CONTRIBUTIONS TO THE DEVELOPMENT OF A NATIONAL GRID INTRASTRUCTURE FOR E-SCIENCE

A DISSERTATION
SUBMITTED TO THE SCHOOL OF COMPUTER SCIENCE
OF THE UNIVERSITY OF ADELAIDE
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

THE UNIVERSITY
of ADELAIDE

By

Shunde Zhang

July 10, 2012

Table of Content

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

e-Science is a terminology denoting modern scientific experiments and studies being carried out with the support of large-scale Grid infrastructures. The essence of a Grid is to enable resource sharing, including compute resources and storage resources, to as many authorised people as possible. This thesis has made several contributions towards building a national grid infrastructure, by designing and implementing new approaches to simplify the use of the Grid so as to enlarge the user base of grid infrastructure and e-Science. These new systems have been deployed as part of the Australian national grid infrastructure, however the approaches used aim to provide a generic solution so that other grid operators and users can also benefit from it.

Our exploration of existing data grid systems has shown that these are not easy for researchers to use, since they require users to have certain IT knowledge, and rarely have a user-friendly interface. My approach is to develop a web portal based upon a widely-used data grid system, iRODS, that is able to make use of geographically distributed storage resources. This new interface not only supports the WebDAV standard, enabling easy drag-and-drop file access, but also provides a web interface, allowing users to share and manage data with a web browser.

Data transfer is a challenge when dealing with large volumes of data and long distances, which leads to problems with stability, reliability and performance. Existing data repositories and data transfer services deliver necessary functionality, but only support a limited number of protocols, which can cause problems with interoperability.

Rather than developing a new data transfer service or modifying current services to support new protocols, my approach focuses on equipping an arbitrary data source with a standard GridFTP interface, so that it can interact with most of the existing data transfer services and grid services. This thesis gave a detailed description of my architecture and evaluation, and demonstrated that this approach adds virtually no overhead to the data source but gives it more flexibility in data transfer.

Compute job submission usually requires users to have a significant level of understanding of the Grid, such as its structure and the usage of its client tools, especially when users are exposed to a complex grid infrastructure with multiple resources. The client tools and interfaces are not easy to use or to develop custom applications. My approach addresses this problem by providing a web portal with a RESTful interface to simplify job submission to multiple grid resources. The RESTful interface also makes it possible for users and application developers to submit massive jobs in a simple way. The portal has a template system to enable quick and easy development of customiszd interfaces to applications running on grid compute resources. The portal therefore provides a generic solution to users across various research domains.

DECLARATION

I, Shunde Zhang certify that this work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

The author acknowledges that copyright of published works contained within this thesis (as listed below*) resides with the copyright holder(s) of those works. I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library catalogue and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

_____

Shunde Zhang

PhD Candidate,
School of Computer Science
University of Adelaide
10 July 2012

LIST OF PUBLICATIONS

The following papers were written based on the work presented in this thesis.

Towards an interoperable International Lattice Datagrid. P. Coddington and et al., in Proceedings of XXV International Symposium on Lattice Field Theory, Regensburg, 2007, Proceedings of Science (LATTICE 2007)

Experiences in Developing a Node of an International Computational Physics Data Grid. P. Coddington and et al., in Proceedings of the Sixth Australasian Workshop on Grid Computing and e-research, Wollongong, 2008

Davis: A Generic Interface for iRODS and SRB. Shunde Zhang, Paul Coddington, Andrew Wendelborn. In proceedings of Grid Computing 2009, Banff, Alberta, Canada. October 2009.

Connecting arbitrary data resources to the grid. Shunde Zhang, Paul Coddington and Andrew Wendelborn. In proceedings of Grid Computing 2010, Brussels, Belgium. Oct 2010.

A national grid submission gateway for eScience. Shunde Zhang, Paul Coddington, Andrew Wendelborn, in proceedings of the 7th IEEE International Conference on e-Science (e-Science 2011), Stockholm, Sweden, December 2011.

ACKNOWLEDGEMENT

First and foremost, I am deeply indebted to my supervisors, Dr. Paul Coddington and Dr. Andrew Wendelborn, for their advice, guidance and supervision throughout my candidature. Their insight and breadth of knowledge is of great help in every stage of my research from problem identification to paper publications. Their enthusiasm in research encouraged me to keep my mind open and pushed me to the next level of achievement. I would like to express my sincere gratitude to both of them. My work would not be completed without their tremendous effort.

I would also like to thank my colleagues and ex-colleagues in eResearch SA and the Australian Research Collaboration Service, such as Daniel Cox, Florian Goessmann, Pauline Mak, Graham Jenkins, Rowan McKenzie, Sam Morrison, Vladimir Mencl, Simon Yin, Markus Binsteiner, Darran Carey and Jim McGovern, who have given me invaluable feedback and advice to my work, and great help in the development and evaluation of my systems. Especially I would like to thank some international peers, such as Reagan Moore, Mike Conway, Wayne Schroeder and Guy Kloss, for exchanging ideas, thoughts and experiences.

Last but never the least, I would like to thank my family for their love, encouragement and continual support at all times. They gave me optimism and happiness so I can finish this long journey.

Chapter One: Introduction

A national grid infrastructure plays an important role in modern research. It facilities various IT services to aid researchers in data process, sharing and storage. The construction of a national grid infrastructure is not trivial; it involves lots of money and resources. Although there are many successful stories around the world, the situation is always different when building your own. This thesis focuses on my contributions to building a national grid infrastructure. It describes in detail the architecture of a national grid infrastructure, as well as the components I developed to make it easier to use, and hence available to a wider range of researchers, with more or less IT expertise, so they can all benefit from it.

This chapter firstly introduces eScience, the Grid and their relationship, then the challenges in building a national grid. It follows by a summary of my work in this project, and ends with an overview of this thesis.

1.1 eScience and the Grid

Modern science heavily relies on data. As technologies evolve, especially electronic sensors and measurers, data collected and generated is getting more accurate and fine grained, hence larger and larger. One example is the Large Hadron Collider (LHC) [1], which generates 1TB of data every day. The analysis of massive amounts of data is not possible using traditional methods, such as spreadsheets or pen and paper. Like other domains, science needs the help of modern Information Technology (IT).

Therefore, the term e-Science was first introduced by Dr. John Taylor to mark the combination of IT and the study of science. It brings scientists a new revolutionary means of doing their daily research with heavy use of IT resources. As addressed by Hey et al. [2], e-Science is backed by a certain large-scale and distributed IT infrastructure, which is generally referred to as the *Grid* [3].

Since the late 1990s, a lot of effort has been spent on developing and improving various grid infrastructures. The grid arose to solve big problems in research disciplines as diverse as high energy physics, bioinformatics, oceanography and earth science. When there is less data to process, store or share with other researchers, a single desktop computer or a portable hard drive is enough. But if the volume of data is massive or data processing is time-consuming, it is essential to run the processing program in parallel, using several computers to reduce overall processing time. High Performance Computer (HPC) systems, or supercomputers, are commonly used as a shared resource for a group of researchers, in a research institute, or a university. Authorized users, such as employees of the organization that owns the supercomputer, have access to the supercomputer, and share it by using a local resource management system (LRMS), which allocates compute resources to users based on pre-defined rules and user requirements, to ensure that the HPC resources are shared fairly among the users.

Resource sharing is also important for data storage. Normally, a hierarchical storage system exists alongside an HPC system. This storage system, generally set up with a fast disk array and a robotic tertiary storage device, has a massive amount of space and has been configured with a proper backup strategy. Users can use the storage up to a limit that is specified by a quota mechanism. A common use case is that a user places

input data in the storage system and then submits a job to a supercomputer; then the supercomputer reads the input data, processes it, and writes output data to the same place, from where the user can retrieve result data. This solution is the standard form of supercomputing, which is suitable for a single organization, and is still the main usage scenario for small groups today.

However, it brings up issues when a user wants to use resources from different organizations, or users from different organizations want to share resources among them. Foster and Kesselman were among the first to look at these issues [4, 5, 6] and developed the concept and architecture of *Grid*, aiming to create a federation for several trusted parties so that every organization is able to contribute their resources to the federation and make use of remote resources if they are allowed to. Around the concept, software stacks and standard specifications were developed. The famous grid middleware, *Globus Toolkit* [7], developed by Foster and Kesselman in late 1990s, provides a standard interface to heterogeneous resources consisting of different operating systems, LRMS and storage systems. Designed as a service-oriented software system [8], it acts as the foundation of a number of grid service providers in the US and Europe, with users from many countries. On the other hand, some grid providers use a modified version of the Globus Middleware, such as gLite [9], or completely different middleware, such as Unicore [10] and Condor [11]. To standardize the interface and behaviour of each grid system, so that they can communicate with each other, the Open Grid Forum (OGF) [12], formerly the Global Grid Forum (GGF), was formed, with the intention to define standards, and with inputs from international user communities. To date, over 150 specification documents have been generated, ranging from storage systems to

computation systems, including ones as low level as network topology, and as high level as programming interfaces. Wide adoption ensures interoperability. For instance, GridFTP protocol [13] is a de facto standard for data transfer in the grid community. A large number of GridFTP-compatible data servers have been deployed across many countries. This makes it easy to transfer data among them, with the use of a compatible GridFTP client, or a data transfer service, such as Globus Online [14].

1.2 The Challenges of using the Grid

Over the past few years, much effort has been put into building grid systems and making the grid simpler and easier to use, for both data grid [15] (focusing on data storage, access and sharing in a distributed environment) and computational grid [5] (focusing on providing uniform interfaces to supercomputers). Early developments were focused on middleware, such as Globus Toolkit, gLite [16], UNICORE [10] being computational grid middleware, and SRB [17]/iRODS [18], dCache [19] being data grid systems. Discipline-specific and regional grid infrastructures have been implemented, such as the LCG project [20] for LHC experiments, and the TeraGrid project [21] for US universities. These systems are very successful in their own areas, and intend to make their system generic for other people to use. However, it is still difficult to copy the entire system to another environment and it does not always give the best outcomes, because their design is more or less aligned with their local requirements. In addition, it is easy to make a middleware generic but not so easy to make a user interface generic. What can be extracted from the existing systems are mostly service components. Their user interfaces

are mostly specific to their users and hard to migrate. Nevertheless, the experience gained from building these grid systems is very important for people who want to set up a new grid infrastructure.

Although a number of grid systems are running around the world, we see several issues when using the grid. From a user's point of view, the grid is somewhat complex. Before a user can use it, there are a number of technical hurdles to overcome, such as the knowledge of basic Linux commands, being able to write scripts, the understanding of the setup of the grid, the use of X.509 certificates, and so on. Ideally, users usually want to be presented a simple and easy-to-use web-based system, where they can do a variety of things, without worrying about the installation of different client applications. From the grid service provider's point of view, they always want to increase the usage of their system, by sharing the resources to more users, and making it easier to more users to get access. They also want their users to be able to use remote resources if no local resource is available. From the application developer's point of view, a simple API to access all resources is what they need. The APIs provided by current grid middleware are far more complicated and needs a fair bit more time to learn and understand, compared to the APIs offered by commercial systems, such as Yahoo and Google. This stops third-party developers from building user-customized grid application on top of the basic infrastructure. All in all, it is not easy to build an easy-to-use grid system, due to the lack of a user-friendly web-based interface and the inflexibility of adopting applications developed by other groups, thus hard to share resources to a broader group of users.

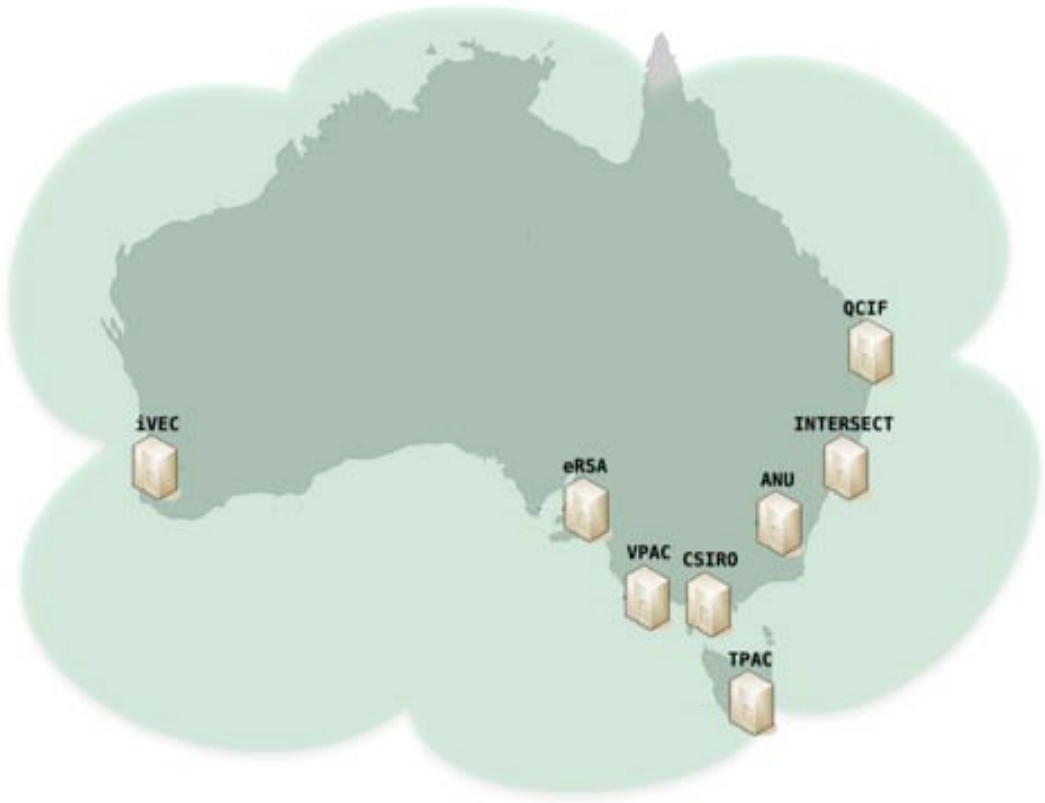

Figure 1-1 Regional Grid Operators in Australia

Australia has several regional HPC centres in major cities, as shown in Figure 1-1. In the early 21st century a federal government project, known as APAC, started to build a basic national grid infrastructure, including system-level grid middleware on top of the existing HPC systems, and central storage systems based on tape devices or disk arrays, with a collection of software that is installed in these HPC systems and used by multiple research disciplines. Existing hardware and software are the essential foundation of a working grid system [2], and indeed, the APAC project has built a working grid system that a number of Australian researchers have taken advantage of in their research. On the

other hand, it is not very user-friendly and thus, users need to have certain IT knowledge and get over a number of technical hurdles before they can benefit from it. After APAC finished, a subsequent project, known as ARCS, was funded by the government, as described in this thesis. ARCS aims to provide further user support on existing services, and to establish a more advanced nation-wide service federation, to aggregate distributed grid resources to be a uniform platform so as to lower the barriers to using the grid, make it simpler and more user-friendly for more researchers to use, and make good use of resources across the country. To achieve this, there are a few challenges to tackle.

Firstly, as users are from different organizations, they normally hold a credential from their home organization. To give them access to a grid service, the Globus toolkit implements the Grid Security Infrastructure (GSI) specification [22], which is based on X.509 certificate technology. Any individual user will be given a certificate to identify them to the grid systems. However, this usually requires quite a lot of effort from the user and the issuer of the certificate, and has a long learning curve because it requires users to remember a different set of username and password, and to learn how to use certificates. Secondly, although each regional grid operator has installed Globus Toolkit as the grid submission gateway for HPC systems, it only offers a web service interface, and requires users to install client software, write a Globus-defined job description file, and use command-line interface tools in order to successfully send a job to the backend HPC systems. It is common for a modern system to provide a web portal to remove the above hassles, and give access to all backend resources. Lastly, storage facilities are available at each grid site, but different sites may provide different interfaces in various protocols. In addition, it is not easy to share data with someone from another organization, since

normally only local users with a local account can access the storage area, and external users can hardly get into the storage system, let alone retrieve the files. This thesis, therefore, talks about how I designed and developed additional components on a national grid infrastructure to tackle these challenges, and provide an easy-to-use one-stop shop for researchers in Australia to take advantages of the existing IT facilities.

1.3 A summary of my work

My research interests lie in the study of different grid technologies, applications and systems. I am also interested in the combination of grid systems and web-based systems. In this project, my main goal is to make the Grid easier to use by users who don't have a lot of IT expertise, and easier for developers to create grid applications. In summary, my contributions are three components in the grid infrastructure. Firstly, I investigated several data grid systems and clustered file systems, and made a significant contribution in the selection of data grid middleware that is used as the foundation of the data grid system in this project; I also evaluated all existing user interfaces of such middleware and identified the missing interface, a web-based interface, which was then developed to benefit both end-users and developers. Secondly, I studied different data transfer solutions and services, to understand their pros and cons. The outcome of this study showed that implementing a data transfer service with multi-protocol support would be complex and messy, and it may reduce the transfer performance too if all data goes through the central data transfer service. To simplify the system structure, I developed a standard GridFTP interface in order to connect arbitrary data sources into the

grid world; therefore users can make use of existing high-performance grid data transfer services to move data. Lastly, in order to make it easier for users to use existing grid computing resources, I developed a web-based grid job submission gateway, with integration of cutting-edge technologies, such as Shibboleth [23], to enable submission of various types of jobs and a huge amount of jobs. After submission, the gateway can re-distribute these jobs to various grid resources to process, based on pre-defined rules. These components can work independently or together. For users who want to store and share data, they only need to use the data portal of the data grid middleware. The standard GridFTP interface is the bridge between the data grid middleware and the computational grid. Input data and result data of grid jobs are stored in the data grid middleware so this data can be kept safely, shared to other researchers and backed up properly. Thus, for users who want to run jobs in the backend HPC systems and share the result, they can upload input data to the data grid via the data portal, and submit the job using the grid job submission gateway, and then examine and share the result data with the data portal again.

1.4 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 presents the concept of the Grid in detail and gives background information, such as the major grid projects and production grid systems in the world. Then, Chapter 3 describes the architecture of a national grid infrastructure using the example of the Australian grid, and explains the motivations for this project. Chapter 4 focuses on my development of a data grid, which

aims to be a national file system that aggregates storage resources across the country and allows researchers to use it as a seamless national data fabric. It starts with the investigation of a data grid middleware system and identification of missing capability, then leads to the architecture of *Davis*, a data portal to the backend storage. Chapter 5 describes *Griffin*, a GridFTP interface for data transfer. It can interface with any arbitrary data source and connect this data source to the grid, so a grid data transfer service can be used to move data among users, grid compute resources and data storage systems. It is then followed by Chapter 6, which talks about a different topic, a grid job submission portal and how it interacts with the existing grid middleware in each regional grid service provider. Finally, the thesis concludes and gives ideas for future work in Chapter 7.

Chapter Two: Background and Literature Review

This chapter presents an introduction to eScience, and an in-depth view of the *Grid*, the technology that has been supporting eScience since 1990s. Definition and characteristics of eScience will be discussed next, with detailed descriptions of its two major forms, data grid and computational grid. Data transfer services and authentication mechanisms in the Grid are also depicted. Then, a few examples of eScience projects are presented, and followed by a brief outline of a cutting-edge technology, the Cloud.

2.1 eScience

The way of doing science has been incredibly changed from a few decades ago. Along with the evolvement of IT technology, scientists more and more rely on computers, in every aspect of their work. Nowadays, each individual scientist has a desktop computer, which is used everyday to write papers, send emails and search other research works. More importantly, a large number of researchers use computers to run simulations, store research data and process instrument-generated data. One example is the Large Hadron Collider (LHC) [1], a high-energy particle accelerator constructed by CERN, lying in a tunnel beneath the border of France and Switzerland. It produces about 15 Petabytes of data annually, with thousands of scientists from hundreds of research institutions around the world involved in the study of experimental results. The analysis and processing of data cannot be completed without the help of supercomputers, while

the storage and movement of this data cannot be done without the help of massive storage facilities and fast network links.

The term *eScience* was first introduced by John Taylor, Director General of the Research Councils of UK [24]:

"*e-Science is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it.*"

This definition is intended to summarize a new era of scientific activities, far beyond the change that Internet has brought to the general public. Its main focus is global collaboration of researchers worldwide, who share not only resources but also data and ideas. Figure 2-1 [25] illustrates the scientific lifecycle supported by eScience: the inner grey circle shows the stages of a typical scientific research progress, while the outer circle surrounding it shows the IT facilities that get involved in scientific activities. The technology enabling eScience consists of compute power, storage, networking, video processing and transmission, and so on. For instance, researchers in different countries have videoconferences regularly to exchange ideas, share research outcomes or do remote presentations. This is one of the basic and simplest forms of collaboration in eScience. This thesis concentrates on a more complex type of collaboration, namely resource sharing, including CPU power sharing and data sharing. This is backed by an advanced technology, the *Grid* technology, which appeared from the late 1990s. The next section will look at the history and features of the *Grid*.

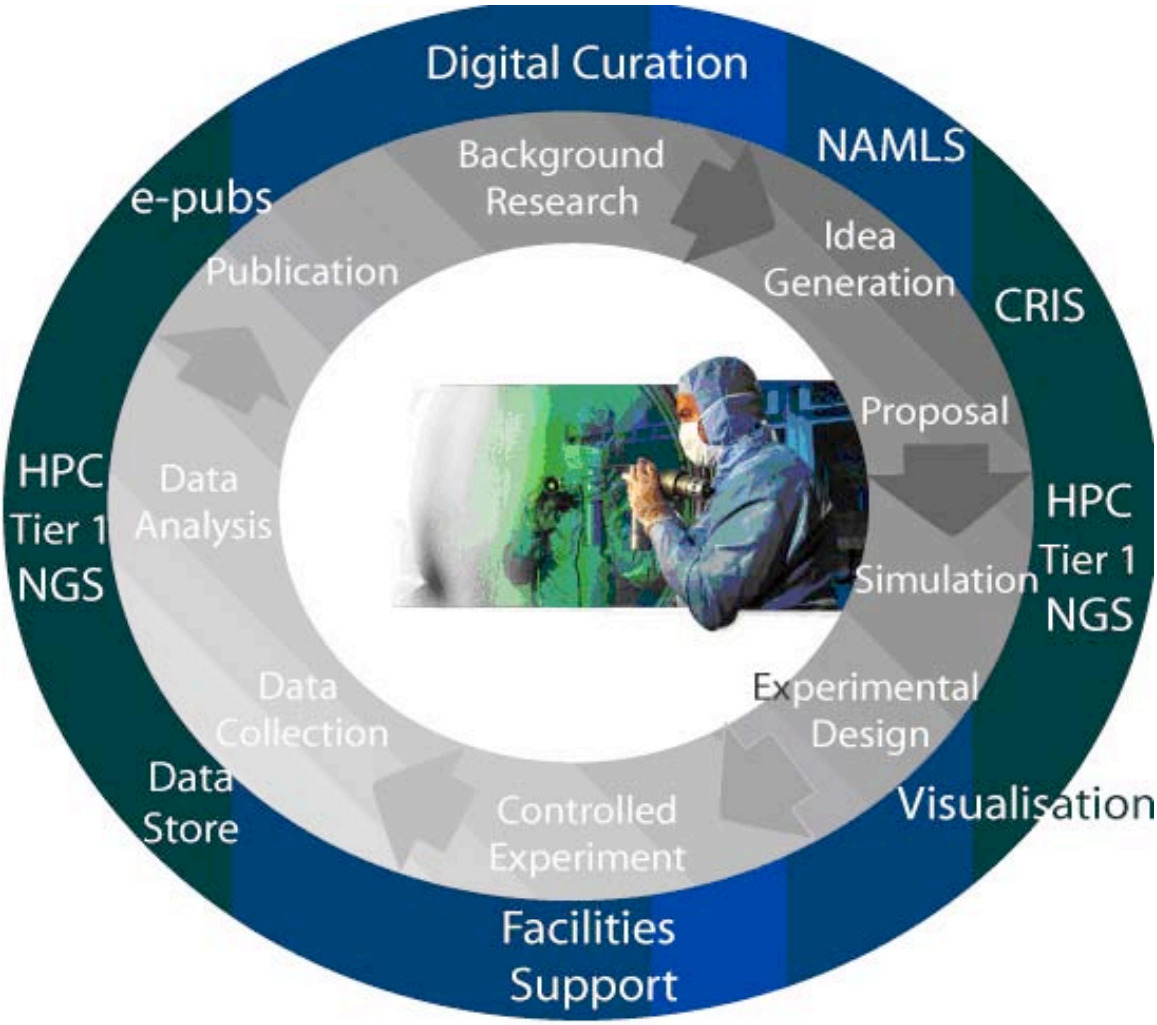

Figure 2-1 eScience supports the complete scientific lifecycle

2.2 The Grid

The concept of Grid appeared in late 1980s and early 1990s. Ian Foster and Carl Kesselman were among the first to propose the concept of the Grid [3], which has led to the standardized Grid systems that are widely used nowadays, and thus Ian Foster is known as the father of the Grid. The purpose of having a *Grid* is sharing resources, just

like anyone can use electricity from the power grid if needed. Before getting into any details of the Grid, it is necessary to describe why the Grid is needed, from a technical perspective.

The design of modern computers restricts hardware parts that can be placed into one single physical machine. Typically, one machine can only have limited number of central processing units (CPUs), from several up to tens of Gigabytes of Random-access memory (RAM), some hard disks and several network interfaces, etc. This configuration is sufficient for personal use, or shared by a limited number of users to run simple tasks. When it comes to complex tasks or multi-tasking, such as CPU-intensive tasks and memory-intensive tasks, one physical machine is not enough. At this time, a cluster is used.

A computer cluster is a group of computers, usually connected with each other through fast local area networks, working together and closely as one single computer. A cluster is often deployed to achieve load balance, high performance and high availability. In academic research, clusters are primarily used for high performance computing, such as computational simulation of climate change and analysis of experimental results. They are normally called supercomputers, and provide compute resources in a Grid environment. A supercomputer normally has homogeneous machines as its nodes, and a central control computer, known as the head node. A popular programming model for supercomputers is Message Passing Interface (MPI), which enables programs to be run in parallel across multiple nodes of a supercomputer. There are a number of resource management systems for users and system administrators to interact with supercomputers, such as Sun Grid Engine [26], the Portable Batch System (PBS) [27]

and Platform's Load Sharing Facility (LSF) [28]. These systems are usually local, available to users with a local account, so they are normally called LRMSs (Local Resource Management Systems), or batch systems.

To remove the hurdles of using multiple remote and heterogeneous supercomputers, the concept of Grid has been developed to share heterogeneous resources across different administrative domains. Figure 2-2 shows the evolution from a single computer to the Grid.

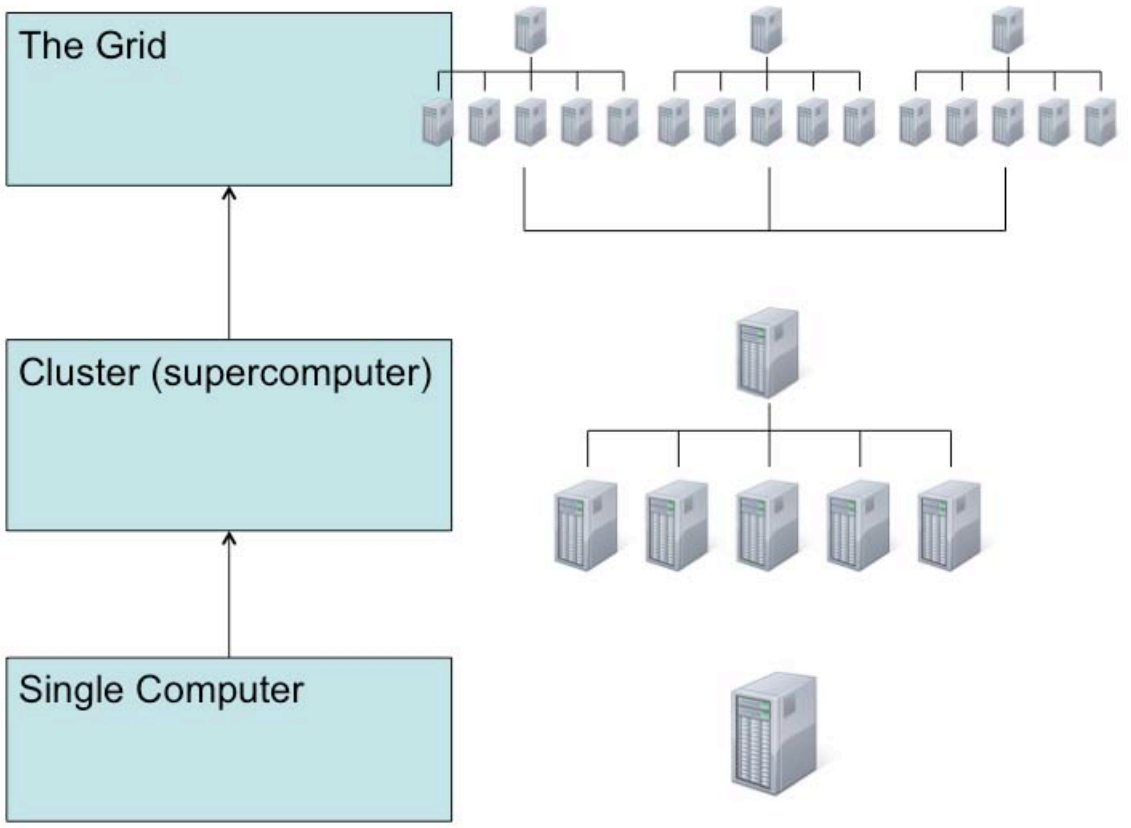

Figure 2-2 From a single computer to the Grid

2.2.1 The Definition

There has been a lot of debate about what is a grid. To briefly outline the features of a grid, Ian Foster presents a checklist with three points [29]. Firstly, a Grid aggregates and coordinates resources from different administrative domains, such as clusters from several organizations and computers in different universities, which raises authentication, authorization and other policy issues when dealing with these distributed systems. This is in contrast to the local management systems. Secondly, resources in distributed systems like the Grid are usually heterogeneous, and since there are lots of them, we need open standards in order for them to be interoperable, and for the development of grid applications that work across them, otherwise it is hard for remote users to utilize the grid resources. Grid services, such as authentication, authorization, resource discovery and resource access, should be standardized to provide a uniform interface to underlying resources. Thirdly, users should be able to specify and achieve nontrivial qualities of services from a Grid, addressing performance, availability, security, etc. A grid consisting of a number of hardware components should offer better utility than the sum of each single one.

