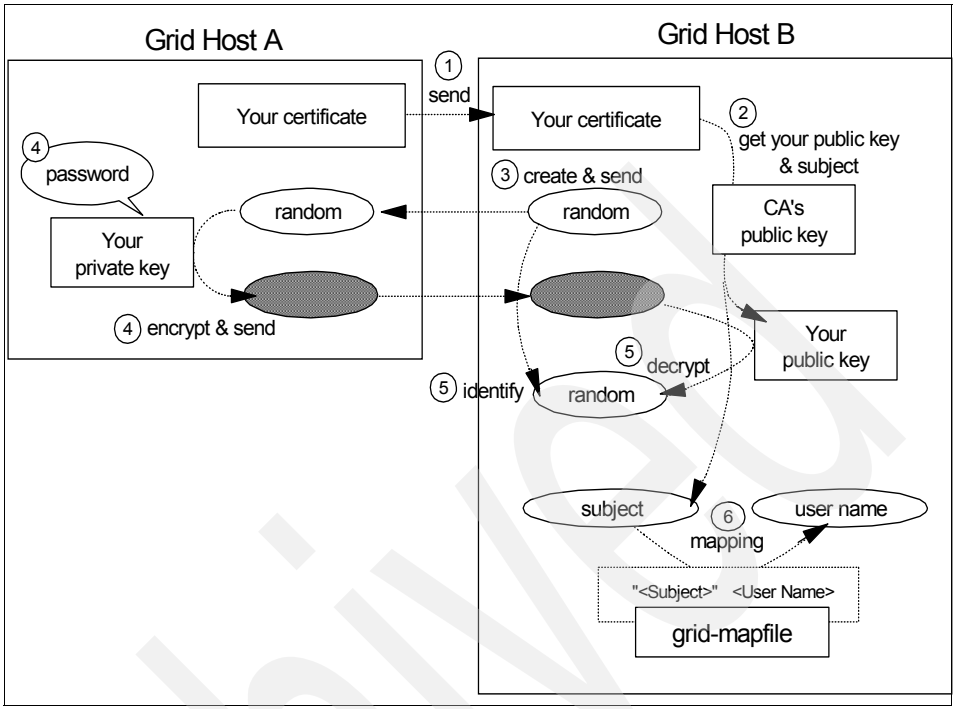

Figure 7-5 Authentication procedure

In grid environments, your host will become a client in some cases, and in other cases, a server. Therefore, your host might be required to authenticate another host and be authenticated by the host at the same time. In this case, you can use the mutual authentication function of GSI. This function is almost the same as explained above, and it proceeds with the authentication steps, and changes the direction of hosts and redoes the procedure.

Briefly speaking, authentication is the process of sharing public keys securely with each other, and authorization is the process that maps your DN to a local user/group of a remote host.

Delegation

Imagine a situation where you distribute jobs to remote grid machines and let them distribute their child jobs to other machines under your security policy. In this situation, you can use the delegation function of GSI, as shown in Figure 7-6 on page 81.

If you are on the side of host A, you can create your proxy at host B to delegate your authority. This proxy acts as yourself, and submits a request to host C on your behalf.

The next steps (see “Proxy creation” on page 80) describe the procedure to create your proxy (proxy creation) at a remote machine, and the procedure to submit a request (see “Proxy action” on page 80) to the other remote host on your behalf (proxy action).

Proxy creation

For proxy creation:

1. A trusted communication is created between host A and host B.
2. You request host B to create a proxy that delegates your authority.
3. Host B creates the request for your proxy certificate, and sends it back to host A.
4. Host A signs the request to create your proxy certificate using your private key and sends it back to host B.
5. Host A sends your certificate to host B.

Proxy action

For proxy action:

1. Your proxy sends your certificate and the certificate of your proxy to host C.
2. Host C gets your proxy's public key through the path validation procedure:
   a. Host C gets your subject and your public key from your certificate using CA's public key.
   b. Host C gets the proxy's subject and your proxy's public key from your proxy's certificate using your public key.
   c. The subject is a Distinguished Name similar to `"O=Grid/O=Globus/OU=itso.grid.com/CN=your name"`. The subject of the proxy certificate is similar to its owner's (your) subject and is similar to `"O=Grid/O=Globus/OU=itso.grid.com/CN=your name/CN=proxy"`. So in order to validate the proxy certificate, host C just has to check that the words that eliminate the words `"/CN=proxy"` from the proxy's subject are just the same as your subject’s. If it is validated, your proxy is authenticated by host C and able to act on your behalf.
3. The proxy encrypts a request message using its private key and sends it to host C.
4. Host C decrypts the encrypted message using the proxy's public key and gets the request.
5. Host C runs the request under the authority of a local user. The user is specified using a mapping file, which represents the mapping between the grid users (subject) and local users (local user name).

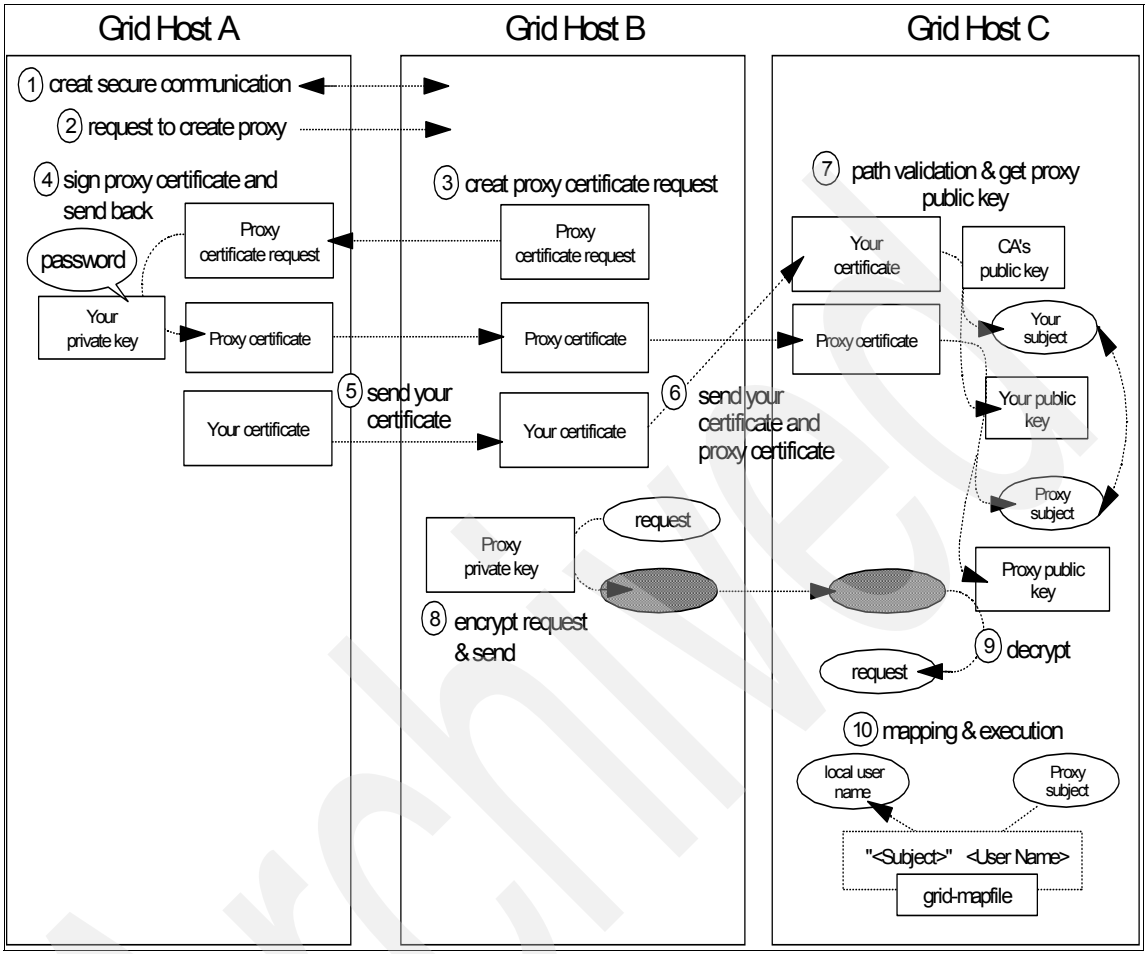

Figure 7-6 Delegation procedure of user's proxy

The procedure in Figure 7-6 represents remote delegation, where a user creates a proxy at a remote machine. There is also a local delegation, where a user creates a proxy certificate at the local machine; for that task, Globus Toolkit uses the **`grid-proxy-init`** command and gatekeeper daemon mechanism.

When you make a proxy on a remote machine (in remote delegation), the proxy's private key is stored on the remote machine, so the super-user of that machine can access your proxy's private key. This delegated credential can be vulnerable to attacks. In order to avoid this, it is recommended that the proxy attain restricted policies from its owner. The standardization of this proxy restriction is

now going on under GSI Working Group of the Grid Forum Security Area, and you can see more details in its Internet draft at:

http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-03.txt

7.2.2 Grid secure communication

While we have gone over the process of using PKI within a grid environment and the different functions of GSI, it is still important to understand the communication mechanisms used within the Globus Toolkit. By default, the underlying communication is based on the mutual authentication of digital certificates and SSL/TLS.

The digital certificates that have been installed on the grid computers provide the mutual authentication between the two parties. We discuss this process in detail later on in this section. The SSL/TLS functions that OpenSSL provides will encrypt all data transferred between grid hosts. These two functions together provide the basic security services of authentication and confidentiality.

Mutual authentication

To allow secure communication within the grid, the OpenSSL package is installed as part of the Globus Toolkit. Within the Globus Toolkit, OpenSSL is a software package that is used to create an encrypted tunnel using SSL/TSL between grid clients and servers.

The process of mutual authentication begins when two grid resources want to share resources. Instead of using a key repository, each grid resource authenticates with one another based on their digital certificate. For example, one grid resource will attempt to establish secure communication with another grid resource. Before the recipient will allow the client access to their resources, they need to authenticate to one another. This process is documented below with the SSL handshake.

SSL handshake

In order to establish the secure communication between the grid server and grid client, a handshake must be established. This handshake is responsible for determining the SSL settings, exchanging public keys and the basis for the mutual authentication process. The handshake process is as follows:

1. A grid client contacts a remote grid server to start a secure session by using a digital X.509 ID certificate.
2. The grid client automatically sends to the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.

3. The grid server responds, automatically sending the grid client the site's digital certificate, along with the server's SSL version number, cipher settings, and so on.
4. The customer's client examines the information contained in the server's certificate, and verifies that:
   a. The server certificate is valid and has a valid date.
   b. The CA that issued the server certificate has been signed by a trusted CA whose certificate is built into the client.
   c. The issuing CA's public key, built into the client, validates the issuer's digital signature.
   d. The domain name specified by the server certificate matches the server's actual domain name.
5. If the server can be successfully authenticated, the grid client generates a unique session key to encrypt all communications with the grid server using asymmetric encryption.
6. The user's client encrypts the session key itself with the server's public key so that only the site can read the session key, and sends it to the server.
7. The server decrypts the session key using its own private key.
8. The grid client sends a message to the server informing it that future messages from the grid client will be encrypted with the session key. The grid server then sends a message to the grid client informing it that future messages from the server will be encrypted with the session key.
9. An SSL-secured session is now established. SSL then uses symmetric encryption (which is much faster than asymmetric PKI encryption) to encrypt and decrypt messages within the SSL-secured *pipeline*.
10. Now that the first grid resources have authenticated, the second grid resource will now authenticate using the same process.
11. Once the session is complete, the session key is eliminated.

As long as both grid resources have a valid digital certificate, the process of mutual authentication will succeed. This is a good example of how grid security uses both symmetric and asymmetric encryption to authenticate and secure data transfer between grid resources. A grid client uses asymmetric encryption to authenticate, and once it is authenticated, it passes symmetric encryption along with a shared secret key to encrypt and decrypt all data traffic between them.

Other grid communication

If you cannot physically access your grid client or server, it may be necessary to gain remote access to the grid. While your operating systems default telnet

program works fine for remote access, the transmission of the data is in clear text. That means that the data transmission would be vulnerable to someone listening or sniffing the data on the network. While this vulnerability is low, it does exist and needs to be dealt with.

To secure the remote communication between a client and grid server, the use of Secure Shell (SSH) can be used. SSH will establish an encrypted session between your client and the grid server.

7.2.3 Grid security step-by-step

In order to better understand the process for accessing grid resources, we have outlined the basic process from start to finish.

Local delegation

This program is used to get a session *proxy* certificate using your long-term certificate.

The proxy certificate is used to authenticate the user and user programs to resources on the grid. For example, the user can run jobs on the grid with the **`globusrun`** command. The **`globusrun`** command is authenticated with the proxy certificate. The proxy certificate is created with the **`grid-proxy-init`** command. A proxy certificate must be created before jobs can be run on the grid. The proxy certificate is a session certificate with a limited or short-lived life time, which is signed by the user certificate. This is functionally equivalent to the Kerberos kinit program or DCE dce_login.

The motive behind this model is to provide for the single sign-on. The single sign-on is the grid-proxy-init. Once the grid proxy certificate is created, this certificate is used for authentication on the grid.

This model works because it creates a certificate trust hierarchy, as shown in Figure 7-7 on page 85.

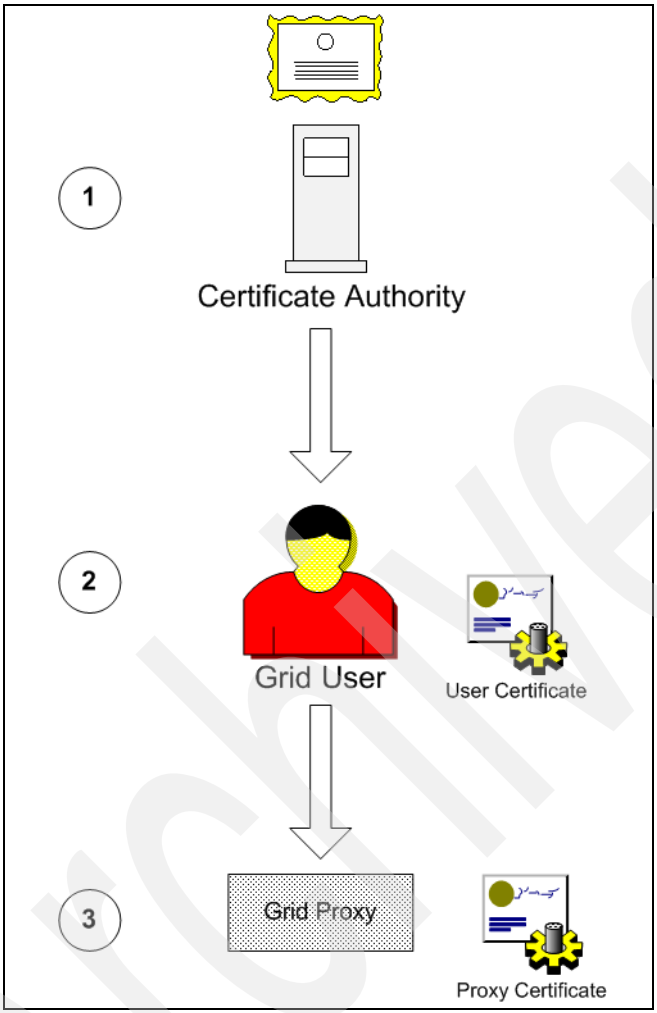

Figure 7-7 Authentication process

The hierarchy is as follows:

1. The remote grid resource trusts the CA. The remote grid resource trusts the CA because it placed the CA's certificate in /etc/grid-security/certificates.
2. The remote grid resource can authenticate the user certificate because it is digitally signed by the CA.
3. The remote grid resource can authenticate the user proxy certificate because it is digitally signed by the user certificate.

It is analogous to meeting three people at a party: CA, Alice, and Proxy. Proxy hands you a card that is similar to Figure 7-8.

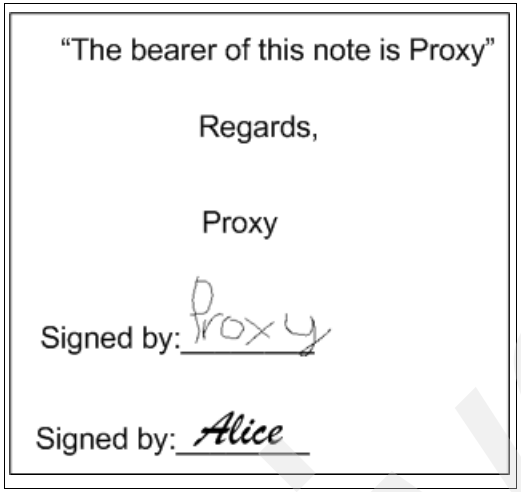

Figure 7-8 Certificate signed by Alice

You are not familiar with Alice's signature, so you take a card from Alice, which is similar to Figure 7-9.

Figure 7-9 Certificate signed by CA

You keep a copy of CA's signature in you wallet. You compare the CA signature on Alice's card to the copy you keep in your wallet and they match. You now have an authenticated copy of Alice's signature, which you compare to the signature on Proxy's card. They match, and you now trust you that you are talking to Proxy. You have authenticated that this person is Proxy.

The **grid-proxy-init** command uses the SSL library to create a proxy certificate that is stored in /tmp/<filename>, where <filename> is equal to x509up_u<uid>, where uid is equal to the UID of the user running **grid-proxy-init**. The permission of this file is “-rw------- owner group” of the user running the command.

This file is an X.509 certificate where the issuer is the user’s primary certificate. Basically, the user’s primary certificate acts like a CA to create this session or *proxy* certificate. The proxy certificate is considered a short-lived certificate. By default, it has a validity period of 12 hours, but this can be specified by the **grid-proxy-init** parameter -hours.

The proxy certificate, as with all X.509 certificates, contains a unique name and public key. The proxy certificate's unique subject name or distinguished name is the primary certificate’s unique name plus "CN=proxy" (limited proxy). This is best illustrated with the **grid-cert-info** and **grid-proxy-info** commands. If the **grid-cert-info** command is run with the file name of our primary certificate, the contents of the certificate are displayed:

```
%grid-cert-info -f.globus/usercert.pem -subject
/C=US/O=IBM/OU=GridLPP/OU=austin.ibm.com/CN=griduser
```

The -subject flag displays the subject or distinguished name (DN).

A complete description of X.509 certificates can be found in RFC 2459. The /tmp/x509_up_u<uid> file created by **grid-proxy-init** contains two other components in addition to the proxy certificate. It also contains the private key of the proxy certificate and the user certificate.

The proxy certificate’s private key is only protected by the file permissions of /tmp/x509_up<uid>. Since the proxy certificate is short lived, a compromised or stolen certificate will become useless at the end of its life.

The user certificate's private key remains encrypted in the $HOME/.globus/userkey.pem file. It can only be accessed with the passphrase that is given when the user certificate is created with the **grid-cert-request** command.

7.3 Grid infrastructure security

Apart from the different GSI components and technologies, there are many other infrastructure security components that are needed to secure the grid. As in other areas of grid design, the grid infrastructure security builds on other security principles. While these security components are optional, they are considered standard within many production networks. We explore some of these basic security concepts and see how they fit into a grid infrastructure.

7.3.1 Physical security

Once again, the security of grid infrastructure is based on other common security fundamentals. The basics involve solid physical security practices for all grid computers. The physical environment of a system is also considered a part of the infrastructure. If the servers are kept in an open room, no matter how secure the applications are designed or how complex the cryptographic algorithms are, the server services can easily be interrupted, such as being powered off, or otherwise tampered with. Therefore, physical access should be controlled and is part of the security policies that need to be defined.

The CA server should be located in a robust, dedicated, and locked room. All accesses should be logged and controlled. The power supply to the servers should never be interrupted. This means an uninterruptable power supply (UPS) must be used. However, a UPS may still run out of electricity after a prolonged period. In such a case, the servers should be able to automatically back up the data and properly shut down.

For maximum security, the network segment where the PKI-sensitive server machines are installed should be physically and logically separated from the rest of the network. Ideally, the separation is done through a firewall that is transparent only for PKI-related traffic. Normally, PKI traffic is reduced to using only a few TCP/IP ports.

7.3.2 Operating system security

A review of the configuration files for each operating system and middleware component within the scope of the project determines how each effectively allows authorized users access based on your security policy and prevents and detects unauthorized access attempts at all times. You should:

- Remove any unnecessary processes from the servers. If the grid server does not need sendmail or an FTP server running, these processes should be disabled.
- Remove any unnecessary users or groups.

- Use strong passwords for all users on the grid server.
- Update your server with the latest updates and security FixPacks. This includes all software the has been installed as well.
- Restrict access to directories that contain security-related information, such as the /.globus directory (in a Globus Toolkit) environment.
- Consider using host IDS to monitor important directories on the server.
- Enable logging and auditing for the server.
- Use a uniform operating system build whenever possible.
- Enable file-level restrictions on important files within the server.
- Make periodic reviews of the operating system every other month to ensure that nothing major has changed.
- Enable anti-virus protection.

7.3.3 Grid and firewalls

Firewalls can be used within a networked environment to logically separate different sets of computers that require additional security. In a grid environment, this is no different. The use of firewalls within a grid design helps restrict network access to computers. The firewall is an important piece of the security infrastructure, so it needs to be carefully analyzed and understood before it is implemented.

7.3.4 Host intrusion detection

A recommended option for further securing your grid computers is to invest in a host intrusion detection (IDS) product. As with any software application that stores important files within the local workstation, host intrusion detection can add a greater defense for anyone manipulating files on the workstation that should not be doing so. If the host IDS product detects a changed file on the server, it can send an alert to a central monitoring workstation to log and alert the necessary people.

An intrusion detection system gathers and analyzes information from various areas within a computer or a network to identify possible security breaches, which include both intrusions (attacks from outside the organization) and misuse (attacks from within the organization). An intrusion detection system uses vulnerability assessment (sometimes referred to as scanning), which is a technology developed to assess the security of a computer system or network.

Intrusion detection functions include:

- Monitoring and analyzing both user and system activities

- Analyzing system configurations and vulnerabilities
- Assessing system and file integrity
- Ability to recognize typical patterns of attacks
- Analysis of abnormal activity patterns
- Tracking user policy violations

Network intrusion detection

There can be a point made for network IDS within a grid environment, but some of that benefit would be lost due to the encryption between grid servers. While a network IDS would be able to use special signatures for standardized network traffic, the introduction of a network-based IDS system would be lost because of the SSL/TLS encryption. While a network IDS system could not see the data payload portion of the packet that is encrypted, the network IDS could respond to events based on the packet header that is unencrypted. Network IDS is best suited for placement where it can analyze unencrypted traffic.

The use of any IDS is an optional component within an architecture, but is strongly recommended for good security practices.

7.4 PKI security policies and procedures

Good security policies and procedures are used to complement the variety of security components that make up a security infrastructure. This is no different in a grid environment, but may take on more importance since you may be dealing with networks out of your control. To help manage this risk, different policies and procedures should be used. These policies and procedures will help build a certain way of managing the security controls.

One of the first steps an organization has to consider when comprehensive security solutions are to be introduced is to define a feasible set of security policies. In the first place, this has little to do with a PKI because security policies need to be in place for any kind of IT infrastructure. Only when the deployment of a PKI has been decided do some additional benefits and issues come up that need to be defined within security policies. The following subsections discuss security policies that primarily relate to a PKI.

7.4.1 Certificate Authority

A PKI must be operated in accordance with defined policies. The deployment of a PKI system in an organization requires the development of security policies and processes for that organization. The demo CA that is provided within the Globus Toolkit provides the software needed in order to build a CA, but unfortunately none of the policies. In this section, we examine some of the basic

policies and expectations that a CA would normally be responsible for. For any type of production CA duties, it is suggested that you examine a commercial vendor to provide these services for you.

The standardization effort has been made to involve security policies in a PKI framework systematically, as outlined in RFC 2527, Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. According to X.509, a certificate policy is “a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements.” A more detailed description of the practices followed by a CA in issuing and otherwise managing certificates may be contained in a Certification Practice Statement (CPS) published or referenced by a CA.

A certificate policy’s extension contains a sequence of one or more policy information terms, each of which consists of a registered object ID (OID) and optional qualifiers. Applications with specific policy requirements will have to recognize the OID meaning in at least the same security domain. If the required policy’s OID is not contained in the certificate extension field, or if any existing critical OIDs are not understood by the application, the application has to reject the client’s request. Security policies also result in processes that have to be in place and subsequently enforced. Processes describe (and/or mandate) the way an infrastructure is utilized by its administrators and users. Processes may include elements, such as:

- The certificate requesting, issuance, distribution, and revocation processes.
- The use of certificates for client authentication
- The use of certificates for securing e-mail communication
- The use of certificates for inter-organization communication
- Procedures to follow when security violations are suspected
- Handling guidelines for private keys and certificates
- Application development guidelines for PKI exploitation (such as user authentication using certificates)

A PKI will alter many existing business processes and require many new ones to support it. These processes can cover technical, organizational, legal, and infrastructure elements of the whole workflow.

- CA key generation
  - Who is involved?
  - How is the process secured?
- CA key backup

  How is a backup of the CA private key accomplished?

- CA key restore

  How is a key restored?
- CA key compromise

  What happens if the key is broken?
- User registration

  How does a user obtain a certificate?
- Certificate revocation

  How is a certificate revoked?

CA implementation

If you are planning on implementing your own Certificate Authority, you will likely build on tools similar to those provided with the Globus Toolkit. The Globus Toolkit provides some of the basic tools for a demo CA within a lab or testing environment, but there is more to building a CA than installing a few scripts.

In order to manage and administer your own CA, you should be aware of some of the other resources and policies that are normally required. If you plan on managing a CA yourself, your plan for implementation must include:

- Required resources and skills
- Required PKI and security process additions and changes
- Recommended implementation time line and dependencies
- Required changes to the technical infrastructure
- Adoption of the CPS, certificate, and security policies
- Required PKI and security policy additions and changes
- All required checkpoints and approvals

7.4.2 Security controls review

When building any new environment or implementing a new software application, it is always a good idea to perform a security health check. A security health check will help determine how these new changes will affect the overall security of the environment and any other areas of change. This can help provide guidance on the overall use of security controls or how you are managing security within your environment. A review of your security controls can help you better understand how security works for your passwords, administration, toolsets, auditing, and monitoring within your environment. This will provide an in-depth review of the site security controls in place and the related processes used within the organization.

7.5 Summary

This chapter has described in some detail the types of considerations involved in security related to grid environments. For specific examples, we have used the Globus Toolkit 4 environment and the PKI infrastructure that it delivers.

8

Design

This chapter provides architectural design considerations for grid computing. Other design topics that will be discussed are different grid topologies, grid infrastructure design, and grid architecture models.

At a glance, the following topics are discussed:

- Grid architecture design concepts
- Different grid topologies
- Grid architecture models
- Building a grid architecture
- Grid architecture conceptual model

8.1 Building a grid architecture

The foundation of a grid solution design is typically built upon an existing infrastructure investment. However, a grid solution does not come to fruition by simply installing software to allocate resources on demand. Given that grid solutions are adaptable to meet the needs of various business problems, differing types of grids are designed to meet specific usage requirements and constraints. Additionally, differing topologies are designed to meet varying geographical constraints and network connectivity requirements. The success of a grid solution is heavily dependant on the amount of thought the IT architect puts into the solution design.

Once the functional and non-functional requirements are known, the IT architect should readily be able to select the type of grid and the best topology required to satisfy the majority of the business requirements. When armed with this information, the high-level grid design will be easier to complete, and by leveraging the use of known grid types and topologies, articulating the solution design will require much less effort.

It is important to focus on starting small and to begin building the basic framework of the design. Rather than setting out to build the desired end state grid solution all at once, consider building the grid solution in a phased approach. The milestone for the initial phase is to provide an *intragrid* solution, which is essentially a grid sandbox that supports a basic set of Grid services. This solution would support a single location built upon the core grid components, such as a security model, information services, workload management, and the host devices. As long as this model supports the same protocols and standards, this design can be expanded as needed.

The first step of the design process is to build a graphical representation of the grid components. The subsequent phases of the design will be focused on the next level of architecture. This phase of the design is a starting point for architects, technical managers, and executives to understand the overall structure of the architecture.

At a glance, the grid architecture design should offer the following:

- The “blueprint” for the detailed conceptual design
- The use of open standards prescribed by the grid framework
- A multi-dimensional tiered and layered view of the grid infrastructure, which demonstrates the ability to logically partition grid resources so that their service consumption does not impact other grid locations
- The middleware components and subsystems for a grid infrastructure integration

- A design for communication to both business and technical personnel, for budget and planning purposes, and to provide application development an illustration of how the shared grid infrastructure will impact the middleware solution design
- The distribution of applications and subsystems
- A means for identifying the necessary technical, infrastructural, and other middleware components and subsystems for a grid infrastructure

8.1.1 Solution objectives

The design objectives provide a basic framework for building the grid infrastructure. The advantage of using design solution objectives is to start documenting certain areas that can affect the overall design. Within your design, you are going to need to make sure that the grid can provide a certain amount of security, availability, and performance. By documenting these different objectives or requirements, it will make your design a lot easier to follow. You will also be able to justify some of your decisions during the course of the design by being able to come back to certain objectives and making sure they were met.

Once the design objectives have been defined, you can separate them into individual subsystems. This allows each design objective to be worked on in parallel, while at the same time providing a cohesiveness for the overall architecture. Once you have documented the core subsystems of the design, you can focus on the different requirements that your grid design will comprise.

When you start building the initial pieces of your design, you need to make sure that your solution objectives line up with the customer's requirements. For a grid design, this is especially important, as there are not only the standard infrastructure components to consider, but specialized middleware and application integration issues as well. Making sure that your solution objectives satisfy your stated requirements will allow you to design a working grid.

Security

Within any networked environment, there is going to be some risk and exposure involved with the security of your infrastructure. Unless the computers are unplugged in a locked room, there is the potential that someone may bypass the security and get access to protected resources. Whether the weaknesses are exploited in the infrastructure, application, configuration, or administration, there is some level of risk.

Security objectives are put in place to help to reduce that risk to an acceptable level. While no design is 100 percent secure, the level of risk is reduced and controlled through the use of security controls. The goal of the security objectives

are to examine the security requirements and implement the necessary tools and processes to reduce the risk involved.

The degree of security involved is based on the type of grid topology and the data the security will be protecting. The security requirements for a grid design within a bank will be completely different from those of an academic institution doing research. Whatever the security requirements may be, the security design objectives for the grid design need to be a central focus for the conceptual architecture.

Considering that the basic grid security model is based on PKI, it is imperative that the security components are designed and thought out carefully. While PKI has been around for a while, there are different components and necessary processes that should be identified. Rushing this process could lead to many problems in the future.

With the PKI architecture being the focus of the initial design, there are still areas that need attention. The infrastructure components (firewalls, IDS, anti-virus, and encryption) and the processes to manage these pieces are all part of the security objectives. Knowing which areas match up with your existing environment is the first step to robust security. The following bullet points are an example of some security questions that will be answered during the course of the design. The first three assume that the enterprise will provide its own certificate authority, which is not usually recommended:

- Where will my CA be deployed and how will we manage it?
- Do I have the necessary processes in place to administer my own CA?
- What are the responsibilities for managing my own CA?
- How will I administer security on the local servers?
- Are my servers of a uniform build or common operating environment?
- Do I have a consistent software build across critical grid infrastructure systems?
- Which processes are running on my servers?
- Will any existing applications conflict with or further expose my grid to any vulnerabilities?