On the other hand, Buyya and Venugopal defined grid as "a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements" [30]. They tried to address the following issues: enable access to distributed resources while preserving complete control over local resources; improve availability of data and provide solutions

to data access; provide a uniform and user-friendly interface to users so they can make use of physically distributed resources.

These definitions provide a high-level abstraction of a Grid. From the usage, grid systems can be categorized into two groups, data grid and computational grid. In essence, all grid systems share some basic, standard services and some common features. The next section will depict these features in detail.

2.2.2 Features

Every grid system, either data-centric, or compute-centric, has some features in common. In all definitions, a grid is all about resource sharing, which could happen between organizations in one country, or between countries. The resources that can be shared include data, software, or compute power. On the other hand, it is necessary to ensure that the resources are shared to the person that is trusted. The concept of Virtual Organization (VO) [4] is created to solve this problem. A VO consists of a group of users that have the same goal, or work on the same project, so that they share their resources with each other, and have access to resources that are necessary to finish their work. In a real grid system, users are grouped into different VOs, so that the sharing can be managed and controlled at a fine-grain level.

Secure access to a grid system is vital and challenging. As grid resources are usually expensive facilities, it should be assured that users are valid and allowed by the policy when they request to access a resource. This raises several issues including the identification of a user, the permissions that a user can have, and the arrangement of

resources to different users. For instance, users that are considered as VIP users should be able to use more compute power than normal users.

Resource sharing is the key concept of a grid, which enables users to spread their work across multiple resources, so that their work can finish quicker and faster, and also they can access and process remote or distributed datasets. On the other hand, when a number of people want to use a particular resource, it is necessary to have a mechanism to allocate resources efficiently, and to avoid race conditions, e.g. a user cannot use a resource that is being used by another user. Normally, resources that can be shared include low-level, hardware-level resource sharing, such as storage resources and compute resources, and high-level, application-level resource sharing, such as sharing a document, or data collected by scientists or generated from instruments. A grid middleware system usually focuses on low-level resource sharing, and aims to standardize the application interfaces so that a third-party application can easily get access to the shared resources and usually expose them via a high-level user interface. The next section will describe grid middleware in detail.

Modern computer systems are usually connected to a fast network. This makes it possible to form an international grid. One good example is the LHC, where data is collected from the instrument, and then transferred via fast networks to collaborator sites in many other countries. Scientists in these countries can use either their local compute systems or computer centres where data is local to process the data, and share the result to more people.

To ensure systems across countries can communicate with each other, standards and open protocols are in place and adopted when building a grid. The Open Grid Forum

[12] is an international organization that consists of members around the world; it organizes workshops and conferences regularly to set standards and specifications. Development of current and new grid systems that follows these standards gives them the capability of future interoperation.

2.3 Grid Architecture and Middleware

According to its required characteristics, a grid can be designed and developed differently, and categorized in different groups.

Differentiated by the usage, a grid can be designed for a particular application, or to be general-purposed. For example, the SETI@Home [31] grid is only used for one purpose, which is to analyse data collected from telescopes to find intelligent life on other planets. General-purpose grids are generally built upon grid middleware, such as Globus Toolkit, to provide generic interfaces that can support various grid applications. This project aims to support users from different research disciplines, hence focuses on grid middleware in order to build a general-purpose grid infrastructure.

With respect to problem solving, grids can be categorized for solving high-throughput problems, or high-performance problems. High-throughput applications are normally designed to analyze massive data that is collected from instruments or experiments, and they usually divide big data analyzing jobs into many independent tasks in order to run these tasks concurrently on different computers. Examples of this kind include the analysis of thousands of particle collisions in LHC project, and the analysis of molecules or protein folding configurations in bioinformatics projects. SETI@Home is

another example that processes the massive data collected from huge telescopes. Condor [11] is a middleware system that enables users to run high-throughput jobs on desktop computers and clusters. On the other hand, high-performance applications are usually simulations and modeling where a large amount of compute resource is needed for a single job. For example, astrophysicists may run simulations of a supernova explosion or black hole collision, car designers may run simulation of car crashes when designing a new car, and meteorologists may run climate model simulations to predict climate change.

General-purpose grids, based on grid middleware, have attracted much funding and resources all over the world, due to its flexibility and utility for many problems, and ability to be shared by many users, for not only high-throughput applications, but also high-performance applications. A few widely-used grid middleware implementations will be examined next.

2.3.1 Globus Toolkit

The Globus Toolkit [8] has been developed since the late 1990s to support the development of service-oriented grid applications and infrastructure. The software is a common framework, or middleware that provides a number of essential components and capabilities as the foundation of a grid infrastructure. Along with the software itself, a broader ecosystem has formed to include users, developers, administrators and researchers who have utilized, contributed to, and standardized the middleware. The Globus Toolkit not only has become a de facto standard grid middleware implementation

that most grid systems around the world are built upon, but has also become the direction of many grid technology studies.

The sophisticated architecture of Globus Toolkit consists of a number of key components, as illustrated in Figure 2-3. It has service containers to host native and user-developed services written in Java, Python or C, and correspondingly, exposes a set of client libraries in Java, Python and C. The key components fall into 4 aspects, namely job execution, data management, information service and security. All of these components are backed by common runtime libraries in Globus Toolkit. In a real deployment, not all of them need to be installed; a site can only install common runtime environment and required components. The latest version of Globus is version 5. However, version 2 and version 4 are the major versions being used widely. There are major differences between versions, especially the interface. As technologies advance, some components are re-implemented with newer technologies and newer architecture. For example, Globus Toolkit version 4 (GT4) provides a web service based user interface while Globus Toolkit version 2 (GT2) and 5 (GT5) offer a simple HTTP-based user interface. More details about differences between versions will be given later.

Figure 2-3 Primary components in Globus Toolkit version 4 [8]

Following sections will give some details into the major Globus components.

GRAM

The Grid Resource Allocation and Management (GRAM) service [32] is the most important service in the execution management category. It is used to query, submit, monitor and cancel jobs in various local or remote compute resources by providing a single protocol and interface to users to simplify the use of compute resources. The GRAM protocol is HTTP-based; in version 4, or GRAM4, a web service interface was

introduced into GRAM; but in GT 5, the web service interface was abandoned and GRAM2 was enhanced to be GRAM5, which is a RESTful-like interface.

Technically, GRAM is not a scheduler but an interface to a native scheduler. GRAM accepts jobs in a specific format, RSL (Resource Specification Language) [33] in GRAM2 and GRAM5, while JDL (Job Description Language) [34] is used in GRAM4. JSDL (Job Submission Description Language) [35] is also supported in GRAM5. Note that JSDL is an OGF standard while RSL and JDL are proprietary formats and only used by Globus. Once a job is submitted to GRAM, GRAM converts it into a native format that the target scheduler uses, and submits the job to the target scheduler. GRAM keeps information of jobs, such as the native job ID in the target scheduler so it can query or cancel the job in the future. To date, GRAM supports a number of job schedulers, including PBS (Portable Batch System), Condor, Platform LSF and SGE (Sun Grid Engine).

MDS

MDS, or the Monitoring and Discovery Service [36], is a framework that collects and stores information about resources for monitoring and discovering purpose. MDS has two components, a registry and a data filter. The registry is an index of resource information and the data filter sends data about local resources to the registry, which can be hierarchical: there can be one central registry that aggregates data from all local registries and therefore shows the whole picture of all resources in a grid centre. Globus has provided a number of clients for users to take advantage of the data in registry. For

example, the WebMDS [37] service provides a web-based view of the registry via XSLT [38] transformations.

GridFTP

The GridFTP protocol [13] is an extension to the standard FTP (File Transfer Protocol) for transferring massive data in a grid environment. It is a major part of the data movement service in Globus Toolkit and in fact, it is the de facto standard data service in the grid world. It is based on FTP because the well-defined architecture of FTP is specific to data movement and allows extensions. The features added into GridFTP are mostly for fast data movement, and security in a grid environment. For example, the third party transfer feature allows data to flow between two data endpoints even when the transfer is initiated by a client residing at a third location; the parallel and striped transfer feature allows multiple simultaneous data streams to maximize the use of available bandwidth; partial file transfer allows the transfer of a portion of a huge file so as to save network bandwidth and time; GridFTP also supports fault tolerance by monitoring the transfer and can restart failed transfer from the failure point; users can also optimize the transfer by setting a proper TCP buffer size according to the current network situation. GridFTP is used in this project and will be discussed in more detail in Chapter 5.

Another service provided by Globus under the data movement service category is RFT (Reliable File Transfer) but it has been replaced by Globus Online in GT5. Globus Online will be described later in this chapter.

GSI

GSI [39], or the Grid Security Infrastructure, is a specification for secure communication between elements, such as applications and users, in a grid environment. GSI is based on the X.509 certificate technology and thus, every user and every service of the grid has a private key and a valid public key signed by a trusted CA (certificate authority). Before any real communication, GSI defines a way for both parties to exchange public keys and verify if the remote party is trusted. If trusts are created, communication can start within encrypted or non-encrypted channels.

Delegation is an extension to the standard SSL protocol defined by GSI in order to set up a mechanism for services to act on behalf of the user. This is useful in many cases where users cannot enter passwords all the time. For example, in a workflow, a user can create a delegated certificate, called a proxy, at the beginning and pass it to the workflow service, and the service can then use this certificate to access other services, such as execution managers, and GridFTP servers, as if the user does that himself/herself. This mechanism can be used for single sign on with using a single proxy to access various services. To make it easy for web portals to benefit from the delegation capability of GSI, MyProxy [40] was developed to act as an online credentials repository system. There are applications, such as web portals, that do not support GSI but need to interact with grid resources, and other users may want to access the grid from where the certificate is not in place. MyProxy can be used in these cases so that users can upload a proxy into MyProxy with password protection, and then retrieve the proxy in a web portal, or a computer without the user's certificate. With a valid proxy, the web portal or the user can access any grid resource the user is entitled to.

2.3.2 gLite

gLite [16, 41] is an effort by the EGEE (Enabling Grids for E-Science in Europe) project [42] to create a reliable and dependable grid infrastructure for the European e-Science community. gLite provides a set of services for building a production-level grid infrastructure encompassing computing and storage resource across the Internet and among a number of countries in Europe, as well as a middleware stack for developers to write grid applications hooking into the fundamental grid resources.

Similar to the Globus Toolkit, gLite follows a *Service Oriented Architecture* and offers five service groups, namely security services, information and monitoring services, data services, job management services and access services. Virtual Organizations (VOs) are the essential elements in security services. Any user must join a VO to gain access to grid resources. VOMS (Virtual Organization Membership Service) [43] is the service maintaining the relationship between users and VOs, as well as a user's roles and capabilities in a VO. VOMS keeps records of information of user certificates, but does not know what permission a VO has on a local system. GUMS [44], the Grid User Management System, is a site tool for resource authorization and provides mappings from VOs to credentials on local resources. When a request or a job is sent to a grid resource with the user's grid certificate, the resource asks GUMS which local identity is used to execute the given task by presenting the certificate. GUMS then returns a preconfigured local identify name to the resource if the user is in a privileged VO, or an error code if the user is not in any VO that can access this resource.

The information service is essential to coordinate resource discovering, monitoring and accounting over a grid as a whole. The current information service in gLite is BDII [45], the Berkerley Database Information Index, which stores information about the whole grid in an LDAP server and updates this information with an external process sitting beside a grid resource. The query of information can be done via standard LDAP query syntax, or via any commodity LDAP tools. This design simplifies the whole structure and makes it easy to integrate with other applications.

Data services in gLite consist of a number of storage elements (SEs), catalogue services and a data movement service. Currently, gLite is packaged with DPM (Disk Pool Manager), targeting disk-based storage, and dCache, targeting large-scale disk array and hierarchical storage systems. FTS (the File Transfer Service) is the major component in data movement service. dCache will be described in detail in section 2.4.2 and FTS in 2.5.1. The primary services in the job management services group include computing elements (CEs) and the workload management service (WMS) [46]. CEs are a set of resources localized at a site that act as bridges between higher-level applications and a cluster of computing machines usually managed by a LRMS (Local Resource Management System), while WMS is responsible for distributing and managing jobs across CEs and SEs. WMS accepts jobs described in a flexible and high-level Job Definition Language (JDL), and forwards jobs to a suitable CE based on the workload and other settings. To date, WMS supports LCG CE (based on GRAM2), gLite CE and CREAM [47]. WMS provides a complete set of tools to create, monitor, and cancel jobs.

CREAM, the Computing Resource Execution and Management system, is a lightweight and robust service that manages CEs efficiently on a Grid. The creation of

CREAM addresses several issues in the legacy LCG-CE, which does not support user proxy delegation properly, performs poorly on modern hardware, is not reliable in some situations where jobs are lost and lacks support. CREAM also comes with several improvements, for example, it has a web service interface, in contrast to the pre-WS interface in LCG-CE; its authentication is based on VMOS proxies; it supports multiple batch systems; it also has an asynchronous notification service to send out updates when job status changes. CREAM is part of the gLite software stack but it can also operate as a standalone service.

2.3.3 VDT

The Virtual Data Toolkit (VDT) [48] is a software stack built by the Open Science Grid (OSG) [49] to provide a production quality software distribution to its member sites and users. It has two goals. Firstly, it aims to create a generic grid middleware that is not dependent on any specific grid operator and can be used completely or partially by any grid service provider. Secondly, it aims to provide a working system on several platforms, a.k.a. operating system and CPU combinations. On the other hand, VDT is a package of existing software; the author does not write any new software but put mature and robust applications together with extensive testing to ensure they can work seamlessly and properly.

Currently VDT includes grid software, such as components from Globus Toolkit, gLite, Condor and other applications developed by grid application developers, as well as non-grid software, such as Apache and MySQL that are used by particular grid

applications. One typical example is the VOMS component, which manages lists of VOs and their members by mapping X.509 certificates. Other grid components can liaise with VOMS to find out if a user is in a specific VO thus can decide whether the user has permission for a specific operation. VOMS relies on Tomcat and a specific version of MySQL, which are both included in VDT. VDT also has a plugin for Globus GRAM service so that GRAM can take advantage of VOMS for job submission. VDT provides an easy and fast way to install all these bits and pieces; without it, system administrators may need to install many of them from source code and do testing themselves.

2.3.4 Other Grid Middleware

Some other grid middleware stacks are notable, as they are popular and being used by a number of groups. I'll touch on several typical examples in this section, to give a brief introduction to them.

Condor [11, 50] aims to address High Throughput Computing (HTC) by developing, deploying and evaluating a software stack and policies. A comparison of HTC and HPC application was given in the beginning of section 2.3. Condor offers tools to increase computing throughput and maximize the utilization of idle workstations or cluster nodes depending on the configuration. From a user's perspective, Condor provides functionalities similar to a traditional queuing system: users submit jobs to a queue in Condor and then Condor distributes jobs to worker nodes based on policies and the requested resource requirements of the job. On the backend, Condor can be configured to use a cluster of compute nodes, or make use of wasted CPU power from

idle desktop workstations. When working with desktop workstations, a monitoring program is installed on all desktop computers, and used to detect if the computer is idle, such as whether its keyboard and mouse are idle. If a desktop computer is idle, Condor migrates jobs and data (if needed) to this machine to execute the job. If someone comes to use this computer when a job is still running, Condor produces a checkpoint signal and moves the job to another idle machine. Condor can also incorporate with Globus Toolkit in two ways: Condor-G [51] redistributes Condor jobs to a local or remote Globus instance so users can use Condor interface to submit jobs to Globus Toolkit; Globus Toolkit has a built-in mechanism to send jobs to Condor like other LRMSs, which means users use Globus interface to send jobs to Condor.

UNICORE [10] (UNiform Interface to COmputing REsources) is a grid middleware enabling a seamless, secure and intuitive access to distributed grid resources and information stored in databases. It was started in 1997 in Germany to provide users access to computing resources by German supercomputer centres. Today, UNICORE is a robust production system being used by a number of supercomputer centres worldwide. The architecture of UNICORE consists of three tiers. The first tier, user tier, includes a graphical user interface, a command-line user interface and an API for integration with other grid applications. The server tier consists of the Gateway and the Network Job Supervisor (NJS). The Gateway is a secure entry point with controlled access to any UNICORE site. If access is given, incoming jobs are forwarded to NJS, which is responsible for virtualization of the underlying computing resources by mapping UNICORE jobs to the target system using system-specific configurations. The third tier, target system tier, is the interface to the underlying supercomputer, or other grid resource

manager like Globus GRAM. Lastly, UNICORE employs X.509 technology in its security model and provides single-sign-on based on certificates.

A meta-scheduler is another form of grid middleware. It gives a single and aggregated view of multiple distributed resource managers so as to enable batch jobs and computational workflows to be executed at the best location. One typical example of a meta-scheduler is GridWay [52], which is a workload management system that performs job execution management and resource brokering on various computing resources that are managed by different local or distributed resource management systems in a single organization or scattered across multiple administration domains. It provides a scheduling framework similar to other batch systems, including a command line interface and an OGF standard API, for job submission, monitoring, and control. GridWay can handle single jobs, array jobs (a job to be run multiple times with different inputs) and workflows, in an unattended and reliable manner. All steps including data staging and scheduling are performed transparently to end-users. The main feature of GridWay is adaptive scheduling and execution. Each job is defined with a requirement expression of expected resources, including basic job definitions (such as the executable, input files, output files) and resource requirements (such as CPU number, memory size, queue number). These requirements are then processed by an internal ranking system in GridWay to work out a suitable resource to submit this job to. GridWay also monitors the performance of each resource and keeps statistics, which will then be used in the ranking system. A running job can be restarted on a different host if conditions are met, for example, another resource with more compute power is available. GridWay is based on

Globus and becomes part of the Globus Toolkit from GT4, which makes it easy to install and work with Globus.

Nimrod/G [53], as shown in Figure 2-4, is a Grid middleware derived from the Nimrod project, which developed a parametric modelling system for users to easily run parameter sweep jobs. Nimrod defines a simple declarative parametric language to describe a parameter sweep job and provides mechanisms that run these jobs automatically and collects results from each individual task. Nimrod/G is built on this principle idea with a scheduler that can distribute jobs across heterogeneous compute resources. Technically, Nimrod/G integrates Nimrod toolkit and Globus toolkit to provide parametric functionality to the Grid. It uses Globus GRAM to submit jobs to the compute resources and Globus MDS to find out the structure of the whole Grid. Nimrod/G provides a client for users to submit and control jobs and then distributes jobs to the Nimrod Resource Broker (NRB) on a remote site. The NRB further forwards the job to its paired Globus GRAM, which sends the job to the queuing system. Nimrod was one of the first tools to enable the use of heterogeneous computing resources in a single computation job. For those who have access to multiple compute resources, the use of Nimrod can shorten the execution time by taking advantage of all available resources, and removing the constraint of having to run a job in only one resource.

2.4 Data Storage in a Grid Environment

The above grid middleware systems are all designed to interface with HPC systems. When it comes to data storage, different technologies are used to build massive data/file systems with tertiary storage backend. Traditionally, compute clusters are equipped with clustered file systems, in order to enable fine-grained control on files and sharing by each node of the compute cluster. To enable collaboration and data sharing by researchers from different places, it is required to access files that are geographically distributed. Thus data grid appears as a new concept to solve that problem. In a data grid, distribution is the key feature, which connects all these geographically distributed, and maybe heterogeneous systems that manage local storage resources. Generally speaking, clustered file systems provide low-level and usually POSIX-compliant file access interfaces, and data grid middleware offers application-level access interfaces.

In this section, I examine Lustre, GPFS, Ceph and XtreemFS as examples of clustered file systems, as well as dCache and iRODS/SRB as examples of data grid middleware.

2.4.1 Clustered File System

In distributed systems, a clustered file system is shared by and mounted on multiple hosts. There are two forms of clustered file systems, either shared-disk or distributed. A shared-disk file system uses a storage area network (SAN) or RAID to provide direct disk access from multiple computers at the block level. A distributed file system is any file system that allows access to files from multiple hosts scattered in a computer network. Instead of providing block level access, access to file storage in distributed file system is conducted by using a network protocol. In this sense, a data grid is also a distributed file system. Another important feature of the examples here is that they are parallel and fault-tolerant. They store data in multiple data storage devices for striping and replication in order to provide high performance data access and maintain data integrity. If one server is down, data can still be retrieved from another storage device. Following sections will look into several examples of clustered file systems.

Lustre

Lustre [54] is a massively parallel distributed file system running on Linux, and can be used along with small workgroup clusters to large-scale HPC systems. The name Lustre is a portmanteau word derived from Linux and Cluster. It was originally

developed at CMU and is now owned by Oracle. The motivation of Lustre is to provide a storage area network (SAN) file system that allows fine-grained sharing of data in a cluster environment. A Lustre file system has three major components, where metadata and real file data are stored separately. The first component is a metadata server (MDS) that maintains a single metadata target (MDT) that keeps namespace metadata, including filename, directories, access permissions, etc. MDT is stored in a local disk file system. The second component consists of one or more object storage servers (OSSs), where real file data is stored. Clients are another component of Lustre that makes use of data with standard POSIX-compliant commands in a single and unified namespace composed of all files and objects in the whole Lustre system.

A typical Lustre system contains tens of thousands of clients, thousands of object storage servers and a failover pair of metadata servers. Object storage servers can be added dynamically to provide flexibility. If more than one object storage server is associated with a file, data of this file is striped across them like RAID 0, which provides significant performance benefits than transferring from a single node. Lock management is another useful feature of Lustre to allow many clients to access one single file concurrently for both read and write without risking into bottlenecks. Also, Lustre implements Lustre Networking (LNET) to provide network connection between disk storage and Lustre system components using SAN technologies. LNET supports many network protocols, such as fast InfiniBand and common IP networks. The main advantage of Lustre is high parallel performance, and it has been deployed on several of the top 30 supercomputers in the world. In addition, TeraGrid has installed Lustre as a

production service and is going to offer Lustre WAN service that consists of nodes across the states [55, 56].

GPFS

GPFS [57, 58], or the General Parallel File System, is a high-performance shared-disk clustered file system developed by IBM for cluster computers. The architecture of GPFS consists of a number of cluster nodes, on which the GPFS file system and applications are running, connected to the shared disks via a switching fabric, which is normally a SAN. The GPFS system works in a similar manner to a general-purpose POSIX file system running on a single machine. Users use a conventional block I/O interface to access the underlying data, which they have no knowledge of. Internally, GPFS stores data in blocks rather than as objects, and each file is broken up into many blocks, which are distributed in one or more physical disks. The size of each block is configurable, and normally less than one Megabyte. When accessing such a file, GPFS can achieve a high input/output performance by striping blocks of data from each of these nodes over multiple disks in parallel, as the combined bandwidth of many physical machines is always higher than only one. GPFS also supports fully parallel access to metadata, such as directory structure and file information, which is stored across multiple nodes like the real data. So when a file is accessed on a particular node, that node is also responsible for metadata management of the requested file. Consequently, it removes the limitation that a directory can only contain a certain number of files. Under such a structure, it is feasible to add or remove disks without stopping the whole system.

One notable feature of GPFS is the distributed locking mechanism, which is used to synchronize parallel read/write disk accesses from multiple nodes, to avoid file content or file metadata corruption. It ensures fast throughput while maintaining file system consistency, regardless of the number of nodes being accessed at the same time.

GPFS has been built on several of the most powerful supercomputers, and is a production service of TeraGrid and DEISA in a WAN environment [59].

Ceph

Ceph [60] is an open source distributed file system originally developed by a PhD student at the University of California, Santa Cruz. Its goal is to be a POSIX-compatible and fully distributed file system without a single point of failure. Since May 2010, its client has been merged into the Linux kernel. The servers run as regular Linux daemons. Ceph has three major components: Object storage devices (OSDs) which store the content of files, Metadata servers (MDS) which store the metadata of inodes and directories, as well as cluster monitors that keep track of active and failed cluster nodes.

Ceph claims to have several outstanding features that make it distinct from other file systems. Firstly, storage nodes (OSDs) can be simply added to the system to expand it seamlessly. Afterwards, data will be migrated into the new devices proactively to maintain the balance of data distribution; so new devices can share the same load as old devices. Secondly, all data in Ceph is replicated across multiple OSDs. If any of them fails, Ceph can replicate data in that failed device to other active devices, to maintain the balance. Once the device is recovered, data can be replicated back to it in parallel from multiple OSDs, to achieve a fast performance in the whole recovery process. Distributing

data across all OSDs avoids the waste of having spare disks like RAID 5. Lastly, the design of MDS makes it possible to adapt its behaviour to the current workload. As the number of files and directories changes over time, the metadata is redistributed dynamically among available metadata servers in order to share the load to them evenly. Moreover, if MDS suddenly receives thousands of requests on a single file, the metadata of this file is replicated across to more servers to distribute the workload.

At the moment, Ceph is under heavy development and is not suitable for production use, according to the developer. Also, it is designed mainly for LAN use and the performance in a WAN environment is not clear.

XtreemFS

XtreemFS [61] is a globally distributed and object-based file system, with metadata and regular data stored on different types of nodes. It is designed to work for wide area networks, with replicated objects for fault tolerance and cached metadata and data to improve performance over high-latency networks, as well as supporting SSL and X.509 certificates to ensure data security over public networks. XtreemFS is open source and POSIX-compatible. Its server can run on Linux, Windows and Mac OS, and it has clients for Linux and Windows. XtreemFS has several features that are beneficial to Grid applications. For example, consider the situation where some large files are regularly read by scientific applications from many locations for analysis, instead of copying the file to these locations, XtreemFS is able to make several replications, and when the file is requested, the nearest copy will be used. As XtreemFS is POSIX-compatible, the application can read transparently from XtreemFS just as reading from a local disk,

without needing to copy the file to its local disk. XtreemFS can also make partial replicas of a big file, if a portion of the file is used more frequently than other parts. This can reduce the time and resource to create and hold the replica. In addition, XtreemFS supports striping, which divides the content of a file into several object stores. When this file is opened by a user, it will be transferred from all these stores to the user in parallel, to take advantage of the available network resources.

One big disadvantage of XtreemFS is that it is based on the assumption that storage resources are all hard disks and it cannot interface with heterogeneous storage systems, such as hierarchical storage systems.

2.4.2 dCache

dCache [19, 62] is a distributed file system developed by Fermilab, DESY and NDGF. It makes use of storage across computers and provides a unified and single view to users so they only see one file system without being aware of the actual location of their files. The architecture of dCache consists of nodes and cells. A node is a physical computer that runs all or some modules of dCache. A cell is a fundamental executable module that has a specific task to perform. Cells may or may not interact with other cells during the execution. Thus, a single node can have one or more cells running on it, and many cells in dCache can have multiple instances running across several nodes so as to achieve load-balance. To categorize by type, cells can be grouped into different types, such as pools and doors. Cells of the same type behave in a similar way. Between nodes and cells, dCache defines another term, domains, referring to the container of running

cells. Technically, a domain is a Java Virtual Machine (JVM) instance, which is an independent process running on the hardware. It is possible to run multiple domains on one node, which means all these domains share hardware resources of the physical machine, such as memory, CPU and network bandwidth. dCache comes with several predefined domains, with each composed of a list of different cells to achieve a certain goal, such as data storage backend, user interface front-end and data movers. dCache is scalable, so if the performance is slowed down by a large number of requests, new hardware can be introduced to run additional instances of the same type of the overloaded node, to allow load-balancing.

dCache offers a few features to make storing data easier, and is able to make replicas of files from one node to another. For example, nodes can be added to and removed from dCache cluster anytime without stopping the whole system. One exceptional feature of dCache is the support of requesting data from a tertiary storage system, such as a tape robot. Because magnetic tapes are usually cheaper than disks, it is often an economical way to store massive data on inexpensive hardware. However, tapes have to be loaded and unloaded by the robot, therefore the access latency is significantly high to access data in the tape. dCache acts as a cache for tertiary storage systems so that frequently-used data is cached in dCache, to decrease the waiting time when accessing a file. If a file in dCache is used less often than other files, is will be removed from the cache to make room for other files. dCache also supports many transfer protocols, as modules, allowing different front-end machines and user interfaces to be plugged in. Another performance feature is hot-spot data migration. dCache is able to monitor and detect if a file is accessed very often, and then several replicas of such file will be created

on multiple servers, to allow load balancing across these servers and thus increase throughput. In addition, dCache has a mechanism to direct data to designated data resources. This is useful for large sites to make good use of their resources in an efficient way.

dCache offers a few user interfaces, such as dCap [63] (dCache Access Protocol), GSIdCap, GridFTP, and recently WebDAV. dCap is dCache's native protocol, which supports password authentication and GSI. In addition, dCache comes with an implementation of the SRM [64] protocol to allow negotiating the actual data transfer protocol and reserving required space beforehand. SRM protocol is an OGF standard and has been adopted by several widely-used storage systems, which means dCache can interactive with them seamlessly. In terms of user authentication, dCache does support X.509 certificate technology and traditional username/password authentication. However, certificates are preferred and supported by the commonly used user interfaces, such as GSIdCap, GridFTP and SRM. dCache also provides a console-based and web-based administration interface for administrators to easily configure the system, and do trouble-shooting if any errors happen.

2.4.3 iRODS/SRB

The Storage Resource Broker (SRB) [17, 65] and the Integrated Rule-Oriented Data System (iRODS) [18, 66] are both developed by the Data Intensive Cyber Environment resource group (DICE) to help with data storage and sharing for grid operators, digital libraries and data archivists. SRB was developed from the late 1990s

and was superseded by its successor, iRODS in 2008. They are two independent products but they derive from the same concept, apart from iRODS having extra modules to make it more adaptive and flexible. Before getting into the details of iRODS, it is necessary to briefly touch on SRB.

SRB is a data grid middleware system for sharing data and metadata distributed across heterogeneous resources using uniform APIs and GUIs. SRB provides an abstraction layer of data object names, sets of data objects, resources, users and groups so that functionalities can be built on it to offer uniform methods. With this layer, complexities of underlying physical infrastructure are hidden from users; instead, users only need to deal with global and logical mappings of actual digital entities registered in SRB. Therefore, a user can access files in any data source, such as an online file system, near-line tapes, relational databases or sensor data streams without needing to know the real location of that file, how that file is stored, or in what data system that file is stored. The virtualization view of underlying data sources provided by SRB is maintained in a metadata catalogue called MCAT, which utilizes a relational database system to store mappings of virtual files and directories and their real locations.

SRB is a successful implementation of a virtual distributed file system to store and share data. It has been deployed on more than 100 research institutions around the world and is used to manage data repositories of Petabytes size [67]. Although SRB has mechanisms to ensure data integrity and authenticity of shared collections, other management tasks are frequently tedious. In fact, most of the business logic in SRB is hardcoded, which makes it hard to customize the behaviour of SRB. Even a small function change requires changes in source code, which may result in unforeseen errors.

To solve this problem and make the system flexible and adaptive, the SRB developers implemented the next-generation system, iRODS, as the successor of SRB.

iRODS has a different architecture, which is classified as adaptive middleware. The adaptive middleware architecture (AMA) [68] provides a way for users to customize and configure the software to meet requirements without making any changes to the source code. To achieve this, a particular methodology, namely Rule Oriented Programming (ROP), is adopted, and this is also why the software is called iRODS. ROP paradigm is used in iRODS to declare user-defined workflows with two important elements, rules and micro services. A rule is a series of management operations, or a workflow, invoked by iRODS processes to complete a particular task in the system, such as creating a new user, creating a new collection, and so on. Each operation in the rule is called a micro service, which is generally a C-function and executable by the system. During customization, a user can change the flow of a rule, or even introduce more micro-services to change the behaviour of iRODS. For example, the default rule of creating a new user consists of two steps, creating a new account and creating a home collection for the user. This can be changed in a configuration file to add more steps, such as sending an email to notify the user that the account has been created. Rules in iRODS support conditions, that is, several rules can be set to one task, but with different conditions. In the previous example, several create-new-user rules can be put in place, e.g. one for creating normal users, and one for creating data curators. iRODS comes with more than 100 micro-services and more will be created by both the developers and the user community. In addition, one can add new micro-services, or recompile current micro-services into iRODS, and this will not affect the running iRODS, as iRODS checks the rule

configuration file periodically to pick up any changes without needing to restart the whole system.

The detailed architecture of iRODS is illustrated in Figure 2-5. It is based on a client-server model with distributed storage and compute resources, and uses a relational database system to maintain metadata of files, including attributes and the states. This is similar to SRB. What is extra to SRB is that iRODS has a rule system which enforces and executes adaptive rules. The rule system is the core of iRODS, where predefined micro-services are invoked on the interpretation of the rule being executed. There are two types of rules, system-level rules and external rules. System-level rules are invoked internally by the server to enforce management policies or start system-level services. External rules are invoked externally by users using a command-line interface to perform a sequence of operations for the user. As to execution time, some rules are executed immediately while others are queued and executed at a later time in the background.

Overall, iRODS provides an abstraction layer for data management processes and policies, which is much broader than the abstraction offered by SRB for data objects, collections, resources users and metadata. These management processes and policies are required to enforce authenticity, data integrity, access controls, data placement and presentation, throughout the whole life cycle of data in iRODS. They are therefore mapped to rules that control the execution of data management operations.

Figure 2-5 iRODS Architecture [18]

Access to iRODS service is only achievable via the iRODS proprietary protocol. However, iRODS comes with several native interfaces and APIs, such as Java library, C library and Python library, to allow connecting from other applications. User communities are also proactively involved in the development and have developed more interfaces for iRODS. I, in this project, have contributed to this development to make it easier for users to use iRODS. Details will be given in chapters 3,4 and 5.

2.4.4 Summary and comparison

The features of the above file systems are listed in the following table for comparison. The table has each column showing all features of each file system, while each row compares a particular feature of all file systems. The first column of each row (except the first row) shows what feature will be compared among these file systems. For example, in the row of 'Data Replication', as all file systems have a built-in mechanism to replicate data among its nodes, it is interesting to figure out whether they use the replication for failover (if one node fails, files can still be read from other nodes with replicas) and/or striping transfer (one requested file will be served from multiple nodes); in the row of 'Node types', it briefly gives the different components (nodes) that the file system is comprised of and the roles that these nodes play.

In summary, I have examined six file systems, most of which are open-source or free to use, except for GPFS which is under a commercial license. Most of them store data in files except for GPFS which stores data in blocks. All of them employ metadata and have replication. I also examined the possibility of deploying each in a WAN environment, as this is a requirement in this project, and most of them have been deployed and tested in WAN except for Ceph. All of them offer multiple client interfaces, which are important to users. In terms of architecture, all of them are designed to use a client-server architecture with multiple components (nodes). It is noted that all server-side components can be set up to have fail-over nodes so as to achieve high availability.

	Lustre	**GPFS**	**Ceph**	**XtreemFS**	**dCache**	**iRODS**
License	GPL	Commer-cial	GPL	GPL	DESY	BSD
Data Primi-tive	Object (file)	Block	Object (file)	Object (file)	Object (file)	Object (file)
Meta-data	Max 2 metadata servers	Distributed across storage servers	Multiple metadata servers	BabuDB (key-value store)	Relational database	Relational database
Data Replica-tion	Failover and striping	Failover and striping	Failover	Failover and striping	Failover	Failover
WAN deploy-ment	TeraGrid	TeraGrid, DEISA	unknown	Experi-menting	Fermilab, Swegrid, NDGF	ARCS, TeraGrid, KEK, NGS
Client interface	Native client, FUSE, CIFS, NFS	Native client, CIFS, WebDAV	Native client, FUSE	Native client, FUSE	NFS, HTTP, WebDAV, GridFTP, SRM, dCap	Native client, FUSE

Node types	Clients, metadata, objects	Client, data	Clients, metadata, objects	Directory service, metadata server, storage server, client	Clients, metadata, objects	Clients, metadata, objects

Table 2-1 Comparison of file systems

2.5 Data Transfer Service

When using any file system, it is easy to copy one file, or a few files, from one location to another. However, when it comes to transferring massive volumes of data, it is essential to use a proper tool to automate the whole process rather than doing this manually. This tool is normally called a Data Transfer Service, which offers several basic functions: a scheduler to start and monitor transfer jobs, a mechanism to ensure data integrity, and a mechanism to recover from failure and restart failed transfer jobs. There are differences between different products; for example, some may allow transfer between data sources using different protocols, and others may require transfer between two homogeneous data sources. In this section, we mainly study three data transfer services, as they are the most popular ones and being used by many users.

2.5.1 gLite FTS

The gLite File Transfer Service (FTS) [69] is a data movement service developed as a component of the gLite grid software stack. It is responsible for moving data sets from one site to another with using a specific network bandwidth that can be controlled by the user. The control ensures the network is shared fairly by most users and also allows users to choose a fast network link when there are several possible routes. The design of FTS is mainly for point-to-point transfers, with a mechanism to monitor network resources and display statistics data of transfers. As part of gLite, FTS is integrated with other gLite components, such as File Catalogue. However, it is also possible to use FTS standalone.

FTS consists of several service components. Firstly, FTS performs data transfers on channels, which are usually tied to network links. Each channel is unidirectional, which means there can be multiple channels between two points. This is to allow the use of different network links if possible. As a consequence, two channels have to be created for a bidirectional transfer link between two points. A number of parameters can be set to a channel, such as the reserved bandwidth. This allows fine-grained control of performance and behaviour of a channel. FTS stores transfer information and states in a relational database, including the source and the destination. When submitting a transfer job, the user needs to give the source and destination, and maybe a dedicated channel. Then, the state of a transfer job can be queried against the database. FTS uses VMOS to assist in user authorization. For example, only administrations of a VO can create or modify channels, and users can only see their own transfer jobs.

The Architecture of FTS is comprised of four components. The web service interface includes user interfaces to submit transfer jobs, get status of current jobs, cancel jobs, create/list/remove channels, add/remove/list VO managers, and retrieve statistics data of a channel or a VO. The FTS channel agents are background services and each agent controls a specific channel in FTS. The agent is responsible for starting and controlling transfers as well as tuning transfers according to the channel parameters. FTS VO agents are another set of background services that are responsible for communicating with VOMS to deal with user authentication and authorization matters. A FTS monitor is another independent component that mainly generates statistics by querying the relational database. It can create a summary of FTS work at hourly, daily or weekly intervals. It can also create summary files of recent transfers from the database, and publish them to the central reporting component of gLite.

gLite supports GridFTP and SRM protocol. Users can use command-line tools or write their own application to interact with FTS via its web service interface. Being part of gLite, FTS is currently being used by the European research community and to transfer CMS data [70].

2.5.2 Stork

The Stork [71] data scheduler is claimed to be the first batch job scheduler specializing in data movement. It aims to mitigate the data movement bottleneck in eScience and data-intensive scientific discovery by offering a solution for planning, scheduling, monitoring and managing data placement tasks and application-level end-to-

end optimization of network I/O for large-scale distributed applications. According to the developers, data-related resources and tasks in Stork are treated as first class entities and not simply the side effect of a computation job. Stork allows users to transfer data between heterogeneous or homogeneous data sources by providing an abstraction layer to various protocols. This abstraction layer also hides the complexity of different network links. Therefore, users only need to tell Stork the source and destination, and Stork takes the responsibilities of managing data jobs, using an appropriate client library and protocol to access the desired data source, selecting appropriate network link and tune network parameters to achieve better performance. Stork can also avoid storage system or network bandwidth overloading by implementing a framework to receive status data from data resources, including their attributes, available free space, and maximum concurrent connection number. Stork will not connect to a data source if there is not enough space to hold more data or no free connections. On the other hand, Stork can make reservation for space before transfer to ensure space will not be taken by other applications in the middle of transferring a big file.

From version 2.0, Stork comes with a “Throughput Estimation and Optimization Service”, which is used to predict the optimal number of parallel streams to achieve best transfer throughput. It can estimate the transfer time when using one stream, by collecting and analyzing history data, so that it can suggest a optimal number of parallel streams and how long it would take to transfer using this number. Thus, this optimal number will be used automatically to perform future transfers.

Stork is normally paired with Condor and DAGMan in a workflow, as illustrated in Figure 2-6. DAGMan (Directed Acyclic Graph Manager) is a meta-scheduler for Condor,

and manages dependencies between jobs at a higher level. Normally, a job includes sub-tasks including stage-in, stage-out and the actual compute task. When it is submitted to DAGMan, staging tasks will be executed by Stork while the actual compute task will be executed by Condor. DAGMan, at the higher level, manages the dependencies between tasks and ensures later tasks will not start if prior tasks are not finished. Stork itself has built-in fault tolerance mechanism to retry failed jobs. To date, Stork is being used by several research disciplines and integrated with several research applications in the US. On the other hand, the installation and administration of Stork is not straightforward due to the lack of documentations, and the software itself is not production-ready.

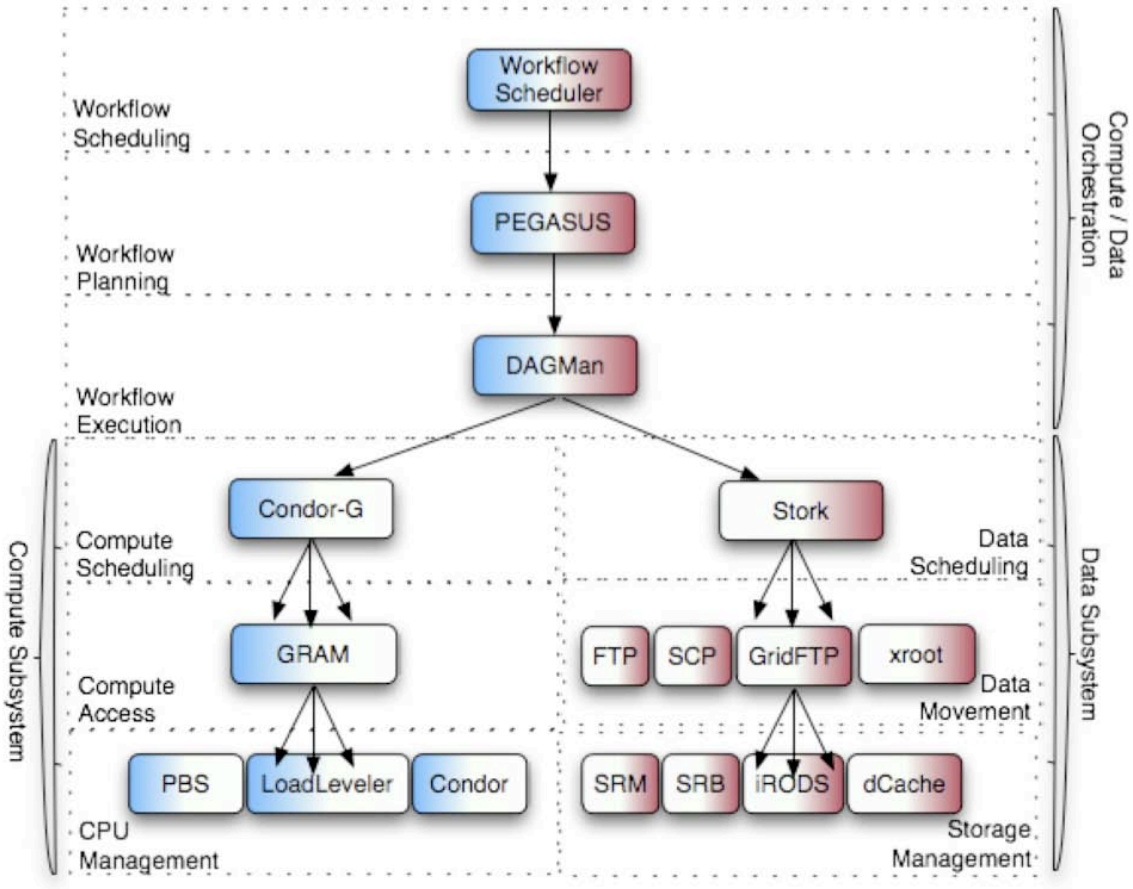

Figure 2-6 Stork and Condor [71]

2.5.3 Globus Online

Globus Online [72, 73] (GO) is a software-as-a-service (SaaS) system intending to provide an easy-to-use data transfer service to researchers. In contrast to traditional grid services that require huge effort to deploy, configure and maintain, GO is a cloud-based service that resides on the Amazon cloud. Research groups, big and small, have requirements to move data regularly or irregularly. However, it would be difficult for a small research group to maintain a complex grid system, given that they do not have

enough IT resource. Outsourcing is a common approach for small businesses to set up email, website and other fundamental IT services for their daily business, so do research groups. GO is designed, developed, maintained and supported by Globus developers. It provides a modern web 2.0 interface with a fire-and-forget capability, requiring no client software installation. With little or no configuration, users can start using GO from any computer anywhere, and don't need to keep an eye on the transfer but will get a notification once the transfer is done.

The architecture of GO, as illustrated in Figure 2-7, consists of a number of user gateways, a number of workers and a profiles and state database. User gateways provide several interfaces to users with different levels of IT knowledge. Ad hoc users may find it easy and sufficient to use a web interface, which is based on the web 2.0 technology. A command-line interface accessed via SSH and a RESTful interface are provided for advanced users to write scripts. The RESTful interface can also be used to integrate with other applications. Once transfer tasks are committed to GO, workers, the entities where these tasks are executed, perform all work related to the task, including connecting to the data sources, and sending notifications to users if task state is changed. GO maintains these tasks and their states in a database. As GO is run in Amazon's Elastic Compute Cloud (EC2), it benefits from all features provided by EC2, notably scalability for workers and fault-tolerance for the database. For now, GO supports GridFTP and FTP protocol and requires a GridFTP/FTP server installed on each data source, or endpoint. With supporting GridFTP, GO conducts third-party transfers between endpoints so user data does not flow through GO, and thus GO does not redirect or store any user data at all. For those who do not have such a server, GO offers a client application, Globus

Connect, to be run on one's desktop or laptop. Globus Connect has a built-in GridFTP server and acts as an agent to receive requests from GO and initiates transfers from the user end. This is also a solution to those with a firewall blocking incoming traffic and requiring transfers to be started from inside the firewall.

Figure 2-7 Globus Online Architecture [73]

GO is currently in beta status, which means more new features will be added frequently. However, it is being used in real scenarios and has been proved to be successful in transferring huge data sets [74].

2.6 Authentication

Authentication in a Grid environment is largely based on X.509 certificate technology, such as the Globus GSI, as described in section 2.3.2. However, certificates are not easy to use in a modern web-based environment, in particular, to achieve single-sign-on across different systems. This section will look into technologies that narrow the gap between traditional grid services and modern web applications.

2.6.1 Shibboleth

Firstly, we examine the cutting-edge architecture and implementation produced by the Shibboleth project [75] for advanced cross-institutional authentication and authorization and single-sign-on in web-based systems. The middleware is called the Shibboleth System, often just called Shibboleth, which is an open-source solution for federated identify-based authentication and authorization based on the standard Security Assertion Markup Language (SAML) [76]. It enables secure access to web-based resource, and allows independent organizations to federate and trust each other in such federation. Attribute-based authorization is employed to protect privacy of user information, and provide controls of resources in the federation. In such environment, users only need to use the credential from their home organization to gain access to any remote resource in the federation, which alleviates the pain of having to register in different systems and remember different passwords.

To build the federation, the Shibboleth System offers two major software components: the Shibboleth Identity Provider (IdP) and the Shibboleth Service Provider (SP). IdPs are deployed next to each organization's user system, acting as an entry point for user login, and supplying user information to SPs if authentication is successful. SPs are sitting alongside each web application; they receive and verify user attributes from IdPs and make decision on whether access should be given based upon this user information. The steps of this process are described as follows:

1. A user first navigates to a Shibboleth-protected web resource using a web browser;
2. The SP redirects the browser to a WAYF (where are you from) page, which presents a list of trusted organizations in the federation;
3. The user selects his/her home organization;
4. The browser is sent to this organization's IdP, which is basically a login page where the user can enter his/her own identity;
5. If username and password are correct, IdP sends the browser back to the original SP, with some security information called an assertion that proves the user has signed on successfully, and then the SP can verify the assertion and request additional information of the IdP and the user; If both the IdP and the user are legitimate, the SP requests user attributes from the IdP and then passes them to the protected web application, which can use this information against its own user policy to make further decision on access.
6. The user is verified to be a valid user, so he can continue to browse the protected area of the web resource.

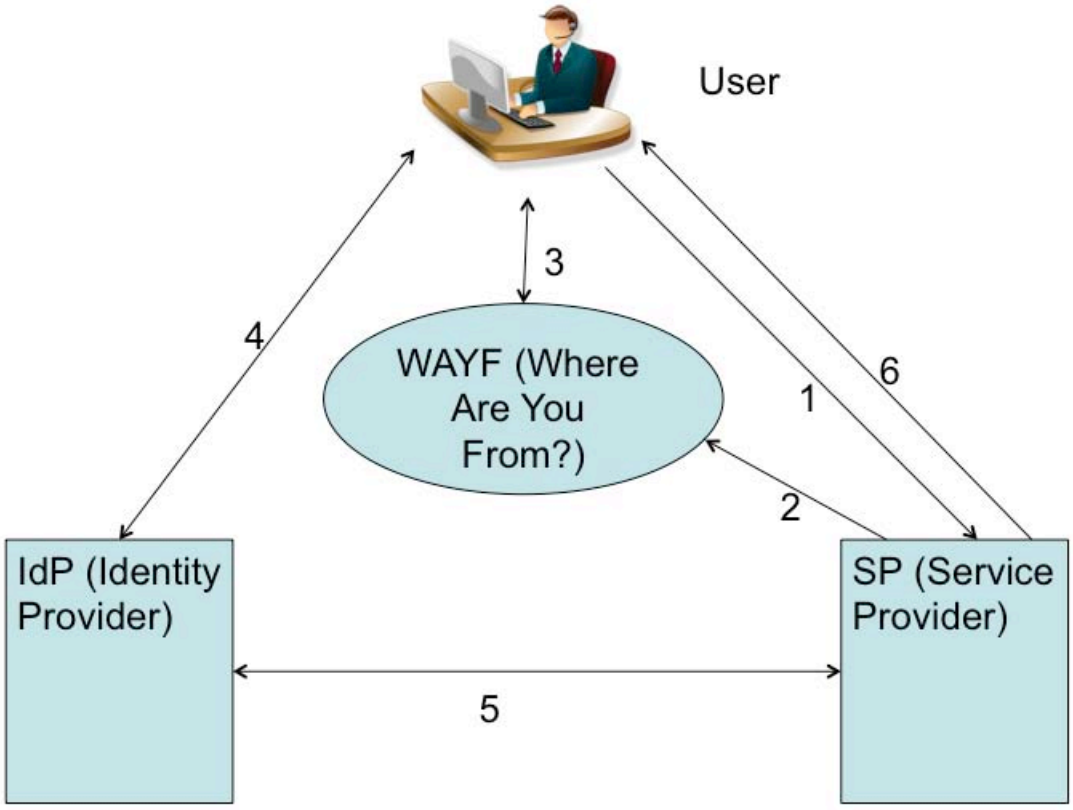

Figure 2-8 Shibboleth Authentication Flow Diagram

A number of Shibboleth federations have now been set up for the academic sector around the world, such as the Australian Access Federation (AAF) [77] in Australia and InCommon [78] in the US. However, being a technology for web-based systems, Shibboleth can only work with an HTTPS-capable web browser [79]. Thus, it is not straightforward to use Shibboleth in a grid environment. The next section will study several technologies created to combine Shibboleth and the grid.

2.6.2 SLCS and GridShib

Several mechanisms have been developed to make Shibboleth work in a grid environment. For example, a few projects [80] in the UK have explored ways to reconcile Shibboleth and the Grid, with using MyProxy as the core, such as ShibGrid [81]. However, these projects use some components that are designed to work in particular within the specific project, so they are not easy to migrate to work in other systems. This section will look into a few generic solutions in detail.

SLCS (Short Lived Credential Service) [82] is a service that issues short-lived X.509 certificates based on a successful authentication. Traditionally, a grid CA issues long-lived certificates, normally with a lifetime of one year. Once issued, a user must keep track of the private key and store it safely. Moreover, the private key is often protected by a user-nominated pass phrase, and re-issuing a long-lived certificate is somewhat time-consuming, usually requiring approvals from several system administrators at different levels. While using a SLCS certificate, a user only needs to request it when it is needed, using the user's own identity. A SLCS certificate can be discarded after use, and easily requested again the second time. Consequently, SLCS is a perfect match to Shibboleth. The integration of SLCS and Shibboleth is accomplished by installing a SP to protect a SLCS service. Thus, when a user requests a new SLCS certificate, he/she needs to authenticate via Shibboleth, and then IdP sends user information to the SP on a successful login. The SP passes user details to SLCS so a certificate request is generated for the user, who can use this request to retrieve a SLCS certificate, including a private key and a public key, which normally expires in eleven

days. The SLCS certificate, conforming to the X.509 specification, can be used to access any grid services.

GridShib [83] is another attempt to integrate SAML-based Shibboleth and PKI-based GSI authentication of the Globus Toolkit. It provides two components, one being a plugin for the Globus Toolkit to obtain attributes about the user from a Shibboleth attribute authority, and the other for a Shibboleth IdP to serve user attributes based on the user's X.509 subject distinguished name (DN). The second plugin maintains a mapping between a user's DN and a local user account name (in the local Linux system). When a valid request comes in, the local principle name will be returned according to the requesting user's DN, and Globus Toolkit gives user permissions based on this local principle name. This is similar to using Globus Toolkit's gridmap file, which maps grid certificates to local user accounts. However, the development of GridShib has ceased [84], and one of its components, the GridShib CA, is continuing in the CILogin project [85]. CILogin issues short-lived certificates to users on a successful authentication with their home organization identities. It supports Shibboleth and OAuth [86], thus any OpenID-compatible credentials, such as Google and Paypal accounts, can be used. Apart from issuing certificates to users, it features a portal delegation service, which issues certificates to a web portal on behalf of a user, and the web portal can use the user's certificate to access grid resources.

2.7 Case Studies

After exploring fundamental technologies and systems in the Grid world, it is interesting to see how the technologies and systems are being employed in a production environment. This section inspects several large distributed computing and storage infrastructures around the world, to get some ideas on what technologies are used and how they are used, and thus use their experiences as a guide in our implementation.

2.7.1 TeraGrid

TeraGrid [21] is an open scientific computational grid infrastructure funded by the US National Science Foundation (NSF). It is a collaboration of twelve partners, integrating high-performance computers, data resources, scientific tools and gateways across the US [87]. The operation of TeraGrid was started from 2004, and now includes more than one petaflop of computing capability and more than 30 petabytes of data storage, with high-speed network connections. In additional, researchers have access to more than 100 discipline-specific databases. It has about 4,000 users from over 200 universities; in this sense, it is a large distributed cyber-infrastructure for open scientific research in the world.

TeraGrid primarily focuses on three objectives. The *Deep* initiative aims to provide assistance to scientists in using a combination of resources, including computing, storage, instruments and visualization. The *Wide* goal enables user access to all TeraGrid resources by developing and tailoring gateways and tools, to allow seamless connections

from desktops, third-party applications, and other grid projects. TeraGrid *Open* involves interoperability with other Grid projects and campus resources in the US and other countries.