Availability

Availability in its simplest terms commonly refers to the percentage of time that a site is up and servicing job requests. Determining how much availability should be built into the design is part of the availability objectives. This leads down the path of discovering how many potential single points of failure exist and how much redundancy should be built into the design. It is inevitable that some

components will fail during a lifetime of usage, but this can be managed by using redundant components where possible.

Whenever you review various availability scenarios, there are always discussions about the amount of availability that is required. In this respect, a grid design is no different from any other infrastructure. A good start is to list the potential components within the design that should be resilient to failure. Once these components have been identified, you can seek out the specific availability options for those components. In the following examples, some different infrastructure options are described.

An important point that needs to be discussed is the availability of dynamic resources within a grid environment. Grid is not like a standard environment where resources are fixed and do not change regularly. Within grid environments, resources are constantly changing according to the membership and participation in the grid. When grid resources are active, they can register with information services within the grid to alert the system of their state. It is important to make sure that when you design your grid, you keep this in mind.

Besides the grid middleware components, the different infrastructure components will also require different levels of availability. Some components will be more critical than others, and it will be up to your design to make sure that you account for this. When going through the different availability requirements, make sure that you account for both the grid and infrastructure components. The following lists are some examples of availability resources that should be accounted for:

- Grid middleware
  - Workload management
  - Grid directory and indexing service
  - Security services
  - Data storage
  - Grid software clustering
- Networks
  - Load-balancing
  - High-availability routing protocols
  - Redundant and diverse network paths
- Security
  - Redundant firewalls
- Datastore
  - Mirroring
  - Data replication
  - Parallel processing

- Systems management
  - Backup and recovery
  - LDAP replicas
  - Alerts and monitoring to signal a failure within the environment

Every so often, different components necessary to the workflow process fail periodically and disrupt availability of the system. You can help mitigate the risk involved by eliminating the single points of failure within your environment through the use of redundant software or hardware components.

To give you a better idea of some different availability targets, the following list presents an example of the expected system availability in a whole year:

- Normal commercial availability (single node): 99–99.5 percent, 87.6–43.8 hours of system down
- High availability: 99.9 percent, 8.8 hours of system down
- Fault resilient: 99.99 percent, 53 minutes of system down
- Fault tolerant: 99.999 percent, 5 minutes of system down
- Continuous processing: 100 percent, 0 minutes of system down

Keep in mind, however, that the redundancy that is added to the grid infrastructure will normally increase the costs within the infrastructure. It is up to the business to help justify the costs that would bring an environment from 99.9 percent availability per year up to 99.99 percent per year. While the difference in time between those two numbers is about eight hours, the costs associated may be too much to justify the increased availability.

Performance

The performance objective for a grid environment is to most efficiently utilize the various resources within the grid. Whether that includes spare CPU cycles, access to a federated databases, or application processing, it is up to you to match the performance goals of the business and design accordingly.

If your application can take advantage of multiple resources, you can design your grid to be broken up into smaller instances and have the work distributed throughout the grid. The goal is to take advantage of the grid as a whole in order to increase the performance of the application. Through intelligent workload management and scheduling, your application can take advantage of whatever resources within the grid are available. Part of the performance is based on the form of workload management to make sure that all resources within the grid are actively servicing jobs or requests within the grid.

8.2 Grid architecture models

There are different types of grid architectures to fit different types of business problems. Some grids are designed to take advantage of extra processing resources, whereas some grid architectures are designed to support collaboration between various organizations.

The type of grid selected is based primarily on the business problem that is being solved. Taking the goals of the business into consideration will help you choose the proper type of grid framework. A business that wants to tap into unused resources for calculating risk analysis within their corporate data center will have a much different design than a company that wants to open their distributed network to create a federated database with one or two of their main suppliers. Such different types of grid applications will require different designs, based on their respective unique requirements.

The selection of a specific grid type will have a direct impact on the grid solution design. Additionally, it should be mentioned that grid technologies are still evolving and tactical modifications to a grid reference architecture may be required to satisfy a particular business requirement.

8.2.1 Computational grid

A computational grid aggregates the processing power from a distributed collection of systems. A well-known example of a computational grid is the SETI@home grid. This type of grid is primarily comprised of low-powered computers with minimal application logic awareness and minimal storage capacity.

Rather than simply painting images of flying toasters, the idle cycles of the personal computers on the SETI@home grid are combined to create a computational grid used to analyze radio transmissions received from outer space in the “Search for Extra Terrestrial Intelligence.”

Most businesses interested in computational grids will likely have similar IT initiatives in common. While they probably will not want to search for extraterrestrials, there will likely be a business initiative to expand abilities and maximize the computer utilization of existing resources through aggregation and sharing. The business may require more computer capacity than is available. The business is interested in modifying specific vertical applications for parallel computing opportunities.

Additional uses for a computational grid include mathematical equations, derivatives, pricing, portfolio valuation, and simulation (especially risk measurement). Note that not all algorithms are able to leverage parallel

processing, data intensive and high throughput computing, order and transaction processing, market information dissemination, and enterprise risk management. In many cases, the grid architecture model is not (yet) suitable for real-time applications.

Computational grids can be recognized by these primary characteristics:

- Made up of clusters of clusters
- Enables CPU scavenging to better utilize resources
- Provides the computational power to process large-scale jobs
- Satisfies the business requirement for instant access to resources on demand

The primary benefits of computational grids are a reduced Total Cost of Ownership (TCO) and shorter deployment life cycles. Besides the SETI@home grid, the World Community Grid™, the Distributed Terascale Facility (TeraGrid), and the UK and Netherlands grids are all different examples of deployed computational grids. The next generation of computational grid computing will shift focus towards solving real-time computational problems.

8.2.2 Data grid

While computational grids are more suited for aggregating resources, data grids focus on providing secure access to distributed, heterogeneous pools of data. Through collaboration, data grids can also include resources such as a federated database. Within a federated database, as illustrated in Figure 8-1 on page 103, a data grid makes a group of databases available that function as a single virtual database. Through this single interface, the federated database provides a single query point, data modeling, and data consistency.

Data grids also harness data, storage, and network resources located in distinct administrative domains, respect local and global policies governing how data can be used, schedule resources efficiently (again subject to local and global constraints), and provide high speed and reliable access to data. Businesses interested in data grids typically have IT initiatives to expand data-mining abilities while maximizing the utilization of an existing storage infrastructure investment, and to reduce the complexity of data management.

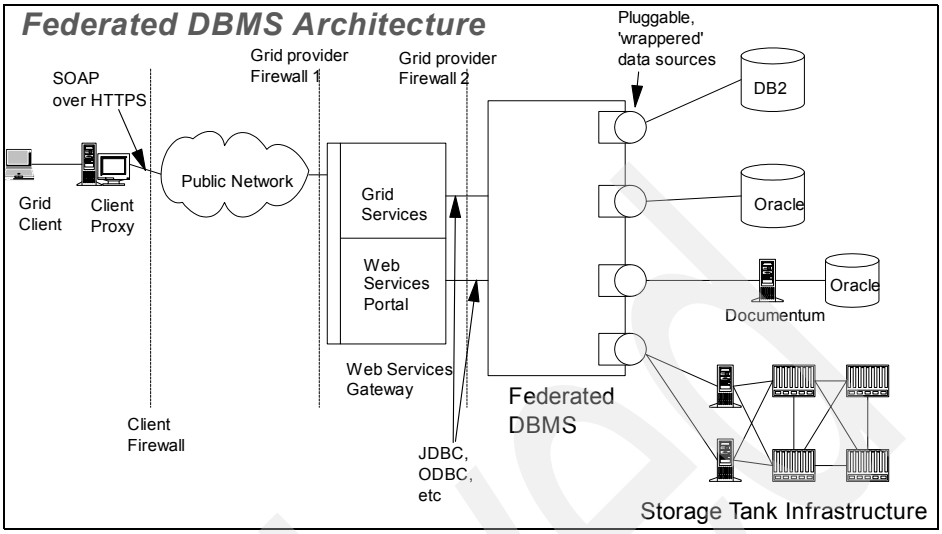

Figure 8-1 Federated DBMS architecture

8.3 Grid topologies

A topology view (see Figure 8-2 on page 104) covers the following spectrum of grids:

- Intragrids
  - Single organizations
  - No partner integration
  - A single cluster
- Extragrids
  - Multiple organizations
  - Partner integration
  - Multiple clusters
- Intergrids
  - Many organizations
  - Multiple partners
  - Many multiple clusters

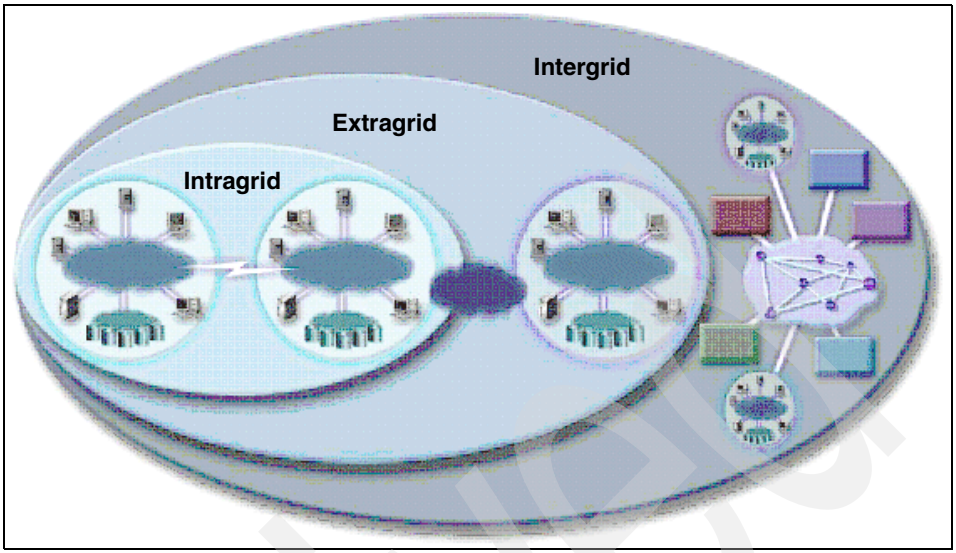

Figure 8-2 Intragrids, extragrids, and intergrids

The simplest of the three topologies is the intragrid, which is comprised merely of a basic set of Grid services within a single organization. The complexity of the grid design is proportionate to the number of organizations that the grid is designed to support, and the geographical parameters and constraints. As more organizations join the grid, the non-functional or operational requirements for security, directory services, availability, and performance become more complex.

As more organizations require access to grid resources, the requirements for increased application layer security, directory services integration, higher availability, and capacity become more complicated.

The resource sharing alluded to is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly protected, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs.

8.3.1 Intragrid

A typical intragrid topology, as illustrated in Figure 8-3 on page 105, exists within a single organization, providing a basic set of Grid services. The single organization could be made up of a number of computers that share a common security domain, and share data internally on a private network. The primary

characteristics of an intragrid are a single security provider, bandwidth on the private network is high and always available, and there is a single environment within a single network. Within an intragrid, it is easier to design and operate computational and data grids. An intragrid provides a relatively static set of computing resources and the ability to easily share data between grid systems. The business might deem an intragrid appropriate if the business has an initiative to gain economies of scale on internal job management, or wants to start exploring the use of a grid internally first by enabling vertical enterprise applications.

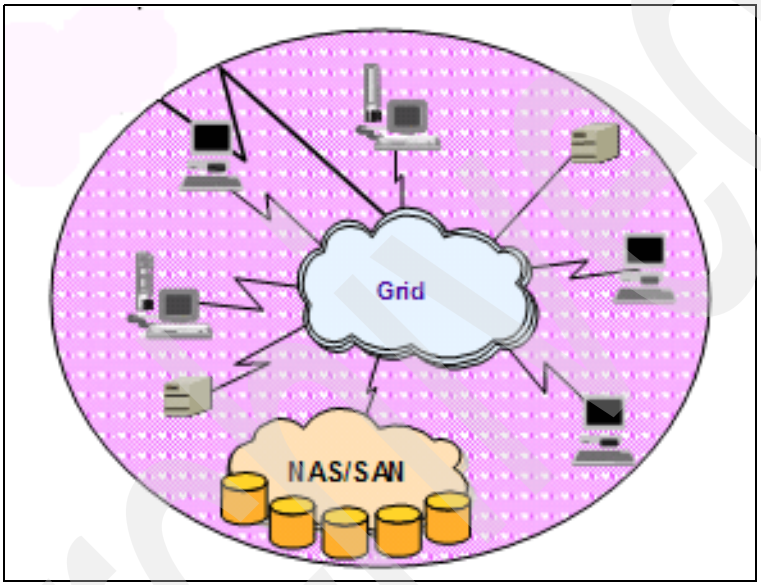

Figure 8-3 An intragrid

8.3.2 Extragrid

Based on a single organization, the extragrid expands on the concept by bringing together two or more intragrids. An extragrid, as illustrated in Figure 8-4 on page 106, typically involves more than one security provider, and the level of management complexity increases. The primary characteristics of an extragrid are dispersed security, multiple organizations, and remote/WAN connectivity. Within an extragrid, the resources become more dynamic and your grid needs to be more reactive to failed resources and failed components. The design becomes more complicated and information services become relevant to ensure that grid resources have access to workload management at run time.

A business would benefit from an extragrid if there was a business initiative to integrate with external trusted business partners. An extragrid could also be used in a B2B capacity and/or to establish relationships of trust.

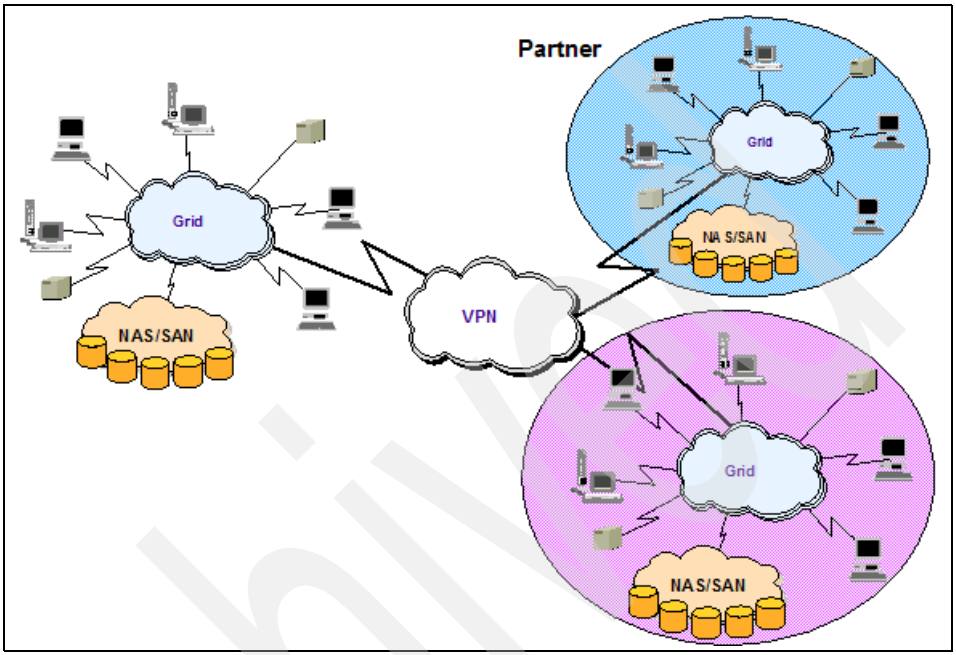

Figure 8-4 Extragrids can exist in several organizations and security providers

8.3.3 Intergrid

An intergrid requires the dynamic integration of applications, resources, and services with patterns, customers, and any other authorized organizations that will obtain access to the grid via the internet/WAN. An intergrid topology, as illustrated in Figure 8-5 on page 107, is primarily used by engineering firms, life science industries, manufacturers, and by businesses in the financial industry. The primary characteristics of an intergrid include dispersed security, multiple organizations, and remote/WAN connectivity. The data in an intergrid is global public data, and applications (both vertical and horizontal) must be modified for a global audience. A business may deem an intergrid necessary if there is a need for peer-to-peer computing, a collaborative computing community, or simplified end-to-end processes with the organizations that will use the intergrid.

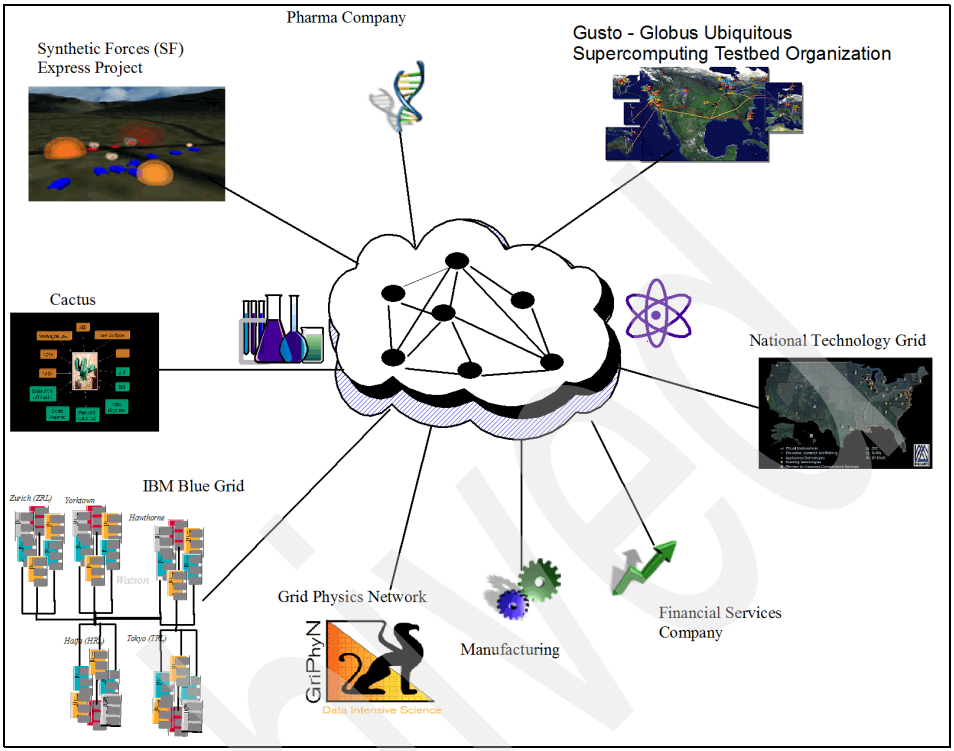

Figure 8-5 Intergrid

8.3.4 e-Utilities

One other type of grid that we should discuss before closing out this section is what we will call e-utility computing. Instead of having to buy and maintain the latest and best hardware and software, with this type of grid, customers will have the flexibility of tapping into computing power and programs as needed, just as they do gas or electricity. But enterprises are coming more and more to see the e-sourcing trend as a continuum—reaching beyond commonplace IT resources on demand to the delivery of business process and management functions integral to the way the organization works.

The *e-sourcing* business model is based on providing the components of IT function that are (largely) standardized and delivered through a service provider model. The attributes of this model include a distributed and shared environment, and generally standardized non-core business processes. The e-utility is used by consumers of the e-utility as building blocks for developing complex e-business solutions. The major properties of e-sourcing environments are a standard solution that requires minimal configuration; pooled resources used to serve

multiple customers; capacity on demand; and scalable, 24x7, always on, high availability, rapidly deployable, minimal operations overhead; shared systems management; and flexible pricing and billing based on either actual usage/consumption of resources, or a calculated flat rate subscription.

8.4 Phases and activities

Deciding which grid type and topology to chose from is just the first step in the grid architecture design. A mature end-to-end design methodology is comprised of distinct phases and activities. The activities in the architecture design phase of the project include a review of the detailed architectural decisions and design documentation for the current infrastructure, conducting interviews and workshops, the modification of the initial high-level design based on new requirements and the results of the detailed assessment, the creation of a detailed modular architecture design, and the creation of the implementation and transition plan.

8.4.1 Basic methodology

For building a grid architecture, using a basic methodology allows the design to follow a consistent path from beginning to end. A methodology is not a cookbook for building a grid architecture, but a way to trace the progress of the design from the kickoff meeting to the final end state. The methodology follows a reproducible set of guidelines that can be used over again based on a set of successful guiding principals for architecture design. A methodology allows the architecture to follow a set of principals that can be documented from beginning to end throughout the design.

We define one such basic design methodology for developing the grid conceptual architecture in the next three sections.

Understanding the business drivers

The first step of any design is to identify and document the business drivers that are the foundation behind building the grid. The business drivers outline the investment and what the end state will accomplish. The business drivers or business strategy is the foundation or reasoning behind building the grid. Whether the goal is to tie together or build a federated database with your suppliers or tie together a set of computers to harness their overall processing power, you should have an end goal in mind before the design begins.

Requirements gathering

The requirements gathering process will help drive the architecture process by helping the technical team work within a set of guidelines for the architecture. By following this process, all of your decisions can be tied back to the basic requirements and business drivers for the design. Along with your solution objectives, the requirements will offer a road map for you to follow work through the design phases.

- Business requirements

  The business requirements are a subset of the business drivers that are focused on solving a specific business need. The business requirements drive important areas within the design, such as the performance and availability of the environment. Helping to understand these key service levels is an important part of the design.

- Infrastructure requirements

  The infrastructure requirements provide the basic framework for how the infrastructure will be designed. There are many different variables for how the grid architecture can be designed and, based on what the requirements will be, will shape how the environment will look.

- Application requirements

  There are many factors that need to be accounted for during the design, and the application is one of them. Possibly one of the most important requirements that must be validated is to ensure that the application in question can be made *grid-aware*. Unless the application can take advantage of the grid resources or split the workload across multiple components, the power of the grid is wasted.

Validate requirements

During the course of some designs, the requirements can change at the last minute or may go undiscovered. Requirements also have a way of changing when you least expect them to, so it is always a good idea to validate them before you proceed. Validating the requirements one last time before the design phase begins is a good way to ensure that all parties agree with the direction of the design.

8.4.2 Recommended steps

The following sections deal with additional recommended methods for developing an optimal grid design. These methods include attending grid design workshops and building prototypes once the design has been completed.

Grid design workshops

The purpose of the grid design workshops is to help all of the parties involved to better understand the variables, options, and considerations that need to be taken into account when developing a grid infrastructure design. Many or most of the grid middleware, technologies, and system components are probably new to many people within the design team and it is always a good idea to hear firsthand from experienced IT professionals the means by which grid infrastructures can be implemented, as well as any pitfalls to watch out for when designing environments for grid computing.

Documentation

An extremely critical means of communicating the design (your *solution*) of your grid infrastructure is via an architecture or solution document. The solution document should start with a high-level overview of the environment and subsequently should drill down into the most detailed configuration diagrams and descriptions possible. You will want to include things like IP addresses, network routes, server names, server architectures, network hardware, and essentially everything you know about the infrastructure at the time your design is completed. In truth, architecture documents are often dynamic, changing as the needs of the system users change and as technologies mature, become obsolete, and are replaced by newer technologies. You should revise your architecture document upon further hardware and software updates so that it accurately reflects the state of the system. Without an accurate architecture document, the system implementation team may get easily confused and not produce the system that was originally designed. Additionally, anyone adding further design changes to the system after the original system architect has moved on will appreciate an up-to-date architecture document, as it will save him or her countless hours of information gathering that would be necessary without an architecture document.

Prototype

Building a prototype of a grid system can save significant time that would otherwise be spent debugging and re-tooling unforeseen system incompatibilities. Your goal in building a prototype should be to produce a small-scale, end-to-end backbone of what your production environment will look like. It should include all interoperating technologies and/or architectures, so that if any incompatibility exists, it will be apparent before the production system is implemented. When all of the kinks are ironed out of your prototype, you will be confident that all of your components will work together properly in your designed infrastructure, and, additionally, you will have some experience in the implementation of such a system. Lessons learned from building the prototype should be reflected in your architecture document and any other directions provided to the implementation team.

8.5 A conceptual architecture

The purpose of the grid conceptual architecture is to establish a common understanding between the business owners and the people architecting and designing the grid infrastructure by describing the grid architecture that will support the client business requirements.

This section highlights some of the common components that you can choose from within the Globus Toolkit. If you are designing a grid architecture using different grid middleware software from Platform, DataSynapse, Avaki, or any other grid software provider, this section should still give you a head start on grid architecture. You will still be faced with decisions on the basic components, such as the security models, workload management, information services, and data sharing.

The conceptual model is a high-level framework consisting of the grid system components and nodes within the design. The nodes represent the different system components and grid middleware that make up the design. Normally, the conceptual model is the first graphical view of the grid infrastructure and is used as a stepping-stone to building a detailed configuration for the grid network. The graphic depiction of the grid environment will allow you to see how the requirements were gathered and how the many grid components will interact with one another.

8.5.1 Infrastructure

The infrastructure represents the physical hardware and software components used to interconnect different grid computers. These components help support the flow of information between grid systems and provide the basic set of services for connectivity, security, performance availability, and management. While many of these infrastructure components supply basic functionality to the grid, many are optional. It will be up to you to decide on the requirements and how well these components match up to the needs of your design.

Security

Chapter 7, “Security” on page 63, provides details about considerations related to security in a grid environment. Please refer to that chapter for more details on security.

One issue not addressed in detail in the chapter referenced above is the used of firewalls. The use of firewalls can provide logical and secure segmentation between grid systems. You might want to use firewalls to protect your networks and grid servers by limiting the types of services and protocols that connect to your computers. By using firewalls within your grid design, you can help limit the

network communication between grid systems and only use protocols that you specify that the firewall will support.

Firewalls are not the only answer to protecting your grid servers, but they do add an additional layer of defense from internal or external users trying to access your systems. Firewalls work by controlling access to network services that your grid computers will be running. Since the network offers a gateway to your grid systems, you want to make sure that you control exactly the services and protocols that can be used to access your systems, as well as who can initiate communications.

For the most up-to-date information regarding the Globus Toolkit and firewalls, you should check out the firewall section on the Globus Web site at:

http://www.globus.org/security/

Some areas you may want to protect within your design are:

- Certificate Authority/Registrant Authority
- Globus Toolkit components, such as MDS, GRIS, and GIIS (For more information about these and other Globus Toolkit components, refer to 7.2, “Components of Globus Toolkit” on page 133.)
- Databases
- All grid servers

Networks

The network design within the grid architecture can take on many different shapes. The networking components can represent the LAN or campus connectivity or even WAN communication between the grid networks. Whatever the case may be, the network’s responsibility is to provide adequate bandwidth for any of the grid systems. Like many other components within the infrastructure, the networking can be customized to provide higher levels of availability, performance, or security.

Grid systems are for the most part network intensive due to security and other architectural limitations. For data grids in particular, which may have storage resources spread across the enterprise network, an infrastructure that is designed to handle a significant network load is critical to ensuring adequate performance.

Systems management

Any design will require a basic set of systems management tools to help determine availability and performance within the grid. A design without these tools is limited in how much support and information can be given about the health of the grid infrastructure. Some networks within a grid architecture can be

dedicated to perform these functions as to not hamper the performance of the grid.

Storage

The storage possibilities are endless within a grid design. How that storage will be secured, backed up, managed, and replicated are some of the questions that the grid design will try to answer. Within a grid design, you want to make sure that your data is always available to the resources that need it. Besides availability, you want to make sure that your data is properly secured, as you would not want unauthorized access to sensitive data. Lastly, you want more than decent performance for access to your data. Obviously, some of this relies on the bandwidth and distance to the data, but you will not want any I/O problems to slow down your grid applications. For applications that are more disk-intensive, or for a data grid, more emphasis can be placed on storage resources, such as those providing higher capacity, redundancy, or fault-tolerance.

8.6 Summary

This chapter provided an overview of some of the key criteria and general methodologies that should be considered when designing a grid computing environment.

9

Web services resource framework

A grid computing environment consists of a set of resources that are being shared, possibly across organizations. A dynamic collection of individuals, institutions, and resources is also known as a virtual organization.

This concern for resource sharing sets a grid computing environment apart from a traditional distributed computing environment. Traditionally, object-oriented distributed systems do not deal with resource sharing and management issues. A grid computing environment is essentially a distributed computing environment that also deals with heterogeneous resource sharing and management.

The sharing of a resource could range from simple file transfers to complex and collaborative problem solving. A resource can potentially be any IT infrastructure component such as software application, database, cluster, network capacity, software licence, storage, and so on.

The resource sharing is required to occur under the control of a well-defined set of conditions and policies. In this context the key issues associated with resource sharing include discovery, authentication, authorization, and access mechanisms.

The resource sharing is further complicated when a grid is introduced as a solution for utility computing, where commercial applications and resources

become available as shareable and on demand resources. However, issues such as metering, accounting and billing, quality of services compliance, and so on, are out of the scope of this book.

This chapter aims to introduce some of the fundamental concepts in resource state management as they are currently defined in the context of a grid computing environment.

9.1 Resource state management using Grid services

In the grid context a resource is assumed to represent some state or data and provides some useful capability via an interface.

An interface associated with a resource defines a logical grouping of operations that can be invoked by its clients.

In the recent past we have observed increasing popularity of service oriented architecture frameworks. The emergence of service oriented architecture (SOA) helps grid resources to advertise their capabilities through a standard service interface.

Web services are open standards-based mechanisms to make services available to whatever client program can take advantage of them. Web services are becoming a popular way to implement various components of a service oriented architecture and many organizations are becoming very familiar with Web services technologies and capabilities.

However, as those familiar with Web services know, Web services are typically stateless. That is, there is no memory between separate transactions invoked on the same service instance. However, for grid computing, the state of a resource or service is often important and therefore may need to persist across transactions.

Other than this little (actually somewhat major) difference, there are many similarities between Grid services and Web services. It would be a shame not to find a way to take advantage of the standards and facilities already provided by Web services when defining and implementing Grid services. We explore this possibility in the discussion that follows and describe what the differences are between Grid services and Web services and how these differences can be addressed.

9.1.1 What a Grid service is

A service interface associated with a grid resource is known as a Grid service. A resource and its state is controlled and managed via Grid services in a Grid environment. A Grid service may require access to more than one resource or vice versa. It is also possible that multiple Grid services access the same resource or a Grid service can create a new instance of a resource every time it is invoked.

Various grid resources may require to interact and integrate with each other depending on business requirements. It is most likely that the resources are hosted in a technologically heterogeneous environment. Therefore, a framework

is required that abstracts environment-specific resource implementation details from the actual interGrid service messaging. A service oriented architecture (SOA) provides such a framework.

It follows that an open standards compliant SOA architecture would make it easier to integrate heterogeneous resources and various layers of the grid architecture. Such an architecture would help us achieve distributed resource sharing across heterogeneous and dynamic virtual organizations, that is, grid computing.

The Global Grid Forum (GGF) has adopted an SOA principles based Open Grid Services Architecture (OGSA) that provides a framework for implementing a Grid.