The architecture of TeraGrid is a service-oriented architecture where each resource exposes itself as a service with well-defined interface and operation. Overall, there are four key sets of components. Firstly, shared policies aggregate distributed resources and provide a uniform interface to all users, such as one single identify to access all resources, a unified support team and documentations to get help. Secondly, TeraGrid provides a coordinated set of Grid middleware, based on familiar systems, such as Globus Toolkit, Condor and SRB, for users to easily port application from one system to another, and for achieving higher-level functionalities, such as single-sign-on, workflow support and data movement. TeraGrid also has a sophisticated monitoring and testing environment to ensure stability of its resources. Lastly, TeraGrid has a geographically distributed user support team that can help users in a fast fashion.

Over the past several years of operation, TeraGrid has supported a number of disciplines, and outstanding projects including the Computational Chemistry Grid (GridChem), Linked Environments for Atmospheric Discovery (LEAD), nanoHUG.org and the Cancer Biomedical Informatics Grid (caBIG).

2.7.2 Open Science Grid

The Open Science Grid (OSG) [88] is a collaboration of large distributed computational resources that supports high-throughput scientific applications. It also

partners with other grid infrastructures, such as TeraGrid and EGEE to form a multi-domain and integrated system for scientists and researchers. OSG does not own any resources. Instead, resource are owned and contributed by its members. OSG provides a standard middleware, VDT (as described in 2.3.3), to member sites with defining a list of essential software and interfaces so that these systems can work together and communicate with each other. For instance, job execution in OSG is done via Condor-G clients and Globus GRAM gateway; data access is via GridFTP or SRM.

OSG is being used by several research communities, and the particle physics community is the large user group, who uses OSG to process and analyze data collected from the Large Hadron Collider (LHC).

2.7.3 LCG

The LHC Computing Grid (LCG) [20] is a computing infrastructure to deliver data analysis solution for all four LHC experiments, namely, ALICE, ATLAS, CMS and LHCb. It works in very close collaboration with EGEE (details will be given in section 2.7.4) and shares responsibility and objectives. Technically, it consists of four tiers connected by high-speed fibre optic cables and the Internet. Tier0 is considered as the CERN centre, responsible for storing all raw data, and it is the only place where all raw data can be found. Tier1 comprises of large computer centres, acting as repositories of the reconstructed data. Tier1 centres are responsible for successive reprocessing of data and distribution of data to lower levels. Tier2 data centres are expected to hold a large subset of the whole data, and provide CPU power for analysis. The LHC experiment generates

15PB of data every year, thus requires a large number of CPU cycles to process. Tier3 is normally a local cluster or even a scientist's desktop that has access to Tier2 facilities.

LCG now is also known as WLCG, Worldwide LCG [89], because it consists of several existing large-scale grid infrastructures around the world, such as OSG in the US and EGEE in Europe. Facilities and resources in these existing grids are contributed to the WLCG project, as either tier1 or tier2, for data storage, analysis and distribution. In this sense, WLCG is like a big Grid. Each LCG experiment has a central job queue and application repository. A member of the experiment can send batch jobs to this central queue, then these jobs will be distributed to an available resource in a tier1 or tier2 that may be anywhere in the world for execution.

2.7.4 Enabling Grids for E-sciencE

The Enabling Grids for E-SciencE (EGEE) project [90] consists of a series of projects funded by the European Commission to build a secure and reliable Grid infrastructure to share computing resources. It does not develop new software; rather, it re-engineers a middleware system based on existing solutions and systems. The produced middleware is call gLite (as described in 2.3.2). EGEE supports users in a wide range of research domains.

EGEE originated from Europe and work generated by the LCG project, and later incorporated members from not only Europe, but also the US and Asia. From a user's perspective, they benefit from EGEE in terms of simplified and secure access to on-demand and large-scale computing resources, as well as the ability to easily share data

and software to other researchers. For grid operators, EGEE means a unified and easy-to-maintain system within a collaboration of regional counterparts. In longer term, EGEE will continue to seamlessly integrate evolving IT technologies and make it available to researchers.

2.7.5 UK National Grid Service

The National Grid Service (NGS) project [91] in UK was initiated with the goal to enable access to geographically distributed IT resources by all UK researchers. NGS aims to provide a common infrastructure to computational and data based facilities across the nation. Basic services offered by NGS include data storage, data transfer, data management, data integration, computational resources, virtual organization management and information registries. These services are tested thoroughly to ensure the stability, and 24-hour user support is provided to solve emergent problems.

The core software stack of NGS is based on VDT. To aid the use of resources and offer a uniform interface to users, the web-based National Grid Service Portal [92] has been developed with single-sign-on through MyProxy. The portal is built upon the JSR168 portlet technology, with functions including MyProxy management, GRAM job submission, file transfer via GridFTP and MDS resource discovery. The recent ShibGrid project [81] integrated Shibboleth into the portal and simplified the login process for users. However, as developed in 2005, the portlet technology is somewhat out-dated, and the project has finished thus no support can be obtained, so it is not straightforward to adopt it into another environment.

2.8 The Cloud

The concept of Cloud [93] has become a hot topic in recent years. Although definitions from different people vary, they share some basic attributes. Firstly, resources are available on an on-demand basis, which is, users can request new resources if needed, and give back if they are no longer required. The appearance of a cloud is of having essentially infinite computing resources, and users pay for what they are using for any length of time. When compared to the Grid, the cloud has a similar vision, which is to manage large-scale resources for users to discover and use, but the cloud solves problems by adopting the virtualization technology, to enable much easier access from users than the Grid and to open opportunities to solve old problems in a new way. For example, as traditional grid systems are hard to scale up without huge effort and resources, some research has been conducted to execute grid jobs on the cloud [94]. Cycle Computing has also built a Condor-managed 30,000-core cluster on the Amazon EC2 cloud [95], which can be used by Grid users in the traditional way but can easily scale up or down according to user requests.

Broadly speaking, Cloud services are provided on an on-demand basis, in an "anything-as-a-service" (XaaS) model [96]. The "anything" in XaaS can be communication, infrastructure, software or platform, etc., and some of them are not new to us. CaaS (Communication as a service) is a generic term for the telephony services and VOIP (voice-over-IP) services that enable long-distance voice and video communication. SaaS (software as a service) appeared a decade ago, and refers to Application Service Providers (ASP) that host software for customers to access from anywhere. One typical

example of SaaS is salesforce.com, which offers a world-class CRM (custom relationship management) package. CaaS and SaaS are mostly user-level and user-oriented services that do not provide any mechanism for users to manage low-level resources. On the other hand, IaaS and PaaS evolved in recent years to offer a way for users, especially developers who know how and want to set up large-scale applications from system level, and this ability is essential to high performance computing or high throughput computing on the Cloud. According to a survey done by Choi et al. [97], IaaS and Paas services are the most common forms of Cloud computing nowadays.

Infrastructure-as-a-Service (IaaS) provides computer infrastructure as a service, with high flexibility, and availability. The payment scheme is also flexible for commercial cloud providers, which allow customers to pay for the amount of resources on an hourly usage basis, and can provide more resources to users in minutes. Amazon EC2 [98] was the first commercial public cloud system, whilst Nimbus [99] and Eucalyptus [100] are two famous open-source cloud systems. They all provide a web-service interface for users to dynamically create and remove virtual machines based on virtualization technology.

Platform-as-a-Service (PaaS) provides software platforms to developers to write their systems and deploy on such platforms. Google App Engine (GAE) [101] is a PaaS service for Java and Python web applications. Developers use GAE APIs to write their code, and deploy on GAE, while GAE handles load-balancing, when the number of users goes up, and fault-tolerance, to ensure stability. GAE also has a proprietary database, BigTable, to store user data, with proper backup.

In the future of my research, it is an interesting topic to expand the current grid-based architecture into the public cloud or cloud-based resources. For example, it would be interesting to find out if batch jobs can be sent to cloud resources to execute.

Chapter Three: Motivation and system architecture

3.1 Motivation

Over the past few years, researchers have been struggling with how to store their data easily, safely and permanently. A massive amount of data is collected everyday from instruments or experiments, and generated by running simulation programs. This data is essential to researchers' study and publication. They may use this data in several ways. For instance, some may want to further process raw data collected from instruments, with desktop PCs or supercomputers. Some may want to store data somewhere for years as archives, current practice is to use a DVD or just keep data in a desktop PC, which is vulnerable to physical damage or disk failure. In additional, it is hard to share data with other people, such as their local colleagues, let alone collaborators at other institutions. As some data is too big to send as email attachments, often they burn a DVD and send it by post.

On the other hand, to support research activities, governments and universities have invested a significant amount of money on storage facilities, located in regional centres. These facilities have terabytes or even petabytes of storage, which is provided to local researchers free of charge or at a very low cost. They often have a tertiary storage device, with proper backup. This is the ideal place to keep valuable data in research. However, there are several obstacles for a researcher to make use of these facilities. Firstly, these systems are typically not easy to use, and often provide only a command-line user interface. This requires users to have some degree of IT knowledge. In some

cases, system administrators need to assist users with data upload or management. Secondly, user accounts are normally created for local users, which makes it hard to give data access to external users. Thirdly, data transfer often goes via ports that are normally opened to the internal network only and blocked by university firewalls for access from outside (some universities do offer VPN software for their users to access from outside, but it is much too slow for data transfer). This makes it hard for users to access data from home, overseas when they are attending conferences, or anywhere outside their office. It is not convenient in many cases when users need to work remotely and want to access their data, e.g. in a coffee shop. Lastly, as storage facilities are distributed in many cities, some national research groups want to take advantage of several of them because they have offices and staff in several locations and want to share data among them. It would be very troublesome for them to remember which storage facility has what data, and the credential to access each storage facility.

Therefore, it is necessary to build a middleware system between the end-users and the existing storage facilities. This middleware should provide a modern and easy-to-use user interface to users. Given that storage facilities are distributed, it is essential to have the middleware interface with these resources, such that users are provided only one single system and can do a variety of operations from anywhere, without needing to know the details or the location of each individual resource. I have studied different distributed file systems as described in chapter 2, and have extensive experience with SRB/iRODS for several years. After considering the available features and the requirements, iRODS has been chosen as the base of the middleware, because iRODS is a mature system derived from SRB with advanced features, and it can take advantage of

distributed storage resources and provide a single view to users. At the time this project was started, iRODS provided the backend functionality that was required by the middleware, but it did not provide a good selection of easy-to-use interfaces for users. Thus, to assist users with using the system, I have developed Davis [102], a WebDAV and web interface for easy connection from users' desktops, as well as Griffin [103], a GridFTP interface for integrating with other grid systems. A brief introduction to these systems will be given later in this chapter, and details will be given in chapters 4 and 5.

The story is similar when considering compute resources. Investments have been made to set up regional supercomputer centres, for local researchers to run data analysis programs and simulation programs. These regional centres often provide an interface to the local batch systems or a grid middleware with web service interface, such as Globus Toolkit. The intention in offering these interfaces is to help users take advantage of the compute resources as much as possible, enable use by national collaborators, and share these resources to remote users when the resources are free, but there are still several obstacles stopping users from using them. Firstly, the existing interfaces are too low-level, either web service interfaces, or console-based command-line interfaces, which require users to have some knowledge of IT and scripting, or even programming. Secondly, the existing resources are distributed, but user authentication is managed by local system administrators, so different user policies are enforced. If a user wants to use multiple distributed resources, he or she may need to apply for memberships in different authoritative domains to get access. Thirdly, network configuration is different for these regional centres, some may have strict control and others may have loose control. Fourthly, it is very hard to find out what resources are available at each regional centre, in

particular, the available space for storage, the installed applications, and the usable compute resources. Lastly, storage and compute facilities work separately, and development is required to get them to work together.

Much work has been done to deal with these issues, as described in section 2.3 and 2.7, but they are either limited in functions, or not easily adapted to a different environment. The traditional grid infrastructure, such as Globus-based grid systems, relies on X.509 certificate technology for authentication, and offers a web service interface to users. It has been designed to be an entry point to local supercomputers, and aims to provide a uniform, high-level programmatic interface, while user interface is not the focus of such grid middleware. This results in different groups developing customized user interfaces, such as the systems we discussed in section 2.7. Consequently, when a new group wants to build its own grid infrastructure, some development has to be carried out to meet its own requirements.

As none of the existing applications can provide all the required features, and it is not easy to adapt existing applications into a different grid environment, in this project, I have implemented a job submission gateway, which provides a uniform interface to all underlying compute resources. The gateway still depends on an existing grid middleware, via which it communicates with local batch systems. From a user's perspective, they only need to interact with this gateway, and do not need to know any details of what is behind it. The system accepts job parameters as input, and distributes the job to an appropriate resource for execution. Based on the existing grid middleware, I developed a few components to facilitate job submission. Firstly, I developed web 2.0 interfaces as the main entrance for easy use of distributed storage and compute resources. Secondly, I

employed Shibboleth as the user authentication method to avoid unnecessary user credentials. Thirdly, I built a resource broker, which is an abstraction layer between the user and the real resources, so that users don't need to know where the resource is. Lastly, I enabled integration of compute resources and storage resources so that users can make use of them from a single interface.

Such a solution to these issues is generic but is deployed and tested in the context of the Australian Grid, which is built and managed by the Australian Research Collaboration Service (ARCS). Over the past few years, ARCS has been looking for a way to improve their services. They have a good amount of compute and storage facilities distributed in several regional data centres. There is a member facility of ARCS in each state of Australia. Each member hosts a few supercomputers, and hundreds of Terabytes of storage. However, not many people know about the details of all the compute facilities or how to use them. This is indeed a challenge to many grid service providers in the world [104, 105, 106], who want to attract more users but fail to do so due to a variety of reasons, such as the absence of user-friendly interfaces to these services, the lack of robustness and stability, inability of handling multiple resources, and failing to provide a modern user experience to most users including those without IT background.

This chapter provides an introduction to the complete grid infrastructure in layers. Then, the details of architecture are given to present the overall picture of how these components are developed to fit into the infrastructure and interact with each other. Finally, I will outline the contributions I have made to this project.

3.2 System Architecture

A general grid infrastructure includes network facilities, hardware, operating systems, middleware and user interfaces. According to the services provided, a different grid operator may use a different set of middleware, or system software. This thesis mainly focuses on three aspects: data storage with an easy-to-use user interface, a fast and reliable data transfer mechanism, as well as a modern and web-based job submission gateway. I focus on not only how each component works individually, but also how they can fit into the infrastructure and work with other components. The components I have built are based on existing software that is chosen as the foundation of this solution, so that there is no need to develop from scratch, and therefore the useful features of this software are retained. The goals of this project are to provide an aggregation of distributed resources, and a uniform and easy-to-use user interface to these grid resources. This section first outlines each layer and component in the complete system structure, and then gives a high-level overview of the two main services of this system. Each component in this system is meant to be generic and conforms to open standards; in principle, any of them can be employed in another grid environment. For example, Davis has been deployed and is running in several organizations interstate and overseas.

3.2.1 The complete system structure

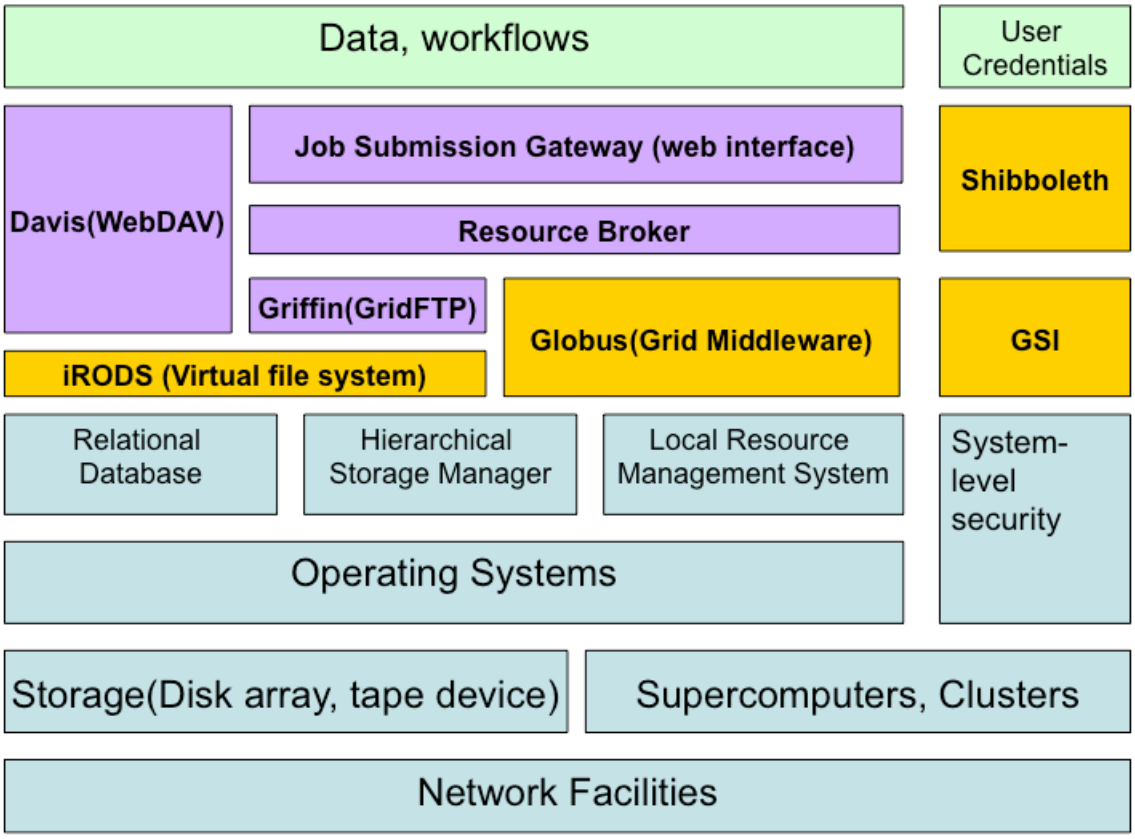

Figure 3-1 Grid Infrastructure Architecture

Figure 3-1 shows an overview of the whole infrastructure, in layers. The bottom layer, or network layer, consists of network facilities, including fibre, switches, routers, and some HPC systems may be equipped with high-speed local network systems, such as Myrinet [107] and InfiniBand [108]. Above the network facilities is the hardware layer: for storage, there are tape devices (tape silo, tape robot) or disk arrays; for HPC, there are different kinds of compute clusters, manufactured by SGI, IBM, etc., or desktop

computers with a special application installed, and used when their resources are free. On top of the hardware layer is the operating system layer, including all system level applications and security means. This layer is the bridge between applications and the hardware. To-date, most of the grid systems are based on various Linux operating systems, such as CentOS, Debian and Scientific Linux. There is no fundamental difference between them, as they are derived from the same core. As such, it is common to find a particular application packaged for several widely-used Linux systems. At this level, security rules are applied by the operating system on the hardware device, and this forms the base of the application-level security method, which is, any application security management task is mapped to a relevant system-level security process. System-level security policies can be applied on different aspects. For instance, user management system deals with local Linux user accounts and groups, which is normally stored in a user database. The user database can reside in the file system, or externally in a LDAP server, or a Kerberos server. The user management system decides who can log in to the system and what credentials they can use. Aligned with the user management system are file system permissions. Comprehensive permissions are set on each unit, a file or a directory, in the file system, to specify whether a particular user can read, write or execute (if executable) a particular file. The permissions ensure a user can only do what the user is allowed to, and will not interfere with other users. In addition, network management systems have firewall policies to block or allow certain IP ranges so that only trusted users can have access.

To provide a universal view to local resources, on top of the operating system level, several resource management systems are deployed to provide basic functions. At this

level, a relational database system provides a storage for tabular data and structured data; a hierarchical storage manager, such as SGI's Data Migration Facility (DMF) [109], provides management and administration of backend tape devices and disk arrays, such that various storage spaces can be accessed via a uniform interface, such as NFS; a Local Resource Management System (LRMS), usually known as a local scheduler, is deployed as the front-end of HPC systems, which normally consists of a head node, used to accept jobs and allocate resources via the LRMS, and worker nodes, where jobs are actually run. Nowadays, there are different kinds of LRMSs, as mentioned in chapter 2, some of them run on commodity servers, while some can make use of normal desktops (e.g. in student labs) when these computers are not being used.

The basic infrastructure, including the layers discussed above, is the foundation of this project. It exists and provides services in each regional grid operator. Our focus in this project is to create a federation of these regional grid operators and provide a single interface to users so that these resources from each individual operator are shared by users from a wider area. To set up a federation, it is usual to implement a grid middleware, on top of the local resource management systems, as illustrated in Figure 3-1. Chapter 2 has investigated several such middleware systems, and some of them are used in this project to provide the base of our federation. In particular, for storage solution, iRODS is chosen as the data grid middleware, because of its ability to interface with difference storage backend, and the flexibility of being able to define customized rules to manipulate data. iRODS is a virtual file system that stores metadata in a relational database and physical files in a remote file system, such as a NFS mount point of a hierarchical file system. For computation, the Globus Toolkit, the de facto standard

in the grid community, is adopted. As a comprehensive grid middleware, Globus provides components ranging from job submission and allocation, to the management of data replication. However, only a subset of all the functions are used in this project, which are the components that accept jobs and redirect jobs to a proper backend local scheduler, and relevant security components as well as information publishing components. Globus Toolkit also interfaces with local shared file systems, which are also accessible by local schedulers. Input files will be placed into these file systems so that local schedulers can read and process them. Also the output files will be written to the same place too, for Globus to retrieve after jobs are processed. These grid middleware systems, iRODS and Globus, both support the GSI security mechanism for authentication. As mentioned in Chapter 2, GSI utilizes the standard X.509 certificate specification and public/private key technology to identify users and maps users to system-level identities. It alleviates the pain of remembering passwords of different systems, instead, one can present a single certificate to the system and the system can map this user to a proper system account, based on the configuration. GSI also makes it possible for a user to delegate privileges to a service so that the service can act on behalf of the user, talk to another system, and so on.

The grid middleware provides a basic platform for higher-level user interfaces; however, these interfaces are too low-level and suitable only for administrators or advanced users. They cannot fulfill the requirements of having a modern web interface with Shibboleth support. In Figure 3-1, on top of iRODS and Globus Toolkit are the components I have implemented in this project. These components include a web interface for iRODS, called Davis [102], a GridFTP interface for arbitrary data sources,

called Griffin [103], and a job submission gateway with a modern web portal and a resource broker, called Grisu2. Davis and Grisu2's web portal are protected by Shibboleth, a web-based single-sign-on technology for users from one organization to access resources in other trusted organizations. Likewise, Shibboleth identifiers will be mapped to the underlying GSI identifiers or the system-level user accounts. The top layer of the figure includes data, workflows and user credentials. They are the real users who make use of the entire grid infrastructure to do simulation, data analysis, and data transfer, etc. in their daily work.

Next, I will focus on the two main services, namely the national file system and the national grid submission gateway. I will also specify my work in particular.

3.2.2 The National File System

Data storage is crucial to many users during their research work, especially those who collect massive data from instruments, experiments, or generated by simulations. Normally, this data is stored in someone's desktop, or in a USB drive, or burnt into a CD/DVD. This may be enough if the user only wants to use the data just by himself/herself. However, when it comes to sharing, such as sharing the data to other researchers in the same research group, or to researcher in another university, or even on the other side of the globe, it would be handy to use a system that can do this easily for them, rather than having users do this manually. Moreover, keeping files in hard disks does not guarantee the data can last for the required length of time. Often data is lost due to hardware failure, or not being taken care of properly.

In order to help researchers to retain their data safely, access their data easily, and share their data conveniently, the National File System (or the Data Fabric) is built in this project to address these issues. It is based on iRODS [18], a virtual file system, and makes use of storage resources in each regional centre to store real data. With this setup, data can have several replicas in different geographical distributed locations, hence invulnerable to severe disasters. Even if one resource is offline for some reason, researchers can still put data in other storage resources, which provides a certain degree of redundancy and reliability. Once in the National File System, researchers can share data easily to users from another state, given that users from the whole country are managed by a central user system. The benefit of this architecture is that it can bring all resources from each regional centre into one big virtual and centrally managed system, so that these resources can be shared among all the operators. For example, some smaller operators may be able to contribute less space, but others may provide more; if the storage in an operator is nearly full, data will be automatically stored in another resource with more free space.

In the beginning of this project, iRODS only offered a basic command-line tool and a very basic web interface, which did not meet the requirements. As the iRODS protocol is a proprietary protocol, there is no commodity client that can be used. This is not like other standard protocols, such as FTP, which has plenty of commercial and open-source clients for download and use. Hence, to use iRODS as the base of this project, I need to develop a few interfaces to bridge the gap between iRODS and users or other systems. I developed two interfaces for iRODS in this project. In particular, to make it easy for users to access the storage resources available in iRODS, I developed a

WebDAV and web interface, called Davis [102]. I also developed a GridFTP interface, called Griffin [103], to facilitate the integration of iRODS with other systems, particularly grid systems.

The architecture of the National File System is shown in Figure 3-2, where distributed storage resources are shown at the bottom, each connected to an iRODS instance. All of these iRODS instances are configured as one iRODS zone, among which one is the master instance, responsible for managing settings, executing rules, and interfacing with users and other systems. My development, Davis and Griffin, are deployed as interfaces to connect to this iRODS zone. In particular, Davis features a web interface for users to access the National File System with a web browser, and a WebDAV interface so users can mount the remote file system as a local drive. Griffin, on the other hand, intends to provide an efficient and reliable way to transfer data from and to the National File System, by providing a GridFTP interface that supports parallel transfer and other features, and makes it easy to use these storage resources for grid compute jobs.

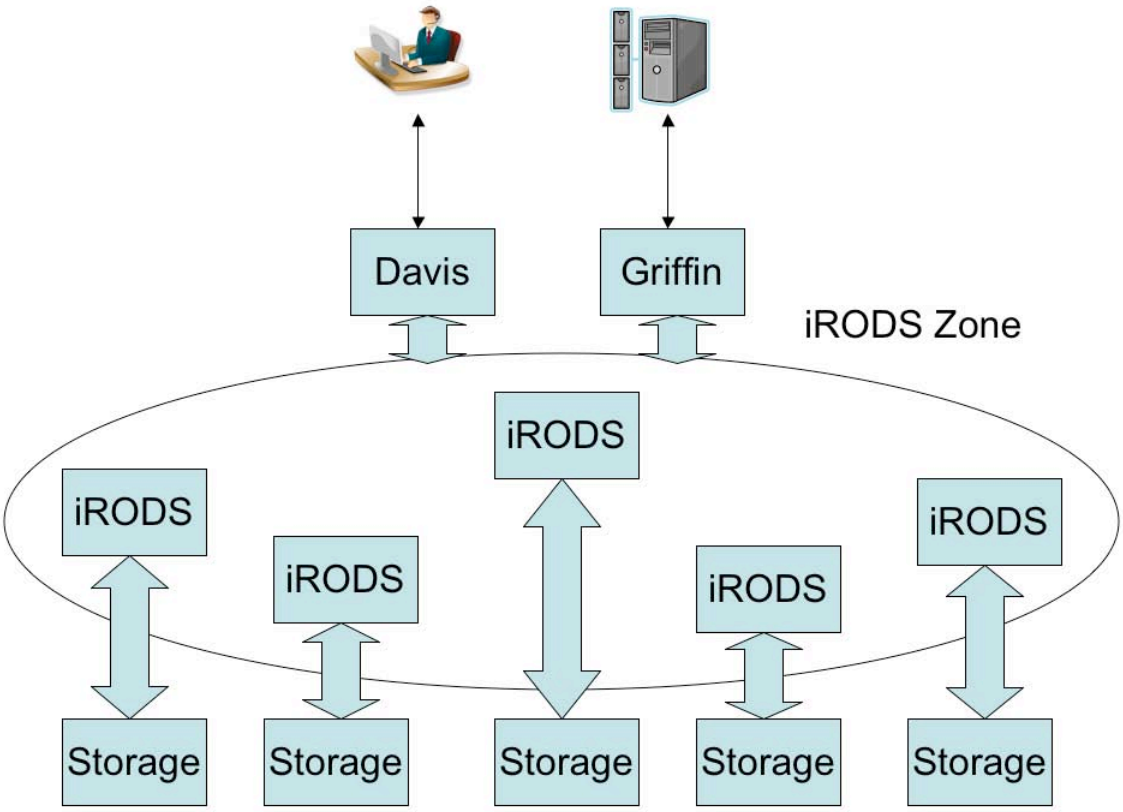

Figure 3-2 Architecture of the national file system

The GridFTP interface is the major component that bridges the National File System and the other main ARCS service (described in the next section), the National Grid Submission Gateway. The Gateway is mainly used for job submission and management, but it also relies on the National File System in terms of data staging and result data storage.