All of the resources (physical or logical) in an OGSA-compliant grid are modeled as Grid services. These Grid services are built on top of a SOA leveraging WEB services technology. This enables a Grid service to use the capabilities of the Web services messaging model, service descriptions, and discovery. Various Web services standards have evolved to enable secure and reliable Web services transactions. The choice of Web-services technology to implement the OGSA-compliant Grid services leverages investment in Web-services architecture and its standards.

9.1.2 Grid services vs. Web services

Although Grid services are implemented using Web-services technology, there is a fundamental difference between a Grid service and a Web-service.

A Web-service addresses the issue of discovery and invocation of persistent services. A Web Services Description Language (WSDL) compliant document points to a location that hosts the Web service.

A Grid service addresses the issue of a virtual resource and its state management. A grid is a dynamic environment. Hence, a Grid service can be transient rather than persistent. A Grid service can be dynamically created and destroyed, unlike a Web service, which is often presumed available if its corresponding WSDL file is accessible to its client. Web services also typically out live all their clients.

This has significant implications for how Grid services are managed, named, discovered, and used. The OGSA model adopts a Factory design pattern to create transient Grid services. Thus, an OGSA Grid service is a potentially transient Web service based on grid protocols using WSDL.

9.1.3 OGSA Grid service requirements

From the OGSA perspective, a grid environment consists of typically few persistent and potentially many transient Grid services. All Grid services must comply with the OGSA-required interface specifications to enable reliable and secure management of a distributed state of virtual resources.

The following are some of the key capabilities that the OGSA Service Model requires a compliant Grid service to provide:

- Creation: This refers to creating new instances of resources associated with a Grid service via an operation. An instance can be newly created or be initialized from a persistent state of a resource.
- Global naming and references: Once we have an instance of a resource, a grid environment requires a unique network-aware reference to a resource instance with information about how to interact with the instance via the Grid service.
- Lifetime management: The lifetime management operation defines the life-span of a resource, mainly dependent on whether a resource expires after a certain time period or immediately.
- Registration and discovery: This set of operations refers to the ability to find Grid service instances and their associated deploy-time and run-time meta data.
- Notification: The notifications are asynchronous messaging mechanisms to notify subscribing clients of certain events such as resource life-time events, property changes, and so on.

The OGSA Grid services also address authorization, concurrency control, and manageability aspects.

There are two standards currently available to implement OGSA-compliant Grid services:

- Open Grid Services Interface (OGSI) Grid services
- Web Services Resource Framework (WSRF) Grid services

Both frameworks provide mechanisms to implement OGSA-compliant Grid services in different ways.

Next we review and compare both approaches at a high level and discuss WSRF in greater detail for the rest of the chapter.

9.1.4 Open Grid Services Interface (OGSI) Grid services

The Open Grid Services Interface defines rules about how OGSA can be implemented using Grid services that are Web services extensions.

The OGSI specification defines a Grid service instance as “a Web service that conforms to a set of conventions expressed by WSDL as service interfaces, extensions, and behaviors.”

The OGSI specification defines Grid services features that include:

- Statefulness
- Stateful interactions
- The ability to create new instances
- Service lifetime management
- Notification of state changes and Grid service groups

The OGSI model requires Grid services to be specified via Grid Web Services Definition Language (GWSDL), which is an extension of WSDL.

The OGSI 1.0 specification defines the following interfaces that should be implemented by a Grid service.

Table 9-1 OGSI interfaces for a Grid service

Interface	Description
GridService	Encapsulates the root behavior of the service model. This interface is mandatory for a OGSA service based on OGSI 1.0.
HandleResolver	The OGSI method of creating an instance of a Grid service returns a handle. This handle is mapped to a reference, which then has enough information to enable client communication with the actual instance of a grid resource via a Grid service. This interface provides the functionality to map a Grid Service Handle (GSH) to a Grid Service Reference (GSR).
NotificationSource	Allows clients to subscribe to notification messages.
NotificationSubscription	Defines the relationship between a single NotificationSource and NotificationSink pair.
NotificationSink	Defines a single operation for delivering a notification message to the service instance that implements the operation.

Interface	Description
Factory	This is the standard operation for creation of Grid service instances.
ServiceGroup	This allows Grid services to be added and removed from a ServiceGroup. A ServiceGroup is a collection of Grid service instances.
ServiceGroupRegistration	This allows Grid services to be added and removed from a ServiceGroup.
ServiceGroupEntry	This defines the relationship between a Grid service instance and its membership within a ServiceGroup.

The portType construct of the WSDL grammar defines the functional interface implemented by a Web service. An OGSI-compliant Grid service component extends the GridService portType. The component may optionally extend other portTypes as listed in the previous table along with any application-specific portTypes, as required. The OGSI model also extends WSDL with mechanisms to specify additional state data descriptions.

The diagram below depicts the layering of various OGSI components.

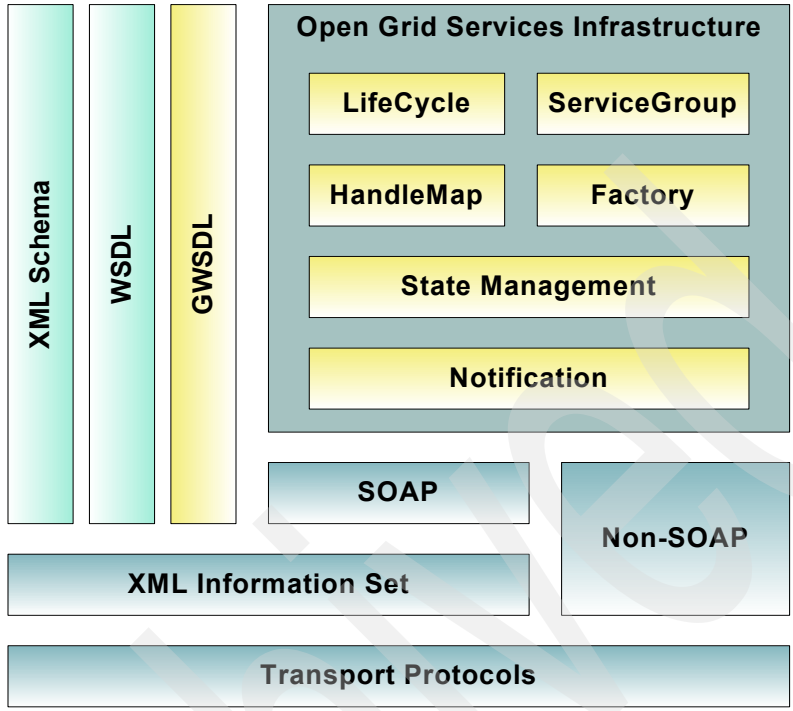

Figure 9-1 OGSI components

Please refer to the *Open Grid Service Infrastructure 1.0* and the *Open Grid Services Infrastructure Primer* documents available from the Global Grid Forum Web site (http://www.ggf.org) for more information about OGSI.

9.1.5 OGSI to WSRF refactoring

The Globus Toolkit 3 (GT3) contains a reference implementation for the OGSI. However, its implementation through extensions to some of the Web services standards and the continuing evolution of Web services has made it more difficult for the Web services and Grid services to continue to merge than originally hoped. The Web Services-Resource Framework (WR-RF) provides a promising solution that can address the needs of Grid services while still holding true to the Web services foundation.

The main issue of contention was the perceived divergence of the OGSI specification from the popular practices in the Web services community at large. The main objective behind the WSRF refactoring is to bring the Grid services and Web services communities closer together.

Please note that it is possible to build and deploy OGSA-compliant, Web services based Grid services using both OGSI and WSRF proposed specifications.

The following itemizes some of the key issues observed with the OGSI approach:

- Too much in one specification: The OGSI did not have a clean separation of functions to support incremental adoption. For example, Table 9-1 on page 120 has a list of the full range of interfaces that can be implemented by a Grid service. The OGSI does not provide a way to partition these functions and adopt them incrementally.

  The WSRF set of specifications partition the equivalent functionality in separate specifications, and they can be adopted incrementally.

- Does not work well with existing Web services and XML tooling: The XML syntax used with OGSI 1.0 causes problems with JAX-RPC standard APIs.

  The WSRF set of specifications use standard XML Schema mechanisms that are familiar to developers and is supported by the existing tooling. The WSRF utilizes WSDL 1.1 compliant methods to associate the XML information model of a resource with a resource's operations instead of Service Data Elements used by OGSI.

- Too object oriented: The OGSI 1.0 models a stateful resource as a Web service that encapsulates the resource's state, with the identity and life cycle of the service and resource state coupled. From a purist Web services point of view, "Web services do not have any state or instances."

  The WSRF set of specifications provides a distinction between the service and the management of stateful entities and their state by that service. The WS-Addressing standard is used by the WSRF set of specifications to formalize the relationship between Web services and the stateful resources.

- Introduction of forthcoming WSDL 2.0 functionality as unsupported extensions to WSDL 1.1: The OGSI exploited features of the WSDL 2.0 draft specification, making it difficult to support the OGSI with existing Web services tooling and runtimes.

  The WSRF set of specifications relies on WSDL 1.1 constructs to avoid incompatibility issues.

The *A Grid Application Framework based on Web Services Specifications and Practices* paper by Parastatidis, et. al. [6], provides further discussion on issues encountered with OGSI specifications.

At a high level the OGSI-to-WSRF refactoring has resulted in the following:

- The notion of a Grid service as a WS-Resource.

- A better separation of functions (listed in Table 9-1 on page 120) by splitting the functionality in separate specifications.
- WS-Notification specification that can be used to build state change notifications using Web services.

The next section formally introduces the Web Services Resource Framework and Web Services Notification families of specifications.

9.2 WSRF fundamentals

In the previous discussion we described a Grid service as a service representation of a resource. A grid resource is normally assumed to represent some state. This section introduces the concept of a WS-Resource and associated modelling concepts that underpin the WSRF and WS-Notification families of specifications.

9.2.1 What a WS-Resource is

The WS-Resource is a construct used to model stateful resources using a Web services architecture framework.

According to WSRF, a stateful resource:

- Has its state data described as an XML document
- Has a well defined life-cycle
- Is known to and accessed by one or more Web services

A stateful resource modelled using the WS-Resource construct can be implemented in a variety of different ways. It can be implemented as a file on a file system or a record in a database table or may reside in memory as an application-specific data structure.

The diagram below depicts the relationships amongst a hypothetical movie scene rendering service and several stateful resources such as the actual scene data, special effects to be applied, and the rendering styles for television and wide-screen display.

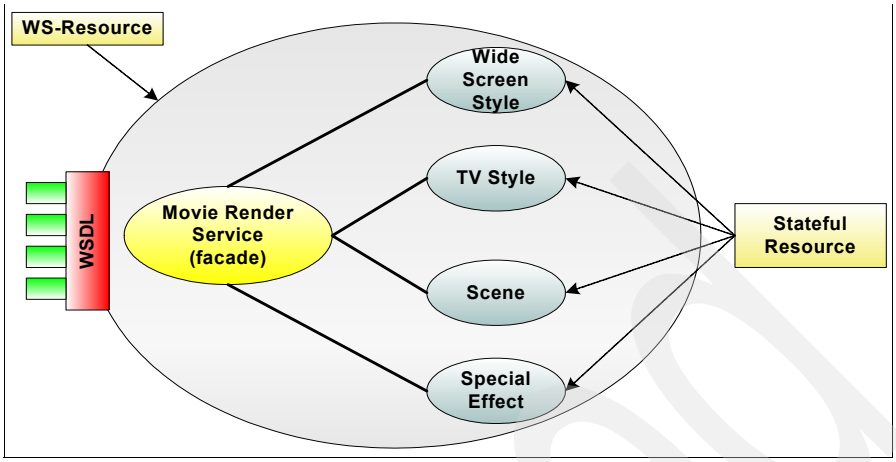

Figure 9-2 Example of a WS-Resource to WSDL relationship model

In the diagram above the resources are modelled as WS-Resources and the movie-rendering service is exposed as a Web service via its WSDL interface file.

The operations available from the movie-rendering service and the attributes of various resources are defined in the WSDL file.

It is important to note that the service and the resources are seen as a single *bundle* via the WSDL file by the service clients. The clients of the movie-rendering service in the above example never deal with a resource instance directly, but implicitly via interactions with the WSRF-compliant movie render service.

This implicit interaction with WS-Resource instances is known as the *Implied Resource Pattern*.

When a stateful resource is associated with a Web service, we refer to the component resulting from the composition of the Web service and the stateful resource as a WS-Resource.

It follows from the WS-Resource Framework discussion so far that a WS-Resource is an association of a Web service and at least one stateful resource. The general understanding of a Web service suggests that a Web service exposes an interface via a portType construct. The portType construct advertises one or more publicly available operations that can be invoked by a Web service client.

A Web service becomes a WS-Resource when a portType definition within a WSDL file is associated with an XML representation of the properties of a stateful resource in a WS-Resource Framework specific way.

Figure 9-3 is an example WSDL fragment that shows the association.

```
<definitions .name=....
.........
   xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
................
         <xsd:element name="atmosphere" type="xsd:boolean" />
         <xsd:element name="water" type="xsd:boolean" />
         <xsd:element name="name" type="xsd:string" />

         <xsd:element name="GenericPlanetProperties">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element ref="atmosphere" minOccurs="1" maxOccurs="1"/>
                  <xsd:element ref="water" minOccurs="1" maxOccurs="1"/>
                  <xsd:element ref="name" minOccurs="1" maxOccurs="1"/>
                  <xsd:any/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>

<wsdl:portType name="SolarSystem" wsrp:ResourceProperties="tns:GenericPlanetProperties">
<operation name=".../>
</wsdl:portType>
................
</definitions>
```

Figure 9-3 Example WSDL fragment showing WS-Resource definition

In the above WSDL fragment, the GenericPlanetProperties are associated with a SolarSystem portType. This association makes the WSDL represent a WS-Resource.

The next section discusses the implied resource pattern.

9.2.2 Implied resource pattern for stateful resources

One of the stated criticisms of OGSI 1.0 was that it was “too object oriented.” The implied resource pattern aims to distinguish between the actual service from the management of stateful resource instances.

The Web services are stateless. Therefore, when Web services operations are involved with dynamic state, these are the following options:

- The state is provided explicitly within the request message.

- The state is maintained implicitly via sub systems with which a Web service interacts.

The implied resource pattern implements the second option from the above. The actual state management and instance management of stateful resources is delegated to an external component. This is the approach selected for WSRF.

The WSRF implementation implicitly passes the resource identifier information when message interaction occurs between a client and a WS-Resource. By implicit it is meant that the client does not explicitly include a resource identifier in its request. Instead, the requisite identifier is implicitly associated with a message exchange. A resource identifier can be dynamically or statically associated with a message exchange.

The implied resource pattern in WSRF parlance utilizes a set of conventions such as XML, WSDL, and WS-Addressing in particular.

The WS-Addressing plays an important role in implementing the implied resource pattern.

The WS-Addressing standardizes the way Web service addresses are represented. Such a representation is known as an End Point Reference (EPR). Besides the Web service address an EPR can also represent enough contextual information to enable client communication with a WS-Resource.

The EPR contains two pieces of information:

- The Web service address information
- The resource properties information that may include an identifier to a resource instance besides other meta data about the service.

In the WSRF, an EPR with a resource identifier is also known as a *WS-Resource qualified end point reference*.

The resource identifier points to a stateful resource used when the Web service is invoked. The Web service maps the identifier to a stateful resource based on its business requirements.

A resource identifier's creation is analogous to creating a new instance of a WS-Resource. A new instance of a WS-Resource can be created via a WS-Resource Factory or some other application. A WS-Resource Factory Web service brings new instances of WS-Resource into existence.

Creating a new instance of a WS-Resource involves the following:

1. Creating a new instance of the resource
2. Assigning a new identifier to the new resource instance

3. Creating an association between the new resource instance and its corresponding Web service

A WS-Resource Factory's operation responsible for creating new instances of WS-Resource may return a WS-Resource qualified EPR or save the equivalent information elsewhere, such as a registry or a database for later retrieval.

Because the stateful resource's identifier is included in a WS-Resource qualified EPR the client is not required to have specific knowledge of the location of the Web service nor the resource identifier.

The actual semantic meaning of a resource identifier is Web service implementation-specific. At the current time there are no specifications that provision a resource identifier definition.

When a client application interacts with a WS-Resource compliant Web service, the XML representation of the concerned EPR is implicitly sent along with the request opaque to the client. If the EPR resource properties contain a resource identifier, then it gets sent along with the rest of the request in a Web service message.

From a client application perspective an EPR represents a pointer to a WS-Resource. The EPR may contain a resource identifier to target a client's interaction with a specific instance of a WS-Resource via a Web service. The resource identifier is required to be unique enough to enable a Web service to uniquely identify a stateful resource instance. The resource identifier is not required to be unique outside the scope of the Web service concerned.

The diagram below depicts how a WS-Resource-qualified EPR gets involved when a client interacts with a WS-Resource.

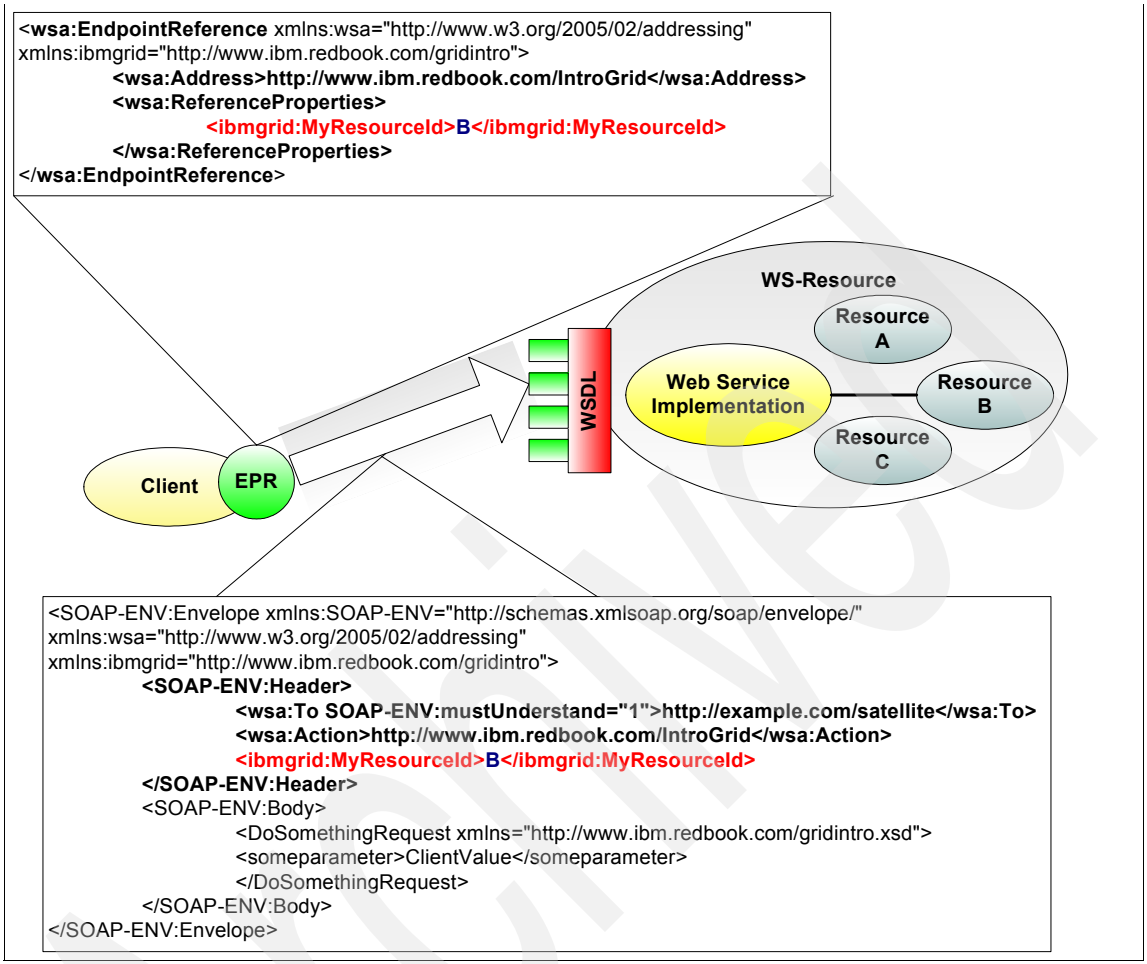

Figure 9-4 Using a WS-Resource qualified endpoint reference

The WS-Addressing specification mandates that the ReferenceProperties part of an EPR must be sent as part of any message that is directed towards a Web service identified by an EPR. How the information is actually sent is dependent on protocol-binding specifics.

In the above example, the client holds an EPR that points to a (fictional) Web service at location http://www.ibm.redbook.com/IntroGrid and identifies a stateful resource B. Because this EPR has a resource identifier in its ReferenceProperties stanza, recall that this becomes a WS-Resource-qualified endpoint reference.

When the client invokes the DoSomethingRequest operation on the Web service portType, the information contained within the ReferenceProperties stanza of the EPR XML document is sent as part of the SOAP header.

The Web service extracts the resource identifier value (B) and locates the corresponding resource to work with and completes the DoSomethigRequest.

Inspecting the SOAP message's *body* element reveals that the client request for the DoSomethingRequest only passes the operation-specific parameter (that is, SomeParameter). The resource identifier is not passed explicitly by the client in the request made. This is the key to the implied resource pattern.

In the next section we review the WSRF and WS-N set of specifications and briefly discuss how they meet OGSA Grid service requirements.

9.3 WS-Resource Framework specifications

The OGSI to WSRF transition is a refactoring exercise for various reasons briefly discussed in 9.1.5, "OGSI to WSRF refactoring" on page 122, which implies that collectively these specifications retain the same functionality present in OGSI.

The OGSI refactoring results in five WSRF specifications and three WS-Notification family specifications. The WS-Notification family of specifications addresses event notification subscription and delivery.

Each of the specifications targets a grouping of functionality. This facilitates the flexible composition of various functionality in an incremental or mix-and-match fashion.

This section gives an overview of the WSRF family of specifications.

The WS-Resource Framework paper by Czajkowski, et. al. [5], summarizes the various WS-Resource Framework specifications, as shown in Table 9-2.

Table 9-2 WS-Resource Framework specifications summary

Specification name	Description
WS-ResourceProperties	Describes associating useful resources and Web services to produce WS-Resources and how elements of publicly visible properties of a WS-Resource are retrieved, changed, and deleted
WS-ResourceLifeTime	Allows a requestor to destroy a WS-Resource either immediately or at a scheduled future point in time

Specification name	Description
WS-RenewableReferences	Annotates a WS-Addressing endpoint reference with policy information needed to retrieve a new endpoint reference when the current reference becomes invalid
WS-ServiceGroup	Creates and uses heterogeneous by-reference collections of Web services
WS-BaseFault	Describes a base fault type used for reporting errors

The following table summarizes the WS-Notification family of specifications.

Table 9-3 WS-Notification Specifications summary

Specification name	Description
WS-BaseNotification	Defines Web service operations to define the roles of notification producers and notification consumers.
WS-BrokeredNotification	Defines Web service operations for a notification broker. A notification broker is an intermediary which, among other things, allows publication of messages from entities that are not themselves service providers. It includes standard message exchanges to be implemented by notification broker service providers along with operational requirements expected of service providers and requestors that participate in brokered notifications.
WS-Topics	Defines a mechanism to organize and categorize topics. It defines three topic expression dialects that can be used as subscription expressions in *subscribe request* messages and other parts of the WS-Notification system. It further specifics an XML model for describing meta data associated with topics.

Figure 9-5 on page 132 provides an overview of WS-Resource Framework and how it relates to other Web service specifications.

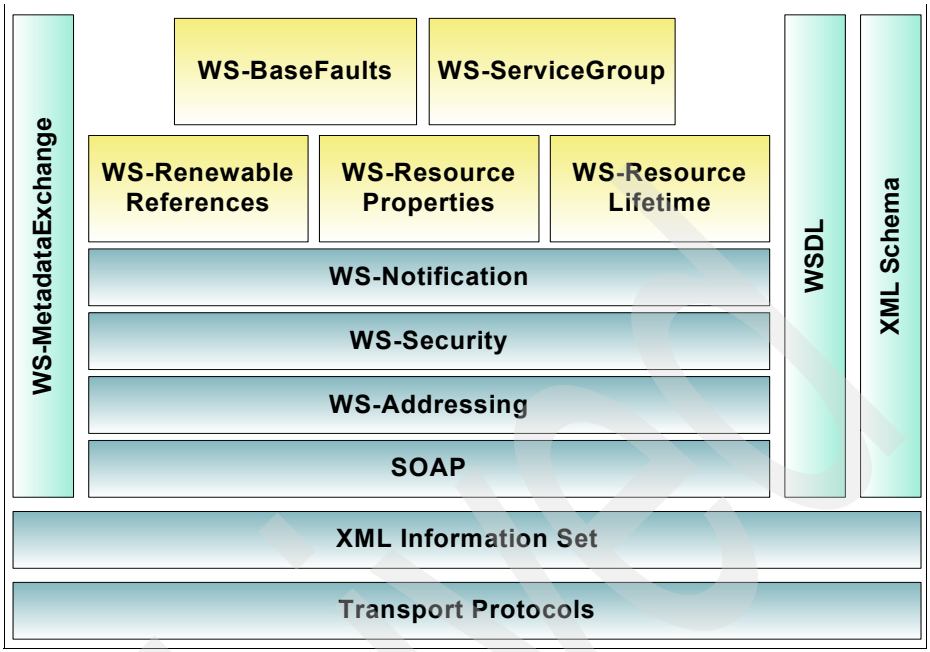

Figure 9-5 WS-Resource Framework with Web service specifications

The diagram above is comparable with Figure 9-1 on page 122.

The *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution* paper by Czajkowski, et. al. [3], maps primary OGSI constructs to WS-Resource Framework and WS-Notification constructs, as shown in Table 9-4.

Table 9-4 OGSI to WS-Resource Framework and WS-Notification map

OGSI	WS-Resource Framework
Grid Service Reference	WS-Addressing Endpoint Reference.
Grid Service Handle	WS-Addressing Endpoint Reference and WS-RenewableReferences.
HandleResolver portType	WS-RenewableReferences.
Service Data Definition	Resource properties definition.
GridService porType service data access	WS-Resourceproperties.
GridService portType lifetime management	WS-ResourceLifetime.

OGSI	WS-Resource Framework
Notification portTypes	WS-Notification.
Factory portType	Now treated as a WS-Resource Factory concept. Please refer to 9.2.2, “Implied resource pattern for stateful resources” on page 126.
ServiceGroup portTypes	WS-ServiceGroup.
Base fault type	WS-BaseFault.
GWSDL	Copy-and-paste. Uses existing WSDL 1.1 interface composition approaches (that is, copy and paste) rather than using WSDL 2.0 constructs.

The following are a few observations based on the OGSI-to-WSRF comparison table above:

- The implied resource pattern and the concept of WS-Resource replaces the GridService interface as defined by the OGSI 1.0 specification.
- The Grid Service Handle (GSH) and Grid Service Reference (GSR) concepts are replaced by the WS-Addressing standard. The EPR introduced by WS-Addressing is equivalent to GSH and GSR.
- The WSRF introduces a standard notification framework for Web services enabling Grid services and Web services to share notification patterns defined by WS-Notification specifications.

Please refer to The *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution* paper by Czajkowski, et. al. [4], for a detailed discussion about each of the items in the table above.

Figure 9-6 on page 134 gives a high-level view of a SolarSystem WS-Resource that implements WS-ResourceProperties interface functions.

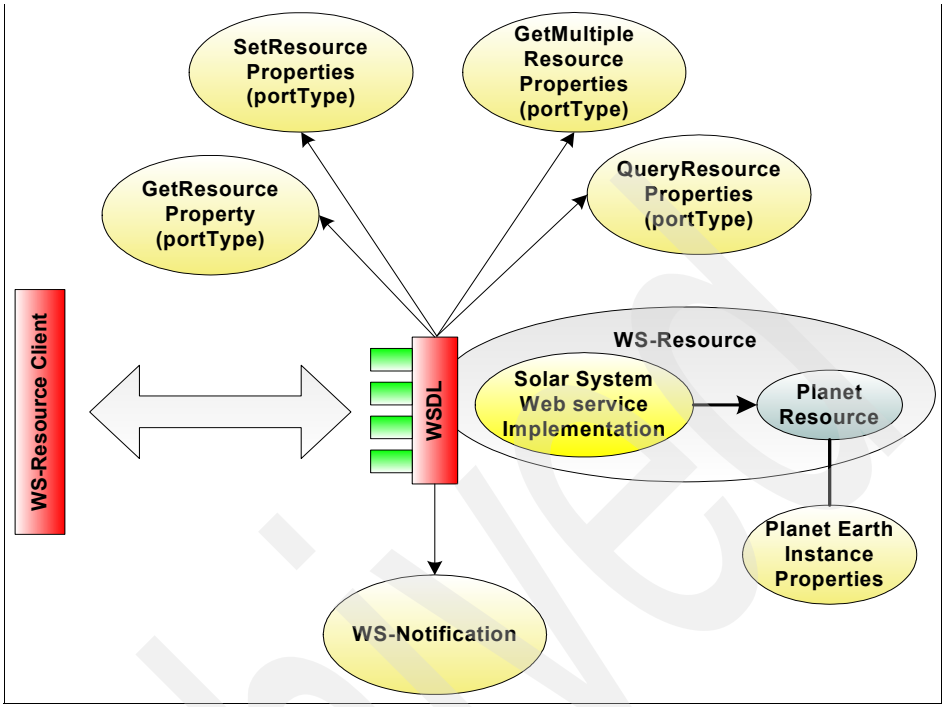

Figure 9-6 A SolarSystem WS-Resource with WS-ResourceProperties interfaces

Please note in the diagram above that the *SolarSystem* Web service is delegating the actual planet instance management to a separate *Planet* resource component.

The WS-Resource client invokes the WS-ResourceProperties interface functions via the information provided in the WSDL file.

The SolarSystem WS-Resource generates a notification when a Planet resource instance's property changes. The notification message format is also declared in the WSDL file.

Figure 9-7 on page 135 is an example GetResourceProperty request and response with our SolarSystem example.

```
<wsrp:GetResourcePropertyRequest xmlns:tns="http://test.org/computersystem" >
        tns:name <!—name of the property to retrieve - ->
</wsrp:GetResourcePropertyRequest>
```

```
<wsrp:GetResourcePropertyResponse xmlns:tns="http://test.org/computersystem" >
        <tns:name> <!—an XML view of the resource property- ->
                Earth
        </tns:name>
</wsrp:GetResourcePropertyResponse>
```

Figure 9-7 Example WS-ResourceProperties request and response for GetResourceProperty operation

The *WS-Resource Framework Interop Workshop #1 - Scenarios (v0.13),* available from the following Web site, has numerous example messages for the rest of the WS-ResourceProperties and WS-Notification operations.

http://www.ibm.com/developerworks/offers/WS-Specworkshops/ws-rf200404.html

The *Understanding WSRF* series of tutorials on IBM developerWorks® also provides numerous WS-Resource examples and further discussion about the fundamentals of the WS-Resource Framework.

The next section discusses the role of the WS-Resource Framework within the Globus Toolkit 4 (GT4).

9.3.1 WS-Resource Framework and Globus Toolkit 4

The WS-Resource Framework introduces the notion of WS-Resource. We have seen earlier in Table 9-4 on page 132 and the subsequent discussion that the notion of WS-Resource replaces the Grid service as it was defined with OGSI 1.0.

When a WS-Resource is packaged as a Grid Archive (GAR) and deployed in a GT4 container, it is recognized by the GT4 container as a valid GT4 WSRF compliant Web service. This is synonymous with a Grid service.

The above also implies that it is also possible to implement a non-OGSA, SOA environment using WS-Resource Framework compliant Web services.

Figure 9-8 on page 136 from the *A Globus Primer* by Ian Foster [7] depicts various Web service deployment scenarios within a GT4 container.

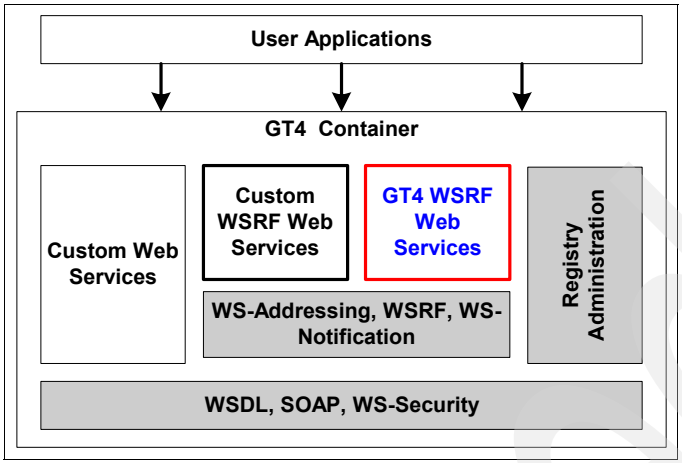

Figure 9-8 GT4 container services

The shaded areas of the above diagram represent a GT4 container's infrastructure components that allow it to host different services.

There are several components that are implemented as WSRF Web services within the GT4 container. A discussion of various GT4 components can be found in Chapter 10, "Globus Toolkit 4 components" on page 141.

At a high level the steps to implement a WSRF-compliant Web service for deployment in a GT4 container are as follows [7]:

1. Define the service interface. This refers to preparing a WSDL file that defines our WSRF service operations and may include resource properties definitions.
2. Implement the service. This refers to developing Java code for the WSRF service operations and associated properties, if any.
3. Define deployment parameters. This refers to preparing a Web Services Deployment Descriptor (WSDD) file for our service that defines various aspects of service configuration.
4. Compile and generate a GAR file. The compilation and GAR file creation involves creating appropriate stub files for handling SOAP messaging and packing the service in a format required by a GT4 container.
5. Deploy the service.

9.4 WSRF references

There are numerous tutorials available in the public domain to assist with WSRF services development and implementation. Some of the useful references are as follows:

- The Globus Toolkit 4 Programmer's Tutorial by Borja Sotomayor

  http://gdp.globus.org/gt4-tutorial/

- Understanding WSRF Parts 1 to 4 by Babu Sundaram

  http://www.ibm.com/developerworks

- Using Eclipse to develop Grid services

  http://www.ibm.com/developerworks/edu/gr-dw-gr-eclipseide-i.html

- Apache WSRF tutorial

  http://ws.apache.org/ws-fx/wsrf/tutorial/

- WSRF.NET Developer Tutorial by Mark Morgan and Glenn Wasson

  http://www.cs.virginia.edu/~gsw2c/WSRFdotNet/WSRF.NET_Developer_Tutorial.pdf

9.5 Summary

This chapter provided an overview of the WS-Resource Framework and how it enables the handling of state information within a Web services context. We also provided information about how it relates to the Open Grid Service Interface standard.

Part 3

Creating a grid environment with the Globus Toolkit 4

10

Globus Toolkit 4 components

The Globus Alliance is made up of organizations and individuals that develop and make available various technologies applicable to grid computing.

The Globus Toolkit, the primary delivery vehicle for technologies developed by the Globus Alliance, is an open source software toolkit used for building grid systems and applications. Many companies and organizations are using the Globus Toolkit as the basis for grid implementations of various types.

To learn more about the Globus Alliance, visit their Web site at:

http://www.globus.org

During the writing of this book, the Globus Toolkit is currently at Version 4. As we will see, the toolkit (as is implied by its name) consists of many components that can be used as the basis to implement a grid computing environment. It is not a complete grid solution, but provides the tools and facilities to address many of the requirements of grid computing. This chapter briefly describes the major components of Globus Toolkit 4.

10.1 Overview of Globus Toolkit 4

Globus Toolkit 4 is a collection of open-source components. Many of these are based on existsing standards, while others are based on (and in some cases driving) evolving standards. Version 4 of the toolkit is the first version to support Web service based implementations of many of its components. (Version 3 had included an OGSI implementation of some components, and Version 2 was not service based at all.)

Though many components have Web service based implementations, some do not, and for compatibility and migration reasons, some have both implementations.

Globus Toolkit 4 provides components in the following five categories:

- Common runtime components
- Security
- Data management
- Information services
- Execution management

Table 10-1 shows a list of components in Globus Toolkit 4, and identifies those that are Web service based and those that are not. In the sections that follow, we describe each of these in more detail.

Table 10-1 List of components in Globus Toolkit 4

	Web service based components				**Non Web service based components**	
Common runtime components	Java WS Core	C WS Core	Python WS Core		C Common Libraries	eXtensible IO (XIO)
Security components	WS authentication and authorization	Community Authorization Service (CAS)	Delegation service		Pre-WS authentication and authorization	Credential Management
Data management components	Reliable File Transfer (RFT)	OGSA-DAI	Data Replication Service (DRS)		GridFTP	Replica Location Service (RLS)
Monitoring and Discovery Services	Index service	Trigger service	Aggregator Framework	WebMDS	MDS2	

	Web service based components				Non Web service based components	
Execution management	WS GRAM	Community Scheduler Framework 4 (CSF4)	Globus Teleoperations Control Protocol (GTCP)	Workspace Management Service (WMS)	Pre WS GRAM	

10.2 Common runtime components

Globus Toolkit 4 includes common runtime components. Common runtime components consist of libraries and tools needed by both types of implementations and used by most of the other components.

10.2.1 Java WS Core

Java WS Core consists of APIs and tools that implement WSRF and WS-Notification standards implemented in Java. These components act as the base components for various default services that Globus Toolkit 4 supplies. Also, Java WS Core provides the development base libraries and tools for custom WS-RF based services. Figure 10-1 on page 144 shows the relation between Java WS Core and other services.

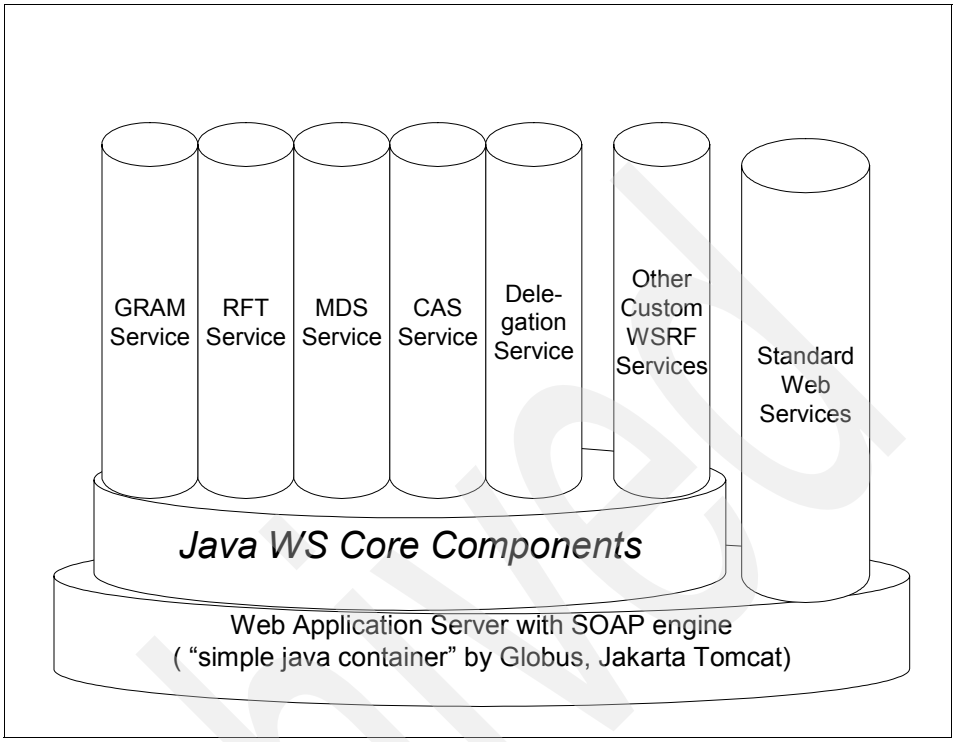

Figure 10-1 Relation between Java WS Core and Globus Toolkit 4 supplied services

For more information about WS-RF, refer to Chapter 9, “Web services resource framework” on page 115. Also, the following link should be of interest:

http://www.globus.org/toolkit/docs/4.0/common/javawscore/

10.2.2 C WS Core

C WS Core consists of APIs and tools that implement WS-RF and WS-Notification standards using C. For more information about C WS Core, look at the following link:

http://www.globus.org/toolkit/docs/4.0/common/cwscore/

10.2.3 Python WS Core

Python WS Core consists of APIs and tools that implement WS-RF and WS-Notification standards with Python. This component is also known as

pyGridWare, contributed by Lawrence Berkeley National Laboratory. For more information about Python WS Core, look at the following links:

http://dsd.lbl.gov/gtg/projects/pyGridWare/
http://www.globus.org/toolkit/docs/4.0/contributions/pythonwscore/

10.3 Security components

Because security is one of the most important issues in grid environments, Globus Toolkit 4 includes various types of security components.

10.3.1 WS authentication and authorization

Globus Toolkit 4 enables message-level security and transport-level security for SOAP communication of Web services. Also, it provides an Authorization Framework for container-level authorization. For more information, refer to Chapter 7, “Security” on page 63. Also look at the following link for more information about those components:

http://www.globus.org/toolkit/docs/4.0/security/message/

10.3.2 Pre-WS authentication and authorization

Pre-WS authentication and authorization consists of APIs and tools for authentication, authorization, and certificate management. For more information, refer to Chapter 7, “Security” on page 63. Also look at the following link for more information about those components:

http://www.globus.org/toolkit/docs/4.0/security/prewsaa/

10.3.3 Community Authorization Service (CAS)

CAS provides access control to virtual organizations. The CAS server grants fine-grained permissions on subsets of resources to members of the community. CAS authorization is currently not available for Web services, but it supports the GridFTP server. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/security/cas/

10.3.4 Delegation service

The Delegation service enables delegation of credentials between various services in one host. The Delegation service allows a single delegated credential to be used by many services. Also, this service has a credential renewal

interface, and this service is capable of extending the valid date of credentials. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/security/delegation/

Also, Figure 10-5 on page 153 provides an example of how this service is used by other services.

10.3.5 SimpleCA

SimpleCA is a simplified Certificate Authority. This package has fully functioning CA features for a PKI environment. In Chapter 11, “Globus Toolkit 4 installation and configuration” on page 155, we use SimpleCA as a Certificate Authority for our grid environment. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/security/simpleca/

Important: It is important to note that the simple CA is only recommended for testing or demo purposes. For any type of production grid, it is recommended that you evaluate commercial PKI solutions that may better suit your needs and remove the responsibility for managing your own CA.

10.3.6 MyProxy

MyProxy is responsible for storing X.509 proxy credentials, protecting them by pass phrase, and enabling an interface for retrieving the proxy credential. MyProxy acts as a repository of credentials, and is often used by Web portal applications. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/security/myproxy/

10.3.7 GSI-OpenSSH

GSI-OpenSSH is a modified version of the OpenSSH client and server that adds support for GSI authentication. GSI-OpenSSH can be used to remotely create a shell on a remote system to run shell scripts or to interactively issue shell commands, and it also permits the transfer of files between systems without being prompted for a password and a user ID. Nevertheless, a valid proxy must be created by using the **grid-proxy-init** command. Fore more information about GSI-OpenSSH, look at the following link:

http://www.globus.org/toolkit/docs/4.0/security/openssh/

10.4 Data management components

Globus Toolkit 4 provides various tools that enable data management in a grid environment.

10.4.1 GridFTP

The GridFTP facility provides secure and reliable data transfer between grid hosts. Its protocol extends the well-known FTP standard to provide additional features, including support for authentication through GSI. One of the major features of GridFTP is that it enables third-party transfer. Third-party transfer is suitable for an environment where there is a large file in remote storage and the client wants to copy it to another remote server, as illustrated in Figure 10-2.

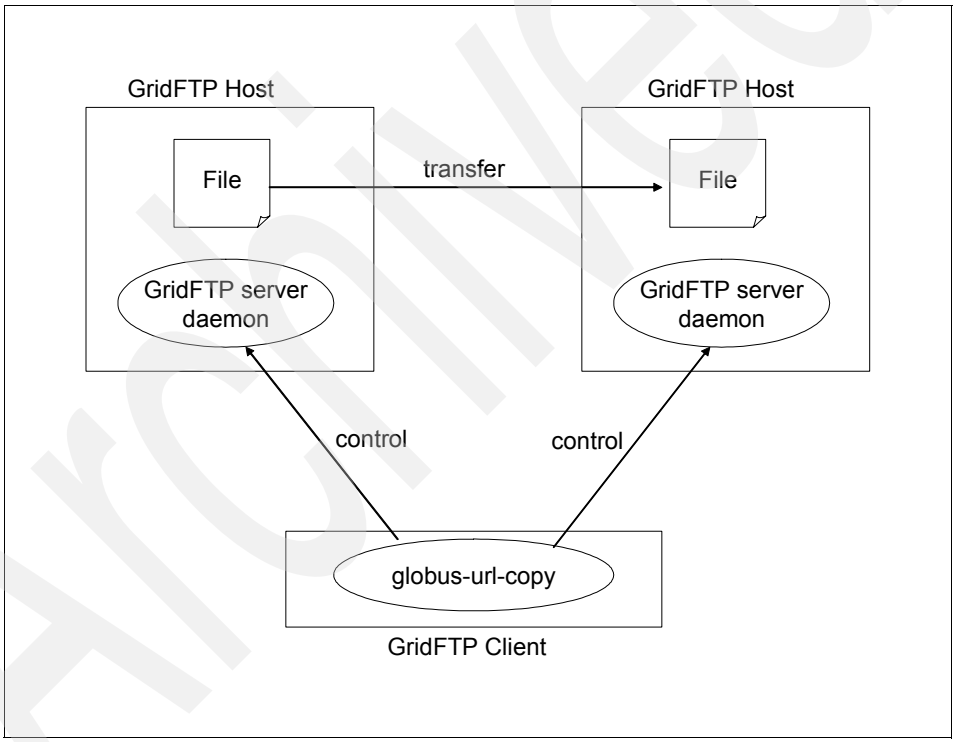

Figure 10-2 GridFTP third-party transfer

For more information about GridFTP, look at the following link:

http://www.globus.org/toolkit/docs/4.0/data/gridftp/

10.4.2 Reliable File Transfer (RFT)

Reliable File Transfer provides a Web service interface for transfer and deletion of files. RFT receives requests via SOAP messages over HTTP and utilizes GridFTP. RFT also uses a database to store the list of file transfers and their states, and is capable of recovering a transfer request that was interrupted. Figure 10-3 shows how RFT and GridFTP work.

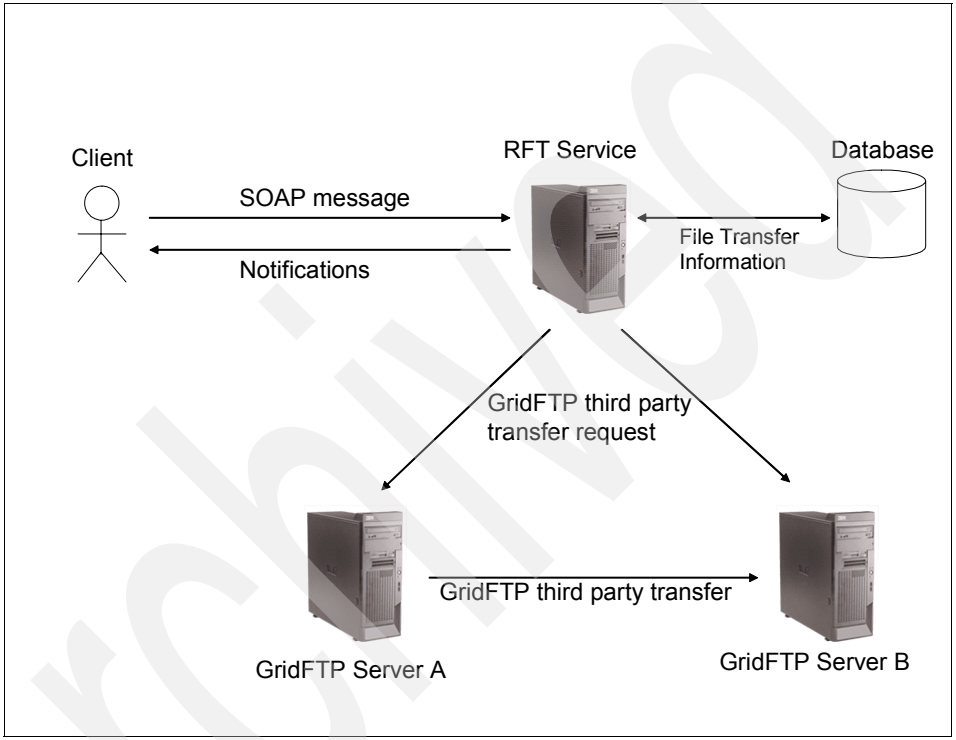

Figure 10-3 How RFT and GridFTP works

For more information about RFT, look at the following link:

http://www.globus.org/toolkit/docs/4.0/data/rft/

10.4.3 Replica Location Service (RLS)

The Replica Location Service maintains and provides access to information about the physical locations of replicated data. This component can map multiple physical replicas to one single logical file, and enables data redundancy in a grid environment. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/data/rls/

10.4.4 OGSA-DAI

OGSA-DAI enables a general grid interface for accessing grid data sources such as relational database management systems and XML repositories, through query languages like SQL, XPat, and XQuery. Currently, OGSA-DAI is a technical preview component. That is, the implementation is functional, but not necessarily complete, and its implementation and interfaces may change in the future. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/techpreview/ogsadai/

10.4.5 Data Replication Service (DRS)

Data Replication Service provides a system for making replicas of files in the grid environment, and registering them to RLS. DRS uses RFT and GridFTP to transfer the files, and it uses RLS to locate and register the replicas. Currently, DRS is a technical preview component. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/techpreview/datarep/

10.5 Monitoring and Discovery Services

The Monitoring and Discovery Services (MDS) are mainly concerned with the collection, distribution, indexing, archival, and otherwise processing information about the state of various resources, services, and system configurations. The information collected is used to either discover new services or resources, or to enable monitoring of system status.

The GT4 provides a WS-RF and WS-Notification compliant version of MDS, also known as MDS4.

The resource properties provided by a WS-RF compliant resource can be registered with MDS4 services for information collection purposes. The GT4 WS-RF compliant services such as GRAM and RFT provide such properties. Upon GT4 container startup these services are registered with MDS4 services.

MDS4 consists of two higher-level services, an Index service and a Trigger service, which are based on the Aggregator Framework that is briefly described next.

10.5.1 Index service

The Index service is the central component of the GT4 MDS implementation. Every instance of a GT4 container has a default indexing service

(DefaultIndexService) exposed as a WSRF service. The Index service interacts with data sources via standard WS-RF resource property and subscription/notification interfaces (WS-ResourceProperties and WS-BaseNotification). A WSRF-based service can make information available as resource properties. An Index service can potentially collect information from many sources and publish it in only one place. Various WSRF registrations with the Index service are maintained as Service Group Entries by the Index service. The contents of the Index service can be queried via XPath queries.

As noted earlier, each GT4 container has a default index service instance registered with it. Therefore, a grid computing site with multiple nodes can potentially have multiple instances of index services available for use. Often virtual organizations configure an instance of Index service to keep track of all relevant resources, containers, and services within their domain.

The following are some of the key features of an Index service:

- Index services can be configured in hierarchies, but there is no single global index that provides information about every resource on the Grid.
- The presence of a resource in an Index service makes no guarantee about the availability of the resource for users of that Index.
- Information published with MDS is recent but not the absolute latest.
- Each registration into an Index service has a lifetime and requires periodic renewal of registrations to indicate the continued existence of a resource or a service.

For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/info/index/

10.5.2 Trigger service

The MDS Trigger service collects information and compares that data against a set of conditions defined in a configuration file. When a condition is met an action is executed. The condition is specified as an XPath expression; that, for example, may compare the value of a property to a threshold and send an alert e-mail to an administrator by executing a script. The name and location of the script can be configured with the MDS Trigger service. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/info/trigger/

10.5.3 Aggregator Framework

The MDS-Index service and the MDS-Trigger service are specializations of a general Aggregator Framework. The Aggregator Framework is a software framework for building software services that collect and aggregate data. These services are also known as aggregator services.

An aggregator service collects information from one of the three types of aggregator sources such as a query source that utilizes WS-ResourceProperty mechanisms to collect data, a subscription source that uses a WS-Notification subscription/notification mechanism to collect data, or an execution source that executes an administrator-provided application to collect information in XML format.

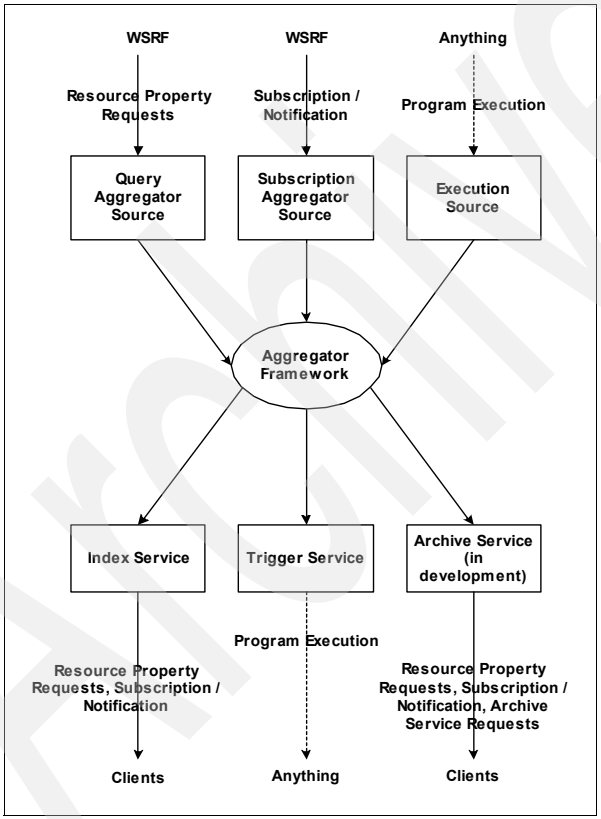

Figure 10-4 MDS4 Aggregator Framework

An aggregator source retrieves information from an external component called an information provider. In the case of a query and subscription source, the information provider is a WSRF-compliant service. For an execution source, the

information provider is an executable program that obtains data via some application-specific mechanism.

For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/info/aggregator/

10.5.4 WebMDS

WebMDS is a Web-based interface to WS-RF resource property information that can be used as a user-friendly front-end to the Index service. WebMDS uses standard resource property requests to query resource property data and transforms data for a user-friendly display. Web site administrators can customize their own WebMDS deployments by using HTML form options and creating their own XSLT transformations. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/info/Webmds/

10.6 Execution management

Globus Toolkit 4 provides various tools that enable execution management in a grid environment.

10.6.1 WS GRAM

WS GRAM is the Grid service that provides the remote execution and status management of jobs. When a job is submitted by a client, the request is sent to the remote host as a SOAP message, and handled by WS GRAM service located in the remote host. The WS GRAM service is capable of submitting those requests to local job schedulers such as Platform LSF or Altair PBS. The WS GRAM service returns status information of the job using WSNotification.

The WS GRAM service can collaborate with the RFT service for staging files required by jobs. In order to enable staging with RFT, valid credentials should be delegated to the RFT service by the Delegation service. Figure 10-5 on page 153 shows how job staging works.

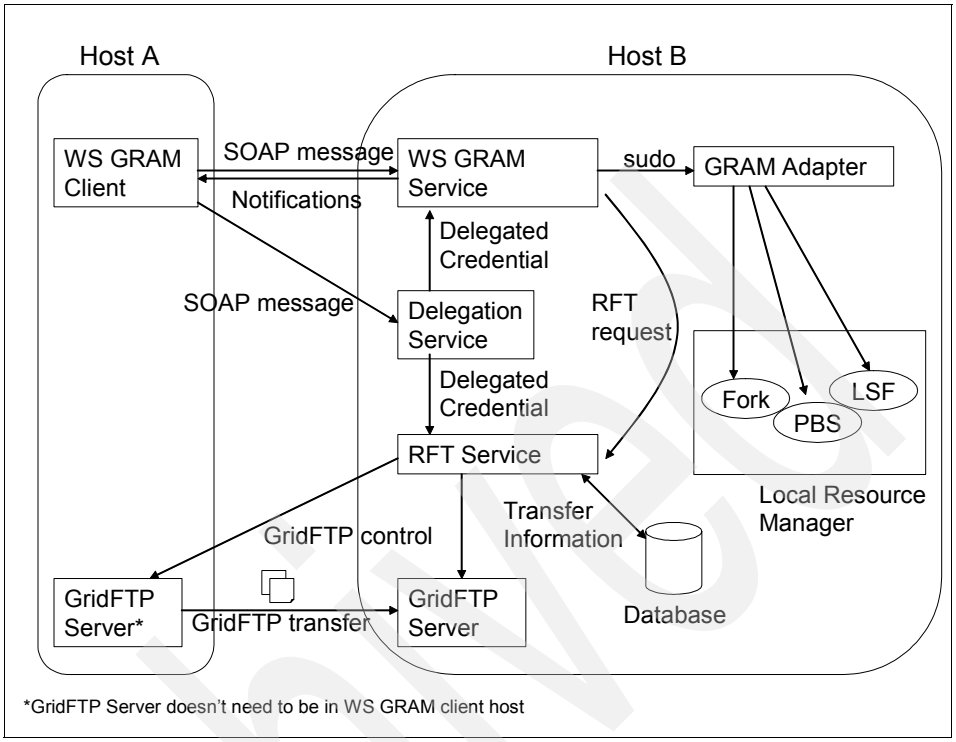

Figure 10-5 Execution of staging job

For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/execution/wsgram/

10.6.2 Community Scheduler Framework 4 (CSF4)

The Community Scheduler Framework 4 (CSF4) provides an intelligent, policy-based meta-scheduling facility for building grids where there are multiple types of job schedulers involved. It enables a single interface for different resource managers, such as Platform LSF and Altair PBS. Currently, CSF4 is a technical preview component. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/contributions/csf/

10.6.3 Globus Teleoperations Control Protocol (GTCP)

Globus Teleoperations Control Protocol is the WSRF version of NEESgrid Teleoperations Control Protocol (NTCP). Currently, GTCP is a technical preview component. For more information, look at the following links:

http://www.globus.org/toolkit/docs/4.0/techpreview/gtcp/
http://it.nees.org/

10.6.4 Workspace Management Service (WMS)

The Workspace Management Service enables a grid client to dynamically create, manage, and delete user accounts in a remote site. Currently, WMS is a technical preview component, and only supports management of UNIX accounts. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/techpreview/wms/

10.7 Summary

This chapter provided a brief overview of some of the components of the Globus Toolkit Version 4. Please refer to the Globus Web site for more information about the toolkit and the details of its components.

In the next chapter, we describe how to install and configure a simple Globus environment suitable for testing various components or creating a demonstration of some of the grid technologies and capabilities.

11

Globus Toolkit 4 installation and configuration

This chapter presents the necessary steps to install and configure Globus Toolkit 4 in a simple environment. You can follow these steps to set up a demo environment suitable for your own testing and to gain experience with some of the components of the Globus Toolkit.

The following topics are discussed:

- How to obtain Globus Toolkit 4
- Packages of Globus Toolkit 4
- Grid environment
- Installation
- Configuration and testing of grid environment
- Uninstallation

11.1 How to obtain Globus Toolkit 4

Globus Toolkit 4 is supported on a variety of operating systems. Binary packages are available for Linux environments (SuSE Linux 9/8, Red Hat Linux 9, Fedora Core Linux 2/3, and Debian 3.1), and Solaris 9. By compiling the source packages, Globus Toolkit 4 can be used on other operating systems such as AIX® and Mac OS X. The Java-based components including the WSRF-compliant WS Java container run in most Java-supported operating systems including Windows.

For the purpose of this book, we use both the binary and source packages of Globus Toolkit 4.0 running on Red Hat Linux 9.

Note: Though our test environment was built using Globus Toolkit 4.0, by the time this book is published, 4.0.1 or later may be available. If using a later release than 4.0.0, some of the information documented here may be slightly out of date.

This version of Globus Toolkit may be obtained at the official Globus Project site:

http://www.globus.org/toolkit/downloads/4.0.0/

Important: Globus Toolkit is distributed under the *Globus Toolkit Public License (GTPL) Version 3*, a liberal open source license. You are allowed to use every tool and the source code in the Globus Toolkit as you like with no restriction. But it is *as is with no warranty*. You can find out more about the GTPL at:

http://www.globus.org/toolkit/legal/4.0/license-v3.html

For bug tracking, the Globus Project provides the following Web site:

http://bugzilla.globus.org/bugzilla/

For platform-specific system requirements for Globus Toolkit 4, please refer to the following Web site:

http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch03.html#s-platform

11.2 Packages of Globus Toolkit 4

Globus Toolkit 4 is available in three ways.

- Download the full binary package from the Globus site.
- Download the full source package from the Globus site.

- Get source codes from a CVS server.

Depending on your environment, you can choose from these ways.

11.2.1 Binary packages

Table 11-1 shows the list of Globus Toolkit 4 binary packages that are available. If you install Globus Toolkit 4 into one of the operating systems described in Table 11-1, you can use binary packages. Otherwise, if you are using operating systems not listed in Table 11-1, you need to obtain the source packages and build binaries.

Table 11-1 List of Globus Toolkit 4 binary packages

Binary packages name	Operating system	Version	Platform
gt4.0.0-ia32-redhat9-binary-installer.tar.gz	Red Hat Linux	9	ia32
gt4.0.0-ia32-fedora2-binary-installer.tar.gz	Fedora Core Linux	2	ia32
gt4.0.0-ia32-fedora3-binary-installer.tar.gz	Fedora Core Linux	3	ia32
gt4.0.0-ia32-debian-binary-installer.tar.gz	Debian Linux	3.1	ia32
gt4.0.0-sun4u-solaris9-binary-installer.tar.gz	Solaris	9	sun4u
gt4.0.0-x86_64-sles9-binary-installer.tar.gz	SuSE Linux	9	x86_64 (Opteron)
gt4.0.0-ia64-sles8-binary-installer.tar.gz	SuSE Linux	8	ia64 (Itanium)

Java WS Core components are also available. This package only includes WSRF-compliant WS Java container and base components. All packages in Table 11-1 include Java WS core components, so you do not need to install both packages. Table 11-2 shows a list of Java WS Core installation packages.