3.2.3 The National Grid Submission Gateway

Much contemporary research requires production and analysis of a large quantity of data. One example is the Large Hadron Collider (LHC), which generates multi terabytes of data every day. In the past several years, a large number of Grid systems have been set up around the world. Among them, some famous Grid systems are designed primarily for a specific discipline, such as the LCG [110], and Earth Science Grid [111], while some others are meant for general purpose, such as TeraGrid [112]. These Grid systems are mostly based on a Grid middleware, which is able to interface with different local resource management systems (LRMS) (sometimes are called batch systems). Several of them are being used widely and should be noted. For example, Portable Batch System (PBS) [113] is a batch system often used in conjunction with Unix/Linux cluster for batch job scheduling. It receives compute jobs from users and allocates resources in the cluster to process these jobs. The Globus Toolkit [114] is a grid middleware system with several key components, such as a grid gatekeeper to receive jobs and re-submit to LRMS, a grid information system, a data access server, and a standard security infrastructure, etc. It provides a standard platform for applications to build upon, with remote APIs for users and developers, but these APIs are mostly low-level, for communication between machines. To make it easy for users to use, grid portals have been developed. Many of them are organization specific or discipline specific systems intending to fulfill certain requirements, such as the CMS (an LHC experiment) grid submission portal [115] that is built to process CMS data with existing CMS specific tools, and the LEAD portal [116] for atmospheric science research. These customized

solutions are useful for their purpose, but often they are hard to adopt and adapt by other organizations and for different applications. The UK's NGS portal [92] and GridSphere [117] aim to be generic grid portals that support job submission of a wide range of applications, file access via GridFTP and grid credential management. However, they are based on the JSR-168 portlet technology, which is mainly designed for implementing web portals by dividing a web page into smaller areas, but it is difficult to implement and not flexible to extend.

In Australia, one regional supercomputer centre has been set up in every state with each managing several supercomputers. In principle, each operator is funded to serve local users, who have the highest priority to use these resources. However, there are a number of research projects jointly funded by several organizations involving researchers in multiple locations. Consequently, any participant of such project should be able to use the resources of the supercomputer centres in these locations. In addition, it is usual that some operators have one or more resources that are often not fully utilized, but for other operators, all resources are almost fully utilized most of the time and there are still jobs being queued for some time. To enable easy collaboration and make the most out of the current investments, this project is set up in order to facilitate sharing of supercomputers amongst all legitimate users in the country.

The sharing of compute resources in different administrative organizations consists of two levels. Firstly, any user with a valid credential, if it is from one of the trusted organizations, should have single-sign-on access to all these resources. Secondly, once a user has access, they should have access to any of the available resources that they are authorized to use, regardless of the resource's actual location. Moreover, the current

interface of the grid middleware is somewhat complex to use, even for developers who have a good understanding of these systems. Therefore, it is required to have a single and simple interface to all these resources, with a meta-scheduler that can take advantage of available resources.

In response to the above requirements, and based on the experiences gained during the development of Grisu [118] (details will be given in chapter 6), a Grid-client framework for developing grid client applications, I developed a sophisticated grid submission gateway, Grisu2, which is a generic grid portal that allows submission of any kind of jobs and does data staging automatically. It was designed to be an Application-as-a-Service system, where applications are pre-installed and provided in such a way that users do not need to know the exact location where their jobs are executed. The National Grid Submission Gateway, a production service based on Grisu2, intended to be the one-stop-shop for researchers across the country to submit jobs, interacts with almost all components in the grid architecture, because a researcher needs to use HPC resources to run jobs, as well as storage resources to keep result files. It consists of a web front end and a grid resource broker backend, which uses Globus Toolkit for remote job execution and monitoring. All components are on the server, so users do not need to download anything before they can start using it. The web front end provides a web portal, a BES web service interface, and a RESTful interface. The web portal is a modern and easy-to-use web interface designed for users who want to submit and manage jobs with minimal effort. The BES web service interface [119] can be used to integrate with third-party applications. The RESTful interface can also be used for integration, and advanced users who know how to write scripts can send jobs from command line or other tools, without a

need to use any client libraries and without the firewall problems that often occur with desktop client applications directly interfacing to Globus. These interfaces read in and write jobs to the grid resource broker, which manages the whole life cycle of a job and keeps other related information. The resource broker, acting as a delegate of the user, communicates with a grid information system, from where it determines which compute resource is suitable to execute a given job, and then interacts with that resource to launch the job and get back results. This architecture also makes it easy to extend and expand; if the number of jobs grows, more nodes can be added to the system to accommodate more jobs; more applications can also be added to the system by configuring new templates and without any development work. The architecture of Grisu2 is shown in Figure 3-3.

Grisu2 is integrated with the Data Fabric, the national file system, via Shibboleth and GridFTP. It also relies on a few components from Globus Toolkit, which is the interface for Grisu2 to access resources in each regional supercomputer centre. In particular, the Monitoring and Discovery System (MDS) is an information system that receives system status from each regional Globus gatekeeper and supplies this information to Grisu2's resource broker, so that Grisu2 can determine which resource is the most suitable location to run a specific job. The Virtual Organization Management System (VOMS) [43] is used to keep mapping information that a user belongs to which Virtual Organization (VO), and this information will be used in the process of authorization. Access to applications or compute resources are given based on the VO. The Globus Gatekeeper in each regional centre uses the Grid User Management System (GUMS) [44] to map grid identities, such as X.509 certificates, to local UNIX accounts

and groups, which are configured with different levels of access in the local scheduler (LRMS).

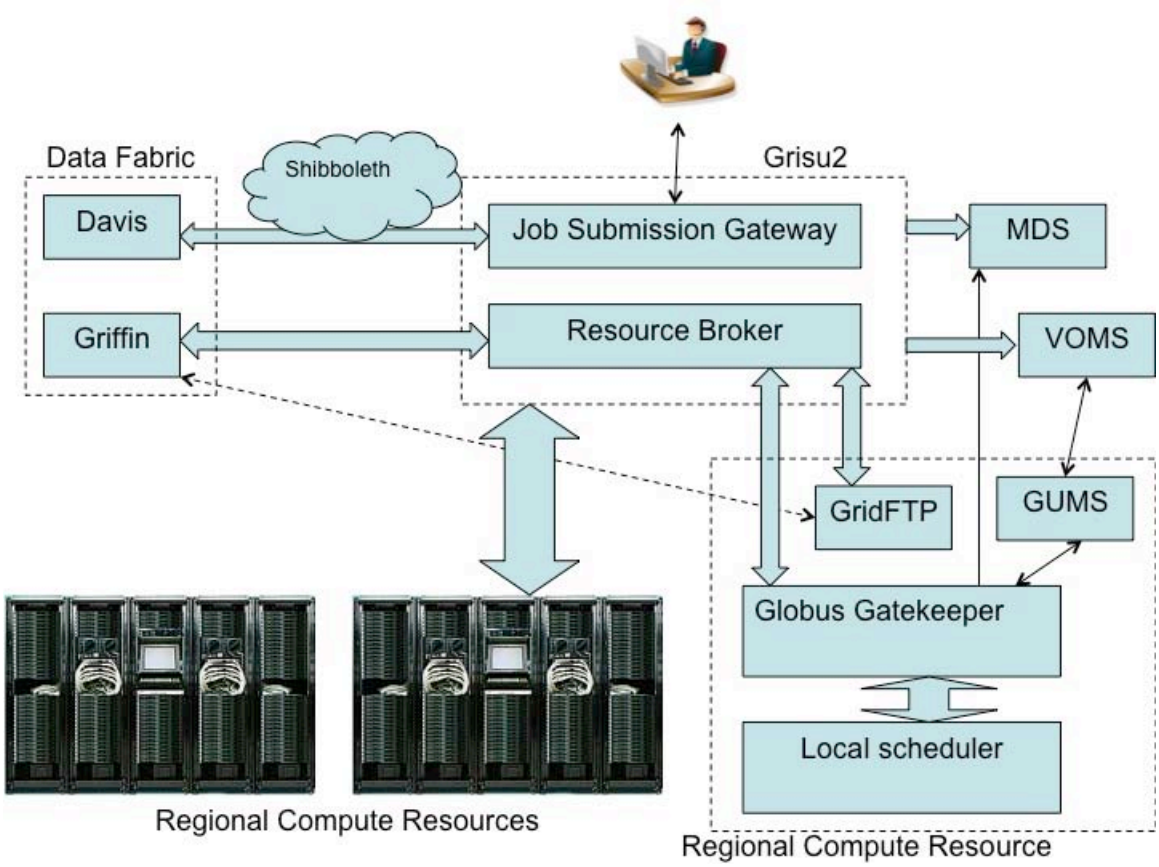

Figure 3-3 the architecture of Grisu2

3.3 My contributions

This thesis makes several contributions towards building a user-friendly grid system for a wide range of user groups: normal users with limited IT expertise, through to advanced users who can write scripts and application developers. I have designed and

developed several interfaces, and deployed them in the Australia Grid for testing, evaluation and production use. Details are listed as follows:

1. I have done a survey on the *Grid*, including computational grid and data grid, that compares major grid projects in the world in terms of their characteristics and features. This study builds a good foundation of the project so I have a good understanding of what is available, what can be reused, and what is missing.
2. I have done research on protocols and clients for data access, and presented the design and implementation of a data portal for accessing data in distributed and heterogeneous storage systems. The portal is built upon the iRODS data grid, and has a web/WebDAV interface that facilitates data access in iRODS. Being a complement to the fundamental data grid middleware, the portal provides various interfaces that are compatible with widely used software, so that users can choose one that they are familiar with to take advantage of the distributed data storage. Details of the portal will be given in chapter 4.
3. I have investigated different ways for efficient and large-scale data transfer, and presented a design of a GridFTP interface to the above data grid system. Traditional data transfer services are normally an external component to data sources, so if data sources on both ends of the transfer do not support the same protocol, data flows via the data transfer service host, which is inefficient and puts considerable loads on the service host. My approach tries to solve this in a different way, by turning any arbitrary data source into a standard and GridFTP-compatible data source, such that the features of GridFTP, such as third-party transfer and parallel transfer, are enabled and used for data transfer.

With the third-party transfer feature, data can travel directly from one end to the other, without going through the data transfer service host. One use case to substantiate this idea is to apply the GridFTP interface to the data grid middleware, iRODS, and this enables the possibility and operability of moving massive data in and out of the data grid, and across to other data systems efficiently and reliably. Details of the GridFTP interface will be given in chapter 5.

4. I have explored mechanisms that help users run massive concurrent jobs when they are exposed to a number of HPC systems with various pre-installed applications, and web portals that allow easy access to HPC systems without needing to know the details of how to run jobs on each system. Then I presented the design and implementation of a national grid submission gateway. It provides several different web interfaces for users with different levels of IT skills, to enlarge the grid user base. These interfaces include a Shibboleth-protected web portal that supports different applications by adopting a template system for users with basic IT skills to easily send jobs to HPCs, a RESTful interface and a web service interface for advanced users to write scripts to run massive jobs or write third-party applications to submit jobs to HPCs. On the backend, the gateway has a grid resource broker that can redistribute jobs to multiple grid resources with automatic data staging. A few case studies are also conducted to prove the usability of this submission gateway and show its importance in users' daily work. Details of the gateway will be given in chapter 6.

Chapter Four: Davis, a web/WebDAV interface for iRODS

4.1 Introduction

Data storage in a Grid environment often comprises of geographically distributed storage resources. Making use of these resources easily and efficiently is a challenge. As described in chapter 2, various cluster file systems and data grid middleware systems have been developed in order to provide a uniform interface to heterogeneous resources. However, these middleware systems often provide low-level interfaces, which are not easy for normal users to use. My goal in this project is to find a simple, standard and easy-to-use interface for the data grid middleware that is chosen as the backend. Section 3.2.2 outlined the infrastructure of the National File System, which has a distributed file system backend with a user-friendly interface, called Davis. This chapter now explains the details of Davis, an important component and the main user interface of the National File System. In the context of this chapter, iRODS is used as the middleware of the National File System, because of the advanced features of iRODS and the expertise gained from several years experience of using SRB, iRODS's ancestor. The native interface of iRODS is not very user-friendly, and more for administrators and developers. I also did a survey on all available SRB/iRODS interfaces, and some popular data access protocols. The conclusion I drew was that existing iRODS interfaces are not sufficient or suitable in this project, thus a new user interface should be developed to fill in this gap. Davis is the new user interface I developed, which is a web-based interface for

iRODS/SRB, in order to provide a simple and easy-to-use interface to users with different levels of IT skills.

Chapter 2 gives a comprehensive introduction to SRB and iRODS. Here I give a brief overview of them as a reminder to the reader, as they are crucial to this work. The SRB project, developed by San Diego Supercomputer Center (SDSC), started in 1996 with a goal of reading from different data sources via a single data access interface [120]. Two years later, the first version was finished and released with a revolutionary new software architecture, which has not been changed since then. It has two major parts: a uniform data access API and a metadata catalogue [17]. The uniform data access API is used to read data from heterogeneous distributed storage resources, such as file systems, database systems, and archival storage systems. On the other hand, the metadata catalogue uses a relational database to store metadata of remote data items and user information. It also provides a virtual view of these items with user-defined collections (like directories of a file system). From a user's point of view, SRB is a virtual file system, transparent to physical data locations, and acts as a single access interface to mass data repositories.

In 2006, iRODS [18] was developed as a successor of SRB. iRODS inherits SRB's architecture, but with an enhanced design and additional functions, the most important of which are a rule engine and micro-services. With the rule engine, customized strategies and user-defined workflows are possible [121]. Note that iRODS is open source under a BSD license, while SRB is not.

Over the past few years, SRB has been used extensively to provide an interface to massive data for many research communities around the world and has become an

essential component of many data grids. For example, a few UK projects use SRB [122]: the Atlas Data Store keeps petabytes of data in tape and provides a SRB front-end with disk caches for fast data access; the Diamond Light Source project registers data on the beamlines into SRB so that this data can be synchronized to a central storage periodically and stored properly; Daresbury SRS project uses SRB for users to query metadata; the Arts and Humanities Data Service stores and manages large distributed data sets in SRB [121]. In the US, SDSC manages a number of shared collections of up to 126 million files and 830 TB for several projects, including data grids, digital libraries and persistent archives [67]. Recently, iRODS has gained more interests as a replacement of SRB, and many of these projects have migrated from SRB to iRODS.

iRODS/SRB developers provide a few basic client tools, such as Scommands/iCommands, InQ/iRODS Explorer and MySRB. Moreover, many big SRB user groups developed their own portals for SRB. However, the iRODS/SRB protocol is a proprietary protocol, which makes it hard to develop a customized client tool or use general tools to access iRODS/SRB. Therefore, it is not easy for general users to use or for developers to integrate with other software. Furthermore, these tools either have limited functionality, or are developed for a particular group.

In order to enable more people to use iRODS/SRB, it is necessary to explore other solutions, including other popular data transfer protocols and best practice for user interfaces. Among the available open standards, WebDAV is an open standard based on HTTP protocol, and extensively supported by many operating systems and applications. The use of WebDAV is as simple as using a file system, even to a user with very basic knowledge of computer systems. Additionally, with a broad range of software support

[123] and the standard HTTP protocol, the integration with other software is simple. Thus, the combination of WebDAV and iRODS/SRB would benefit not only the end users but also the developers.

Next, I will briefly introduce the system structure of iRODS/SRB (more details can be found in chapter 2), with emphasis on authentication methods, as they are one of the major requirements in this project. The following section investigates their client tools and portals. Section 4.4 gives details of a few popular file transfer protocols for comparison. After a description of the state of the art, reasons are given for the choice of WebDAV in this project. The detail of my implementation, a WebDAV gateway to iRODS/SRB called Davis, is illustrated in Section 4.5 and some use cases are described in Section 4.6. Lastly, this chapter ends with a conclusion of the current work.

4.2 An overview of SRB and iRODS

In principle, SRB and iRODS are designed as a Virtual File System, with a similar structure: a metadata catalogue and a file access API [124]. The metadata catalogue (MCAT of SRB or ICAT of iRODS) stores name-value pairs (for SRB) or name-value-unit triples (for iRODS) in a relational database as metadata associated with an object in iRODS/SRB, and presents mechanisms to save, delete or query metadata. The file access API offers a virtual file system for users to access remote data in a similar way to using local file systems, and it is configurable to get access to data in various storage media, such as a remote NFS-mounted file system, a tape device, a disk array or even a zip file. On top of that, iRODS provides a rule engine, which accepts pre-defined

sequences of actions to be executed in particular events like a workflow system [125]. Note that, iRODS/SRB uses a proprietary protocol requiring a specific port, which needs a dedicated iRODS/SRB client for communicating with it.

By default, iRODS/SRB only supports username/password and GSI authentication. Recently, Shibboleth [75] has evolved to become a major authentication method in education and research sector, with support from many open source software tools, such as Sakai [126, 127]. The benefit of using Shibboleth is that users can use only one set of username and password to access different systems. However, iRODS/SRB itself does not support Shibboleth directly. One possible solution is to use a Short Lived Credential Service (SLCS) server [82], which is also used to provide Shibboleth authentication to Globus services [128]. The SLCS server generates a complete certificate (with private key and public key) for valid Shibboleth Identity Provider (IdP) users. This certificate lasts only eleven days by default. After that, a new certificate has to be requested again. In the login process, the user requests a SLCS certificate with an IdP credential, and the SLCS server issues a certificate to the user. Then the SLCS certificate is submitted to iRODS/SRB for GSI authentication, as shown in the following diagram.

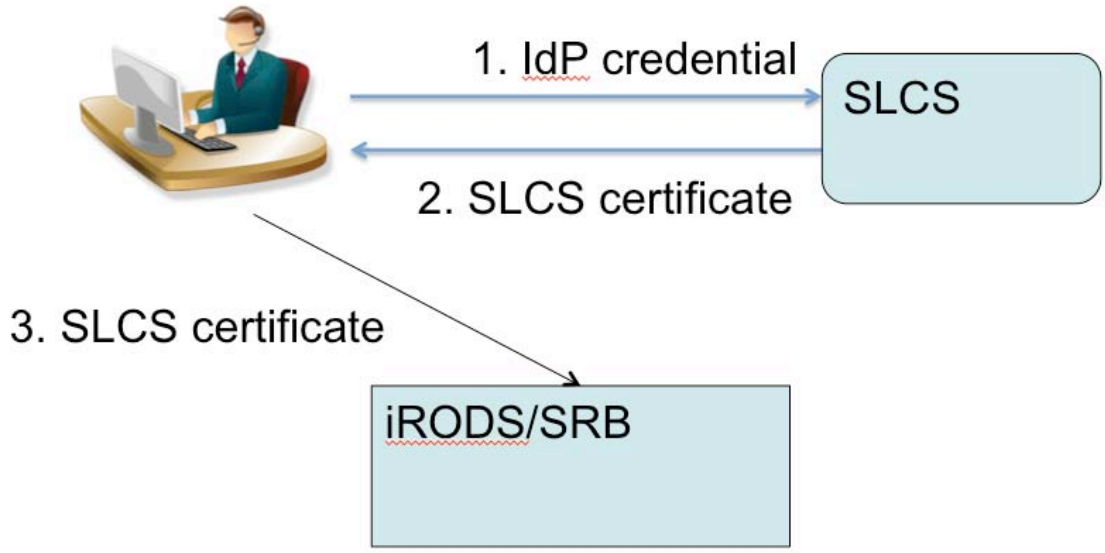

Figure 4-1 GSI login with SLCS certificate

4.3 A survey of existing client tools

There are a number of existing iRODS/SRB tools. Some of them are developed by the iRODS/SRB developers, and others are developed by the user community. This survey[1] looks at the major functions of each tool. In particular, the authentication methods, supported operating systems, user interface and the underlying technology are emphasized. At the end of the survey, a summary will be given to justify my choice of WebDAV over other options. Generally, these tools can be categorized into the following types.

[1] This survey was done in 2009 as a precursor to the work on Davis, and since then some of the client tools have been improved

4.3.1 Command-Line Interface

iCommands [129] are a group of command-line utilities written in C and developed by the iRODS/SRB developers. They provide client tools to normal users, including functions of data transfer and metadata management, with versions for Windows, Mac and Linux. iCommands are Unix-like commands with names starting with ‘i’ to access iRODS objects and metadata. For example, icd acts like cd of a Unix system to change current directory. It accepts password or GSI authentication. iCommands are also handy to system administrators, providing system level functions for management of accounts, zones, and resources, as well as backups and synchronization for objects. Similarly, **Scommands** [130] are a set of command-line utilities for SRB with the same purpose as iCommands, providing similar functions.

Java CoG Kit [131] is a set of command line tools written in Java for Globus. Particularly, its globus-url-copy command supports SRB protocol, which means a user can use srb://server.name/path/to/file as the source or destination of the command to copy files from or to SRB. This command allows password and GSI authentication, but it can only transfer files and has no metadata functions.

4.3.2 Graphical User Interface

InQ [130] is a native Windows GUI written in C++ offering a browser tool for data transfer (drag and drop) and metadata query on SRB, while **iRODS Explorer** [129] is the counterpart for iRODS. Both of them only support password login and can only run

on Windows. Moreover, iRODS Explorer does not allow drag-and-drop from/to desktop at the first version when we started this project.

Hermes [132] is a Java GUI jointly developed by the ARCHER project [133] and ARCS (the Australian Research Collaboration Service). It provides a rich client for data transfer and metadata management, with versions for Windows, Mac and Linux. It also supports different authentication methods, such as SRB account, GSI, MyProxy and Shibboleth. Hermes is based on Apache's Common-VFS [134], which provides a common API for accessing various file systems. The support for iRODS is under development.

JUX [135] is a Java GUI developed by CCIN2P3. It is based on JSAGA [136] and provides a browser view to SRB. It also supports preview of images and audio files. As a Java program, it can run on Windows, Linux and Mac. JUX doesn't provide functions for metadata manipulation. iRODS will be supported in a future version.

vbrowser [137] is another Java GUI client for SRB 3.0, developed by VL-e (Virtual Laboratory for e-Science) mainly for the research of functional Magnetic Resonance Imaging (fMRI). It is intended to be a single access point to the grid, and provides mechanisms to access SRB and GridFTP servers. It is based on VLET, which is an abstraction layer to various middleware components, including data resources [138]. It is also extensible, so that users can develop plugins to view and process files. For example, a workflow plugin was implemented to allow users to invoke workflows in vbrowser interface [138]. However, vbrowser doesn't have any features for metadata management, and it does not support SRB 3.5 or iRODS.

YourSRB [139] is another Java GUI aiming to offer SRB data access as well as fine-grained metadata control to users who require advanced functions, such as modifying metadata of multiple files in a batch. Users can create a metadata schema and apply that to all SRB objects. It also has a function for users to create a SRB federation. It does not support iRODS. The development of YourSRB is guided by the use cases of materials science and maritime archaeology research groups in James Cook University.

4.3.3 Web Interface/Portal

MySRB [130] is a web interface to SRB developed by the SRB developers. It also gives users a tool for data transfer and metadata management from a web browser, but it does not support drag-and-drop files from/to desktop. **iRODS web browser** gives similar functionalities to iRODS users. Note that it only supports iRODS username/password authentication without GSI or shibboleth.

SRB Plugin for Plone [140] was developed by the ARCHER project [133] and enables a SRB view in the Plone web content management system for file browsing/downloading and metadata management. This integration makes it easier to use SRB for other Plone components.

4.3.4 Other Interfaces

SRB-DSI [141] provides a GridFTP interface for SRB. It only allows file transfer without the ability of metadata management. The GridFTP interface makes it possible to

integrate SRB with Globus components. There is no equivalent GridFTP DSI for iRODS yet.

Jargon [129] is a Java API for developers and is capable of doing a variety of file operations and metadata manipulation. It does not have any user interface. Jargon is the base of all Java GUIs described above and of Davis. Jargon supports SRB and iRODS. **Prods** [129] provides a similar PHP API to iRODS.

4.3.5 Limitations of existing interfaces

To summarize, the above iRODS/SRB tools and other popular tools have limitations. Firstly, they are mostly custom clients specific to iRODS/SRB, and this requires users to learn their usage. Secondly, some iRODS/SRB client tools, especially web portals, are designed for a particular user group's needs, which makes them difficult to be adopted in a different environment; also some of them only support SRB but not iRODS. Thirdly, some only work for a particular operating system, and those implemented in Java can work across multiple operating systems but do not provide a native look-and-feel on the OS. Furthermore, they don't follow open standards, because the iRODS/SRB protocol is not an open standard. Additionally, iRODS and SRB use special ports, 1247 and 5544 respectively, which are not commonly opened on firewalls, so that these ports need to be opened for iRODS/SRB tools to get access. Lastly, it is not easy to integrate iRODS/SRB with other software, thus development is needed in many cases.

Therefore, it is necessary to look for another open-source-based and easy-to-use solution to overcome these disadvantages. According to the functionality presented by the current iRODS/SRB tools, as a guideline, the new tool must provide the following functions: essential file operations, such as creating, moving, renaming deleting files and directories; file permissions, such as granting/revoking a permission to/from an iRODS/SRB user on a file; metadata management, such as adding, deleting, modifying iRODS/SRB metadata; authentication, at least password authentication.

In order that iRODS/SRB can be used without modification, it is appropriate to implement a gateway that accepts an open standard protocol and sits in front of iRODS/SRB. The next section explores a few popular file transfer protocols and justifies my choice of WebDAV as a standard interface to iRODS/SRB.

4.4 Standard protocols and the choice of WebDAV

This section investigates a few popular and widely used file transfer protocols and tools, in the commercial sector or open source community. After comparing their features and network properties, WebDAV is chosen and shown to meet the above guidelines.

4.4.1 Popular file transfer protocols

File Transfer Protocol (FTP) is a basic and popular protocol for data transfer and distribution but does not support metadata. **Common Internet File System** (CIFS) [142] is an extension of the Server Message Block (SMB) protocol, and designed to request

files and print services over a local network for all systems, while **Samba** (http://www.samba.org) is a typical SMB server. Most desktop operating systems, such as Linux, Mac and Windows, have built-in SMB/CIFS client. However, SMB/CIFS doesn't have metadata features and the port it uses (445) is typically blocked by firewalls.

Amazon Simple Storage Service (Amazon S3) (http://aws.amazon.com/s3) is a service that offers storage to the Internet users. It exposes proprietary REST and SOAP interfaces so users normally need to use a client application to access, but it is easy for developers to integrate with their own clients. Objects in Amazon S3 are identified with a unique key, and can be shared to any user. HTTP and BitTorrent (http://www.bittorrent.org) are the two protocols for data download.

WebDAV [123] stands for "Web-based Distributed Authoring and Versioning". It adds a set of extensions to the HTTP protocol so that users can edit and manage files on remote web servers collaboratively. WebDAV is defined in an IETF standard, RFC 2518 [143]. It uses the term *resource* for a Uniform Resource Identifier (URI), and the term *collection* for a set of URIs. WebDAV employs the following basic HTTP methods:

- GET – get contents and details, e.g. modification time, of a resource (file)
- PUT – upload a resource (file)
- DELETE – delete a resource (file) or collection (directory)
- HEAD – retrieve details of a resource, e.g. modification time

  It also extends HTTP by adding the following methods:

- PROPFIND — retrieve properties, stored as XML, from a resource. It is also overloaded to retrieve the collection structure (directory hierarchy) of a remote system.

- PROPPATCH — change and delete multiple properties on a resource.
- MKCOL — create a collection (directory).
- COPY — copy a resource (file) or collection (directory) from one URI to another.
- MOVE — move a resource (file) or collection (directory) from one URI to another.
- LOCK — put a lock on a resource. Locks can be shared or exclusive.

The locking mechanism of WebDAV provides a way of serializing accesses to a resource, and prevents another user from writing to a file while it is being edited. The use of locking is decided by the WebDAV client. Moreover, there is a number of software products with built-in WebDAV client [123], such as Microsoft Office and PhotoShop.

4.4.2 The reasons for choosing WebDAV

Among existing open standards, WebDAV is the best option to meet our requirements. Firstly, WebDAV is an open standard, and most operating systems, such as Windows, Linux and MacOS have built-in WebDAV clients, which provide a generic interface that has the same look-and-feel as the OS itself, thus no extra software needs to be installed. Secondly, WebDAV uses HTTP(S) for any connection, which is commonly opened in most firewalls (if not, a HTTP proxy can be used). Moreover, it is easy and straightforward to maintain only one service for providing access to files via the web or desktop folders in a standard way. Furthermore, HTTP(S) interface is extendable, which makes it possible to implement functions not supported by standard WebDAV protocol, such as permission setting and metadata management. Lastly, it is easy to integrate

HTTP/WebDAV with other software, as many of them support HTTP natively. Also WebDAV resources can be mounted to any OS and appear as a normal file system. Software can read and write data in it without modification.