Table 11-2 List of Java WS Core packages

Binary packages	Operating system	Version	Platform
ws-core-4.0.0-bin.zip ws-core-4.0.0-bin.tar.gz	All java vm-enabled operating systems	-	-

You can obtain those packages from the following page:

http://www.globus.org/toolkit/downloads/4.0.0/

11.2.2 Source packages

Table 11-3 shows the list of Globus Toolkit 4 source packages that are available.

Table 11-3 List of Globus Toolkit 4 source packages

Source packages name	Description
gt4.0.0-all-source-installer.tar.bz2 gt4.0.0-all-source-installer.tar.gz	Source packages with all components
ws-core-4.0.0-src.zip ws-core-4.0.0-src.tar.gz	Source packages with only Java WS core components

You can obtain those packages from the following page:

http://www.globus.org/toolkit/downloads/4.0.0/

You can also obtain individual packages from a CVS repository. Table 11-4 shows the list of major packages available in CVS.

Table 11-4 Major packages available in CVS repository

Source package name	Description
wsrf	WS Core packages
ws-transfer	RFT packages
ws-mds	WS MDS packages
ws-gram	WS GRAM packages

In order to obtain the packages from CVS, type the following command:

```
cvs -d :pserver:anonymous@cvs.globus.org:/home/globdev/CVS/globus-packages \
checkout (package-name)
```

11.3 Grid environment

Figure 11-1 on page 159 introduces a conceptual grid environment after a Globus Toolkit installation. In this chapter we take you through the steps required to install and configure this environment. There are three servers:

- CA

This is a Certificate Authority host. We use SimpleCA, which is included in the Globus Toolkit 4 package, as a Certificate Authority.

- Host A, host B

  These are the grid nodes. We install Globus Toolkit 4 packages to those hosts.

The user's names are different on host A (auser1) and host B (buser1), but they share the same grid user ID, which is known as the Distinguished Name:

```
/O=Grid/O=Globus/OU=redbook.ibm.com/CN=grid user 1
```

Note: In a grid environment, users use X.509 certificates to distinguish themselves from other users. So each grid user has one X.509 certificate, and the subject of the X.509 certificate is defined as the Distinguished Name.

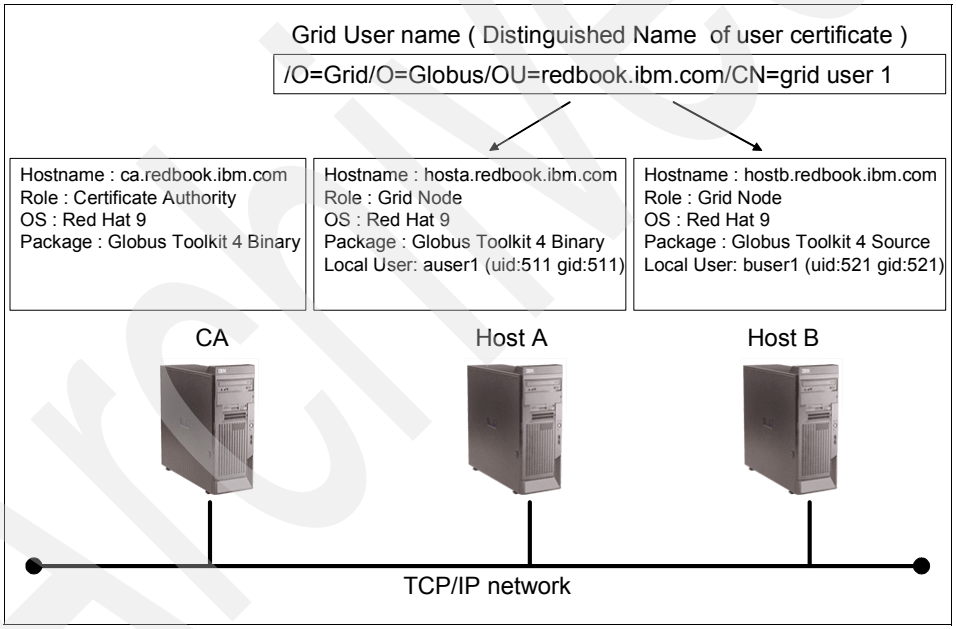

Figure 11-1 System overview after installation

We install software with the versions shown in Table 11-5 on page 160, and this book uses these versions during the installation process. Also, the installation directory of each software component is listed in Table 11-5 on page 160. If you want to install software with different versions or directories, make sure you specify your own version and directory each time you submit a command.

Table 11-5 Version and directory of each Globus software component

Name of software	Version	Directory
Globus Toolkit 4	4.0.0	/usr/local/globus-4.0.0
IBM Java SDK	1.4.2	/opt/IBMJava2-142
Apache Ant	1.6.3	/usr/local/apache-ant-1.6.3

At first, we explain how to install the servers using both binary and source packages. Then we show you how to configure CA, host A, and host B.

11.4 Installation

In order to install Globus Toolkit 4, we need to configure some tools that are essential for the Globus Toolkit 4 installation. After installation of those tools, we will install Globus Toolkit 4 using those tools.

11.4.1 Installing required software for Globus Toolkit 4 installation

Table 11-6 shows a list of software we need for Globus Toolkit 4 installation.

Table 11-6 List of required software for Globus Toolkit 4 installation

Software name	Recommended version
Java SDK (IBM / Sun / BEA)	1.4.2 or later
Apache Ant	1.5.1 or later
gcc	3.2.1 and 2.95.x are tested (avoid 3.2)
GNU tar	-
GNU sed	-
zlib	1.1.4 or later
GNU Make	-
sudo	-
PostgreSQL (or other JDBC compliant database)	7.1 or later (if using PostgreSQL)

Most of the packages in Table 11-6 are installed after Red Hat Linux 9 installation. Therefore, we only show how to install the IBM Java SDK and

Apache Ant, which are *not* installed during the Red Hat Linux 9 installation. Installation and configuration of PostgreSQL are described in “Configuration and testing of RFT” on page 180.

IBM Java SDK installation

To install IBM Java SDK:

1. Obtain IBM Java SDK from the following URL:

   http://www.ibm.com/developerworks/java/jdk/linux140/

   > **Note:** You may alternatively use the Sun Java SDK. The installation procedure of Sun Java SDK is similar to IBM Java SDK. You may obtain Sun Java SDK from the following URL:
   >
   > http://java.sun.com/j2se/1.4.2/download.html

2. Install IBM Java SDK. Example 11-1 shows the installation procedure of IBM Java SDK.

Example 11-1 Installation of IBM Java SDK

```
[root@hosta]# rpm -ivh IBMJava2-142-ia32-SDK-1.4.2-2.0.i386.rpm
Preparing...                ########################################### [100%]
   1:IBMJava2-142-ia32-SDK  ########################################### [100%]
```

3. Add environmental variables for IBM Java SDK. Example 11-2 shows an example of the /etc/profile.

Example 11-2 Example of /etc/profile

```
...(unrelated information omitted)

export JAVA_HOME=/opt/IBMJava2-142
export PATH=$JAVA_HOME/bin:$PATH
```

> **Note:** We set up environmental variables in /etc/profile in order to make those variables available to all users on the same host. You can put those variables into *(userhome)*/.bash_profile if you do not want to share those variables between users.

4. Log out and log in. (Instead, you may type `source /etc/profile` to ensure the variables are set and available.)
5. To test the IBM Java SDK installation, type `java -version`. If you see the version, then IBM Java SDK is properly installed (see Example 11-3 on page 162).

Example 11-3 Test IBM Java SDK installation

```
[root@hosta]# java -version
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
Classic VM (build 1.4.2, J2RE 1.4.2 IBM build cxia32142-20050609 (JIT enabled:
jitc))
```

Apache Ant installation

To install Apache Ant:

1. Obtain Apache Ant from the following URL:

   http://ant.apache.org/

2. Extract the Apache Ant archive. Example 11-4 shows the extract procedure of Apache Ant.

Example 11-4 Extraction of Apache Ant

```
[root@hosta]# tar xvzf apache-ant-1.6.3-bin.tar.gz -C /usr/local
apache-ant-1.6.3/bin/ant
apache-ant-1.6.3/bin/antRun
...(unrelated information omitted)
```

3. Add environmental variables for Apache Ant. Example 11-5 shows the example of /etc/profile.

Example 11-5 Example of /etc/profile

```
...(unrelated information omitted)

export ANT_HOME=/usr/local/apache-ant-1.6.3
export PATH=$ANT_HOME/bin:$PATH
```

4. Log out and log in. (Instead, you may type `source /etc/profile` to set the variables and make them available.)

5. To test Apache Ant installation, type the **ant** command (see Example 11-6). This command will initially fail because a build.xml file is missing, but this output means Apache Ant is working.

Example 11-6 Test Apache Ant installation

```
[root@hosta]# ant
Buildfile: build.xml does not exist!
Build failed
```

11.4.2 Preparing the OS for Globus Toolkit 4 installation

Before you install Globus Toolkit 4, there are few things that need to be prepared.

Users in each host

Add the users in Table 11-7 for Globus Toolkit 4 installation and configuration.

In a Linux environment, users can be added with a command such as that shown in the following example, where the password is provided with the -p parameter.

Example 11-7 Adding a user with the adduser command

```
adduser buser1 -p buserpw
```

Table 11-7 Users for our Globus Toolkit 4 installation and configuration

Host name	User name
ca	globus
hosta	globus
	auser1
hostb	globus
	buser1

Time settings

You should make sure to synchronize the system time of all the machines in your environment. GSI certificates use timestamps and are very sensitive to the time. If the system time of your grid environment is not set correctly, errors might occur when you use GSI certificates. For this reason, it is strongly recommended that you set up a time server, such as NTP, in your grid environment, and set the time correctly on all of your systems. Use of a time server is especially important in a distributed environment where a single administrator cannot easily ensure the correct setting of system clocks.

In order to configure NTP, look at following procedures:

1. On the machine that is designated to be the time server (in our case, CA host), edit the /etc/ntp.conf file as a root user. Leave the two lines shown in Example 11-8 as the only uncommented ones, commenting out all of the other lines with a leading # character.

Example 11-8 /etc/ntp.conf of ntp server

```
server 127.127.1.0 # local clock
```

```
driftfile /etc/ntp/drift
```

2. On the machine that is designated to be the ntp client (in our case, host A and host B), edit the /etc/ntp.conf file as the root user. Leave the two lines shown in Example 11-9 as the only uncommented ones, commenting out all of the other lines with a leading # character.

Example 11-9 /etc/ntp.conf of ntp client

```
server (ip address of ntp server) # time server
driftfile /etc/ntp/drift
```

3. In all hosts as the root user, configure the ntp daemon to run on the next boot (see Example 11-10).

Example 11-10 Configure ntp server to run on the next boot

```
[root@hosta]# chkconfig ntpd -on
```

4. In all hosts as the root user, start the ntp service (see Example 11-11).

Example 11-11 Starting ntp server

```
[root@hosta]# service ntpd start
```

5. Check if the time is synchronized with the ntp server by using the **ntpq** command. If you get an asterisk (*) before the time server name, then your ntp service is properly configured (see Example 11-12).

Example 11-12 Check the time setting with the ntpq command

```
[root@hosta]# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*ca.redbook.ib LOCAL(0)         6 u  516 1024  377    0.931   -2.258   0.262
```

Note: You may have to wait a few minutes before the ntp service synchronizes the time between systems.

Firewall settings

If you have a firewall in your environment, you should open the TCP ports listed in Table 11-8 on page 165 in order to use the services/components of Globus Toolkit 4. Take a look at your firewall settings and make sure those ports are open.

Table 11-8 TCP port numbers used by Globus Toolkit 4

TCP port number	Application
2811	GridFTP
8080	Globus container (non-secure mode)
8443	Globus container (secure mode)

11.4.3 Installing Globus Toolkit 4

You may install Globus Toolkit 4 in many ways.

In this section, we introduce both binary and source package installation. Installation of the binary package is extremely fast, while installation using the source package will take longer, as would be expected. As a guideline, we have provided the approximate time it took for us to install the packages in our environment in Table 11-9. Your experience may vary.

Table 11-9 Approximate time for installation (with Intel® Pentium® 4 3GHz, memory)

Package type	Approximate time
Binary	1 min.
Source	83 min.

Installation from binary package

To install from a binary package:

1. Obtain the Globus Toolkit 4 binary package from the Globus site. For more information, see 11.2.1, “Binary packages” on page 157.
2. Extract the binary package as the Globus user (see Example 11-13).

Example 11-13 Extracting binary package

```
[globus@hosta]$ tar xvzf gt4.0.0-ia32-redhat9-binary-installer.tar.gz \
-C /tmp
```

3. Set environmental variables for the Globus location. Example 11-14 shows how to set up the environmental variables.

Example 11-14 Set up the environmental variables for Globus

```
[globus@hosta]$ export GLOBUS_LOCATION=/usr/local/globus-4.0.0
```

4. Create and change the ownership of directory for user and group globus. (See Example 11-15 on page 166.)

Example 11-15 Create and change the ownership of directory

```
[globus@hosta]$ su
Password:
[root@hosta]# mkdir $GLOBUS_LOCATION
[root@hosta]# chown globus:globus $GLOBUS_LOCATION
[root@hosta]# exit
exit
[globus@hosta]$
```

5. Configure and install Globus Toolkit 4 (see Example 11-16).

Example 11-16 Configure and install Globus Toolkit 4

```
[globus@hosta]$ cd /tmp/gt4.0.0-ia32-redhat9-binary-installer
[globus@hosta]$ ./configure --prefix=$GLOBUS_LOCATION
checking for javac... /usr/java/j2sdk1.4.2_08/bin/javac
checking for ant... /usr/local/apache-ant-1.6.3/bin/ant
configure: creating ./config.status
config.status: creating Makefile

[globus@hosta]$ make 2>&1 | tee build.log
cd gpt-3.2autotools2004 && OBJECT_MODE=32 ./build_gpt
build_gpt ====> installing GPT into /usr/local/globus-4.0.0
...(unrelated information omitted)

[globus@hosta]$ make install
ln -s /usr/local/globus-4.0.0/etc/gpt/packages
/usr/local/globus-4.0.0/etc/globus_packages
/usr/local/globus-4.0.0/sbin/gpt-postinstall
...(unrelated information omitted)
config.status: creating fork.pm
..Done
```

Installation from source package

To install from a source package:

1. Obtain the Globus Toolkit 4 source package from the Globus site. For more information, see 11.2.2, “Source packages” on page 158.
2. Extract the source package with the Globus user ID (see Example 11-17).

Example 11-17 Extracting source package

```
[globus@hostb]$ tar xvzf gt4.0.0-all-source-installer.tar.gz -C /tmp
```

3. Set environmental variables for the Globus location. Example 11-18 on page 167 shows how to set up the environmental variables.

Example 11-18 Set up GLOBUS_LOCATION environmental variables for Globus

```
[globus@hostb]$ export GLOBUS_LOCATION=/usr/local/globus-4.0.0
```

4. Create and change the ownership of the directory for user and group Globus (see Example 11-19).

Example 11-19 Create and change the ownership of directory

```
[globus@hostb]$ su
Password:
[root@hostb]# mkdir $GLOBUS_LOCATION
[root@hostb]# chown globus:globus $GLOBUS_LOCATION
[root@hostb]# exit
exit
```

5. Configure and install Globus Toolkit 4 (see Example 11-20).

Example 11-20 Configure and install Globus Toolkit 4

```
[globus@hostb]$ cd /tmp/gt4.0.0-all-source-installer
[globus@hostb]$ ./configure --prefix=$GLOBUS_LOCATION
checking build system type... i686-pc-linux-gnu
checking for javac... /usr/java/j2sdk1.4.2_08/bin/javac
checking for ant... /usr/local/apache-ant-1.6.3/bin/ant
configure: creating ./config.status
config.status: creating Makefile

[globus@hostb]$ make 2>&1 | tee build.log
cd gpt-3.2autotools2004 && OBJECT_MODE=32 ./build_gpt
build_gpt ====> installing GPT into /usr/local/globus-4.0.0
...(unrelated information omitted)

[globus@hostb]$ make install
/usr/local/globus-4.0.0/sbin/gpt-postinstall
running /usr/local/globus-4.0.0/setup/globus/setup-globus-common..[ Changing to
/usr/local/globus-4.0.0/setup/globus ]
...(unrelated information omitted)
config.status: creating fork.pm
..Done
```

11.5 Configuration and testing of grid environment

After the installation of the Globus Toolkit, each element of your grid environment must be configured.

11.5.1 Configuring environmental variables

Before starting the configuration process, it is useful to set up the GLOBUS_LOCATION environmental variables in either /etc/profile or *(userhome)*/.bash_profile. To save time upon subsequent logins from different user IDs, we specified GLOBUS_LOCATION in /etc/profile (see Example 11-21).

Also, Globus Toolkit provides shell scripts to set up these environmental variables. They can be sourced as follows:

```
source $GLOBUS_LOCATION/etc/globus-user-env.sh (sh)
source $GLOBUS_LOCATION/etc/globus-user-env.csh (csh)
```

The Globus Toolkit also provides shell scripts for developers to set up Java CLASSPATH environmental variables. They can be sourced as follows:

```
source $GLOBUS_LOCATION/etc/globus-devel-env.sh (sh)
source $GLOBUS_LOCATION/etc/globus-devel-env.csh (csh)
```

In this book, to save time upon subsequent logins, we specify globus-user-env.sh and globus-devel-env.sh in /etc/profile so that all users can use the grid environment.

Example 11-21 Example of /etc/profile

```
...(unrelated information omitted)

export GLOBUS_LOCATION=/usr/local/globus-4.0.0
source $GLOBUS_LOCATION/etc/globus-user-env.sh
source $GLOBUS_LOCATION/etc/globus-devel-env.sh
```

11.5.2 Security set up

In this book, we use SimpleCA, which is a wrapper of OpenSSL CA functionality.

Important: Before setting up a certificate authority (CA), make sure to synchronize the system time of all the machines in your environment. For more information, refer to 11.4.2, “Preparing the OS for Globus Toolkit 4 installation” on page 163.

Note: Make sure Globus Toolkit 4 is also installed on the CA host, as the SimpleCA package is provided in the Globus Toolkit. See “Installing Globus Toolkit 4” on page 165 for installation procedures.

Installation of CA packages

To install CA packages:

1. Log in to the CA host as a Globus user.
2. Invoke the setup-simple-ca script, and answer the prompts as appropriate See Example 11-22. This script initializes the files that are necessary for SimpleCA.

Example 11-22 Setting up SimpleCA

```
[globus@ca]$ $GLOBUS_LOCATION/setup/globus/setup-simple-ca

WARNING: GPT_LOCATION not set, assuming:
         GPT_LOCATION=/usr/local/globus-4.0.0



    C e r t i f i c a t e   A u t h o r i t y   S e t u p

This script will setup a Certificate Authority for signing Globus users
certificates.  It will also generate a simple CA package that can be
distributed to the users of the CA.

The CA information about the certificates it distributes will be kept in:

/home/globus/.globus/simpleCA/
/usr/local/globus-4.0.0/setup/globus/setup-simple-ca: line 250: test: res:
integer expression expected

The unique subject name for this CA is:

cn=Globus Simple CA, ou=simpleCA-ca.redbook.ibm.com, ou=GlobusTest, o=Grid

Do you want to keep this as the CA subject (y/n) [y]: y

Enter the email of the CA (this is the email where certificate requests will be
sent to be signed by the CA): (type mail address)globus@ca.redbook.ibm.com

The CA certificate has an expiration date. Keep in mind that once the CA
certificate has expired, all the certificates signed by that CA become invalid.
A CA should regenerate the CA certificate and start re-issuing ca-setup
packages before the actual CA certificate expires.  This can be done by
re-running this setup script.  Enter the number of DAYS the CA certificate
should last before it expires.
[default: 5 years (1825 days)]: (type the number of days)1825

Enter PEM pass phrase: (type ca certificate pass phrase)
Verifying - Enter PEM pass phrase: (type ca certificate pass phrase)
...(unrelated information omitted)
```

```
setup-ssl-utils: Complete
```

Setting up security in each grid node

After performing the steps above, a package file has been created that needs to be used on other nodes, as described in this section. In order to use certificates from this CA in other grid nodes, you need to copy and install the CA setup package to each grid node.

1. Log in to a grid node as a Globus user and obtain a CA setup package from the CA host. Then run the setup commands for configuration (see Example 11-23).

Example 11-23 Set up CA in each grid node

```
[globus@hosta]$ scp globus@ca:/home/globus/.globus/simpleCA \
/globus_simple_ca_(ca_hash)_setup-0.18.tar.gz .
[globus@hosta]$ $GLOBUS_LOCATION/sbin/gpt-build \
globus_simple_ca_(ca_hash)_setup-0.18.tar.gz gcc32dbg
[globus@hosta]$ $GLOBUS_LOCATION/sbin/gpt-postinstall
```

> **Note:** A CA setup package is generated when you run the **`setup-simple-ca`** command in Example 11-22. Keep in mind that the name of the CA setup package includes a unique CA hash.

2. As the root user, submit the commands in Example 11-24 to configure the CA settings in each grid node. This script creates the /etc/grid-security directory. This directory contains the configuration files for security.

Example 11-24 Configure CA in each grid node

```
[root@hosta]# $GLOBUS_LOCATION/setup\
/globus_simple_ca_[ca_hash]_setup/setup-gsi -default
```

> **Note:** For the setup of the CA host, you do not need to run the **`setup-gsi`** script. This script creates a directory that contains the configuration files for security. The CA host does not need this directory, because these configuration files are for the servers and users who use the CA.

Obtain and sign a host certificate

In order to use some of the services provided by Globus Toolkit 4, such as Grid FTP, you need to have a CA signed host certificate and host key in the appropriate directory.

1. As root user, request a host certificate with the command in Example 11-25 on page 171.

Example 11-25 Request a host certificate

```
[root@hosta]# grid-cert-request -host `hostname`
```

2. Copy or send the /etc/grid-security/hostcert_request.pem file to the CA host.
3. In the CA host as a Globus user, sign the host certificate by using the **grid-ca-sign** command.

Example 11-26 Sign a host certificate

```
[globus@ca]$ grid-ca-sign -in hostcert_request.pem -out hostcert.pem

To sign the request
please enter the password for the CA key: (type ca passphrase)

The new signed certificate is at:
/home/globus/.globus/simpleCA//newcerts/01.pem
```

4. Copy the hostcert.pem back to the /etc/grid-security/ directory in the grid node.

Obtain and sign a user certificate

In order to use the grid environment, a grid user needs to have a CA signed user certificate and user key in the user's directory.

1. As a user (auser1 in hosta), request a user certificate with the command in Example 11-27.

Example 11-27 Request a user certificate

```
[auser1@hosta]$ grid-cert-request
Enter your name, e.g., John Smith: grid user 1 (type grid user name)
A certificate request and private key is being created.
You will be asked to enter a PEM pass phrase.
This pass phrase is akin to your account password,and is used to protect your
key file.
If you forget your pass phrase, you will need to obtain a new certificate.

Generating a 1024 bit RSA private key
.....................................++++++
...++++++
writing new private key to '/home/auser1/.globus/userkey.pem'
Enter PEM pass phrase: (type pass phrase for grid user)
Verifying - Enter PEM pass phrase: (retype pass phrase for grid user)
...(unrelated information omitted)
```

2. Copy or send the *(userhome)*/.globus/usercert_request.pem file to the CA host.

3. In CA host as a Globus user, sign the user certificate by using the **grid-ca-sign** command (see Example 11-28).

Example 11-28 Sign a user certificate

```
[globus@ca]$ grid-ca-sign -in usercert_request.pem -out usercert.pem

To sign the request
please enter the password for the CA key:

The new signed certificate is at:
/home/globus/.globus/simpleCA//newcerts/02.pem
```

4. Copy the created usercert.pem to the *(userhome)*/.globus/ directory on the grid node.
5. Test the user certificate by typing `grid-proxy-init -debug -verify` as the auser user. With this command, you can see the location of a user certificate and a key, CA's certificate directory, a distinguished name for the user, and the expiration time. After you successfully execute **grid-proxy-init**, you have been authenticated and are ready to use the grid environment.

Example 11-29 Testing user certificate installation

```
[auser1@hosta]$ grid-proxy-init -debug -verify

User Cert File: /home/auser1/.globus/usercert.pem
User Key File: /home/auser1/.globus/userkey.pem

Trusted CA Cert Dir: /etc/grid-security/certificates

Output File: /tmp/x509up_u511
Your identity:
/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/OU=redbook.ibm.com/CN=grid
user 1
Enter GRID pass phrase for this identity:
Creating proxy .........++++++++++++
.................++++++++++++
 Done
Proxy Verify OK
Your proxy is valid until: Thu Jun  9 22:16:28 200
```

Note: You may copy those user certificates to other grid nodes in order to access each grid node as a single grid user. But you may not copy a host certificate and a host key. A host certificate is needed to be created in each grid node.

Set mapping information between a grid user and a local user

Globus Toolkit 4 requires a mapping between an authenticated grid user and a local user. In order to map a user, you need to get the distinguished name of the grid user, and map it to a local user.

1. Get the distinguished name by invoking the **grid-cert-info** command.

Example 11-30 Obtaining distinguished name

```
[auser1@hosta]$ grid-cert-info -subject -f /home/auser1/.globus/usercert.pem
/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/OU=redbook.ibm.com/CN=grid
user 1
```

2. As a root user, map the local user name with the distinguished name by using the **grid-mapfile-add-entry** command, as seen in Example 11-31.

Example 11-31 Map a grid user and local user

```
[root@hosta]# grid-mapfile-add-entry -dn \
"/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/OU=redbook.ibm.com/CN=gri
d user 1" -ln auser1

Modifying /etc/grid-security/grid-mapfile ...
/etc/grid-security/grid-mapfile does not exist... Attempting to create
/etc/grid-security/grid-mapfile
New entry:
"/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/OU=redbook.ibm.com/CN=gri
d user 1" auser1
(1) entry added
```

> **Note:** The **grid-mapfile-add-entry** command creates and adds an entry to /etc/grid-security/grid-mapfile. You can manually add an entry by adding a line into this file.

3. In order to see the mapping information, look at /etc/grid-security/grid-mapfile (see Example 11-32).

Example 11-32 Example of /etc/grid-security/grid-mapfile

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/OU=redbook.ibm.com/CN=gri
d user 1" auser1
```

4. To check for consistency of the mapfile, submit **grid-mapfile-check-consistency**. If you get no response from this command, then it means the grid-mapfile is consistent.

Example 11-33 Check the consistency of grid-mapfile

```
[root@hosta]# grid-mapfile-check-consistency
```

11.5.3 Configuration of Java WS Core

The Java WS Core container is installed as a part of the default Globus Toolkit 4 installation. There are a few things you need to configure before you start Java WS Core.

Setting up Java WS Core environment

The Java WS Core container uses a copy of the host certificate and a host key. You need to copy and change the owner of those files before you start the Java WS Core container.

As a root user, copy hostcert.pem and hostkey.pem to containercert.pem and containerkey.pem in /etc/grid-security/. Then change the owner of the new files to Globus (see Example 11-34).

Example 11-34 Copying host certificate and key to container certificate and key

```
[root@hosta]# cp hostcert.pem containercert.pem
[root@hosta]# cp hostkey.pem containerkey.pem
[root@hosta]# chown globus.globus containercert.pem containerkey.pem
```

Verifying the installation and configuration of Java WS Core

To verify that the Java WS Core has been installed successfully and that grid security has been implemented correctly, complete the following procedure:

1. As a Globus user, run the following command to start the container:

   ```
   globus-start-container
   ```

   If you do not use a secured container, then type following command:

   ```
   globus-start-container -nosec
   ```

2. When the process is complete, a message indicates that the container is open for Grid services, as shown in Example 11-35.

Example 11-35 Starting the Java WS Core container

```
[globus@hosta]$ globus-start-container -nosec
2005-06-09 11:31:41,192 ERROR service.ReliableFileTransferImpl [main,<init>:73]
Unable to setup data base driver with pooling.Connection refused. Check that
the hostname and port are correct and that the postmaster is accepting TCP/IP
connections.
2005-06-09 11:31:41,848 WARN
service.ReliableFileTransferHome[main,initialize:97] All RFT requests will fail
```

```
and all GRAM jobs that require file staging will fail.Connection refused. Check
that the hostname and port are correct and that the postmaster is accepting
TCP/IP connections.
Starting SOAP server at: http://192.168.1.103:8080/wsrf/services/With the
following services:

[1]: http://192.168.1.103:8080/wsrf/services/TriggerFactoryService
[2]: http://192.168.1.103:8080/wsrf/services/DelegationTestService
...(unrelated information omitted)
[51]: http://192.168.1.103:8080/wsrf/services/ManagedJobFactoryService
2005-06-09 11:32:10,359 INFO  impl.DefaultIndexService
[Thread-9,processConfigFile:99] Reading default registration configuration from
file: /usr/local/globus-4.0.0/etc/globus_wsrf_mds_index/hierarchy.xml
2005-06-09 11:32:11,398 ERROR impl.QueryAggregatorSource
[Thread-11,pollGetMultiple:149] Exception Getting Multiple Resource Properties
from
http://192.168.1.103:8080/wsrf/services/ReliableFileTransferFactoryService:
java.rmi.RemoteException: Failed to serialize resource property
org.globus.transfer.reliable.service.factory.TotalNumberOfBytesTransferred@1fd1
0fa; nested exception is: org.apache.commons.dbcp.DbcpException: Connection
refused. Check that the hostname and port are correct and that the postmaster
is accepting TCP/IP connections.
```

Note: **`globus-start-container`** may take some time to complete.

Note: With the **`globus-start-container`** command, you will see many exceptions regarding RFT. This is because we have not configured RFT yet, and therefore these messages are normal. If you do not want these messages, go to “Configuration and testing of RFT” on page 180 and configure RFT first.

Executing Counter Sample program

Globus Toolkit 4 includes sample programs. Counter Sample is one of the samples in Globus Toolkit 4. Counter Sample contains a CounterService and counter client. CounterService has two key operations:

createCounter Create a new counter resource and return the end point reference of the resource.

add Add a value to the specified counter resource.

CounterService is deployed into the container during the installation process by default, so you only need to use the client program to try Counter Sample. To try the sample, follow these procedures:

1. If your Java WS Core container is not running, start your container by typing the following command:

   ```
   globus-start-container [-nosec]
   ```

   If you do not want to run the container in secure mode, then use the -nosec option.

   Make sure the CounterService entry is shown when you start your container (see Example 11-36).

Example 11-36 Part of globus-start-container output

```
...(unrelated information omitted)
[15]: https://192.168.1.103:8443/wsrf/services/CounterService
...(unrelated information omitted)
```

2. Log in to your grid node with a user that has grid user certificates.
3. Type the **grid-proxy-init command** to authenticate and create the proxy certificate (see Example 11-37).

   Note: If you are using non-secure container, you do not need this step.

Example 11-37 Submitting grid-proxy-init command

```
[auser1@hosta]$ grid-proxy-init
Your identity:
/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/OU=redbook.ibm.com/CN=grid
user 1
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Your proxy is valid until: Tue Jun 14 21:41:25 2005
```

4. Create a counter resource by typing following command:

   ```
   counter-create -s <URI of CounterService> > <epr file name>
   ```

   Output of this command includes an end point reference string, so you need to redirect the output to file. See Example 11-38.

Example 11-38 Create counter resource

```
[auser1@hosta]$ counter-create -s \
https://192.168.1.103:8443/wsrf/services/CounterService > test.epr
```

5. Add a value to the counter resource by typing the following command:

```
counter-add -e <epr file name> <value to add>
```

Output of this command shows the result after addition. You may try several times to see how it works (see Example 11-39).

Example 11-39 Add values to counter resource

```
[auser1@hosta]$ counter-add -e test.epr 3
3
[auser1@hosta]$ counter-add -e test.epr 4
7
```

Troubleshooting

The following are a few common errors that may occur and what you might do to correct them.

- The following message appears during the **globus-start-container** command.

  ```
  Failed to start container: Failed to initialize 'ManagedJobFactoryService'
  service [Caused by: [SEC] Service credentials not configured and was not
  able to obtain container credentials.;
  ```

  This may be due to not having properly created container certificates. Also, this error appears when you do not have a grid-mapfile. Make sure you follow the steps in 11.5.2, “Security set up” on page 168.

- The following message appears during the **globus-start-container** command.

  ```
  Failed to start container: Container failed to initialize [Caused by:
  Address already in use]
  ```

  This is because you have another container or program running. You may need to stop the container or program in order to make this command work.

- The following message appears during the **counter-create** command.

  ```
  Error: ; nested exception is:
          GSSException: Defective credential detected [Caused by: Proxy file
  (/tmp/x509up_u511) not found.]
  ```

  This is because you have tried to access a secured container without an activated proxy certificate. You need to run the **grid-proxy-init** command in order to make this command work.

11.5.4 Configuration and testing of GridFTP

You need to configure GridFTP before RFT, because GridFTP is required by RFT. GridFTP is already installed during the default installation process. You

only need to configure GridFTP as a service daemon so that you can transfer data between two hosts with GridFTP.

Setting up GridFTP environment

In order to install GridFTP, follow the procedures below.

1. Assign the service name gsiftp to TCP port 2811 in /etc/services as you see in Example 11-40.

Example 11-40 Example of /etc/services file

```
...(unrelated information omitted)
gsiftp          2811/tcp                        # GridFTP
```

2. Create the /etc/xinetd.d/gsiftp file with the entry in Example 11-41.

Example 11-41 Example of /etc/xinetd.d/gsiftp

```
service gsiftp
{
instances               = 100
socket_type             = stream
wait                    = no
user                    = root
env                     += GLOBUS_LOCATION=/usr/local/globus-4.0.0
env                     += LD_LIBRARY_PATH=/usr/local/globus-4.0.0/lib
server                  = /usr/local/globus-4.0.0/sbin/globus-gridftp-server
server_args             = -i
log_on_success          += DURATION
nice                    = 10
disable                 = no
}
```

3. Restart xinetd daemon (see Example 11-42).

Example 11-42 Restarting xinetd daemon

```
[root@hosta]# service xinetd restart
Stopping xinetd:                                          [  OK  ]
Starting xinetd:                                          [  OK  ]
```

Note: You may also start your GridFTP server by the command below.

```
globus-gridftp-server -S
```

For more information, see the following link:

http://www.globus.org/toolkit/docs/4.0/data/gridftp/admin-index.html

Verifying the installation and configuration of GridFTP

To verify that GridFTP has been installed successfully, complete the following procedure:

1. Log in to your grid node with the user who has grid user certificates.
2. Type a **grid-proxy-init** command to authenticate and create the proxy certificate.
3. Type the following GridFTP client command to make sure your GridFTP is configured properly (see Example 11-43).

```
globus-url-copy <sourceURL> <destURL>
```

Example 11-43 Using GridFTP with globus-url-copy command

```
[auser1@hosta]$ echo "GridFTP Test" > /tmp/gridftptest.txt
[auser1@hosta]$ globus-url-copy gsiftp://hosta/tmp/gridftptest.txt \
file:///tmp/gridftptest_copied.txt
[auser1@hosta]$ cat /tmp/gridftptest_copied.txt
GridFTP Test
[auser1@hosta]$ globus-url-copy file:///tmp/gridftptest_copied.txt \
gsiftp://hosta/tmp/gridftptest_copied2.txt
[auser1@hosta]$ cat /tmp/gridftptest_copied2.txt
GridFTP Test
```

4. Try third-party transfer with the **globus-url-copy** command (see Example 11-44).

Example 11-44 Third-party transfer with globus-url-copy command

```
[auser1@hosta]$ echo "ThirdParty GridFTP Test" > /tmp/thirdparty.txt
[auser1@hosta]$ globus-url-copy gsiftp://hosta/tmp/thirdparty.txt \
gsiftp://hostb/tmp/thirdparty.txt
[auser1@hosta]$ ssh buser1@hostb
buser1@hostb's password:
Last login: Thu Jun  9 19:36:31 2005 from hosta.redbook.ibm.com
[buser1@hostb]$ cat /tmp/thirdparty.txt
ThirdParty GridFTP Test
[buser1@hostb]$ ll /tmp/thirdparty.txt
-rw-r--r--    1 buser1   buser1         24 Jun  9 19:36 /tmp/thirdparty.txt
```

Important: In Example 11-44, the owner of the created file is buser1 (not root). This is because GridFTP uses GSI for authentication, and /etc/grid-mapfile was used to map the grid user and local user. Take a look at /etc/grid-security/grid-mapfile. Refer to “Security set up” on page 168 and Chapter 7, “Security” on page 63, for more information.

Note: In order to enable third-party GridFTP transfer, you need to install and configure other hosts, such as hostb, with the same steps. Refer to previous sections for installation and configuration procedures.

Troubleshooting

The following are some possible error conditions or symptoms that may come up in your testing along with possible resolutions.

- It takes long time to transfer a small data file using globus-url-copy

  Make sure your name server is configured properly. Look at /etc/resolv.conf to make sure name resolution of your grid node is configured properly.

- The following message appears during the **globus-url-copy** command.

  ```
  globus_gsi_gssapi: Error with gss credential handle
  globus_credential: Valid credentials could not be found in any of the
  possible locations specified by the credential search order.
  Valid credentials could not be found in any of the possible locations
  specified by the credential search order.
  ```

  This is because you have tried to access the secured container without an activated proxy certificate. You need to run the **grid-proxy-init** command in order to make this command work.

11.5.5 Configuration and testing of RFT

After you configure GridFTP, you may configure RFT. RFT is used by WS GRAM during stage-in and stage-out.

Attention: Before configuring RFT, make sure you follow the instructions in “Configuration and testing of GridFTP” on page 177.

Setting up PostgreSQL

In order to use RFT, you need to configure a JDBC-compliant database. In this book, we install PostgreSQL as the database. Follow the procedures below to install and set up PostgreSQL.

1. If you do not have PostgreSQL in your node, install PostgreSQL with rpm commands as a root user (see Example 11-45).

Example 11-45 Installing postgres

```
[root@hosta]# rpm -ivh /mnt/cdrom/RedHat/RPMS/postgresql-libs-7.3.2-3.i386.rpm
[root@hosta]# rpm -ivh /mnt/cdrom/RedHat/RPMS/postgresql-7.3.2-3.i386.rpm
```

```
[root@hosta]# rpm -ivh \
/mnt/cdrom/RedHat/RPMS/postgresql-server-7.3.2-3.i386.rpm
```

2. Start the PostgreSQL server. Also, set postgresql to start after reboot (see Example 11-46).

Example 11-46 Starting PostgreSQL

```
[root@hosta]# service postgresql start
Initializing database:                                    [  OK  ]
Starting postgresql service:                              [  OK  ]
[root@hosta]# chkconfig postgresql on
```

3. RFT requires PostgreSQL to accept a connection from the network. Edit the PostgreSQL settings file (/var/lib/pgsql/data/postgresql.conf) to allow connection from the network (see Example 11-47).

Example 11-47 Example of /var/lib/pgsql/data/postgresql.conf

```
...(unrelated information omitted)
#       Connection Parameters
#
tcpip_socket = true (change the value from false to true )
...(unrelated information omitted)
```

4. Add the following line to /var/lib/pgsql/data/pg_hba.conf to allow access from your host (see Example 11-48).

Example 11-48 Example of /var/lib/pgsql/data/pg_hba.conf

```
...(unrelated information omitted)
local  all      all             ident   sameuser
host    all     all     (ip address of RFT host) 255.255.255.255         trust
```

5. Restart the PostgreSQL server to activate the new settings (see Example 11-49).

Example 11-49 Restarting PostgreSQL

```
[root@hosta]# service postgresql restart
Stopping postgresql service:                              [  OK  ]
Starting postgresql service:                              [  OK  ]
```

Adding a new database for RFT to PostgreSQL

You need to create a database that RFT uses in PostgreSQL. Follow the procedures below:

1. As a postgres user, submit the command in Example 11-50 on page 182 to create the RFT database.

Example 11-50 Creating RFT database

```
[postgres@hosta]$ createdb rftDatabase
CREATE DATABASE
```

> **Note:** The postgres user is automatically generated during the PostgreSQL package installation shown in Example 11-45 on page 180.

2. Create tables by using the sql scripts that are included in the Globus Toolkit 4 package (see Example 11-51).

Example 11-51 Creating tables in RFT database

```
[postgres@hosta]$ psql -d rftDatabase -f \
$GLOBUS_LOCCATION/share/globus_wsrf_rft/rft_schema.sql
CREATE SEQUENCE
CREATE SEQUENCE
psql:/usr/local/globus-4.0.0/share/globus_wsrf_rft/rft_schema.sql:22: NOTICE:
CREATE TABLE / PRIMARY KEY will create implicit index 'request_pkey' for table
'request'
CREATE TABLE
psql:/usr/local/globus-4.0.0/share/globus_wsrf_rft/rft_schema.sql:57: NOTICE:
CREATE TABLE / PRIMAR
Y KEY will create implicit index 'transfer_pkey' for table 'transfer'
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE INDEX
```

3. Register the Globus user to PostgreSQL server (see Example 11-52).

Example 11-52 Registering Globus user

```
[postgres@hosta]$ createuser globus
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) y
CREATE USER
```

Verifying the installation and configuration of RFT

To verify that the RFT has been installed successfully, complete the following procedure:

1. Log in to your grid node with the Globus user.
2. Start the secure container (see Example 11-53).

Example 11-53 Starting the Java WS Core container in secure mode

```
[globus@hosta]$ globus-start-container
```

```
Starting SOAP server at: https://192.168.1.103:8443/wsrf/services/
With the following services:

[1]: https://192.168.1.103:8443/wsrf/services/TriggerFactoryService
[2]: https://192.168.1.103:8443/wsrf/services/DelegationTestService
...(unrelated information omitted)
[51]: https://192.168.1.103:8443/wsrf/services/ManagedJobFactoryService
2005-06-09 21:46:49,805 INFO  impl.DefaultIndexService
[Thread-9,processConfigFile:99] Reading default registration configuration from
file: /usr/local/globus-4.0.0/etc/globus_wsrf_mds_index/hierarchy.xml
```

Attention: Make sure you do not have an RFT error after you start your container. If you still have an RFT error like in Example 11-35 on page 174, check the settings again. See “Troubleshooting” on page 184 for more information.

3. In another console, log in to your grid node with the user who has grid user certificates.
4. Create an RFT description file. A sample RFT description file is show in Example 11-54.

Example 11-54 Sample of RFT description file (rfttest.xfr)

```
#true=binary false=ascii
true
#Block size in bytes
16000
#TCP Buffer size in bytes
16000
#Notpt (No thirdPartyTransfer)
false
#Number of parallel streams
1
#Data Channel Authentication (DCAU)
true
# Concurrency of the request
1
#Grid Subject name of the source gridftp server
/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/CN=host/hosta.redbook.ibm.
com
#Grid Subject name of the destination gridftp server
/O=Grid/OU=GlobusTest/OU=simpleCA-ca.redbook.ibm.com/CN=host/hostb.redbook.ibm.
com
#Transfer all or none of the transfers
false
#Maximum number of retries
10
```

```
#Source/Dest URL Pairs
gsiftp://hosta.redbook.ibm.com/tmp/fileInHostA.txt
gsiftp://hostb.redbook.ibm.com/tmp/fileFromHostA.txt
```

Note: A template for rfttest.xfr in Example 11-54 is located in the file below:

```
$GLOBUS_LOCATION/share/globus_wsrf_rft_client/transfer.xfr
```

For more information, look at the following site:

http://www.globus.org/toolkit/docs/4.0/data/rft/rn01re01.html

5. Run the RFT job by invoking the **rft** command, as below.

Example 11-55 Executing RFT file transfer with rft command

```
[auser1@hosta]$ echo TestFromHostA > /tmp/fileInHostA.txt
[auser1@hosta]$ rft -h hosta.redbook.ibm.com -r 8443 -f rfttest.xfr
Number of transfers in this request: 1
Subscribed for overall status
Termination time to set: 60 minutes

 Overall status of transfer:

 Overall status of transfer:
Finished/Active/Failed/Retrying/Pending
Finished/Active/Failed/Retrying/Pending
1/0/0/0/0
0/1/0/0/0
All Transfers are completed

[auser1@hosta]$ ssh buser1@hostb
buser1@hostb's password:

[buser1@hostb]$ cat /tmp/fileFromHostA.txt
TestFromHostA
```

Troubleshooting

To troubleshoot:

- The following message appears during the **globus-start-container** command.

```
2005-06-09 21:41:12,135 ERROR service.ReliableFileTransferImpl
[main,<init>:73] Unable to setup database driver with pooling.A connection
error has occurred: FATAL:  No pg_hba.conf entry for host
(XXX.XXX.XXX.XXX), user globus, database rftDatabase
```

This message appears because PostgreSQL is not configured properly. Look at /var/lib/pgsql/data/pg_hba.conf and check if there is an entry for your host ip.

- The following message appears during the **globus-start-container** command.

```
2005-06-13 16:10:55,374 ERROR service.ReliableFileTransferImpl
[main,<init>:73] Unable to setup database driver with pooling.Connection
refused. Check that the hostname and port are correct and that the
postmaster is accepting TCP/IP connections.
```

  This message appears because the container can not connect to PostgreSQL. Check whether PostgreSQL is running. Check the configuration as described in “Setting up PostgreSQL” on page 180.

- The following message appears during **rft** command execution.

```
Exception in thread "main" Error during startup processing. Caused by
java.lang.NumberFormatException: For input string:
```

  Or:

```
Number of transfers in this request: 0
Exception in thread "main" Error during startup processing. Caused by
AxisFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
 faultSubcode:
 faultString: java.lang.NullPointerException
```

  Or:

```
Exception in thread "main" Error during startup processing. Caused by
java.lang.ArrayIndexOutOfBoundsException: 10 >= 10
```

  These messages appear because the RFT description file (rfttest.xfr) is inconsistent. Check the RFT description file.

11.5.6 Configuration and testing of WS GRAM

Most of the settings for WS GRAM are completed automatically during the Globus Toolkit 4 installation process. You do need to set the environment for the **sudo** command.

Setting up WS GRAM environment

In order to configure WS GRAM, you need to configure the **sudo** command. Follow the procedures below:

1. As a root user, type in the **visudo** command (see Example 11-56 on page 186).

Example 11-56 Type visudo command

```
[root@hosta]# visudo
```

2. Add the lines in Example 11-57 into the field.

Example 11-57 Example of visudo entry

```
...(unrelated information omitted)
# Globus GRAM entries

globus  ALL=(auser1,auser2) NOPASSWD:
/usr/local/globus-4.0.0/libexec/globus-gridmap-and-execute -g
/etc/grid-security/grid-mapfile
/usr/local/globus-4.0.0/libexec/globus-job-manager-script.pl *

globus  ALL=(auser1,auser2) NOPASSWD:
/usr/local/globus-4.0.0/libexec/globus-gridmap-and-execute -g
/etc/grid-security/grid-mapfile
/usr/local/globus-4.0.0/libexec/globus-gram-local-proxy-tool *
```

Note: All entries in Example 11-57 should be in one line.

Note: Do not forget to put a list of your local user names in the ALL=() clause, as shown in Example 11-57.

Verifying the installation and configuration of WS GRAM

To verify WS GRAM installation, complete the following procedure:

1. Log in to your grid node as the Globus user.
2. Start the secure container (see Example 11-53 on page 182).
3. In another console, log into your grid node with the user who has grid user certificates.
4. Run the command in Example 11-58. If you find a file, then WS GRAM is configured properly.

Example 11-58 Running simple WS GRAM command

```
[auser1@hosta]$ globusrun-ws -submit -c /bin/touch /tmp/createdfile
Submitting job...Done.
Job ID: uuid:1b4e3cb2-d966-11d9-9f76-0011250d31d9
Termination time: 06/11/2005 04:14 GMT
Current job state: Active
Current job state: CleanUp
Current job state: Done
```

```
Destroying job...Done.
[auser1@hosta]$ ls /tmp/createdfile
/tmp/createdfile
```

Submitting a WS GRAM job using a job definition file

You may define a WS GRAM job with a job definition file.

Simple echo job definition file

Example 11-59 shows a simple echo job definition file. In order to submit the WS GRAM job, type the commands as in Example 11-60.

Example 11-59 Example of echo_job.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<job>
    <executable>/bin/echo</executable>
    <argument>This file is written by WS GRAM job with job definition
file.</argument>
    <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
    <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
</job>
```

Note: ${GLOBUS_USER_HOME} is a Globus-supplied variable. For more information, look at the following link:

http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html

Example 11-60 Submitting simple echo job with globusrun-ws command

```
[auser1@hosta]$ globusrun-ws -submit -f echo_job.xml
Submitting job...Done.
Job ID: uuid:2139fdca-d9e6-11d9-afb5-0011250d31d9
Termination time: 06/11/2005 19:30 GMT
Current job state: Active
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
[auser1@hosta]$ cat ~/stdout
This file is written by WS GRAM job with job definition file.
```

WS GRAM multiple job

Example 11-61 on page 188 shows a multiple job definition that echoes a string to both host A and host B. In order to submit this WS GRAM job, type the commands as in Example 11-62 on page 188.

Example 11-61 Example of multijob.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<multiJob xmlns:gram="http://www.globus.org/namespaces/2004/10/gram/job"
     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <job>
        <factoryEndpoint>
<wsa:Address>https://hosta.redbook.ibm.com:8443/wsrf/services/ManagedJobFactory
Service</wsa:Address>
            <wsa:ReferenceProperties>
                <gram:ResourceID>Fork</gram:ResourceID>
            </wsa:ReferenceProperties>
        </factoryEndpoint>
        <executable>/bin/echo</executable>
        <argument>This file is the first file written by WS GRAM job with
multiple job definition file.</argument>
        <stdout>${GLOBUS_USER_HOME}/stdout_multi_1</stdout>
        <stderr>${GLOBUS_USER_HOME}/stderr_multi_1</stderr>
        <count>2</count>
    </job>
    <job>
        <factoryEndpoint>
<wsa:Address>https://hostb.redbook.ibm.com:8443/wsrf/services/ManagedJobFactory
Service</wsa:Address>
            <wsa:ReferenceProperties>
                <gram:ResourceID>Fork</gram:ResourceID>
            </wsa:ReferenceProperties>
        </factoryEndpoint>
        <executable>/bin/echo</executable>
        <argument>This file is the second file written by WS GRAM job with
multiple job definition file.</argument>
        <stdout>${GLOBUS_USER_HOME}/stdout_multi_2</stdout>
        <stderr>${GLOBUS_USER_HOME}/stderr_multi_2</stderr>
        <count>1</count>
    </job>
</multiJob>
```

Example 11-62 Submitting multiple jobs with globusrun-ws command

```
[auser1@hosta]$ globusrun-ws -submit -f multijob.xml  -J
Delegating user credentials...Done.
Submitting job...Done.
Job ID: uuid:08d97932-dc1f-11d9-9f2c-0011250d31d9
Termination time: 06/14/2005 15:23 GMT
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
Cleaning up any delegated credentials...Done.
[auser1@hosta]$ cat /home/auser1/stdout_multi_1
```

```
This file is the first file written by WS GRAM job with multiple job definition
file.
This file is the first file written by WS GRAM job with multiple job definition
file.
[auser1@hosta]$ ssh buser1@hostb
buser1@hostb's password:
[buser1@hostb]$ cat /home/buser1/stdout_multi_2
This file is the second file written by WS GRAM job with multiple job
definition file.
```

Note: -J option in Example 8-57 is used to delegate the credentials to each GRAM host.

WS GRAM job with stage in and stage out

Example 11-63 on page 190 shows a job definition with file stage in and file stage out. This job copies /bin/echo binary to host B, executes the **echo** command with the copied echo binary in host B, copies output files from host B to host A, then cleans up the work files in host B. See Figure 11-2 for more details.

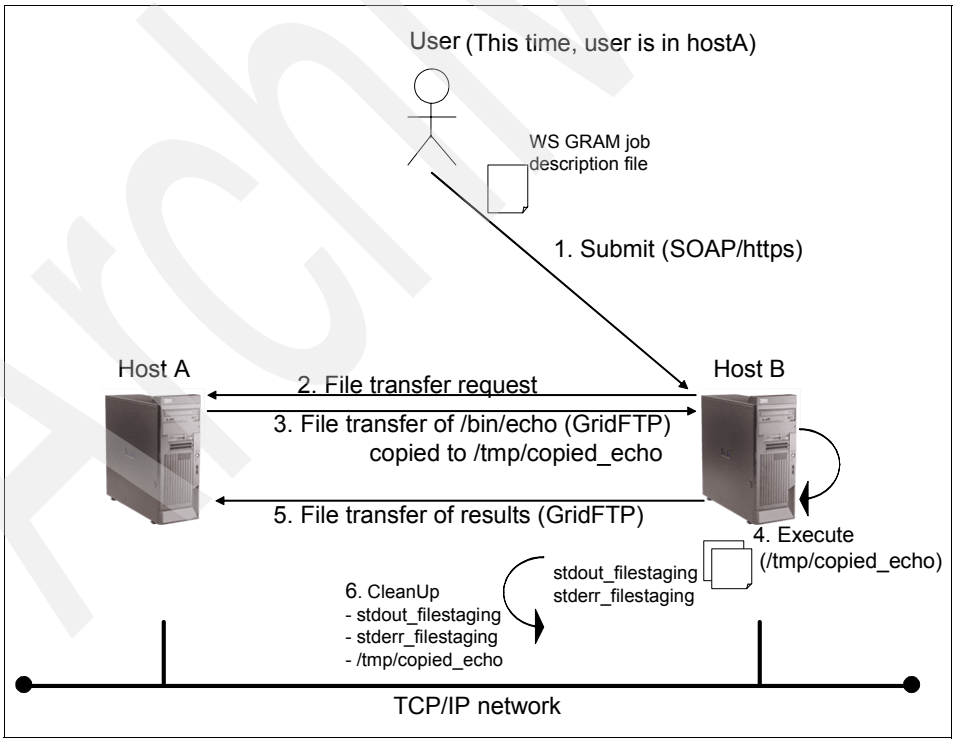

Figure 11-2 Overview of file staging GRAM job

Example 11-63 Example of filestaging.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<job  xmlns:gram="http://www.globus.org/namespaces/2004/10/gram/job"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <factoryEndpoint>
<wsa:Address>https://hostb.redbook.ibm.com:8443/wsrf/services/ManagedJobFactor
yService</wsa:Address>
        <wsa:ReferenceProperties>
            <gram:ResourceID>Fork</gram:ResourceID>
        </wsa:ReferenceProperties>
    </factoryEndpoint>
    <executable>/tmp/copied_echo</executable>
    <argument>Staging sample executed in hostb.</argument>
    <stdout>${GLOBUS_USER_HOME}/stdout_filestaging</stdout>
    <stderr>${GLOBUS_USER_HOME}/stderr_filestaging</stderr>
    <fileStageIn>
        <transfer>
            <sourceUrl>gsiftp://hosta.redbook.ibm.com//bin/echo</sourceUrl>
<destinationUrl>gsiftp://hostb.redbook.ibm.com//tmp/copied_echo</destinationUrl
>
        </transfer>
    </fileStageIn>
    <fileStageOut>
        <transfer>
<sourceUrl>gsiftp://hostb.redbook.ibm.com/${GLOBUS_USER_HOME}/stdout_filestagin
g</sourceUrl>

<destinationUrl>gsiftp://hosta.redbook.ibm.com/tmp/stdout_from_hostb</destinati
onUrl>
        </transfer>
        <transfer>
<sourceUrl>gsiftp://hostb.redbook.ibm.com/${GLOBUS_USER_HOME}/stderr_filestagin
g</sourceUrl>
<destinationUrl>gsiftp://hosta.redbook.ibm.com/tmp/stderr_from_hostb</destinati
onUrl>
        </transfer>
    </fileStageOut>
    <fileCleanUp>

<deletion><file>gsiftp://hostb.redbook.ibm.com/tmp/copied_echo</file></deletion
>

<deletion><file>gsiftp://hostb.redbook.ibm.com/${GLOBUS_USER_HOME}/stdout_files
taging</file>
</deletion>

<deletion><file>gsiftp://hostb.redbook.ibm.com/${GLOBUS_USER_HOME}/stderr_files
taging</file>
```

```
</deletion>
    </fileCleanUp>
</job>
```

In order to the submit WS GRAM job, type the commands shown in Example 11-64.

Example 11-64 Submitting file staging job with globusrun-ws command

```
[auser1@hosta]$ globusrun-ws -submit -f filestaging.xml -S
Delegating user credentials...Done.
Submitting job...Done.
Job ID: uuid:dd7e461a-dc3b-11d9-bc0b-0011250d31d9
Termination time: 06/14/2005 18:49 GMT
Current job state: StageIn
Current job state: Active
Current job state: StageOut
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
Cleaning up any delegated credentials...Done.
[auser1@hosta]$ cat /tmp/stdout_from_hostb
Staging sample executed in hostb.
[auser1@hosta]$ cat /tmp/stderr_from_hostb
[auser1@hosta]$
```

Note: The -S option shown in Example 11-64 is used to delegate the credentials to each host during staging.

11.5.7 Testing of MDS4

The configurations of MDS4 are completed automatically during the Globus Toolkit 4 installation process. In order to test the MDS4 function, follow the procedure below:

1. Log in to your grid node as the Globus user.
2. Start a secure container.(see Example 11-53 on page 182).
3. In another console, log in to your grid node with the user who has grid user certificates.
4. Run the command shown in Example 11-65. If you receive a list of services, then MDS4 is properly configured.

Example 11-65 Using wsrf-query to obtain information from MDS4

```
[auser1@hosta]$ wsrf-query -s \
https://hosta.redbook.ibm.com:8443/wsrf/services/DefaultIndexService "//*"
```

```
<ns0:IndexRP xmlns:glue="http://mds.globus.org/glue/ce/1.1"
xmlns:ns0="http://mds.globus.org/index"
xmlns:ns1="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-dra
ft-01.xsd"
xmlns:ns10="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.
2-draft-01.xsd" xmlns:ns2="http://schemas
...(unrelated information omitted)
```

11.6 Uninstallation

In order to uninstall Globus Toolkit 4, follow the procedure below.

1. If you have running WS Core containers, stop them.
2. As a root user, delete the directories listed below. (See Example 11-66 for commands.)
   - $GLOBUS_LOCATION (/usr/local/globus-4.0.0/)
   - /etc/grid-security/
   - (If you do not need Apache Ant) $ANT_HOME (/usr/apache-ant-1.6.3)

Example 11-66 Removing Globus directories

```
[root@hosta]# rm -rf /usr/local/globus-4.0.0/
[root@hosta]# rm -rf /etc/grid-security/
[root@hosta]# rm -rf /usr/apache-ant-1.6.3/
```

3. If you have changed /etc/profile in 11.5.1, “Configuring environmental variables” on page 168, remove the following lines:
   - export GLOBUS_LOCATION=/usr/local/globus-4.0.0
   - source $GLOBUS_LOCATION/etc/globus-user-env.sh
   - source $GLOBUS_LOCATION/etc/globus-devel-env.sh
4. Remove the GridFTP service settings by removing the `gsiftp 2811/tcp` line in /etc/services.
5. Remove the GridFTP daemon settings. (See Example 11-67 for the commands.)

Example 11-67 Remove GridFTP settings

```
[root@hosta]# rm /etc/xinetd.d/gsiftp
[root@hosta]# service xinetd restart
Stopping xinetd:                                           [  OK  ]
Starting xinetd:                                           [  OK  ]
```

6. Remove the Globus user. (See Example 11-68 for the commands.)

Example 11-68 Removing Globus user

```
[root@hosta]# userdel -r globus
```

7. Remove the two entries for the Globus user in /etc/sudoers by typing `visudo` and editing the file.
8. If you do not need PostgreSQL, uninstall the following rpm packages:
   – postgresql-libs
   – postgresql
   – postgresql-server

Example 11-69 Removing postgres rpm files

```
[root@zeta]# rpm -e postgresql-server
[root@zeta]# rpm -e postgresql
[root@zeta]# rpm -e postgresql-libs
```

Note: If you get dependency errors with Example 11-69, remove packages that depend on postgresql packages.

9. If you do not need IBM Java SDK, uninstall the rpm package by issuing the following command (Example 11-70).

Example 11-70 Removing IBM Java SDK

```
[root@hosta]# rpm -e IBMJava2-142-ia32-SDK
```

11.7 Summary

In this chapter we provided step-by-step instructions for setting up a grid in an environment based on Globus Toolkit 4. This environment is relatively basic and does not include all of the Globus Toolkit 4 components. However, it does provide a representative environment that can be used for self-education, testing, and creating a demonstration of certain grid capabilities.

In the next chapter, we describe a sample grid application that can be executed in the environment that we have just installed and configured.

Part 4

Grid demonstration application

12

Demonstration application

This chapter describes a demonstration application built to explore some of the functionality provided by the Globes 4 Toolkit.

Important: The application as described below was built and tested in the environment described in Chapter 11, “Globus Toolkit 4 installation and configuration” on page 155. This application is provided *as is* and is intended as a learning tool for the reader. For information related to obtaining the sample application’s source code and building the application, please refer to Appendix B, “Additional material” on page 231.

The application is a system that takes Scalable Vector Graphics (SVG) files (see http://www.w3.org/TR/SVG) and uses nodes on a grid to render a set of JPEG files representing sub-images of the complete image. As it is a demonstration system, certain design decisions and assumptions have been made to accelerate development.

The three components of the system are:

- RenderClient: This is a Java application with a graphical interface for the user that drives the rendering work on the grid and displays the resulting sub-images into a final large image. There is only one running in the grid.
- RenderWorker: This is a Java application with no graphical user interface that converts one sub-image of the SVG file into a JPEG file. There are one or more running on each node in the grid. Due to the strong parallelism inherent

in rendering an SVG file to multiple JPEG sub-images, the more nodes in the grid, the faster the SVG file will be fully rendered. You can run one or more RenderWorkers on each node, but depending on the available cycles and networking capabilities, you will reach a point of diminishing returns.