4.5 The implementation – Davis

Davis is a WebDAV gateway to any iRODS/SRB server regardless of its location. It has two major functions: the WebDAV service, and browser mode. WebDAV service provides basic WebDAV functions to WebDAV clients that support WebDAV version 1 and 2, such as the built-in clients of Windows, Linux and MacOS, as well as other third-party software. Browser mode enables users to use a web browser, such as Internet Explorer or Firefox, for using extra functions that are not provided by WebDAV clients. Basically, browser mode shows on a browser the details of an iRODS/SRB collection, such as creation time of files and file size. On that page, users can download a file, change file permissions and modify metadata. A screenshot of browser mode is shown in Figure 4-2, which shows a list of files in the user's home directory on the background and a popup dialog for managing metadata. The metadata dialog lists all metadata of the current object and allows users to add/change/remove metadata. Browser mode also has a permission dialog that shows a list of permissions of a file and provides a form on the right where the user can change the permissions of this file.

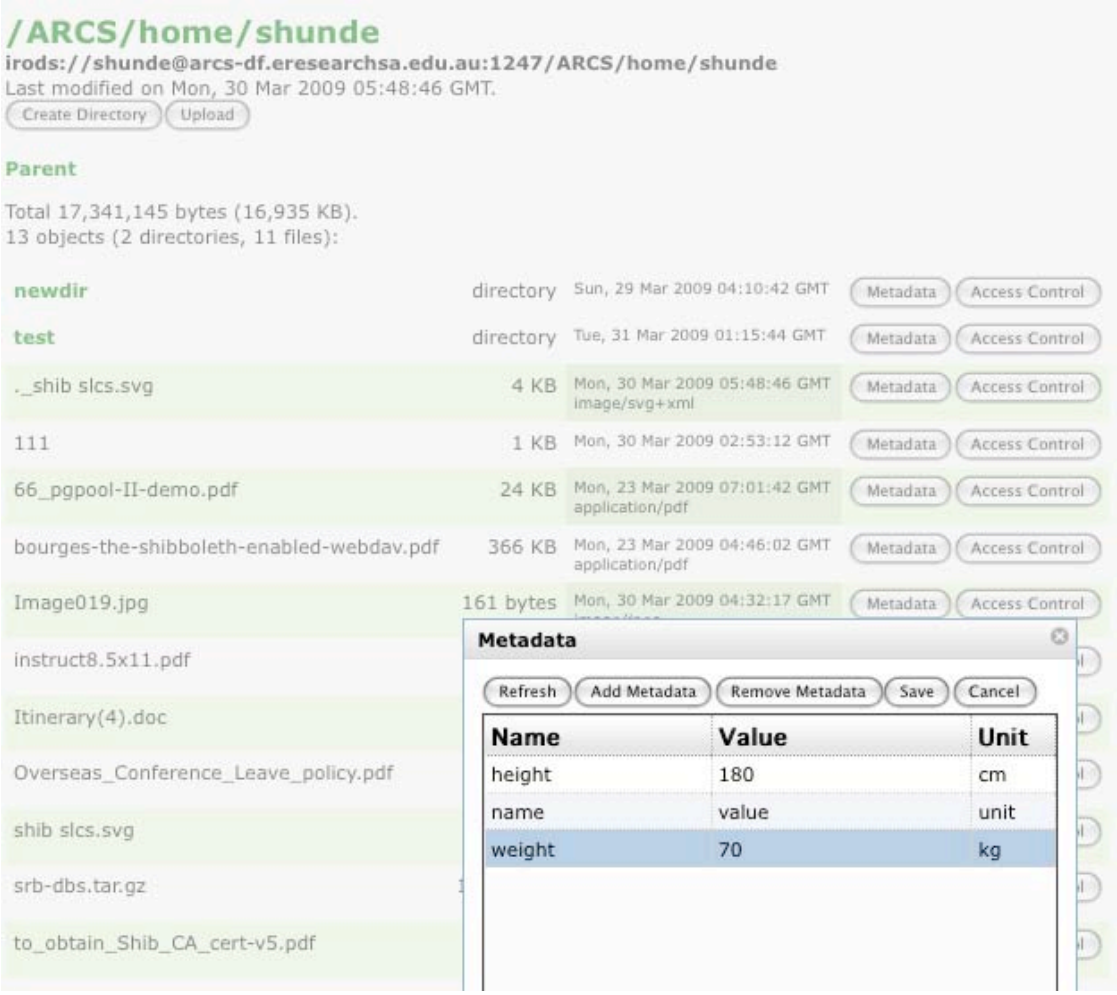

Figure 4-2 A screenshot of browser mode

The architecture of Davis can be described as three modules, as shown in Figure 4-3, and Jargon API as the bridge for these modules to communicate with iRODS/SRB. The first module, WebDAV handler handles all file operation requests from WebDAV clients that talk WebDAV protocol. Table 4-1 lists the implemented methods in Davis, as well as the corresponding iCommands, and iRODS/SRB operations.

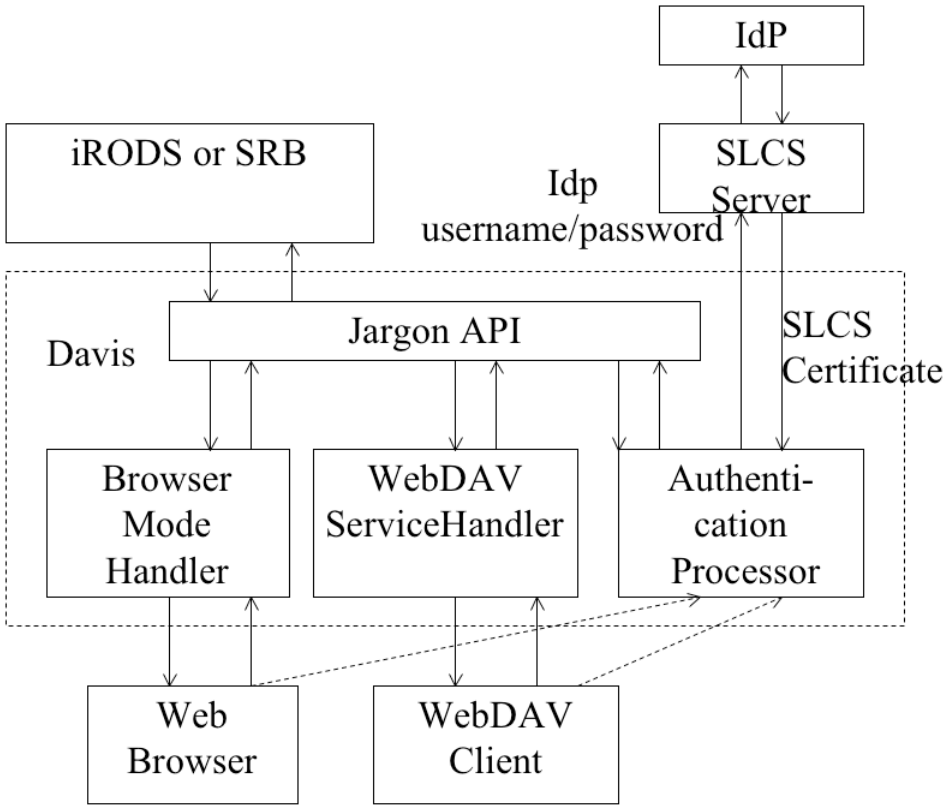

Figure 4-3 System structure of Davis

Browser mode handler provides web pages for metadata management and permissions setting. To simplify the structure, the asynchronous JavaScript and XML (AJAX) technology is exploited to convey requests from the web pages to Servlets. In addition, dojo toolkit (http://dojotoolkit.org), an open source modular JavaScript library, is chosen to implement the front-end web page as it provides extensible JavaScript widgets, including AJAX widgets, for web development. POST method is used by the AJAX widget to submit data from the web page to the handler, while GET method with a collection's URL returns a web page with the details of this collection.

WebDAV method	Equivalent iCommand	Davis operation
COPY	icp	Copy a file or directory; if directory, it recursively copies all child directories
DELETE	irm	Delete a file or directory; if directory, it recursively deletes all child directories
GET	iget	Download a file. If the target is a directory, it is assumed to be accessed by a browser so that a HTML page is returned
MKCOL	imkdir	Create a new directory
MOVE	imv	Move a file or directory to another place
PRODFIND	ils	Get details of a directory, such as sub directories and files
PUT	iput	Upload a file
HEAD		Return basic information about the specified resource, e.g. last modified time
OPTIONS		Get a list of supported methods for the target resource
LOCK		Lock a resource before editing
UNLOCK		Unlock a locked resource
POST		Not used by WebDAV; used by browser mode to communicate via AJAX.

Table 4-1 Davis operations

Authentication Processor is an extensible component where different login methods can be implemented and plugged into. In order to support different client types and different authentications, it does several checks in the login process, to identify an appropriate one for users. Note that not all authentication types are compulsory, and any of them can be switched off according to needs. The whole diagram is shown in Figure 4-4. Currently, Shibboleth is employed to protect HTTP requests. However, communications between IdPs and the SP are still using HTTPS; just the user data is transferred between the users and Davis via HTTP. If a user logs in successfully via Shibboleth, the user will be mapped to an iRODS user by the shared token attribute (which can be configured to use another attribute that is unique to all users) from the Shibboleth session.

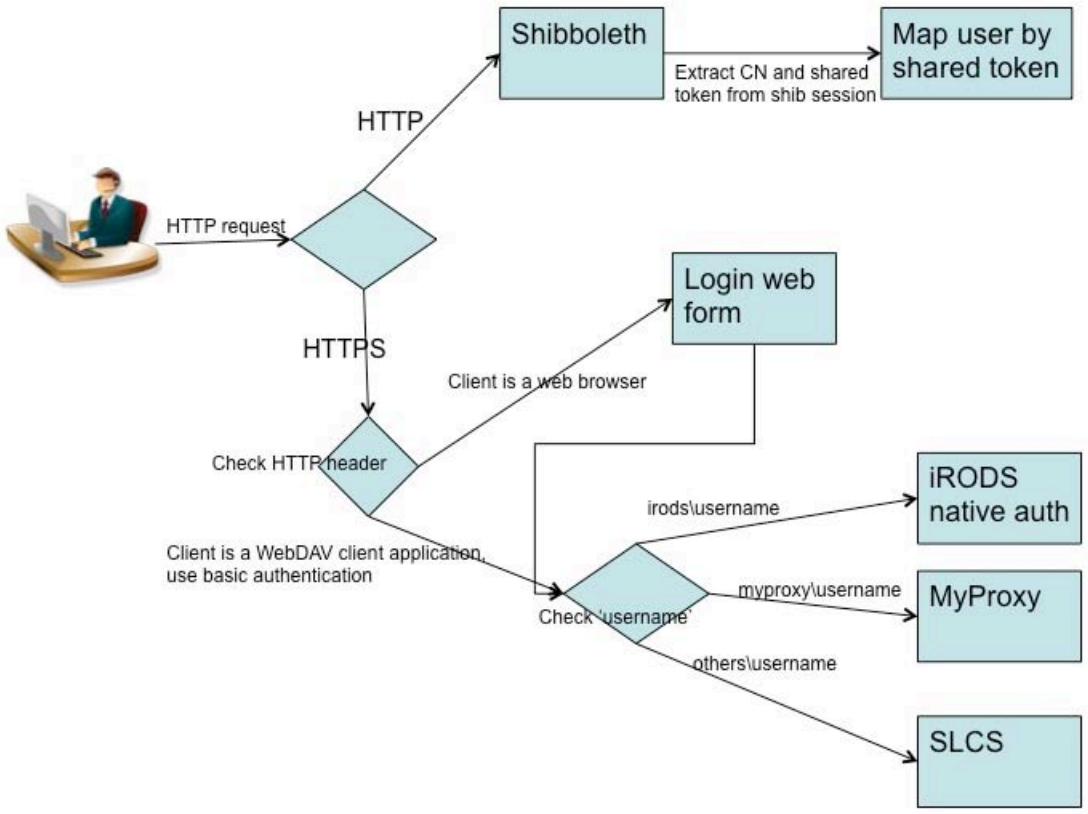

Figure 4-4 Login process

If users use HTTPS to access Davis, Davis checks the HTTP header to find out the client type: a web browser or a WebDAV client application. If the client is a web browser, a web-based login form will be displayed for users to enter username and password. If the client is a WebDAV client application, only basic authentication can be used, and HTTPS connection is highly recommended so passwords will be sent as plain text for basic authentication. When Davis receives the username and password from the client, either from the login form or basic authentication, the processor will examine the username string. To support different methods, the username is checked against the form

of ‘method\account’, where the prefix *method* is a key to locate and apply an appropriate method. Firstly, the processor checks if the methods is in the list of known and static values, for example, ‘irods’ means iRODS user system, and ‘myproxy’ means using a user proxy from a preconfigured MyProxy server. If none of these is matched, the method name will be used as an IdP name to fetch a SLCS certificate, then the certificate is used for GSI authentication on iRODS/SRB. If neither method nor backslash is presented, the authentication method that has been set as the default will be used. If no credential is found, an HTTP status code 401 is returned for users to enter username/password. As iRODS/SRB only supports username/password or GSI, all different authentication methods need to be translated into these two methods. For example, Shibboleth authentication mentioned above is actually GSI authentication to iRODS with the help of SLCS. After the authentication is successful, requests are directed to the appropriate handler according to the request’s HTTP method.

Data transfer using Davis benefits from the HTTP protocol. For instance, as an HTTP session is connection-less, there is no live connection between the client and the server, which saves server resources. In addition, the HTTP header *Ranges* allows the client to specify the starting point (offset of a file) of a file download, which enables resumable download and segmented download. Resumable file downloads make use of server resources more efficient by allowing a user to temporarily suspend the download of a large file and then resume the download later. If the network access is temporarily interrupted or if the computer is put to sleep, the user doesn't have to resume the download from the beginning when access is restored. Segmented downloads divide a big file into segments and start off multiple threads to download each segment. When one

segment ends, the client starts a new segment automatically. Downloading multiple segments concurrently can significantly improve data download speed.

4.6 Performance Evaluation

This experiment compares the data transfer performance of Davis with iCommands in terms of transfer rate in a local network. The aim of this experiment is to demonstrate that adding Davis to iRODS/SRB doesn't add too much overhead in the transfer, rather, it is an acceptable trade-off in order to benefit from the features users get from Davis.

The test environment was set up on a server with two quad-core Intel Xeon 3.16GHz CPUs and 16GB memory, running CentOS 5. iRODS 2.0.1 was used in the test with Davis on the same machine, and Davis runs on Jetty 6.1.12 with 1.5GB heap size JVM and accepts HTTP and HTTPS requests for this test. Using HTTPS is recommended in a production service, as it is the only practical mechanism for protecting user credentials. As a consequence, it also protects the data. The client is a VM with 512MB memory and two Intel Xeon 3.20GHz CPUs. Both the server and the client are in the same local network (1Gbps Ethernet). Six tests have been performed to transfer files of 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1024MB, and 2048MB. The six tests are, file upload with iput, file download with iget, file upload via Davis (HTTP and HTTPS) with cURL, file download via Davis (HTTP and HTTPS) with cURL. iRODS is configured to use 4MB window size and maximum 16 threads as these values are recommended by Yoshimi Iida (https://www.irods.org/index.php/Lyon-KEK).

The real number of threads is chosen by iget and iput according to the actual file size. If the file size is less than 32MB, one thread will be used. Other than that, 16 threads will be used. Each test is done 20 times and the average is shown on Figure 4-5.

The results show that Davis (HTTP) performs similarly to iCommands. Transfer via HTTPS is about 2-3 times slower because the overhead of SSL encryption and decryption. However, the transfer rate of HTTPS transfers is steady, mostly between 10 to 20 MBs. Similar HTTPS transfer tests have been done on a 100Mbps Ethernet network, which is a common network connection to a user's desktop, and the transfer rate I got from those tests ranges from 9MB/s to 11MB/s, which is the maximum transfer rate one can achieve from a 100Mbps Ethernet network. Thus, if a user is connected at 100Mbps they will see no difference in performance in using Davis compared to iCommands.

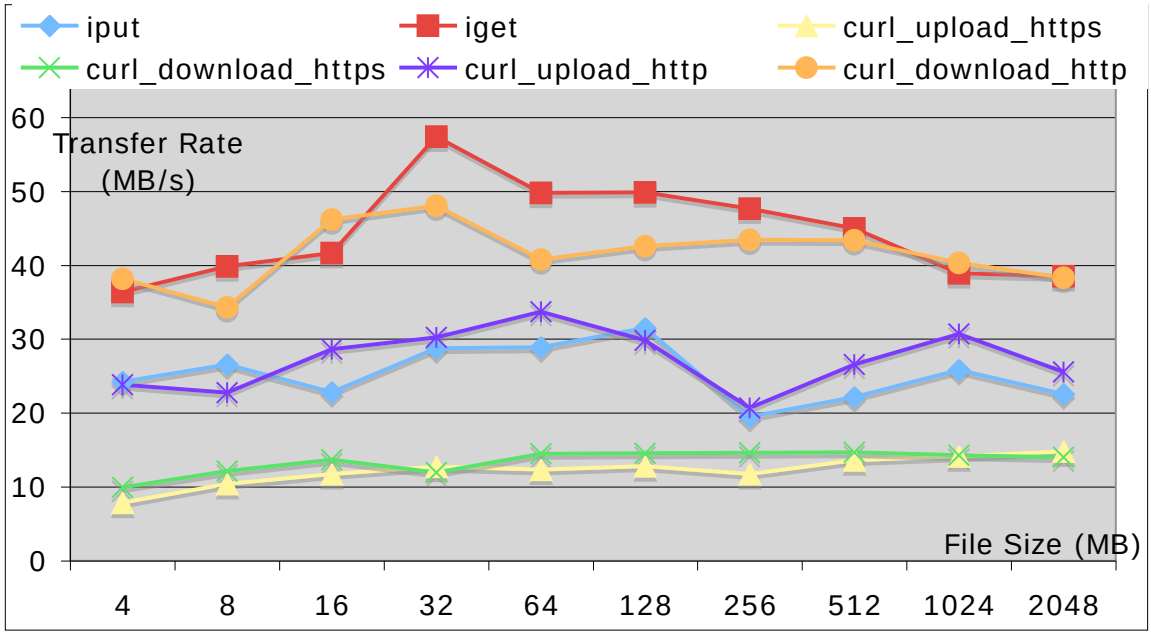

Figure 4-5 Test Results of iCommands and cURL

4.7 Use cases

The integration of WebDAV and iRODS/SRB opens a window to use iRODS/SRB by more and more researchers. The following are a few examples.

4.7.1 Easy access

The biggest benefit of using WebDAV as the gateway is that iRODS/SRB collections can be mounted to the client system as a local folder; thus, various WebDAV clients, such as Windows Explorer, Mac Finder, and Gnome Nautilus, can be used to access data in iRODS/SRB, without requiring iRODS/SRB clients to be installed on the

file system. Accessing iRODS/SRB is as easy as accessing a local hard disk, and the use of iRODS/SRB is the same as using a native program, with features like drag-and-drop. The user group will be expanded to more people, especially Windows or Mac users or those without much IT expertise.

There are WebDAV clients for hand-held devices, such as smartphones (e.g. iPhone) and tablets (e.g. iPad). A test has demonstrated that browsing and getting files from SRB is achievable on iPhone. A demo is available on YouTube: http://www.youtube.com/watch?v=o2yu8ZJBOck.

Accessing from a PDA-like handheld device is very useful, especially for monitoring experiments remotely. For example, if an experiment generates data continuously and saves it into iRODS/SRB, the researcher can monitor the process at home, or while travelling, by checking the data or metadata generated in iRODS/SRB via Davis, without staying in the laboratory, and this does not require any development or help from a system administrator.

4.7.2 Integration with other applications

It is easy to integrate iRODS with other applications via Davis. The following are some examples where this has been demonstrated.

Some software, such as Microsoft Office and PhotoShop, allows users to open a file directly from WebDAV, without getting it to the local drive. After editing, the file can be directly saved to iRODS/SRB via WebDAV immediately. Furthermore, as a WebDAV resource can be mounted to the system as a normal file system, legacy

software can be used to read and write files on WebDAV without the need to implement support for iRODS, which requires development with iRODS APIs. This is very helpful for users who want to run a compute job but with data stored in iRODS. They can easily mount the iRODS collection via Davis and keep using the existing application software.

Another example is, via the HTTP interface, a resource URL of Davis can be set in a Handle.NET system [144] to be mapped to a unique HDL Identifier, which is a HTTP URL. If a user puts it into a web browser, Handle.NET system resolves the unique HDL Id and redirects the user to Davis, and the requested file will be downloaded automatically.

THREDDS (Thematic Realtime Environmental Distributed Data Services) Data Server [145], a popular system for sharing environmental science and remote sensing data, supports reading data via WebDAV. A test has been done to supply data stored in SRB to THREDDS Data Server via Davis. As a result, this data is available to users via the OPeNDAP protocol [146], which is supported by THREDDS. Moreover, as THREDDS Data Server can be configured to work with Web Mapping Service (WMS), it can generate a WMS overlay with its data; Google Earth can then display this overlay [147].

4.8 Conclusion

This chapter has evaluated a number of existing iRODS/SRB client tools, which shows that the current tools are either of limited functionality, having potential firewall restrictions to the user's desktop, compatible with SRB but not iRODS, or customized for

special needs. Normally, a new user has to spend much effort to use iRODS/SRB. Thus, investigating a new client tool has been conducted with references to standard protocols and existing tools. Based on the investigation, a WebDAV gateway for iRODS/SRB has been implemented, for the purpose of being a generic and one-stop solution for users, developers and systems. Software described in this chapter (Davis) is available at http://projects.arcs.org.au/trac/davis. Davis is deployed as a production service by ARCS, as part of the ARCS Data Fabric.

The next chapter will talk about another important component in the National File System, namely Griffin, a GridFTP interface for arbitrary data sources (here I use it for iRODS). Because Davis transfers data using the HTTP protocol, it inherits the limitations of HTTP, and therefore is not efficient to transfer a huge amount of data. That's when Griffin is useful, because it is designed especially for data transfer and has the ability to interact with data transfer services.

Chapter Five: Generic GridFTP Interface for Data transfer

5.1 Introduction

This chapter describes another important component in the National File System, the GridFTP interface. As discussed in chapter 3, the National File System stores multi-Terabytes of data and requires frequent data transfers. Also, the National File System needs to be integrated with other Grid components to facilitate data staging. Hence, it is necessary to have a component especially for efficient data transfer and easy integration with the Grid. This chapter describes why this becomes an issue and how I address it.

Scientific grid systems are important tools for modern researchers to analyze, store and share data. Most of them provide not only computational services but also data services. Among the existing scientific grid systems, many of the computational services are built on a OGF-compliant grid middleware system, such as Globus Toolkit [8] and gLite [9]. Some other groups adopt Condor [148], a different scheduling system, which can integrate with Globus via Condor-G [51]. However, the mechanism used to store and access data is not unique. As illustrated in Chapter 2, various clustered file systems offer low-level and posix-like interfaces to users, whilst data grid systems are designed to be a virtual file system, offering application-level interfaces to storage space without the users knowing the physical location of data, such as SRB [17] and its successor, iRODS [18], as well as dCache [19]. The way of accessing these data grid systems varies. For example, dCache provides GridFTP interface and HTTP interface, so that grid submission systems can easily access data from it. Some others, such as iRODS,

implement their own proprietary protocols so grid job submission systems have no way of directly getting data out of it. Chapter 4 describes my work on implementing a WebDAV interface for iRODS, which is suitable for transferring small amounts of data, such as file size less than 4GB. When it comes to a larger amount of data, a HTTP-based WebDAV interface is inefficient, so a faster interface such as GridFTP should be used.

It becomes a common requirement to transfer data between heterogeneous data sources as the amount of data increases and the data is placed in different data storage systems. When a new protocol is developed, considerable work has to be done to integrate it into the existing system. Typically, users have to use one client to copy data from the source to an intermediate space, such as a local storage space, and then use another client to move data from the intermediate space to the destination. This is error-prone and inefficient. Because of having an extra step, the user has to sit in front of a terminal or write a script to automate the whole process. If anything goes wrong, the user has to do a lot of debugging work to find out the error.

This chapter describes a different approach, which presents to users a standard GridFTP interface, through which the user can transfer data from and to an underlying arbitrary data source, thus it is compatible with most grid systems, and can leverage the advantages of GridFTP protocol in data transfer. It solves two problems in a single solution. Firstly, it enables grid systems to access data from any arbitrary data source. Secondly, transferring data between two data sources with different protocols becomes possible without the use of an intermediate point, and is more reliable, and manageable by using a data transfer service. The structure of such system is not complicated: it has a GridFTP interface front-end that receives and interprets GridFTP commands, and a

generic file system framework that translates the commands to a backend data source via a plugin. The generic file system framework abstracts required file operations for each data source to implement a specific plugin, so that the system can be used with any arbitrary data source by configuring a corresponding plugin.

The next section gives an overview of related work. Then, section 5.3 describes the features of the GridFTP protocol and the reasons why it has been chosen. Section 5.4 gives some details of the generic file system framework. Section 5.5 explains the implementation and Section 5.6 analyzes its performance through some experiments. Lastly, a summary is given in Section 5.7.

5.2 Related work

Currently, the Globus toolkit only supports GridFTP, HTTP and SSH protocol [149]. If users want to stage data from/to a data source that Globus doesn't support, they have to do that outside of Globus. For example, a client of that data source can be installed on each worker node, then in the beginning of a PBS script, a command is issued to retrieve data using the client. Another solution is to use a copy queue [150], which is a special queue only for copying files, so that the data source client is only needed on the Globus gatekeeper, and there is no need to install it on all worker nodes. For each compute job, a copy job will first be submitted to the copy queue to transfer files, and then a compute job will be submitted to the normal queue to process the data. These approaches require additional software to be installed into the grid system. In addition, the status of data staging is not monitored or managed, so if the copy is

incomplete, the grid system cannot be notified of it, which may result in a waste of resources to process incomplete data.

Some work has been done to help users with transferring data between heterogeneous data sources, and some desktop applications have been developed. It is noted that these solutions cannot be used by Globus to stage data. Instead, they mainly focus on data transfer from a user's desktop. Firstly, in order to access different file systems from a single application, a few libraries have been designed, such as JSAGA [151] and commons-vfs [152]. They both provide a pluggable framework for building applications that allow users to access different data sources in a uniform way with the use of conversions and plugins. The difference is, commons-vfs only has APIs for data access while SAGA/JSAGA offers a comprehensive API including not only data access functions but also functions to submit jobs to various grid systems. Correspondingly, JUX [153] is a Java GUI implemented based on JSAGA, and Hermes [132] is another Java GUI implemented based on commons-vfs. However, these frameworks aim to provide a generic file access interface, and are hard to extend for extra functions. For example, GSI (the Grid Security Infrastructure, details in section 2.3.1) authentication is the main authentication mechanism for GridFTP, so the underlying data source must support GSI authentication natively, or our implementation needs to provide a mapping from GSI to the data source's native authentication method. Based on the experience of using the aforementioned frameworks, implementing GSI in an arbitrary data source is not trivial, and may require some low-level coding or complex configuration. What's more, I not only want to develop an interface for data transfer, but also want to offer advanced functions, such as the conductor plugin described in section 5.7.2, and it is hard

to extend these frameworks for that function. Therefore, I designed a simple framework, influenced by these frameworks and the java IO interface, aiming to offer the required and essential functions for data access, and the ability to be extended to support any future requirements.

Similar functions have been added to data transfer services, such as stork [71], which supports a new data system by adding a new script to the scheduler, where the transfer is done by invoking a corresponding client of that particular data system. PAFTP [154] is a WSRF-based data transfer system that supports multiple protocols. It employs an application interface that wraps file access and operations over different protocols. These solutions need to copy data from the source to the destination via an intermediate place if both ends do not support the same protocol, which is not efficient and may use unnecessary network bandwidth.

Globus Toolkit offers two solutions for users to integrate a third-party data source with Globus. The implementation [155] of the GridFTP server provides a modular pluggable interface to a storage system, which is called Data Storage Interface (DSI) [156]. The DSI presents a modular abstraction layer to a storage system, and hides the details and complexities of GridFTP protocol. Adding a new DSI requires implementing a few interfaces, and some interactions at the GSI-level for authentication and authorization. The other way is to use the Globus XIO system [157], which is an abstraction framework aiming to provide a simple Open/Close/Read/Write (OCRW) I/O API to Globus Toolkit so that developers of Globus can access data with a single API regardless of the actual location of the data. Being part of the Globus Toolkit, it can work with Globus GridFTP server seamlessly. Globus GridFTP server can use XIO to read

data from one data source, and write this data to another data source, or vice versa. However, these two solutions rely on the common runtime libraries of Globus software stack (as mentioned in section 2.3.1), such as GSI libraries, which makes them hard to develop, deploy and use, because Globus is developed in C and not cross-platform. A UNIX emulator is needed for Globus to run on Windows, adding extra complexity to installation and maintenance. Globus XIO system only provides common APIs for OCRW functions, and lacks common APIs to deal with other aspects, such as permissions, of the underlying file system. Although this can be overcome with the use of plug-ins or external callouts, it requires additional components from inside or outside Globus, hence results in a more complicated system to set up and maintain.

The observations above show it hard to transfer data with a proprietary data source using existing tools. The Globus Toolkit does offer a way to interface its GridFTP server with a data source. However, it is not easy to implement, as developers need to have a deep understanding of the Globus middleware framework. Next I will introduce GridFTP protocol and its advantages, then describe my idea of using a generic file system framework to bridge between a GridFTP interface and an existing data source.

5.3 GridFTP protocol

This section gives an overview of the GridFTP protocol, and outlines the benefits of adopting it.

5.3.1 An overview of GridFTP protocol

GridFTP protocol version 1 was released in 1999 [13]. The second document, GFD-21 [158], addresses some issues in the first version and gives some improvements to the GridFTP protocol. In 2005, GridFTP protocol version 2 [159] was released, with more advanced features. In essence, GridFTP is based on the File Transfer Protocol (FTP) [160] with some extensions for secure, reliable and high-performance data movement. Important features of GridFTP protocol include:

- Third party transfer: To transfer large datasets between two data sources, it is more efficient to transfer them directly, rather than going through the client. GridFTP protocol offers a way for the client to create control channels to two data sources, and initiate direct data channel between the two sources. During the transfer, the client can monitor and further control the data flow via the established control channels. By this means, data goes from the source to the destination fast and efficiently without going through an intermediate point.
- Security: GridFTP protocol employs RFC 2228 (FTP Security Extensions) [161] to allow Grid Security Infrastructure (GSI) [162] authentication and other security options. After authentication, the control channel between the client and the server is encrypted. The data channel can also be encrypted if necessary. For instance, when creating data channel during a third-party transfer, it is essential to verify both parties by data channel authentication.
- Extended block mode: Extended block mode extends the FTP's block mode in the header to support parallel transfer, striped transfer and partial transfer. Data

being transferred is split into blocks with each block consisting of offset and length of the data in its header, and real data in its body. Therefore, the client can transfer portions of a file from a particular point, or use multiple streams to transfer a big file in parts. The order of different parts arriving at the server does not matter, as the server can work out the position by the offset and length in header. In most situations, parallel transfer has proved to be faster than single stream transfer [163].

- Reliable and restartable transfer: Fault recovery can be achieved by a mechanism defined in the data channel protocol to restart a failed transfer. This is useful when suffering a network outage or server outage during the transfer of a huge file. Restarting a failed transfer can save resources required to start from the beginning. To verify a transfer, a mechanism has been added to GridFTP protocol to compare the checksums of the source file and the new copy after the transfer is finished.
- Alternative transfer protocol: Apart from TCP, UDP protocol can also be utilized to transfer data, which has been shown to outperform TCP on high-bandwidth and high-delay networks [164].

5.3.2 The benefits of adopting GridFTP protocol

GridFTP protocol is the de facto standard for data transfer in the grid world. Supporting GridFTP in a data source means that the source can be accessed by existing grid applications, including job submission systems, grid clients, workflow systems and

data transfer services. Also, the advantages of GridFTP features for enhancing transfer performance, such as parallel transfer and data channel reuse, enable the implementation of a fast data transfer interface [155]. Applying a GridFTP interface on top of a slow or legacy data server interface can potentially improve data transfer performance. Monitoring and automating data transfer is also feasible by using data transfer services, many of which support GridFTP. Not all protocols are suitable to transfer large datasets, due to the limitations in the protocol itself, or the implantation of its clients. For example, HTTP protocol has a time out problem, which makes the client disconnect from the server if transferring a large file for too long; also, from my experience, most HTTP clients can only transfer a single file whose size is up to 4 GB in a 32 bit system. If the file is larger than that, the client may get an error before it can reach the end of the file. GridFTP protocol does not suffer from this limitation. Instead, it has a mechanism to send data in small blocks and keep the control channel alive if the transfer takes a long time.

5.4 Generic file system framework for GridFTP

In order to access arbitrary file systems in a uniform way, I have designed a generic file system access framework as a backend to the GridFTP interface. By means of this framework, any request coming from the GridFTP interface will be translated to a standard file system operation. To make it generic, the framework is lightweight and simple, supporting only the basic and commonly implemented file operations required by GridFTP. Other file system features are not useful to GridFTP so there is no need to implement them. The design of this framework is specific to the requirements of the

GridFTP protocol, especially advanced features for data transfer, in order to minimize overhead between the front-end GridFTP interface and the backend data system. As the GridFTP protocol is mainly used for data transfer, this framework does not require other functions, such as metadata manipulation (and GridFTP protocol doesn't support that anyway).

The simple top-down structure is illustrated in Figure 5-1, with four major objects. The top one is the only instance in the system corresponding to the data source, with all configurations and controls at the data source level. Each user, once authenticated, will be associated with a connection object, which is the second one from the top. The connection object keeps all user-relevant information (one connection object will be created for each user login), because different users will have a different view of the data source, with different permissions. Thus, the connection object can determine whether a file object, the third one from the top, can be created for a particular path, and what functions can be called, based on the user's permissions. Once a file object is created, it is used to retrieve information of the target file, or do some operations on it. The bottom one is mainly used to access the content of a file object, as long as the file object is initiated with proper permissions. An adaptor for a data grid system must implement all four interfaces.

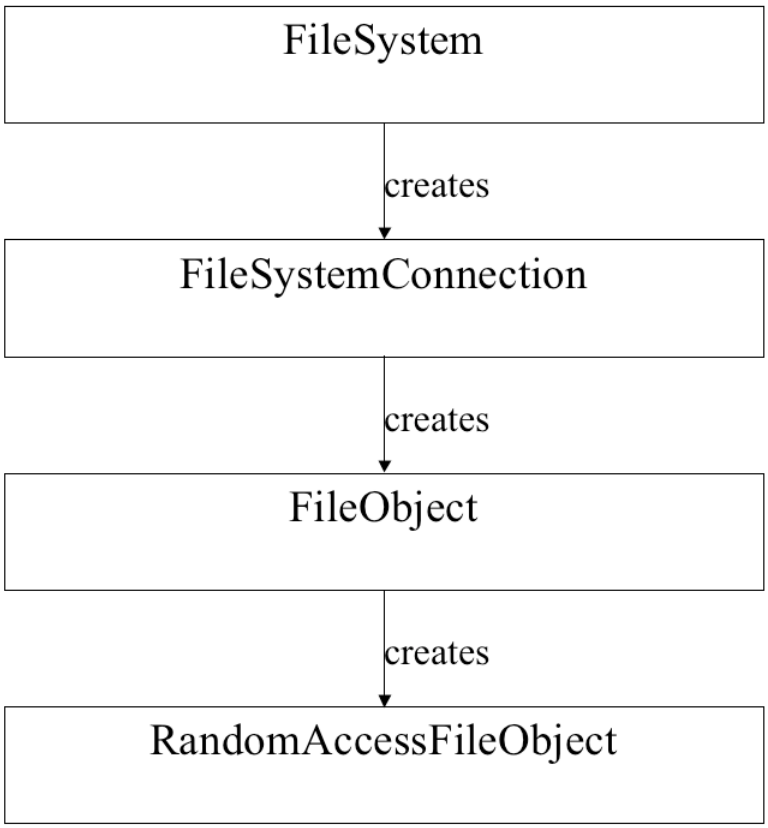

Figure 5-1 The structure of the generic file system framework

The *FileSystem* object is the root of the hierarchy. It has the methods as shown in Figure 5-2. This object presents the underlying data system in the GridFTP interface and is called on startup and shutdown. In particular, method *init()* is called to examine the configurations and check the connectivity to the data source during the startup phase. If there is an error, for instance, the configuration is not correct or the underlying data system is not accessible, an exception will be thrown so that the GridFTP server can decide whether it can continue to run. Likewise, method *exit()* is invoked during shutdown to release resources and close connections to the data system if there are any.

Method *getSeparator()* returns the separator used in paths, given that it may be different in some systems, such as back slash in Windows and forward slash in Linux. Method *createFileSystemConnection()* will be executed when a user connects to the GridFTP interface and a GridFTP session is created for that user. Correspondingly, a connection to the underlying data system will be created, according to the GSI credential. This method returns a *FileSystemConnection* object, which represents a session to the data system. If the underlying data system supports GSI authentication, the same GSI credential used to authenticate to GridFTP interface will be used to authenticate to the underlying data system. However, if the underlying data system does not support GSI authentication, some kind of mapping mechanism will be employed.

```
public String getSeparator();
public void init() throws IOException;
public FileSystemConnection
        createFileSystemConnection(GSSCredential credential) throws
        FtpConfigException, IOException;
public void exit();
```

Figure 5-2 FileSystem interface

FileSystemConnection holds a connection to the underlying data system for the user. It consists of a few system-wide and user-related methods. *getHomeDir()* and *getFreeSpace()* are self-descriptive. *getUser()* returns the user name of the underlying

system associated to this session. *isConnected()* indicates whether the connection is opened, while *close()* will terminate the current connection. Method *getFileObject()* generates a *FileObject* for a particular path in the GridFTP system context.

```
public FileObject getFileObject(String path);
public String getHomeDir();
public String getUser();
public void close() throws IOException;
public boolean isConnected();
public long getFreeSpace(String path);
```

Figure 5-3 FileSystemConnection interface

FileObject, illustrated in Figure 5-4, represents a data object in the data source. It is designed to reflect the *java.io.File* class, providing methods to get the file name, path, canonical path, length, last modified time, its parent and its children. In addition, it allows users to check whether this file exists, or whether this object is a file or a directory. Users can also use this object to make a new directory, delete or rename the file or directory, or set the last modified time. Method *getPermission()* returns an abstracted permission number, which can only be no access (0), read privilege (1), write privilege (2) or read/write (3). GridFTP protocol does not require complex permission management functions, such as user/group permissions in UNIX; all that it cares about is the permission of the current user, so the generic framework only needs to find out the

permissions for the user associated to the current connection. Returning a simple numbered permission is the easiest and most generic way to address this issue, and hide the complexities in different data sources. Method *getRandomAccessFileObject()* returns a *RandomAccessFileObject*, as shown in Figure 5-5, for accessing the contents of this file object.

```
public String getName();
public String getPath();
public boolean exists();
public boolean isFile();
public boolean isDirectory();
public int getPermission();
public String getCanonicalPath() throws IOException;
public FileObject[] listFiles();
public long length();
public long lastModified();
public RandomAccessFileObject
        getRandomAccessFileObjec(String type) throws IOException;
public boolean delete();
public FileObject getParent();
public boolean mkdir();
public boolean renameTo(FileObject file);
public boolean setLastModified(long t);
```

Figure 5-4 FileObject interface

RandomAccessFileObject provides methods to read and write contents of the corresponding *FileObject*. The design is similar to *java.io.RandomAccessFile*, for developers' convenience. Several *read()* methods and *write()* methods are the major part

of this object to read file content to a byte array or write content of a byte array to the file. Other methods include *seek()* to jump to a certain point in the data object and *length()* to return the length of the data object. Unlike the streaming objects, which can only read or write in sequence, the *RandomAccessFileObject* offers a way for the front-end GridFTP interface to easily read or write data from any position in the file, which is used in the extended block mode to enable parallel transfer and restarting after a connection problem in the middle of a transfer. Consequently, the underlying data source must support random access so that it can be plugged into this framework.

```
public void seek(long offset) throws IOException;
public int read() throws IOException;
public int read(byte[] b) throws IOException;
public int read(byte[] b, int off, int len) throws IOException;
public void close() throws IOException;
public String readLine() throws IOException;
public void write(int b) throws IOException;
public void write(byte[] b) throws IOException;
public void write(byte[] b, int off, int len) throws IOException;
public long length() throws IOException;
```

Figure 5-5 RandomAccessFileObject interface

5.5 The implementation

To demonstrate this concept, a production system, called *Griffin*, has been developed. The implementation includes a basic GridFTP interface, the generic file system framework, and two adaptors: one for the iRODS data grid system, and one for the local file system. The purpose of the implementation is to demonstrate that any client compatible with GridFTP can access the underlying data source via the GridFTP front-end and the generic file system framework, where iRODS is used as an example of an arbitrary data source. The implementation generated in this work will also be used as the interface to a production iRODS system. To access other data sources, a relevant adaptor can be developed, and configured in *Griffin*. In addition, the implementation is to substantiate that with this structure, data transfer performance will not be affected. Instead, the transfer rate may be better than the native protocol as it benefits from the advantages of the GridFTP protocol.

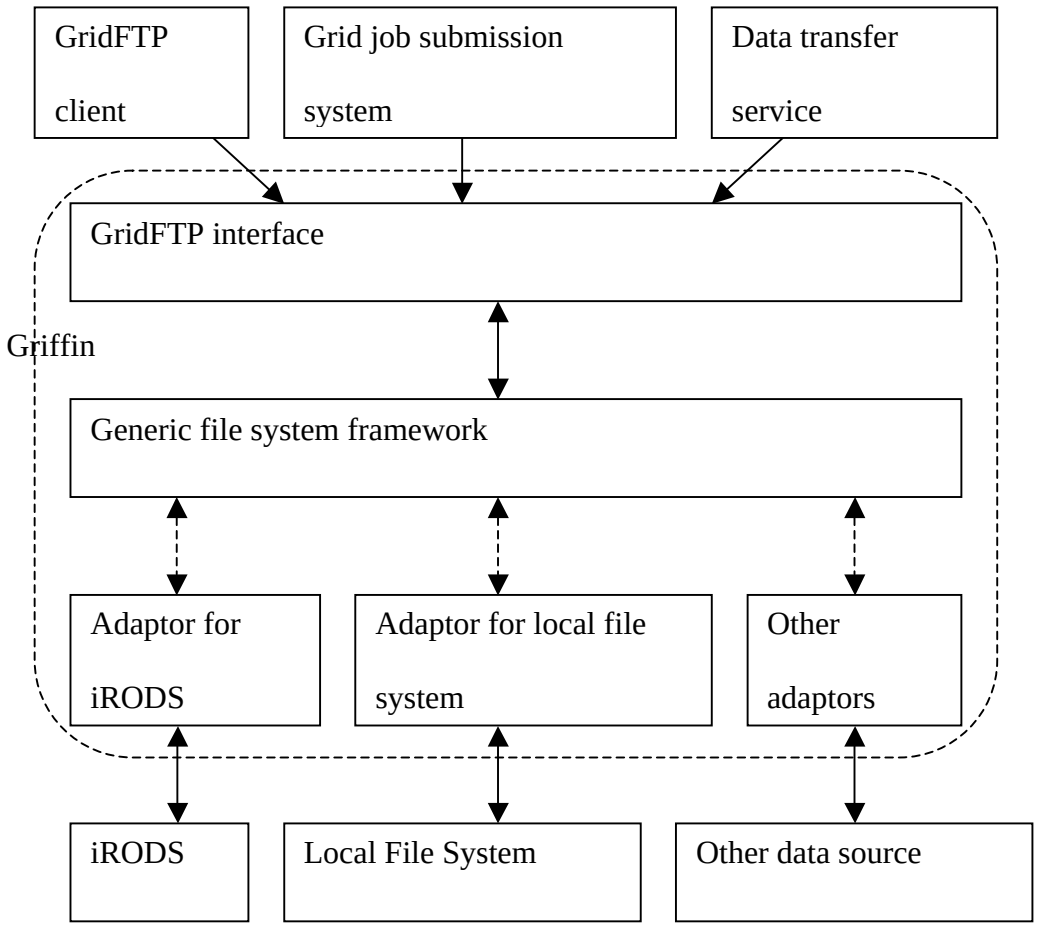

Figure 5-6 The architecture of Griffin

The implementation, illustrated in Figure 5-6, is written in Java and based on the Spring framework [165], which offers an architecture for developers to easily develop a modular and pluggable application. GridFTP interface is the base of the application. It manages all client connections with a parser to parse GridFTP commands and invoke them with relevant command classes. The implementation of GSI authentication is based on jGlobus [166]. The generic file system framework is implemented as stated in section 5.4. Adaptors for data sources rely on the relevant client libraries. For example, the

adaptor for iRODS relies on iRODS's Java library, Jargon [167]. Having the generic framework makes it possible to implement adaptors for other file system libraries, such as commons-vfs and JSAGA, so that *Griffin* can be used to access all data sources that are supported by these libraries. The binary of *Griffin* is lightweight, stand-alone and self-contained, with all dependent Java libraries. Therefore, it does not rely on any Globus components, which makes it easy to install and maintain. In addition, being a Java application, it is possible to be run on most operating systems without recompiling. For instance, if a user wants to submit a grid job to analyze data that is located on a desktop computer, an instance of *Griffin* could be run on this desktop machine, no matter if it is Windows or Linux (this is contrast to the Globus GridFTP server, which requires a UNIX emulator to run on Windows). Globus job submission can then stage in data from the desktop and stage out data to the desktop, without the need to copy data to and from somewhere that is accessible by the Globus job submission node, such as a dedicated GridFTP server, or a file system that is mounted to the worker nodes.

With the modular design and the help of Spring dependency injection framework [168], changing the underlying data system only requires a change in an XML configuration file, without a need to recompile the whole system. Figure 5-7 is a sample of a portion of the *Griffin* configuration file, with the iRODS adaptor. As seen, the configuration is simple, and only needs the iRODS server's hostname and port number.

```
<bean id="server" class="au.org.arcs.griffin.server.impl.GsiFtpServer"
singleton="true">
   <property name="name" value="GSI FTP Server" />
   <property name="options" ref="options" />
   <property name="resources" value="griffin-resources"/>
   <property name="fileSystem" ref="fileSystem" />
</bean>
<bean id="fileSystem"
class="au.org.arcs.griffin.filesystem.impl.jargon.JargonFileSystemImpl"
singleton="true">
   <property name="serverName" value="localhost" />
   <property name="serverPort" value="1247" />
   <property name="serverType" value="irods" />
</bean>
```

Figure 5-7 A configuration with iRODS adaptor

To run *Griffin* with a local file system, we only need to replace the *<bean id=”fileSystem”>* portion with Figure 5-8, and restart *Griffin*.

```
<bean id="fileSystem"
class="au.org.arcs.griffin.filesystem.impl.localfs.LocalFileSystemImpl"
singleton="true">
   <property name="rootPath" value="/data/data-fabric" />
   <property name="userManager" ref="userManager" />
</bean>
```

Figure 5-8 A configuration with local file system adaptor

5.5.1 Parallel transfer

The implementation of parallel transfer is a notable challenge. Without knowing the features of an arbitrary data source in advance, the current implementation is based on the assumption that single connection and random access are always supported; otherwise the data source cannot be plugged into *Griffin*. If the data source only supports streaming, it is not straightforward to plug it into *Griffin*. Suppose a user starts a transfer in extended block mode with parallel streams, data blocks arrive in no particular order, however, writing to the data source should be in order as it only supports streaming. Thus, if a block of a latter position comes in first, it has to be cached by *Griffin* before writing it to the data source. If the file being transferred is a big file, it may end up running out of local space. Thus, the current implementation opted not to support data sources that only support streaming. In addition, for the best performance of *Griffin*, and further to the first

assumption, I also made the second assumption that the network from the client to *Griffin* is always slower than the network from *Griffin* to the data source, otherwise the transfer would not benefit from the advanced features of GridFTP. Hence, it is recommended to place *Griffin* as close to the data source as possible, e.g. on the same host as the data source or in the same LAN. Thus, the bottleneck becomes the network from the client to *Griffin*, which can take advantage of parallel transfer between the client and *Griffin* to enhance the transfer rate, as shown in Figure 5-9. If the network between the client and *Griffin* is faster than the network between *Griffin* and the data source, data transfer will be delayed by *Griffin* as *Griffin* only opens one stream to the data source.

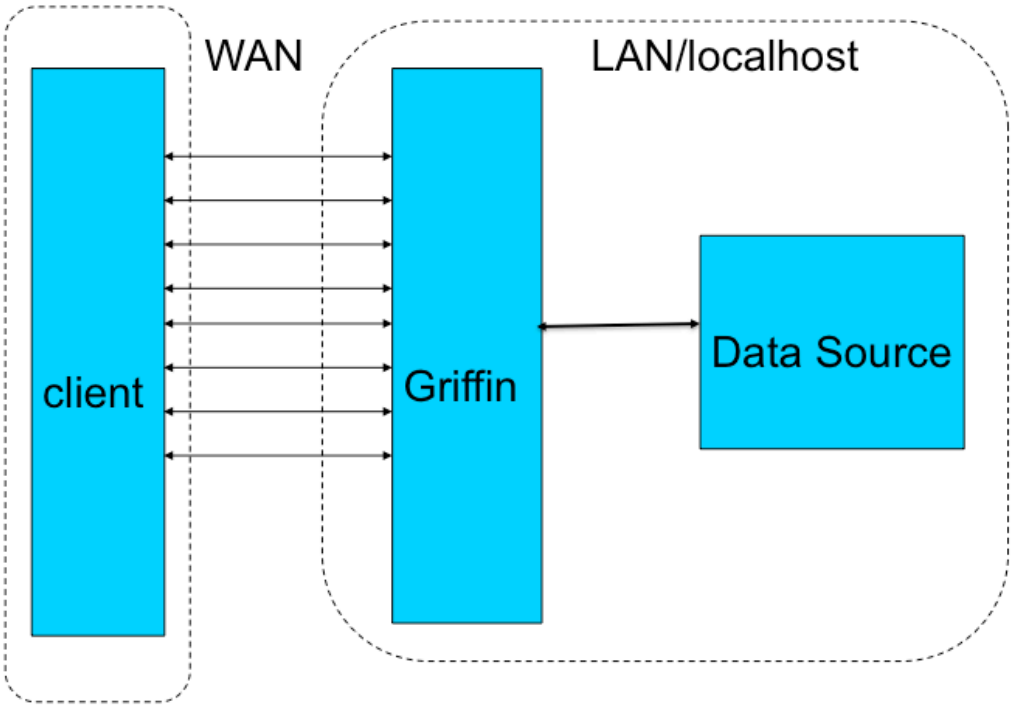

Figure 5-9 Parallel transfer between Griffin server and the client

5.5.2 Authentication and authorization

Authentication and authorization are the most significant issues when doing multiple-protocol mappings. During GridFTP authentication, a GSI credential will always be passed through from the front-end GridFTP interface to the generic file system framework, to verify whether this user is authenticated to access the underlying data source. The decision is made by the file system object of the adaptor based on the presented GSI credential. If the user is not valid, an exception will be thrown to the GridFTP layer and an error code will be returned to the GridFTP client. If the authentication is successful, a connection to the data source will be created with the user information. Authorization is done according to the user information attached to the connection. How the authentication and authorization are processed in the underlying data system differs depending on whether the data system supports GSI authentication. For those data systems that support GSI authentication, such as iRODS, the given GSI credential will be passed to the data system for validation. If the GSI credential belongs to a valid user in the data system, this user will be used in the current connection for all subsequent operations, with the permissions this user has. For those data systems that do not support GSI authentication, some kind of mapping mechanism should be in place to map between GSI credentials and the underlying data system's local credentials.

One typical example of the mapping mechanism is the local file system. The Globus implementation of GridFTP maps GSI credentials to local system users, thus file permissions in GridFTP context are aligned with the native file system permissions, and files are owned by real operating system users. From the GridFTP interface, any user can

access any file in the operating system if the user has access to it. File permission settings in *Griffin* for local file system are different from the Globus GridFTP implementation. In essence, *Griffin* acts as a virtual file system, with file contents in local file system and user/permission information in its XML configuration file. The configuration specifies a directory in the file system as the root of the file system in *Griffin*. Users in *Griffin* are not mapped to operating system users. Instead, all files are owned by the single operating system user who runs *Griffin*, whose permission are managed at the application level, by maintaining a user and permission configuration file in XML format, which defines groups, users, and mappings as the basic elements. An example is displayed in Figure 5-10. In this example, a group is an element that holds the real permission, which can be read only or read-write. Permissions can be assigned to a single path, or a list of paths that conform to a pattern. For example, /${user}/** means all files under current user's home directory. A user is an entity in a group, with a default directory, usually its home directory. A mapping maps a DN to a user in the system. Thus, when a user logs in with GSI authentication, *Griffin* works out its home directory by getting the user element from the mapping, and the permissions by the relevant group. The benefit of the *Griffin* permission system is that users can only see what they are allowed to, without exposing the whole file system to users. In addition, this would work with any operating system, including Windows, because at the file system level, all files are owned by a single user, without the need of switching to other system users, which is not possible for Windows. The disadvantage is that all files have to be under a single directory, unless some soft links are created to aggregate directories to one place.

```
<groups>
   <group name="users" >
      <permissions>
         <permission flag="rw" path="/${user}/**"/>
         <permission flag="r" path="/"/>
      </permissions>
   </group>
</groups>
<users default-dir="/${user}">
   <user uid="user" fullname="Test User">
      <group-ref name="users"/>
   </user>
</users>
<mappings>
   <mapping dn="/C=AU/O=APACGrid/OU=SAPAC/CN=Shunde Zhang"
uid="user"/>
</mappings>
```

Figure 5-10 An example of permissions for local file system

5.6 Performance Measurements

Three tests were conducted to show the usability and efficiency of Griffin. The first two tests compare the data source's proprietary protocol and GridFTP protocol: the first test transfers files of different sizes; the second test transfers a large amount of small files. These two tests show that data transfer rate of Griffin is similar or better than the data source's own protocol. The third test compares Griffin and Globus GridFTP server: both will be used to transfer the same files. This test shows that Griffin performs similar to Globus GridFTP server.

The first test transfers files in different sizes to compare the transfer rate of the data source's own interface and Griffin. In this test, iRODS is used as the underlying data source, so iCommands is used to transfer data to compare with Griffin. The test environment, illustrated in Figure 5-11, includes a server and a client. The server has two quad-core Intel Xeon X5460 3.16MHz CPUs and 16GB memory, running 32bit CentOS 5.2, and iRODS 2.1. Griffin was set up on the server as the GridFTP interface for this iRODS and allocated 786MB heap size in the JVM. This server is in Adelaide with a 1Gbps Ethernet network interface. With this setup, data in iRODS can be accessed via native iRODS interface or GridFTP interface. The client, on the other hand, is an IBM xSeries 346 with two hyper-threaded Intel Xeon 3.20GHz CPUs, 4GB memory and a 1Gbps Ethernet interface, running 64bit CentOS 5.4, in Melbourne. The WAN connection between Adelaide and Melbourne includes two 10Gbps links. iCommands and GridFTP client (globus-url-copy) are installed on the client machine. We compare the upload rate and download rate between them, with file sizes of 256MB, 512MB,

1GB, 2GB, 4GB, 8GB to 16GB, and eight threads are used for both iCommands and Griffin transfers. For small files, fewer threads may have less overhead while for larger files, more threads may give better performance. This test is to compare the performance between two clients in the same situation but not to look for the best parameters for a particular client; therefore, using a static thread number is acceptable and makes the test easy to run. I keep the number of threads fixed, as my objective is to compare the performance between two clients in the same situation; optimizing the number of threads does not materially affect this comparison.

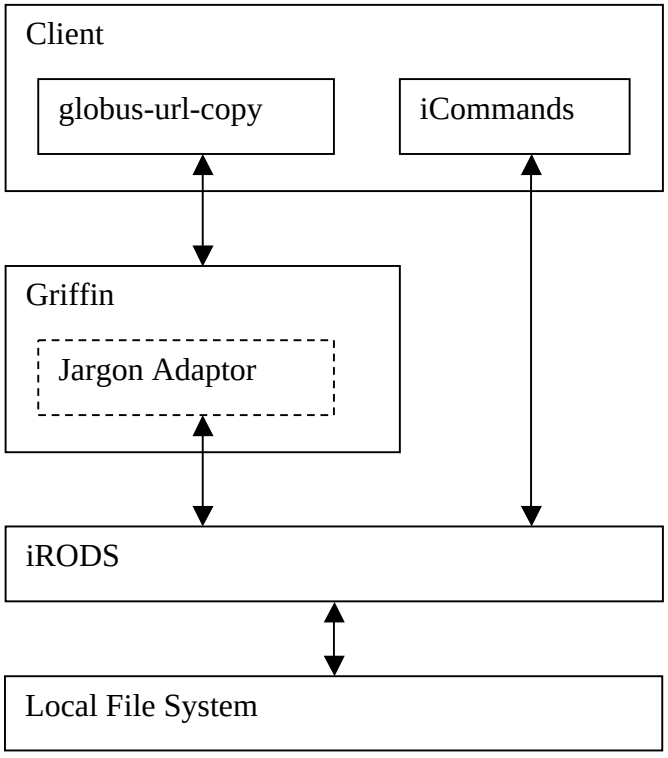

Figure 5-11 The setup of test env. for comparing iCommands and Griffin

Having each test run 10 times, the result in Figure 5-12 shows the average transfer rate in Mbytes per second. The results show that when uploading a file, iCommands and *Griffin* perform at a similar rate if the file is less than 8GB, however, if the file is 8GB or over, *Griffin* is faster than iCommands. When downloading, the performance of *Griffin* is almost constant across the size range in the test, while iCommands performs faster for small files but gets slower as the file size gets bigger.