- RenderSourceService: This is a Globus Toolkit 4 grid service, deployed into a Globus Toolkit 4 container. It is initialized by the RenderClient and hands out work instructions to RenderWorker processes on the grid. There is only one running in the grid.

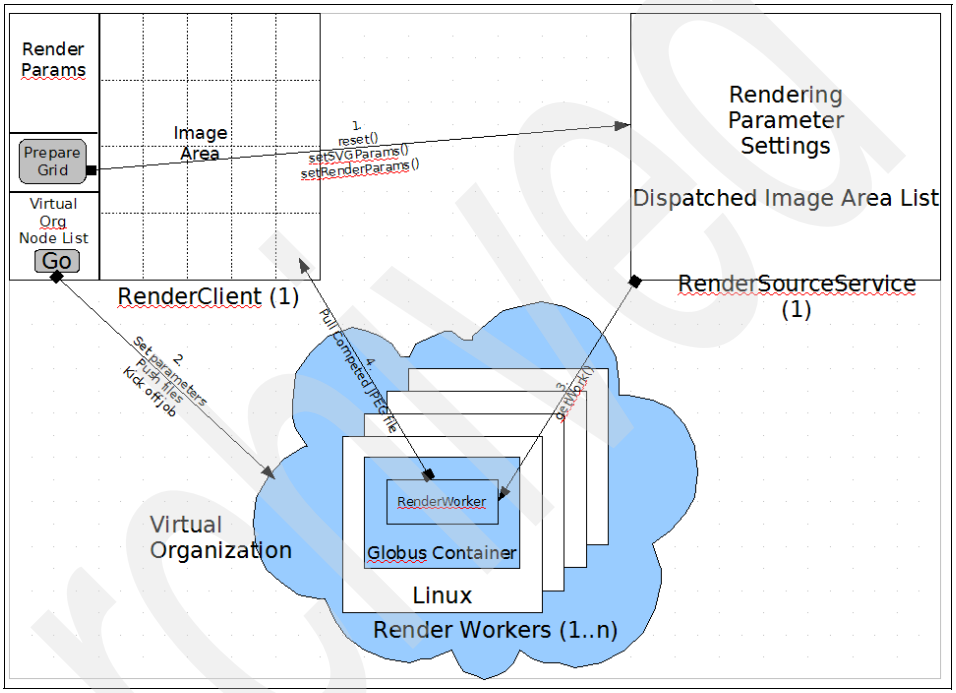

Figure 12-1 Demonstration application architecture

We use the following Globus Toolkit 4 features in this demonstration application:

- Grid service: A stateful Java class with methods using complex parameter passing and return objects
- MDS: Registration and query of nodes participating in the virtual organization
- Security: Grid proxies and certificates for secure execution of tasks and file transfers
- RFT / GridFTP: High-performance file transfers
- GRAM: Staging all files required for the RenderWorker to the node, executing the RenderWorker, and staging back the resulting JPEG file

The following sections detail the design and implementation of each component.

12.1 RenderClient

The RenderClient is a large Java application using Swing classes to present the user interface of the SVG rendering system to the user.

12.1.1 The Graphical User Interface (GUI)

The GUI consists of several parts. Most of the text fields are pre-filled with reasonable defaults in an effort to make it easier to use. Changing the default values requires editing and rebuilding the source code of the RenderClient.

Here is the overall screen layout. We examine each section in detail below.

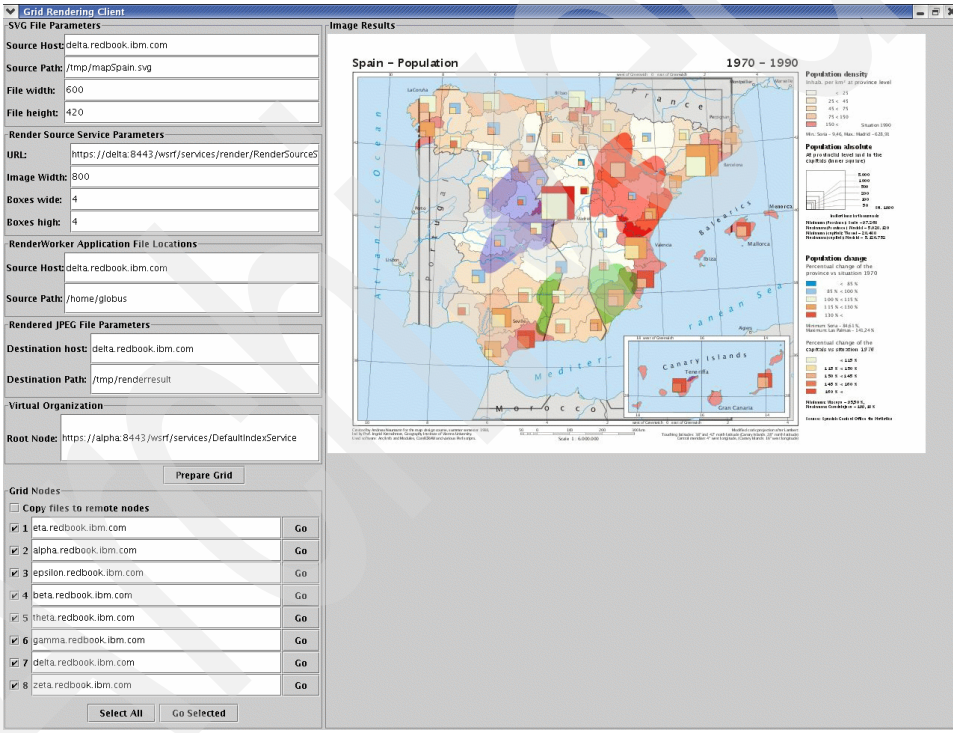

Figure 12-2 The Full RenderClient screen

The left side of the screen holds a number of grouped text fields that allow the user to customize the work to be done and buttons to initialize and launch the rendering process. The right side displays the rendered JPEG files, tiled into a single full image.

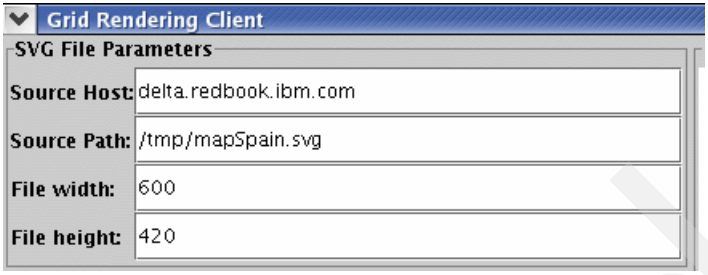

Figure 12-3 SVG File Parameters

The user provides the host and path to the SVG file to be rendered. Currently the demonstration assumes the SVG file is on the same machine running the RenderClient. The user also gives an indication of the natural height and width of the file. You can enter estimated values or examine the SVG file contents for a line near the top of the file that looks similar to:

```
<svg viewBox="0 0 600 420" width="600" height="420"
```

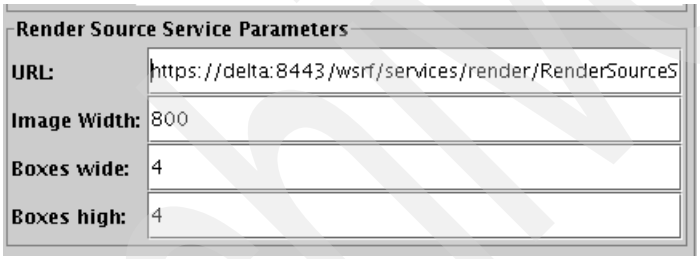

Figure 12-4 RenderSource Service Parameters

The user provides the full URL where the RenderSourceService is running in a Globus Toolkit 4 container. A suggested URL is pre-filled, as the format of the URL is very specific. The desired width of the resulting full JPEG file is given. The RenderWorker will scale as appropriate to give this resulting size, while the aspect ratio of the SVG file will be preserved in the JPEG file. Finally, the user specifies how many boxes wide and high the SVG will be broken into for rendering. The number of RenderWorkers launched will be (boxes wide * boxes high). A rule of thumb is to specify boxes high and boxes wide in proportion to the number of nodes in the virtual organization.

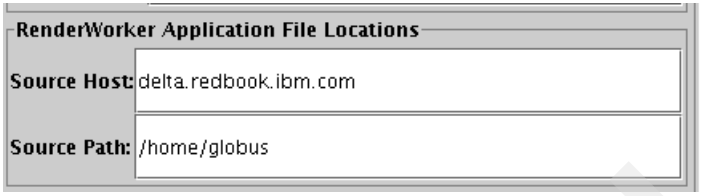

Figure 12-5 RenderWorker Application File Locations

The user provides the host and path to the location of all files required to run RenderWorker on the nodes. Currently, the demonstration application assumes that the host holding the RenderWorker files is running a Globus Toolkit 4 container, or at least the RFT service.

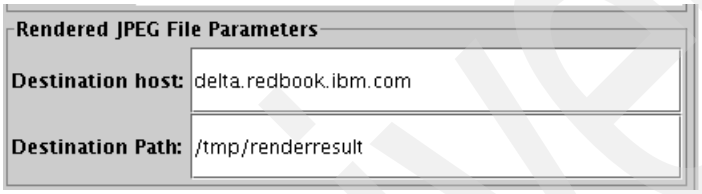

Figure 12-6 Rendered JPEG File Parameters

The user provides the host and desired path where the rendered JPEG files will be placed. The demonstration application currently assumes that the host will always be the host running the RenderClient.

Figure 12-7 Virtual Organization

The user provides the full URL to a DefaultIndexService running in a GT4 container that is defined as the root node of the virtual organization. The URL is queried to get a list of all nodes in the VO.

Figure 12-8 Prepare Grid button

When all of the above fields are filled in, clicking Prepare Grid initializes the RenderSourceService, queries the virtual organization, and builds a list of nodes in the virtual organization in the following section of the graphical interface.

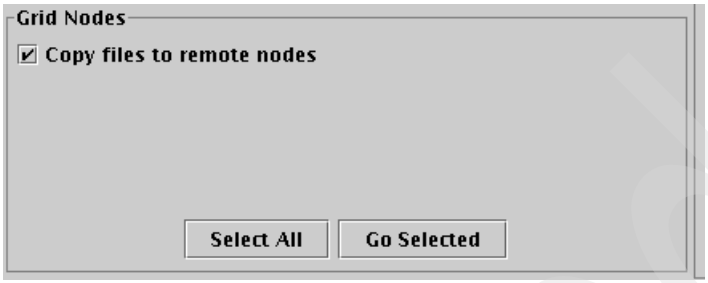

Figure 12-9 Grid nodes before clicking Prepare Grid

The “Copy files to remote nodes” box can be checked to force the staging of all files required by the RenderWorker to each node. Since this takes a non-trivial amount of time and network bandwidth, the user can uncheck this box if each node has already had the latest version of all files staged at least once.

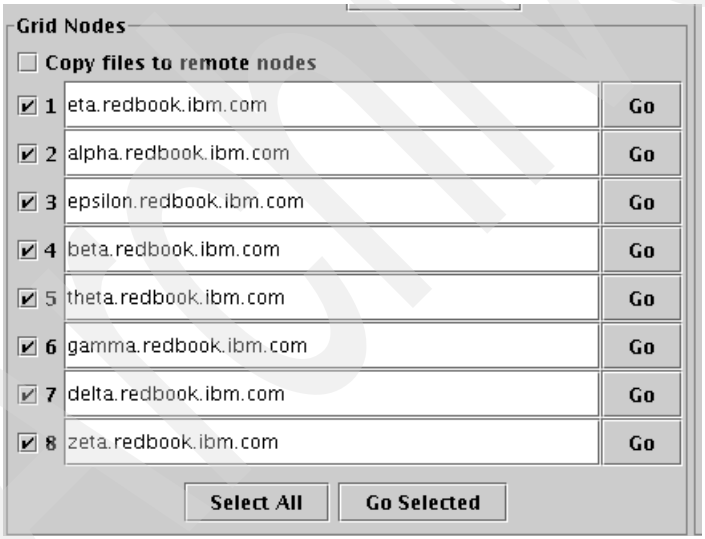

Figure 12-10 Grid nodes after clicking Prepare Grid

After the Prepare Grid button is clicked, this section of the GUI shows the list of all nodes in the virtual organization. Individual sub-images can be rendered by clicking the Go button to the right of a host name. If the user wants to start the rendering of the remaining sub-images, nodes can be selected or deselected for work.

After the user clicks Go Selected, the RenderClient provides a visual indication of the current state of the process on each worker node. The full image area is drawn with lines showing how the SVG file will be broken into sub-images and rendered by worker nodes on the grid. Text and color coding is used to track each job's progress.

There are six possible states that a job can be in on this demonstration grid:

- Preparing
- Pushing
- Rendering
- Retrieving
- Complete
- Failed

Each state and its visual representation is described below.

Preparing

Figure 12-11 Preparing job state

Preparing is the first job state entered after the user clicks Go Selected. The application creates and prepares a Globus JobDescriptionType object. This object defines everything required to prepare and execute a program on a remote system. The client then creates and initializes a GramJob object, which is used to pass the JobDescriptionType object to the Globus GRAM subsystem. Calling GRAM's submit() method launches the request to run the program on the remote system. A listener is added to the GRAM job to catch the notifications for changes in status, which drive the state changes shown on the screen.

Note that in this demonstration application each GRAM job is launched in its own thread, allowing multiple simultaneous GRAM jobs to be launched and monitored for state changes.

From a problem determination standpoint, several things can go wrong during this stage. Below are a few of the problems we encountered while developing and testing this application:

- Cannot communicate with the ManagedJobFactoryService running in the Globus container on the worker node. We found that some systems were

silently running iptables, which blocked communications, so all systems must permanently turn off iptables or do `/etc/init.d/iptables stop` after each reboot.

- Incorrect parameters defined in the JobDescriptionType and GramJob objects.

To enable debugging and tracing, edit /usr/local/globus-4.0.0/container-log4j.properties. Comment or uncomment lines at the bottom of the file to get more information for GRAM and RFT/GridFTP.

Pushing

Figure 12-12 Pushing files to worker node state

Pushing is the term used by the RenderClient for GRAM's StageIn state. StageIn means copying the list of requested files from their source on the network to the destination worker node. Note that any number of files can be designated for staging in, and they can be sourced from different servers on the network. The only caveat is that GRAM must be able to find the file on the attached storage via the file: Protocol or via a Globus container or RFT/GridFTP server via the gsiftp: protocol.

Several things can potentially go wrong during this stage, including:

- Not being able to locate the files to be staged in.
- Destination directory on the worker node does not exist—GRAM will not automatically create a directory path for you.
- Insufficient privilege to write the file to its destination location on the worker node.
- When running multiple RenderWorkers on a single worker node, GRAM is not smart enough to only copy the files once. We have seen the JVM crash when a JAR file was in the process of being overwritten by one job while being used by another job.

Rendering

Figure 12-13 Rendering Image on worker node state

Rendering is the term used for the RenderClient for GRAM's active state. Active means the program defined in the JobDescriptionType object is about to be launched or is currently executing.

Several things can potentially go wrong during this stage, including:

- The command line defined in the JobDescriptionType failed.
- The program's environment is not fully set up, including JAVA_HOME, GLOBUS_LOCATION, CLASSPATH, LIBPATH, and LD_LIBRARY_PATH.
- Globus security fails due to non-existent or expired proxy credentials.
- Insufficient privilege to create the program's output files.
- Program-specific failures. In the case of this demo, the Batik library requires communication with the machine's X Windows server, so we had to set the DISPLAY environment variable. Also, an administrator must do 'xhost +' on each worker node or Batik cannot open the X Windows DISPLAY to run its code. An alternative is to do 'xhost +' on a single node and modify the shell script's DISPLAY variable to point to that host.

It is a good idea to build your target application such that it can be run and debugged by hand on a worker node, minimizing the amount of effort once you get to the GRAM part of your development and testing.

The most important thing we discovered for this state is that even though the job is run under the auspices of a certain user ID on the worker node, for example, globus, the program does not inherit any environment that the user would normally see if logged into a system with that user ID. In the end we had to provide a complete environment setup for the program to run, so we made a shell script, called runrenderworker, that performed all environment setup, and then as the last step launched the target Java application, RenderWorker. This has the benefit of getting RenderWorker to run successfully on the worker nodes, but the detriment is the directory paths to all required software components (Java runtime, Globus installation, system libraries, and so on) had to be hardcoded in the shell script. Therefore, all worker nodes had to be identically configured to allow for successful execution across the grid.

Retrieving

Figure 12-14 Retrieving image from worker node state

Retrieving is the term used for the RenderClient for GRAM's StageOut state. StageOut occurs after the program finishes execution and pulls any requested files back from the worker node to another location on the network. For our demonstration, the generated JPEG file was pulled back to the machine running the RenderClient application.

Several things can potentially go wrong during this stage, including:

- Files expected to exist for StageOut do not exist due to an incorrect definition in the JobDescriptionType object or failure of the program.
- Insufficient privilege to copy the program's output files to the destination system.

Complete

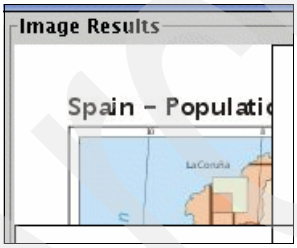

Figure 12-15 Job complete state

Complete is the term used for the RenderClient for GRAM's done state. Complete occurs after files are staged out to the destination system. Our demo application located the rendered JPEG file by its expected file name and displays it at the proper coordinates in the Image Results part of the screen.

Several things can potentially go wrong during this stage, including:

- The expected JPEG file does not exist, due to a failure described above.
- The JPEG file is actually not in JPEG format due to an error with the RenderWorker.

Failed

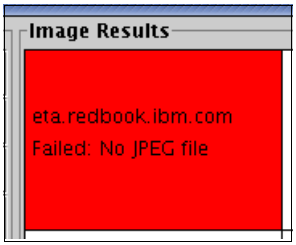

Figure 12-16 Job failed state

Failed is the term used for the RenderClient for GRAM's failed state. A job can fail for any of the reasons described above, at any state of the process. In the demo application we try to catch as many failure modes as possible to give the user a hint as to what failed. Failures are unusual once a system goes into production, but during development GRAM can fail in several ways. There are several places to look for help, both information messages and error messages, when something goes awry:

- The output of the Globus container where the RenderClient is running. You may see security failure-related errors, file StageIn and StageOut transfer errors, and so on.
- The terminal window where you executed the RenderClient. The demo application emits many messages when an error is determined. You can also uncomment desired System.out.println lines and rebuild the RenderClient to see trace information.
- The output of the Globus container where the RenderSourceServce is running. You may see security failure-related errors and grid service API errors. You can also uncomment desired System.out.println lines and rebuild the RenderSourceService to see trace information.
- The contents of the StageIn target directory on the worker node will show if all files are being successfully staged in.
- The stdout and stderr files created by GRAM for your remote application. Messages from both the shell script and RenderWorker are shown. You can also uncomment desired System.out.println lines and rebuild the RenderWorker to see trace information.
- The contents of the StageOut target directory on the system running RenderClient will show if all JPEG files are being successfully staged out.

View of successful completion of job

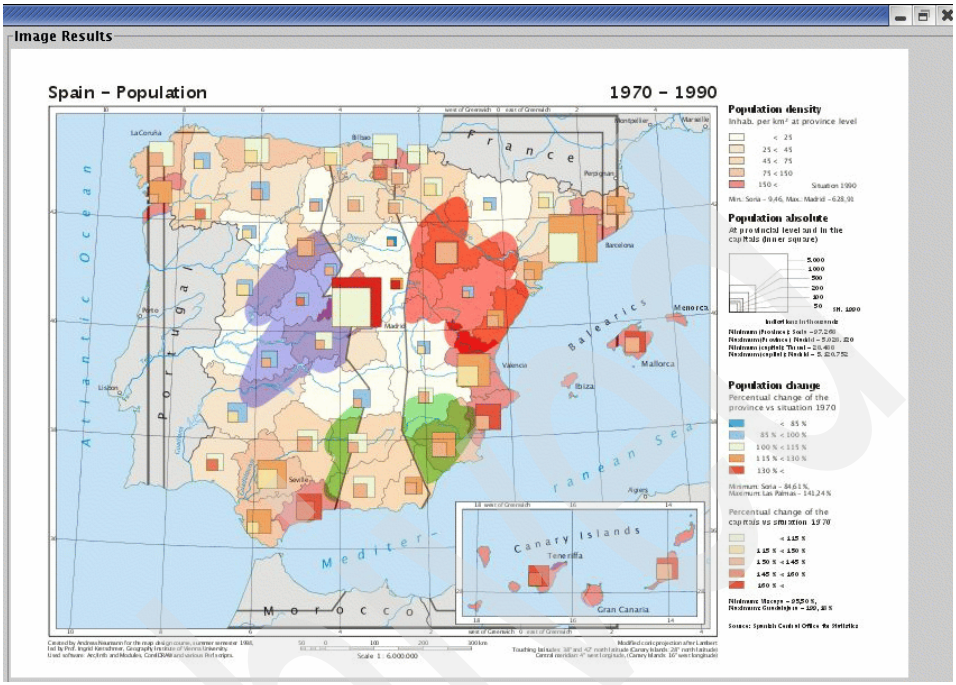

Figure 12-17 Resulting image area

The entire right side of the GUI window is reserved for the resulting sub-image JPEGs.

12.1.2 RenderClient source code

> **Attention:** This demonstration application is available to be downloaded as described in Appendix B, “Additional material” on page 231. Detailed descriptions of the source code and how to develop grid applications using the Globus Toolkit are beyond the scope of this book. However, some information is provided below for those interested in modifying or adapting this code for their unique environments.

The source code to the RenderClient application is defined in two classes, RenderClient.java and GRAMLocator.java.

We highlight some interesting aspects of the RenderClient application below:

- A special inner class called SVGImage is used to maintain the state of each of the sub-images.
- When the Prepare Grid button is clicked, the GramLocator object is used to query all nodes in the virtual organization and populate the list on the screen. Then the internal state of image dispatching is reset and the area for the resulting JPEG images is cleared and segmented. Finally, the createRenderSourceInstance() method uses Globus facilities to locate the RenderSourceService and call its reset(), setSVGParams(), and setRenderParams() methods.
- When the Go Selected button is clicked, a check is made to make sure at least one node is selected for processing. Then each sub-image is dispatched to each selected worker node in round-robin fashion via threads.
- If one of the Go buttons is clicked, the next non-dispatched sub-image is dispatched to the corresponding worker node via a thread.
- The WorkerDispatch class handles the actual dispatching of each sub-image rendering job. Doing this in a thread allows the parallel execution and asynchronous status update of each sub-image. The run() method is used to set up the job via the Globus JobDescriptionType class. JobDescriptionType is a complex class that is used to set up all aspects of a task to be executed on a remote node. Normally, a subset of the full object setting will get the job done, but you may need to set more fields, depending on your project's requirements.
  - Environment variables
  - Working directory
  - Executable name
  - Executable parameters
  - Location of stdout and stderr files created by the remote task
- The source and destination of all files to be transferred to the remote node are also defined in the JobDescriptionType object. Normally, the file description is in URL format starting with "gsiftp://..." signifying the Globus RFT/GridFTP facilities are to be used to perform the transfers. Note that our demonstration application requires 18 files to be staged to the remote node:
  - A shell script, called runrenderworker, that sets up the full environment (Java and Globus locations, CLASSPATH, LIBPATH, and others), runs grid-proxy-init, and then launches the RenderWorker java application
  - A Jar file containing the RenderWorker application
  - A Jar file containing the stub classes required to communicate with the RenderSourceService

  - Thirteen Jar files from the Apache Batik project, which contains the classes that actually convert SVG files to JPEG files
  - The SVG file whose sub-images are to be rendered into a JPEG file
  - Source and destination of the JPEG file to be transferred back to the system running the RenderClient (called Staging In)
- When the JobDescription is fully set up, a GramJob object is created and used to submit the job to the Globus GRAM facility.
- A listener is added to the GramJob that catches all changes in the job's state, as reported by GRAM. At each state change, the graphical interface is updated with color coding and status messages. When GramJob returns, the job has completed, either succeeding or failing, so the code attempts to load and display the JPEG file that was expected to be returned.
- A special inner class called SVGImagePanel is used to force the proper repainting of each sub-image on the screen.
- A helper class called GRAMLocator is used to query the root node of the virtual organization and return a list of nodes registered in that virtual organization.

12.2 RenderWorker

The RenderWorker is a standalone Java application (with no GUI) that is launched by GRAM on a worker node. It is a single Java file called RenderWorker.java.

It creates a JPEG file corresponding to a given sub-image of the SVG file and exits. This is a design choice we made in order to make the design and execution of the RenderWorker straightforward.

When launched, it checks that its two required parameters are available. The first parameter is the full URL of the RenderSourceService and the second parameter is the sub-image number that it should render.

It uses Globus methods to connect to the RenderSourceService and call the getWork() method. getWork() returns a Java class that describes everything the RenderWorker needs to know to create a JPEG image, including the rectangle of interest in the SVG file, the size of the resulting JPEG file, and the names of the SVG and the JPEG file.

Note that the source code uses a number of Java classes that are generated from the WSDL definition of the RenderSourceService's grid service API, which is standard procedure when building and working with Globus 4 grid services. The most important object is GetWorkResponse, a compound generated Java

object that contains all of the information needed for the RenderWorker to do its job.

The code uses the Apache Batik library's JPEGTranscoder class to set up the proper parameters for the generation of the JPEG image, then generates the JPEG file.

One thing you might notice in the code is the following:

```
static
{
Util.registerTransport();
}
```

This is the prescribed way to avoid a `No socket factory for https` error in any Globus code working in secure mode.

12.3 RenderSourceService

The RenderSourceService is a straightforward Globus Toolkit 4 grid service that maintains an internal state and has several public methods. Its purpose is to hand out sub-image chunks of work when a RenderWorker somewhere on the grid makes a request.

The service currently has several Globus 4 resources defined, but we do not currently exploit any Globus Toolkit 4 resource facilities in this application.

The service is set up to start an instance when the container it is deployed into is started. It calls its own reset() method to initialize the internal state, where none of the sub-images have been marked as dispatched to remote nodes.

The setSVGParams() and setRenderParams() methods are called by the RenderClient to pass the user-supplied SVG and JPEG parameters and prepare for dispatch of a new series of RenderWorkers.

When a RenderWorker calls getWork(), the service takes the sub-image number passed in, calculates the image coordinates, builds an object describing the work to be done, and hands it back to the RenderWorker.

Note that all methods along with their input and output parameters are defined in WSDL so other applications on the grid can call the methods and interpret the return values properly.

12.3.1 Alternative architecture

As they stand today, the RenderClient and RenderSourceService share some of the tasks of dispatching work to the nodes; specifically, they must stay in sync with which sub-images have been dispatched. In hindsight, it would have been better to give the RenderSourceService full *scheduler* responsibility.

The RenderClient would continue to collect the parameters from the user and pass them to the RenderSourceService as is done today. The design would change so when the user clicks Go or Go Selected, the RenderClient would simply ping the RenderSourceService. The RenderSourceService would do all job creation, file staging, and fire off the RenderWorkers. The RenderClient would register for any job state changes and gain access to the resulting JPEG file via Globus Toolkit 4's Resource facility.

12.4 DirectoryTree of important files in demo

Note: In the various scripts and other files that are described below, we have hard coded many of the directory paths based on our specific environment. This constrains us to having identical environments and directory structures on each of our grid nodes. This may not be a practical constraint in many environments. More complex scripts and the use of environment variables or parameters may be required.

<top directory of our project>

buildclient: batch file to build RenderClient

```
source $GLOBUS_LOCATION/etc/globus-devel-env.sh
javac -classpath ./build/stubs/classes/:$CLASSPATH
com/ibm/redbook/gridintro/render/clients/RenderClient.java
javac -classpath ./build/stubs/classes/:$CLASSPATH
com/ibm/redbook/gridintro/render/clients/GRAMLocator.java
```

buildworker: batch file to build RenderWorker

```
source $GLOBUS_LOCATION/etc/globus-devel-env.sh
javac -classpath batik-transcoder.jar:../build/stubs/classes/:$CLASSPATH com/ibm/redbook/gridintro/render/worker/RenderWorker.java
jar cf RenderWorker.jar com/ibm/redbook/gridintro/render/worker/RenderWorker.class
```

buildservice: batch file to build RenderSourceService

```
export GLOBUS_LOCATION=/usr/local/globus-4.0.0
./globus-build-service.sh -d com/ibm/redbook/gridintro/render -s schema/gridintro/RenderSourceService_instance/RenderSourceService.wsdl
#copy up the generated jar files to the main directory
cp build/lib/com_ibm_redbook_gridintro_render.jar .
cp build/lib/com_ibm_redbook_gridintro_render_stubs.jar .
```

deployservice: batch file to deploy RenderSourceService into a GT4 container

```
/usr/local/globus-4.0.0/bin/globus-deploy-gar com_ibm_redbook_gridintro_render.gar
```

undeployservice: batch file to undeploy RenderSourceService from a GT4 container

```
/usr/local/globus-4.0.0/bin/globus-undeploy-gar com_ibm_redbook_gridintro_render
```

startcontainer: batch file to start a GT4 container

```
/usr/local/globus-4.0.0/bin/globus-start-container
```

runclient: batch file to run RenderClient

```
source $GLOBUS_LOCATION/etc/globus-devel-env.sh
rm -rf /tmp/*.jpg
java -DGLOBUS_LOCATION=$GLOBUS_LOCATION -classpath ./build/stubs/classes/:$CLASSPATH
    com/ibm/redbook/gridintro/render/clients/RenderClient
    http://127.0.0.1:8080/wsrf/services/render/RenderService
```

runworker: batch file to run RenderWorker in standalone mode for testing

```
./runrenderworker https://192.168.1.111:8443/wsrf/services/render/RenderSourceService 1
```

build.xml: build files provided by the Globus Service Build Tools project - http://gsbt.sourceforge.net, unchanged by our work

globus-build-service.sh: build file for Globus services, provided by the Globus Service Build Tools, unchanged by our work.

client-config.wsdd: configuration file used as input to the build process, provided by the Globus Service Build Tools, unchanged by our work

namespace2package.mappings: file that maps abstract namespaces of our project to concrete class names implementing the service

```
http\://www.globus.org/namespaces/render.gridintro.redbook.ibm.com/RenderSourceService_instance=c
om.ibm.redbook.gridintro.render.stubs.RenderSourceService_instance
http\://www.globus.org/namespaces/render.gridintro.redbook.ibm.com/RenderSourceService_instance/b
indings=com.ibm.redbook.gridintro.render.stubs.RenderSourceService_instance.bindings
http\://www.globus.org/namespaces/render.gridintro.redbook.ibm.com/RenderSourceService_instance/s
ervice=com.ibm.redbook.gridintro.render.stubs.RenderSourceService_instance.service
```

com_ibm_redbook_gridintro_render.gar: Grid Archive file containing the RenderSourceService, ready for deployment. This is the output file of the buildservice process.

runrenderworker: batch file that is the main executable, staged to the worker nodes.

```
#!/bin/bash

# this script is pushed to the remote node and executed to run the RenderWorker application
# two arguments: URL of RenderSourceService and block number to render
# echo Running RenderWorker kickoff script

#delete stdout and stderr to assist with debugging - GRAM appends every job
#rm -f stdout
#rm -f stderr

# set up the path to the Java runtime
export JAVA_HOME=/usr/java/j2sdk1.4.2_08
```

```
# echo JAVA_HOME is $JAVA_HOME

# set up the path to the globus install
export GLOBUS_LOCATION=/usr/local/globus-4.0.0
# echo GLOBUS_LOCATION is $GLOBUS_LOCATION

# run the globus script to set up the classpath with all of the globus jars
# echo sourcing globus environment in $GLOBUS_LOCATION
source $GLOBUS_LOCATION/etc/globus-devel-env.sh
# echo sourced globus environment

# add render demo specific jars to classpath
# echo adding required worker jars
export
CLASSPATH=RenderWorker.jar:com_ibm_redbook_gridintro_render_stubs.jar:batik.jar:batik-dom.jar:bat
ik-svg-dom.jar:batik-css.jar:batik-rasterizer.jar:batik-transcoder.jar:batik-bridge.jar:batik-gvt
.jar:batik-util.jar:batik-ext.jar:batik-xml.jar:batik-script.jar:batik-awt-util.jar:batik-parser.
jar:$CLASSPATH
# echo final classpath is $CLASSPATH

# set up the path to the Globus and system libraries
export LIBPATH=/usr/local/globus-4.0.0/lib:/usr/lib:/lib
# echo LIBPATH is $LIBPATH

# set up another path to Globus libraries
export LD_LIBRARY_PATH=/usr/local/globus-4.0.0/lib
# echo LD_LIBRARY_PATH is $LD_LIBRARY_PATH

# having problems with credentials on each worker node, so manually ran grid-proxy-init -hours
100000 to avoid
# set up proper user for security credentials for this application
# echo setting X509_USER_PROXY
#ID=`/usr/bin/id -u`
#X509_USER_PROXY="/tmp/x509up_u$ID"
#echo set X509_USER_PROXY to $X509_USER_PROXY
# request security credentials for this application
# echo doing grid-proxy-init
#$GLOBUS_LOCATION/bin/grid-proxy-init
# echo did grid-proxy-init

# for some reason the Batik library needs to have X Windows' DISPLAY variable set
# this has to be done to the machine before this script is run
#echo setting DISPLAY and running xhost +
export DISPLAY=192.168.1.103:0.0
#export DISPLAY=localhost:0.0
#/usr/X11R6/bin/xhost +

# now execute the application with the passed in URL to the RenderSourceService
echo launching RenderWorker class with parameter $1
$JAVA_HOME/bin/java com/ibm/redbook/gridintro/render/worker/RenderWorker $1 $2
# echo completed RenderWorker, exiting
```

RenderWorker.jar: jar file containing the compiled RenderWorker java class.

batik-awt-util.jar: Apache Batik jar files, staged to the worker nodes.

- batik-bridge.jar
- batik-css.jar
- batik-dom.jar
- batik-ext.jar
- batik-gvt.jar
- batik.jar
- batik-parser.jar

- batik-rasterizer.jar
- batik-script.jar
- batik-svg-dom.jar
- batik-transcoder.jar
- batik-util.jar
- batik-xml.jar

mapSpain.svg: test SVG file provided by the Apache Batik project, staged to the worker nodes

com_ibm_redbook_gridintro_render_stubs.jar: jar file generated by the Globus service build process, containing Java objects for RenderSourceService methods, staged to the worker nodes.

Note that the tree for the Java classes may follow the package naming convention you choose, but the other configuration files must be placed in very specific places in order to be found and processed during the build and deployment stages.

./com/ibm/redbook/gridintro/render:

deploy-jndi-config.xml: instructions to the container about how to deploy the RenderSourceService

```
<?xml version="1.0" encoding="UTF-8" ?>
<jndiConfig xmlns="http://wsrf.globus.org/jndi/config">
     <service name="render/RenderSourceService">
          <resource name="home" type="org.globus.wsrf.impl.ServiceResourceHome">
          <resourceParams>
          <parameter>
          <name>factory</name>
          <value>org.globus.wsrf.jndi.BeanFactory</value>
          </parameter>
          </resourceParams>
          </resource>
     </service>
</jndiConfig>
```

deploy-server.wsdd: instructions to the container about how to deploy the RenderSourceService

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <service name="render/RenderSourceService" provider="Handler" use="literal" style="document">
        <parameter name="className" value="com.ibm.redbook.gridintro.render.impl.RenderSourceService"/>
        <wsdlFile>share/schema/gridintro/RenderSourceService_instance/RenderSourceService_service.wsdl</wsdlFile>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="handlerClass" value="org.globus.axis.providers.RPCProvider"/>
        <parameter name="scope" value="Application"/>
        <parameter name="providers" value="GetRPProvider"/>
        <parameter name="loadOnStartup" value="true"/>
    </service>

</deployment>
```

./com/ibm/redbook/gridintro/render/clients:

RenderClient.java: source file for RenderClient application

GRAMLocator.java: source file for RenderClient application

./com/ibm/redbook/gridintro/render/impl:

RenderSourceService.java: source file for RenderSourceService GT4 service

RenderSourceServiceQNames.java: source file for RenderSourceService GT4 service

./com/ibm/redbook/gridintro/render/worker:

RenderWorker.java: source file for RenderWorker application

./schema/gridintro/RenderSourceService_instance:

RenderSourceService.wsdl: Web Services Description Language (WSDL) file defining methods and parameters for RenderSourceService

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="RenderSourceService"

targetNamespace="http://www.globus.org/namespaces/render.gridintro.redbook.ibm.com/RenderSourceService_instance"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.globus.org/namespaces/render.gridintro.redbook.ibm.com/RenderSourceService_instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />

<!--============================================================
T Y P E S
============================================================-->
<types>
<xsd:schema
targetNamespace="http://www.globus.org/namespaces/render.gridintro.redbook.ibm.com/RenderSourceService_instance"
    xmlns:tns="http://www.globus.org/namespaces/render.gridintro.redbook.ibm.com/RenderSourceService_instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- REQUESTS AND RESPONSES -->

<xsd:element name="reset">
     <xsd:complexType/>
</xsd:element>
<xsd:element name="resetResponse">
     <xsd:complexType/>
</xsd:element>


<xsd:element name="setSVGParams">
     <xsd:complexType>
     <xsd:sequence>
          <xsd:element name="uriSVG" type="xsd:string"/>
          <xsd:element name="svgDocWidth" type="xsd:int"/>
          <xsd:element name="svgDocHeight" type="xsd:int"/>
     </xsd:sequence>
     </xsd:complexType>
</xsd:element>
<xsd:element name="setSVGParamsResponse">
```

```
    <xsd:complexType/>
</xsd:element>

<xsd:element name="setRenderParams">
    <xsd:complexType>
    <xsd:sequence>
        <xsd:element name="uriJPEG" type="xsd:string"/>
        <xsd:element name="imageWidth" type="xsd:int"/>
        <xsd:element name="imageHeight" type="xsd:int"/>
        <xsd:element name="blocksWide" type="xsd:int"/>
        <xsd:element name="blocksHigh" type="xsd:int"/>
    </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="setRenderParamsResponse">
    <xsd:complexType/>
</xsd:element>

<xsd:element name="getWork" type="xsd:int"/>
<xsd:element name="getWorkResponse">
    <xsd:complexType>
    <xsd:sequence>
        <xsd:element name="uriSVG" type="xsd:string"/>
        <xsd:element name="uriJPEG" type="xsd:string"/>
        <xsd:element name="blockNumber" type="xsd:int"/>
        <xsd:element name="numBlocksWide" type="xsd:int"/>
        <xsd:element name="numBlocksHigh" type="xsd:int"/>
        <xsd:element name="svgBlockX" type="xsd:int"/>
        <xsd:element name="svgBlockY" type="xsd:int"/>
        <xsd:element name="svgBlockWidth" type="xsd:int"/>
        <xsd:element name="svgBlockHeight" type="xsd:int"/>
        <xsd:element name="imageBlockX" type="xsd:int"/>
        <xsd:element name="imageBlockY" type="xsd:int"/>
        <xsd:element name="imageBlockWidth" type="xsd:int"/>
        <xsd:element name="imageBlockHeight" type="xsd:int"/>
    </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="getLastJPEGRP">
<xsd:complexType/>
    </xsd:element>
<xsd:element name="getLastJPEGRPResponse" type="xsd:string"/>

<!-- RESOURCE PROPERTIES -->

<xsd:element name="LastJPEG" type="xsd:string"/>

<xsd:element name="RenderResourceProperties">
    <xsd:complexType>
    <xsd:sequence>
        <xsd:element ref="tns:LastJPEG" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    </xsd:complexType>
</xsd:element>

</xsd:schema>
</types>

<!--============================================================
M E S S A G E S
============================================================-->
<message name="ResetInputMessage">
    <part name="parameters" element="tns:reset"/>
</message>
<message name="ResetOutputMessage">
    <part name="parameters" element="tns:resetResponse"/>
</message>

<message name="SetSVGParamsInputMessage">
    <part name="parameters" element="tns:setSVGParams"/>
</message>
<message name="SetSVGParamsOutputMessage">
    <part name="parameters" element="tns:setSVGParamsResponse"/>
</message>
```

```
<message name="SetRenderParamsInputMessage">
    <part name="parameters" element="tns:setRenderParams"/>
</message>
<message name="SetRenderParamsOutputMessage">
    <part name="parameters" element="tns:setRenderParamsResponse"/>
</message>

<message name="GetWorkInputMessage">
    <part name="parameters" element="tns:getWork"/>
</message>
<message name="GetWorkOutputMessage">
    <part name="parameters" element="tns:getWorkResponse"/>
</message>

<message name="GetLastJPEGRPInputMessage">
    <part name="parameters" element="tns:getLastJPEGRP"/>
</message>
<message name="GetLastJPEGRPOutputMessage">
    <part name="parameters" element="tns:getLastJPEGRPResponse"/>
</message>

<!--============================================================
P O R T T Y P E
============================================================-->
<portType name="RenderSourceServicePortType"
    wsdlpp:extends="wsrpw:GetResourceProperty"
    wsrp:ResourceProperties="tns:RenderResourceProperties">
<operation name="reset">
    <input message="tns:ResetInputMessage"/>
    <output message="tns:ResetOutputMessage"/>
</operation>
<operation name="setSVGParams">
    <input message="tns:SetSVGParamsInputMessage"/>
    <output message="tns:SetSVGParamsOutputMessage"/>
</operation>
<operation name="setRenderParams">
    <input message="tns:SetRenderParamsInputMessage"/>
    <output message="tns:SetRenderParamsOutputMessage"/>
</operation>
<operation name="getWork">
    <input message="tns:GetWorkInputMessage"/>
    <output message="tns:GetWorkOutputMessage"/>
</operation>
<operation name="getLastJPEGRP">
    <input message="tns:GetLastJPEGRPInputMessage"/>
    <output message="tns:GetLastJPEGRPOutputMessage"/>
</operation>
</portType>
</definitions>
```

There are also a significant number of files created under the build directory as a result of the build process. Luckily, the build process is highly automated and you should not need to worry about anything in here. You can safely delete the entire build tree and re-run all build scripts.

Some of the products of the build process are .java source files, which are then compiled and used later in the build process. These tend to be the Java definition of resources and helper classes for complex objects passed in and back from Globus service methods. These source files can be useful to track down strange compiler errors or runtime bugs dealing with calling these methods.

Double-check the spelling of all method and parameter variable names. Remember that by convention a parameter called *value* will have corresponding methods called getValue() and setValue().

It is always better to take an existing file and make slight modifications rather than trying to write one from scratch, for example, namespace2package.mappings, deploy-jndi-config.xml, deploy-server.wsdd, <>QNames.java, <>.wsdl.

Part 5

Appendixes

A

IBM software portfolio for grid computing

This appendix provides a short summary of some IBM software that has particular application for grid environments.

IBM Application Workload Modeler

This can help you allocate existing system resources more efficiently by modeling, generating real traffic on your network, and evaluating the network performance of existing workloads.

IBM Cloudscape/Apache Derby

IBM Cloudscape™ V10.0 is a pure, open source-based Java relational database management system that can be embedded in Java programs and used for online transaction processing (OLTP). A platform-independent, small-footprint (2MB) database, Cloudscape V10.0 integrates tightly with any Java-based solution. It has been donated to the Apache Software Foundation and is now named Derby.

DB2 Connect Family

DB2® Connect™ connects LAN-based systems and their desktop applications to your company's mainframe and minicomputer host databases. Designed to address the needs of organizations that require robust connectivity from a variety of desktop systems including workgroup/departmental and LAN-based systems to mainframes and iSeries™ database servers.

DB2 Everyplace Family

This creates secure embedded mobile data management solutions easily using the DB2 Everyplace® Database. Use industry standard SQL to store and query data in the high-performance, small footprint relational database.

DB2 Universal Database Family

This is the premier IBM database and data management products.

Mathematical Acceleration Subsystem (MASS)

Mathematical Acceleration Subsystem consists of libraries of tuned mathematical intrinsic functions, available in versions for the AIX and Linux platforms. MASS libraries offer improved performance over the standard

mathematical library routines, are thread-safe, and support compilations in C, C++, and Fortran applications.

Rational Application Developer for WebSphere Software

Quickly design, develop, analyze, test, profile, and deploy Web, Web services, Java, J2EE, and portal applications with this comprehensive IDE. Optimized for IBM WebSphere® software, and supporting multi-vendor runtime environments, IBM Rational® Application Developer for WebSphere Software is powered by the Eclipse open source platform so developers can adapt and extend their development environment to match their needs and increase their productivity. When used with the IBM Software Development Platform, developers can access a broad range of requirements and change management functions directly from Rational Application Developer for WebSphere Software. Adapt and extend your development environment with Eclipse-based plug-ins to match your needs.

IBM Tivoli Access Manager Family

IBM Tivoli® Access Manager is an award winning, policy-based access control solution for e-business and enterprise applications that is in the leader quadrant of Gartner's Magic Quadrant. Tivoli Access Manager for e-business can help you manage growth and complexity, control escalating management costs, and address the difficulties of implementing security policies across a wide range of Web and application resources.

IBM Tivoli Configuration Manager

IBM Tivoli Configuration Manager provides the ability to capture your best practices for software distribution, automate those best practices, and enforce corporate standards. It helps you gain total control over your heterogeneous enterprise software and hardware.

IBM Tivoli Enterprise Console

IBM Tivoli Enterprise™ Console® provides sophisticated, automated problem diagnosis and resolution to improve system performance and reduce support costs. The new enhancements focus on time to value and ease of use with out-of-the-box best practices to simplify and accelerate deployment. The

auto-discovery feature allows you to understand the environment and process events appropriately. The Web console enhances visualization while providing remote access to events and console operations.

IBM Tivoli Intelligent Orchestrator

IBM Tivoli Intelligent Orchestrator helps you to improve return of IT assets and increase server utilization. It helps boost server-to-administrator ratios by automatically triggering the provisioning, configuration, and deployment of a solution into production. This automated process supports servers, operating systems, storage, middleware, applications, and network devices. IBM Tivoli Intelligent Orchestrator extends the benefits of the IBM Tivoli Provisioning Manager. It intelligently and dynamically issues instructions to Tivoli Provisioning Manager, which then uses automation packages to maintain server availability and meet required service levels in accordance with business priorities. It provides the why, where, and when of a complete orchestration solution.

IBM Tivoli License Manager

IBM license management software offerings help companies achieve a total software asset management solution, enabling planning, management and optimization of enterprise-wide software assets.

The IBM Tivoli Management Framework

The IBM Tivoli Management Framework is the foundation for a suite of management applications that are making systems and network management easy. This shields administrators from platform-specific details of day-to-day operations. Common operations such as deploying applications and routine network maintenance can be performed with a single action; administrators are no longer required to repeat the same operation for each platform on your enterprise. Deploy applications to literally thousands of machines with one operation, all the while ensuring the applications remain available.

IBM Tivoli Monitoring for Virtual Servers

IBM Tivoli Monitoring for Virtual Servers centrally monitors server virtualization and consolidation resource performance and availability at the enterprise level for efficient and cost-effective IT operations. IBM Tivoli Monitoring for Virtual

Servers allows for quick problem identification, notification, and correction, and provides tasks to automate and perform routine operations.

IBM Tivoli OMEGAMON XE Family

IBM Tivoli OMEGAMON® XE for Distributed Systems offers a unique approach to enterprise management “proactivity and advanced automation,” which is especially important as IT structures become increasingly complex and heterogeneous. An integrated approach to management, Tivoli OMEGAMON XE for Distributed Systems enables you to see and manage your entire distributed enterprise from a single point of control.

IBM Tivoli Provisioning Manager

IBM Tivoli Provisioning Manager automates manual tasks of provisioning and configuring servers and virtual servers, operating systems, middleware, applications, storage, and network devices acting as routers, switches, firewalls, and load balancers.

IBM Tivoli System Automation for Multiplatforms

IBM Tivoli System Automation for Multiplatforms manages the availability of business applications and middleware running on single Linux and AIX systems or clusters on IBM zSeries®, pSeries®, iSeries, and xSeries®, or other Intel-based servers, according to customer-defined goals.

IBM Tivoli Universal Agent

IBM Tivoli Universal Agent collects information via numerous data providers including SNMP, ODBC, and FILE to monitor almost any device or application connected to a TCP/IP network. IBM Tivoli OMEGAMON solutions can then reveal consolidated views of performance and availability to help you diagnose and pinpoint problems more quickly.

WebSphere Application Server

The core of the WebSphere portfolio, this product is the industry's leading J2EE and Web services application server, delivering a high-performance and extremely scalable transaction engine for dynamic e-business applications.

WebSphere Application Server Network Deployment

WebSphere Application Server Network Deployment provides an operating environment with advanced performance and availability capabilities in support of dynamic application environments. In addition to all of the features and functions within the base WebSphere Application Server, this configuration delivers advanced deployment services that include clustering, edge-of-network services, Web services enhancements, and high availability for distributed configurations.

WebSphere Extended Deployment

WebSphere Extended Deployment, together with WebSphere Application Server Network Deployment, delivers a high-performance, easily manageable, and dynamically scalable environment for distributed WebSphere applications that leverages the principles and concepts of proven IBM systems. It provides:

- WebSphere resource virtualization and pooling using node groups and dynamic clusters
- Dynamic adjustment of WebSphere resources through application placement
- Integration with Tivoli Intelligent Orchestrator (optional, available separately) for enterprise-wide autonomic provisioning
- Introduction of operational policies to distributed WebSphere environments and intelligent routing and dynamic workload management according to established goals

IBM WebSphere MQ

IBM WebSphere MQ V6.0 delivers improved ease of use and manageability to provide a flexible and proven foundation for your enterprise service bus (ESB).

WebSphere Studio Application Monitor

WebSphere Studio Application Monitor helps improve application availability and performance by providing deep-dive real-time problem detection, analysis, and repair. Diagnostics at the method level can pinpoint code problems, which can help an architect or developer quickly fix a problem.

IBM Director

IBM Director is the industry-leading client/server workgroup manager. IBM Director tools provide customers with flexible capabilities to realize maximum system availability and lower IT costs. With IBM Director, IT administrators can view and track the hardware configuration of remote systems in detail and monitor the usage and performance of critical components, such as processors, disks, and memory.

IBM Remote Deployment Manager

Remote Deployment Manager (RDM) facilitates remote deployment of both IBM and non-IBM systems. RDM allows for remote unattended installation of new and existing systems. RDM helps automate deployment tasks such as initial operating system installation, BIOS updates, and disposal of retired systems. All of these tasks can be done without visiting the remote system, reducing travel and labor costs.

IBM ServerGuide

IBM ServerGuide™ is a tool that simplifies the process of installing and configuring IBM @server™ and IBM @server xSeries servers. ServerGuide goes beyond mere hardware configuration by assisting with the automated installation of Windows server operating systems, device drivers, and other system components, with minimal user intervention.

IBM Virtual Machine Manager

IBM Virtual Machine Manager (VMM) is an extension to IBM Director that allows you to manage both physical and virtual machines from a single console. With VMM, you can manage both VMware ESX Server and Microsoft® Virtual Server

environments using IBM Director. VMM also integrates VMware VirtualCenter and IBM Director for advanced virtual machine management.

Cluster Systems Management

Cluster Systems Management (CSM) for AIX and Linux is designed for simple, low-cost management of distributed and clustered IBM @server pSeries and xSeries servers in technical and commercial computing environments.

Parallel ESSL

Parallel ESSL is a scalable mathematical subroutine library that supports parallel processing applications on the IBM RS/6000® SP Systems and clusters of IBM pSeries and RS/6000 workstations.

LoadLeveler

LoadLeveler® manages both serial and parallel jobs over a cluster of servers. This distributed environment consists of a pool of machines or servers, often referred to as a LoadLeveler cluster.

General Parallel File System

The IBM General Parallel File System (GPFS) is a high-performance shared-disk file system that can provide fast, reliable data access from all nodes in a homogenous or heterogeneous cluster of IBM UNIX servers running either the AIX 5L™ or the Linux operating system.

B

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246778

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246788.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
DemoGridApp.zip	A zip file including the source files and other supporting files required for the sample application described in Chapter 12, “Demonstration application” on page 197
GT4SampInst.zip	A zip file containing a sample script that we used to quickly install Java, Ant, and Globus Toolkit 4.0

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space	1 MB minimum.
Operating System	Any OS supporting Java environment. Our examples are based on Linux.

How to use the Web material

GT4SampInst.zip

The content of this zip file is a sample bash shell script that we used to install and reinstall our grid nodes whenever we needed to rebuild our environment. It is customized for our environment and assumes specific host name and IP addresses for NFS shares that contain the installation images for Java, Ant, and Globus Toolkit 4.0. We do not provide any specific instructions on modifying this for your environment, but thought you might find it useful if you want to automate the install task once you have done it a few times manually. Chapter 11, “Globus Toolkit 4 installation and configuration” on page 155, provides the step-by-step instructions for manually installing an environment similar to ours.

Extract the shell script (gt4_install.sh) from the GT4SampInst.zip file to an environment that supports the bash shell. Edit this file to meet your specific requirements.

DemoGridApp.zip

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. To use the files and application, follow the directions provided below in conjunction with detailed information about the application provided in Chapter 12, “Demonstration application” on page 197.

This application was developed and tested in the environment as described in Chapter 11, “Globus Toolkit 4 installation and configuration” on page 155.

Important: The demonstration grid application described in this book and available for download as described above utilizes the open source Batik toolkit available from Apache.org. See:

http://xml.apache.org/batik/

This toolkit provides libraries of functions to handle and manipulate SVG files. Before building and running our sample application, you should obtain the Batik toolkit from the Apache Web site referenced above. Specifically, the following jar files need to be available and specified in your CLASSPATH environment variable to compile and execute our sample application.

- batik-awt-util.jar
- batik-bridge.jar
- batik-css.jar
- batik-dom.jar
- batik-ext.jar
- batik-gvt.jar
- batik-parser.jar
- batik-rasterizer.jar
- batik-script.jar
- batik-svg-dom.jar
- batik-transcoder.jar
- batik-util.jar
- batik-xml.jar
- batik.jar

In addition, the Batik package includes sample SVG files that can be used to test and run this application. The two samples we use for testing are:

- mapSpain.svg
- tiger.svg

Note that all scripts follow the installation paths shown earlier in the book. If you chose different install paths you will have to adjust the scripts.

To perform the source code build process

To do this:

1. Run the buildservice script. This must be done first to generate the services binding files required for the rest of the sample application, and build and package the RenderSourceService code. This must build cleanly to continue the process.

2. Run the buildworker script. This builds and packages the RenderWorker code.
3. Run the buildclient script. This builds and packages the RenderClient application.

To perform the code deployment process

To do this:

1. Ensure the Globus container running the RenderSourceService is stopped. You can do this by typing `^C` or closing the terminal window that the container is running in.
2. Run the undeployservice script. This will uninstall a previously installed version of the RenderSourceService from your local system's Globus container. This allows for a clean install of a new version of the service in the next step.
3. Run the deployservice script. This installs the previously built RenderSourceService into your local system's Globus container. This service should be installed into a Globus container on only one machine on the network. The client application defaults to this service running on the same machine as the client, so if you choose a different machine you will have to adjust the client's entry field to point to the proper machine.

To bring up the system for grid processing

To do this:

1. Run the startcontainer script in its own terminal window. You must do this for every node that you want to participate in the grid or host the RenderSourceService. This launches the Globus container, making it available for dispatch of RenderWorker processes, and on one machine makes the RenderSourceService ready for work. Globus will take some time to launch and will produce many status messages. These are the key long-running process for the operation of the grid, so only shut down the container when you do not expect to submit any work to that particular node, or if you need to deploy a new version of the service.
2. Optionally, run the runworker script. This allows you to perform stand-alone testing of a RenderWorker process against a running RenderSourceService, without the overhead of the full GUI client.
3. Run the runclient script. This launches the GUI RenderClient application and lets you test the rendering system on your running grid.

.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 238. Note that some of the documents referenced here may be available in softcopy only.

- *Introduction to Grid Computing with Globus*, SG24-6895
- *Grid Computing: Solution Briefs*, REDP-3891
- *Grid Computing Products and Services*, SG24-6650
- *Grid Computing in Research and Education*, SG24-6649
- *Grid Computing with the IBM Grid Toolbox*, SG24-6332
- *Grid Services Programming and Application Enablement*, SG24-6100
- *Globus Toolkit 3.0 Quick Start*, REDP-3697
- *Enabling Applications for Grid Computing with Globus*, SG24-6936
- *Fundamentals of Grid Computing*, REDP-3613

Other publications

These publications are also relevant as further information sources:

- J. Joseph, M. Ernest, and C. Fellenstein, *Evolution of grid computing architecture and grid adoption models*, IBM Systems Journal Vol 43, No 4, 2004.
- M. Baker, A. Apon, C. Ferner, and J. Brown, *Emerging Grid Standards*, page. 43-50, IEEE Computer, April 2005.
- I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana, *Modelling Stateful Resources with Web Services*, http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf, March 2004.

- K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution*, http://www.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf, March 2004.
- K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, *The WS-Resource Framework*, http://www.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf, March 2004.
- S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, *A Grid Application Framework based on Web Services Specifications and Practices*, http://www.neresc.ac.uk/ws-gaf/A%20Grid%20Application%20Framework%20based%20on%20Web%20Services%20Specifications%20and%20Practices%20v1.0.pdf, 2003.
- I. Foster, *A Globus Primer*, http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf, May 2005.
- I. Foster, C. Kesselman, *The Grid: Blueprint for a new Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, CA, 1998.
- I. Foster, C. Kesselman, and S. Tuecke, *The Anatomy of the Grid-Enabling Scalable Virtual Organizations*, The Globus Alliance, http://www.globus.org/research/papers/anatomy.pdf.
- I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell and J. Von Reich, *The Open Grid Services Architecture, Version 1.0*, http://forge.gridforum.org/projects/ogsa-wg, January 2005.
- S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt, *Open Grid Services Infrastructure (OGSI) Version 1.0*, Global Grid Forum, http://www.ggf.org, June 2003.
- *Open Grid Service Infrastructure Primer*, Global Grid Forum, http://www.ggf.org, August, 2004.
-
- N. Nagaratnam, P. Janson, J. Dayka, A. Nadalin, F. Siebenlist, V. Welch, I. Foster, S. Tuecke, *Security Architecture for Open Grid Services,* http://www.cs.virginia.edu/~humphrey/ogsa-sec-wg/OGSA-SecArch-v1-07192002.pdf.

Online resources

These Web sites and URLs are also relevant as further information sources:

- Apache Ant Web page

  http://ant.apache.org/
- Apache Batik

  http://xml.apache.org/batik/
- Apache WSRF tutorial

  http://ws.apache.org/ws-fx/wsrf/tutorial/
- Distributed Management Task Force (DMTF)

  http://www.dmtf.org/
- Global Grid Forum (GGF)

  http://www.ggf.org
- Globus

  http://www.globus.org
- Globus Toolkit 4 Programmer's Tutorial by Borja Sotomayor

  http://gdp.globus.org/gt4-tutorial/
- Globus WSRF

  http://www.globus.org/wsrf/
- GridFTP

  http://www.globus.org/grid_software/data/gridftp.php
- Internet Engineering Task Forfe

  http://www.ietf.org/
- Open Grid Services Architecture (OGSA)

  http://www.globus.org/ogsa/
- OGSA-DAI

  http://www.ogsadai.org.uk/
- Open Grid Services Interface (OGSI)

  http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf
- Organization for the Advancement of Structured Information Standards (OASIS)

  http://www.oasis-open.org/

- OASIS Web Services Resource Framework

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
- pyGridWare

  http://dsd.lbl.gov/gtg/projects/pyGridWare/
- Scalable Vector Graphics specification

  http://www.w3.org/TR/SVG
- Understanding WSRF Parts 1 to 4 by Babu Sundaram

  http://www.ibm.com/developerworks
- Using Eclipse to develop Grid services

  http://www.ibm.com/developerworks/edu/gr-dw-gr-eclipseide-i.html
- Web Services Activity

  http://www.w3.org/2002/ws/
- Web Services Interoerability

  http://www.ws-i.org/
- Web Services Interoperability Organization (WS-I)

  http://www.ws-i.org/
- World Wide Web Consortium (W3C)

  http://www.w3.org/
- *WS-Resource Framework Interop Workshop #1 - Scenarios (v0.13)*

  http://www.ibm.com/developerworks/offers/WS-Specworkshops/ws-rf200404.html
- WSRF.NET Developer Tutorial by Mark Morgan and Glenn Wasson

  http://www.cs.virginia.edu/~gsw2c/WSRFdotNet/WSRF.NET_Developer_Tutorial.pdf

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

H

I

J

K

L

M

N

S

T

U

V

W

X

IBM

Redbooks

Introduction to Grid Computing

Introduction to Grid Computing

Learn grid computing basics

Understand architectural considerations

Create and demonstrate a grid environment

In the past several years, grid computing has emerged as a way to harness and take advantage of computing resources across geographies and organizations. In this IBM Redbook, we describe a generalized view of grid computing including concepts, standards, and ways in which grid computing can provide business value to your organization. In a nutshell, grid computing is all about virtualization that enables businesses to take advantage of a variety of IT resources in order to be more responsive to demands of the business and increase availability of applications while reducing both infrastructure and management costs.

There are many products and toolkits available from IBM and other companies that enable different aspects of grid computing. One of the most well known toolkits is the Globus Toolkit. Globus Toolkit 4 provides components and services conforming to existing and evolving standards that can be used as the basis for a grid computing solution. In the second half of this book we provide instructions for installing and configuring a simple Globus environment that can be used to demonstrate various aspects of grid computing and to build a proof of concept environment. We also describe, and provide as additional material, a sample grid application that can be used to demonstrate, test, and teach more about the grid computing concepts introduced in this book.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6778-00 ISBN 0738494003