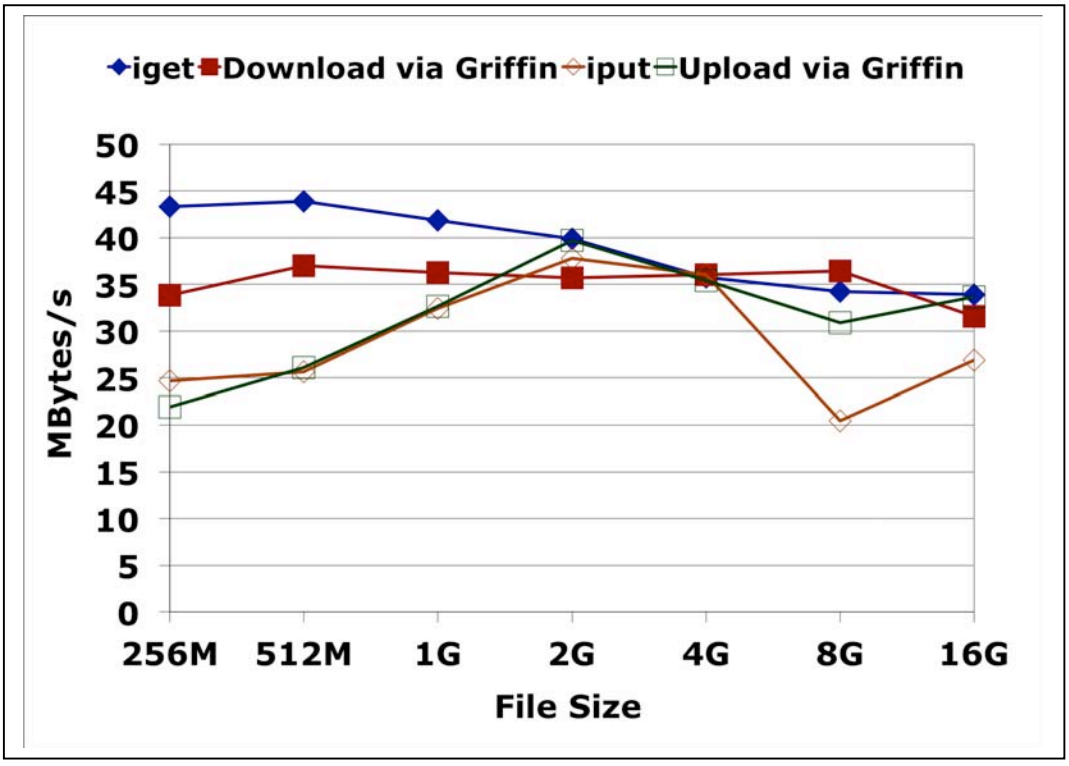

Figure 5-12 A comparison between iCommands and Griffin

Another test was conducted to compare the rate of transferring many small files. The same test environment was used for this test, to copy one thousand 1MB files, i.e. a

total of 1GB, in both directions. At the time of this test, iCommands only supported transferring one file at a time and multiple threads for one file are not useful due to the small file size. Thus, for iCommands, default parameters were used, which is one connection between the client and the server for commands and data. However, the latest version of the GridFTP client supports multiple control channels to allow upload and download of multiple files concurrently. Furthermore, the feature of data channel reuse only needs to establish the data channel once when transferring multiple files, thus saving a significant amount of time in channel creation if it is to transfer many small files. For *Griffin*, the parameters for GridFTP client were "–cc 20 –p 1", which creates 20 control channels between the client and the server, and for each control channel, one data channel is used to transfer the file content. Thus, there would be a total of 20 control channels plus 20 data channels to transfer the data. Likewise, download and upload were executed for both clients. Each test was done 10 times and the average elapsed time is shown in Table 5-1. The result indicates that *Griffin* works faster than iCommands, because the GridFTP client has the ability to initiate multiple control channels and data channels, as well as reusing an existing data channel to transfer multiple files.

Test	Time Elapsed	Transfer Rate
iput	177 seconds	5.65 Mbytes/s
Upload via *Griffin*	142 seconds	7.04 Mbytes/s
iget	237 seconds	4.22 Mbytes/s
Download via *Griffin*	180 seconds	5.56 Mbytes/s

Table 5-1 Elapsed time of transferring 1000 files

The third test, as shown in Figure 5-13, is to compare *Griffin* with the Globus GridFTP server, on the same test environment as the above tests. This test shows that the performance of Griffin is as good as that of Globus GridFTP server; on the other hand, it also shows that the same transfer rate can be achieved by using GridFTP protocol, no matter in what language the system is implemented. In this test, Globus GridFTP server has been installed on the server from VDT 1.10.1. *Griffin* has been re-configured with a local file system adaptor, so that it reads and writes data in the local file system. Test data is stored in the same partition as the iRODS resource. Similar to the first test, I transferred files with sizes of 256M, 512M, 1G, 2G, 4G, 8G and 16G, via *Griffin* and Globus GridFTP, with 16 threads. I chose 16 because it is the number that can maximize the use of network bandwidth.

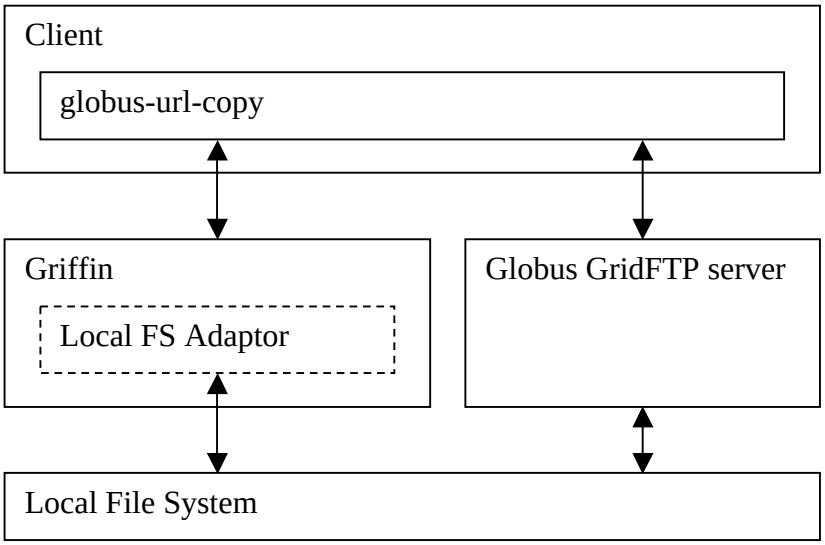

Figure 5-13 The setup of test environment for comparing Globus GridFTP server and Griffin

Each transfer test was done 10 times and the average values are presented in Figure 5-14 with error bars showing the error in the mean for these 10 measurements. The result demonstrates that *Griffin* performs very closely to Globus GridFTP server in data upload and data download.

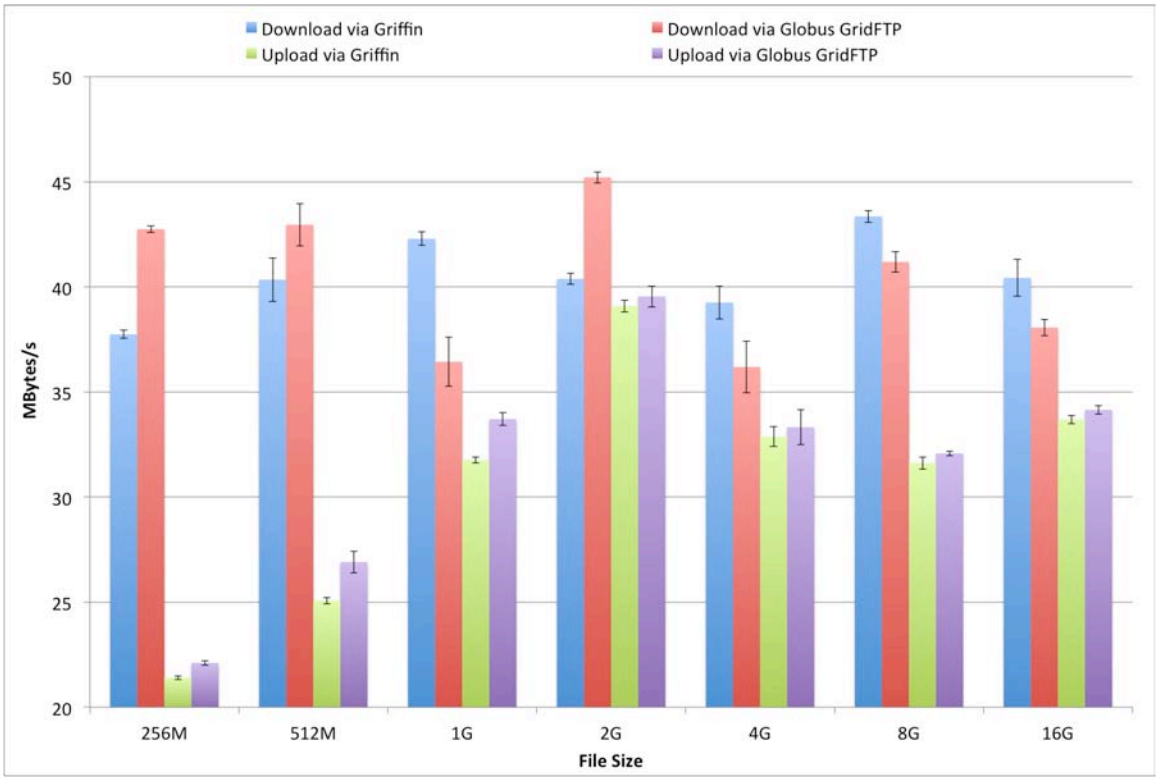

Figure 5-14 A comparison between Globus GridFTP server and Griffin

5.7 Extensions

I have discussed the basic *Griffin* architecture and performance evaluation in the above sections; on the other hand, the design of *Griffin* makes it easy to be extended to fulfill other functions, while preserving the basic GridFTP structure. For example, the

generic file system framework allows plugins to be written for other file systems, apart from iRODS. Alternative GridFTP command implementations can be developed to replace default implementations, to provide a special function. This section describes two examples of *Griffin* extensions.

5.7.1 MongoDB adaptor

The MongoDB adaptor [169] has been developed by an external developer to provide a GridFTP interface to GridFS [170], the file storage to store large objects in MongoDB. With this adaptor, grid users and grid services are able to access files that are stored in MongoDB, through the use of grid certificates and the commonly supported GridFTP protocol. This is a good example to connect another data source into the grid.

5.7.2 Griffin Conductor

The goal of using *Griffin* Conductor is to provide striping transfer feature to those data sources that are in form of a distributed file system, such as iRODS and dCache that are virtual file systems with the ability to replicate data into multiple distributed data resources. Consider the situation where *Griffin* or a GridFTP interface is deployed on top of each resource of a geographically distributed file system, as shown in Figure 5-15, which ensures that each *Griffin*/GridFTP server can serve data from a local data source, hence avoiding network latency between *Griffin*/GridFTP server and the data source. However, a user who is given several available *Griffin*/GridFTP endpoints may not know

which one is the most suitable, and system administrators may also want a particular user to store files in a specific data source. Therefore, on top of existing *Griffin*/GridFTP servers, I designed and deployed *Griffin* Conductor, which is configurable to orchestrate data transfer for each user to the best-matched (according to pre-defined rules) *Griffin*/GridFTP server. The essential idea of this mechanism is third-party transfer: the client connects to one GridFTP server but receives data from another (this is not the same as the normal third-party transfer users usually do, because the transfer target is decided by *Griffin* Conductor but not the user).

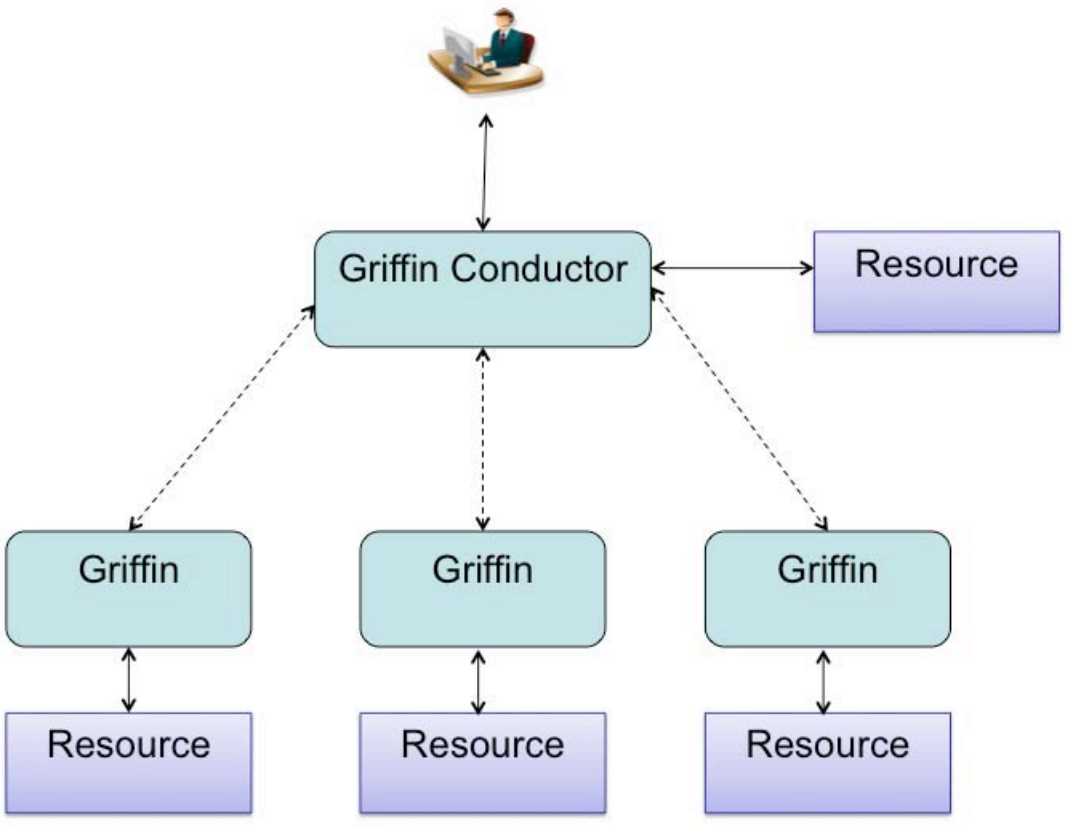

Figure 5-15 Deployment of Griffin on top of distributed file system

Data Upload

Griffin Conductor directs files to specific resources according to pre-defined rule implementations (a sample of the rules will be shown later in this section). Various rules may include: geographical location, a user always uploads files to the nearest resource, which can be determined by the user's IP address; identity of the user, users from a particular group all store files in one specific resource. Because every file system is different, the current implementation simplifies the use case and assumes that the upload of a single file can only go to one resource. Uploading portions of a single file to multiple resources needs the underlying file system to combine these portions into a complete file in the background, which places another requirement to the data source. Thus, to make it more generic, I implement *Griffin* Conductor to direct upload data stream to one resource only. The whole progress is shown in Figure 5-16, in 6 steps:

1. A user sends an upload request
2. *Griffin* Conductor checks against the distributed file system to see if the file already exists or the user has permission to upload files to the given location
3. *Griffin* Conductor uses pre-defined rules to locate a destination *Griffin*/GridFTP server, which connects to the targeted resource, and requests connection information, such IP and port
4. The destination *Griffin*/GridFTP server returns connection information to *Griffin* Conductor
5. *Griffin* conductor sends the connection information back to the client

6. The client makes connection to the destination *Griffin*/GridFTP server directly to send data

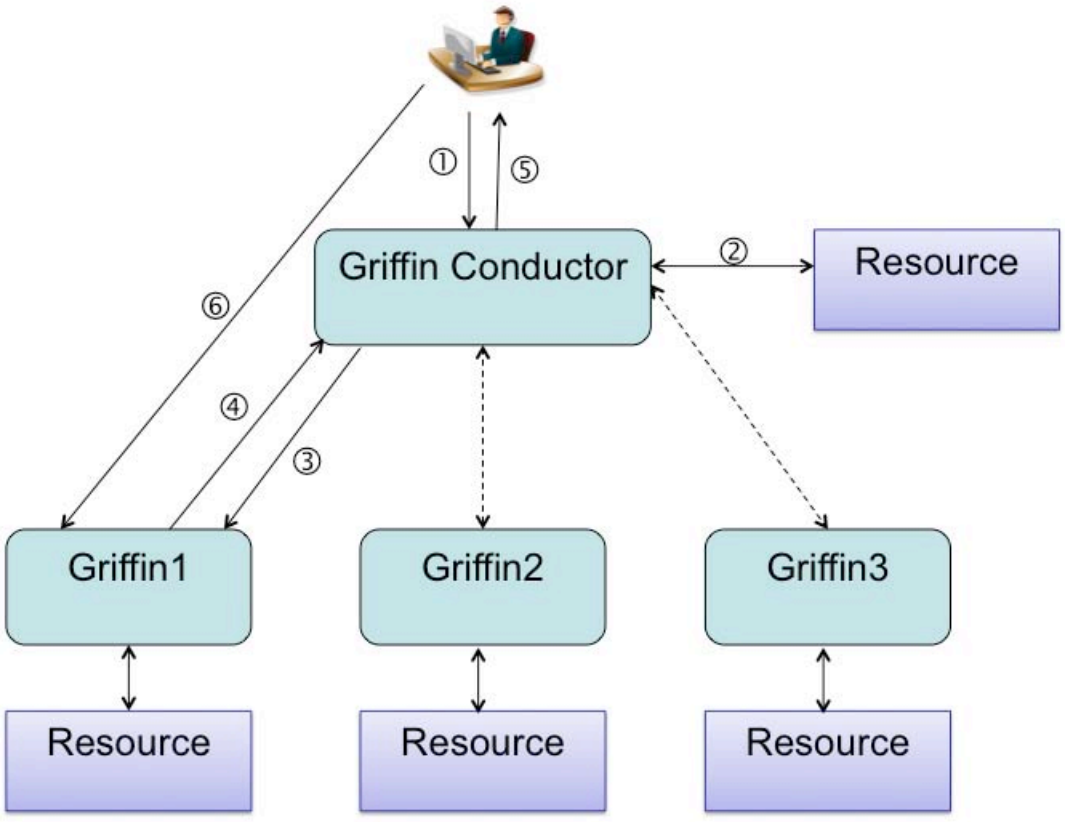

Figure 5-16 Data upload via Griffin Conductor

Data Download

Downloading via *Griffin* Conductor works similarly to uploading. The difference is, when deciding the resource where the client downloads from, *Griffin* Conductor needs to query the distributed file system to check which resources have a replica of the requested file, and the connection information sent to the client will be one of these

resources. As the query is specific to the distributed file system, a plugin structure is employed to provide a common interface to the different query mechanisms. In this implementation, I developed a plugin for iRODS system. In addition, if there is more than one replica, the client can request striping transfer, which is to download concurrently from all of these resources, as illustrated in Figure 5-17, to improve download performance.

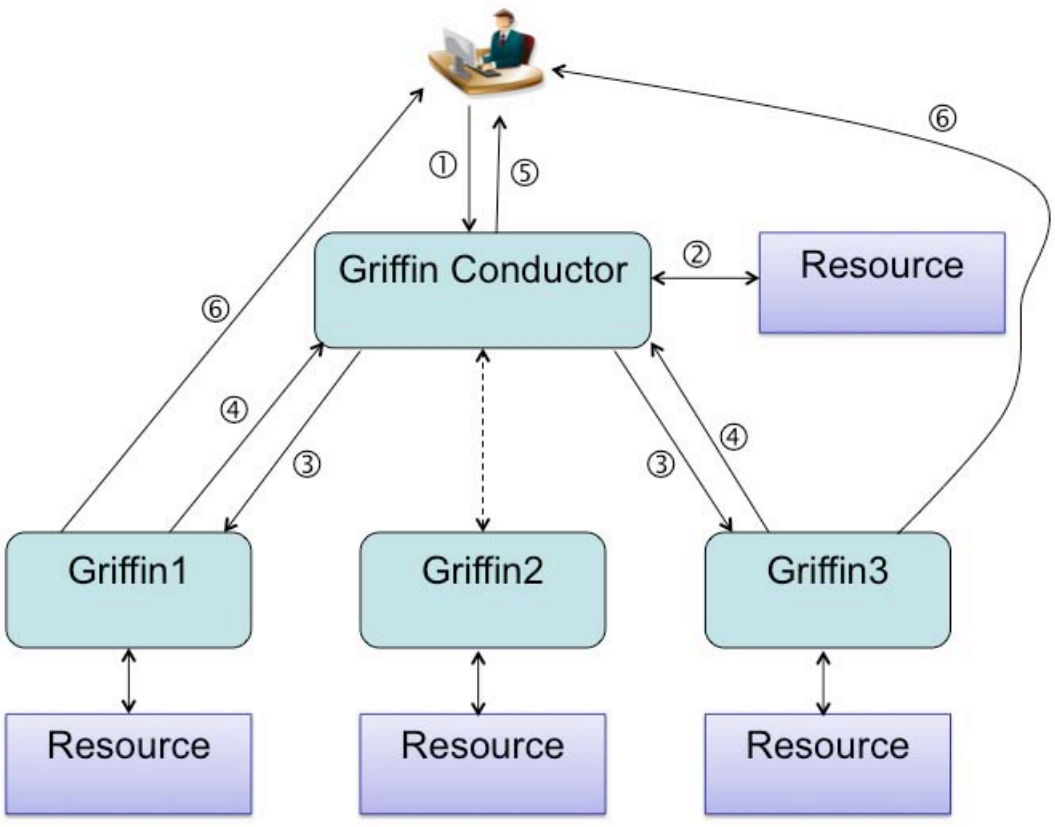

Figure 5-17 Data Download via Griffin Conductor

Implementation

Currently, *Griffin* Conductor is implemented to work with iRODS, which in our case has geographically distributed resources. The conductor redirects clients to the nearest *Griffin* server for upload and download, based on the client's geographical location, which can be worked out with the client's IP address and a Geo-IP database (http://www.maxmind.com/app/ip-location). The Geo-IP database keeps records of IP ranges in each city or area of every country, from which the Conductor knows the location of the client. The Conductor also has a configuration file, to keep a list of existing *Griffin* servers, and a priority list that users from each state should access in order. Figure 5-18 shows a sample configure file. By combining the configuration file and Geo-IP database, *Griffin* Conductor can then work out where to direct the client.

```
<?xml version="1.0" encoding="UTF-8"?>
<location-selector>
        <locations>
                <location id="vpac" hostname="arcs-df.vpac.org" port="2810">
                        <resource name="arcs-df.vpac.org"/>
                </location>
                <location id="ivec" hostname="arcs-df.ivec.org" port="2810">
                        <resource name="arcs-df.ivec.org"/>
                </location>
                <location id="ersa" hostname="arcs-df.eresearchsa.edu.au" port="2810">
                        <resource name="arcs-df.eresearchsa.edu.au"/>
                </location>
                <location id="tpac" hostname="arcs-df.sf.utas.edu.au" port="2810">
                        <resource name="arcs-df.tpac.org.au"/>
                </location>
                <location id="uq" hostname="arcs-df.hpcu.uq.edu.au" port="2810">
                        <resource name="arcs-df.qcif.org"/>
                </location>
                <location id="intersec" hostname="arcs-df.ac3.edu.au" port="2810">
                        <resource name="arcs-df.ac3.edu.au"/>
                </location>
        </locations>
        <selector>
                <!-- ACT -->
                <if country="AU" region="01">
                        <select priority="1">intersec</select>
                        <select priority="2">vpac</select>
```

```
                </if>
                <!-- NSW -->
                <if country="AU" region="02">
                        <select priority="1">intersec</select>
                        <select priority="2">vpac</select>
                </if>
                <!-- NT -->
                <if country="AU" region="03">
                        <select priority="1">ersa</select>
                        <select priority="2">vpac</select>
                </if>
                <!-- QLD -->
                <if country="AU" region="04">
                        <select priority="1">uq</select>
                        <select priority="2">intersec</select>
                        <select priority="3">vpac</select>
                </if>
                <!-- SA -->
                <if country="AU" region="05">
                        <select priority="1">ersa</select>
                        <select priority="2">vpac</select>
                </if>
                <!-- TAS -->
                <if country="AU" region="06">
                        <select priority="1">tpac</select>
                        <select priority="2">vpac</select>
                        <select priority="3">intersec</select>
                </if>
                <!-- VIC -->
                <if country="AU" region="07">
                        <select priority="1">vpac</select>
                        <select priority="2">intersec</select>
                </if>
                <!-- WA -->
                <if country="AU" region="08">
                        <select priority="1">ivec</select>
                        <select priority="2">ersa</select>
                        <select priority="3">vpac</select>
                </if>
                <!-- other areas, e.g. other countries -->
                <else>vpac</else>
        </selector>
</location-selector>
```

Figure 5-18 An example configuration of Griffin Conductor

Related Work

The Globus GridFTP does support striping [155]. Its system architecture consists of PI (Protocol Interpreter) and DTP (Data Transfer Process) modules, where PI is the front-

end that receives requests and DTP handles actual data movements. The communication between PI and DTP is not specific in the official GridFTP protocol, so in the Globus GridFTP, it is implemented to use some proprietary protocol. Furthermore, although each DTP is deployed on each data source, it cannot be accessed directed by the GridFTP client without PI. Griffin Conductor has an more advanced architecture: it uses the standard GridFTP protocol to communicate with other GridFTP servers on each data source; and the service on each data source is a real GridFTP server, which can be accessed directly by any GridFTP client.

5.8 Conclusion

This chapter describes a generic, lightweight and easy-to-use approach to connect an arbitrary data source to the grid, by using a generic file system framework with a GridFTP interface. It allows any data source to be accessed by the grid, enables transferring large datasets, has very little impact on performance for shipping data, and makes it possible to move data between data sources with different protocols easily, efficiently and reliably. The implementation is a standalone GridFTP service based on Java, and does not rely on Globus software stack, so it can run on most operating systems. It is suitable for data sources that offer a Java API. The implementation of Griffin can be obtained from http://projects.arcs.org.au/trac/griffin.

In the next chapter, I will discuss a different topic, which is the National Grid Submission Gateway mentioned in section 3.2.3. It is an important module in the grid infrastructure but it is not part of the National File System. However, it interacts with the

National File System via Griffin. The next chapter illustrates the Gateway in detail and how it is integrated with the work I have done here.

Chapter Six: Grid Submission Gateway

6.1 Introduction

This chapter explores the details of the National Grid Submission Gateway, as briefly outlined in section 3.2.3. It is not a part of the National File System, like the components discussed in Chapter 4 and 5. However, it is closely integrated with the National File System and relies on the National File System in several aspects including data storage and staging. They both play an important role in the national grid infrastructure and form a major user interface frontend for users to take advantages of the resources in the infrastructure.

Modern research requires production and analysis of a large quantity of data. One example is the Large Hadron Collider (LHC), which generates multi terabytes of data every day. Scientists need to use massive storage devices and high performance/throughput computing (HPC/HTC) systems nowadays in their research to store, process and manage the collected data. Over the past several years, a lot of effort has been spent on helping researchers with limited IT expertise to use these complex computing facilities easily. A number of middleware, frameworks and systems have been implemented and deployed. Chapter 2 described a variety of these frameworks and systems that are being used currently.

Over the past few years, the Australian Research Collaboration Service (ARCS, http://www.arcs.org.au) has been investigating a way to help users to more easily use distributed heterogeneous grid resources, which are set up based upon existing

middleware, such as Globus Toolkit, but in different network environments, with different user policies and administrative domains. Grisu is an end-user's tool developed a few years ago, offering a simple web service interface to the multiple Globus Toolkit instances, and allowing easy customization using templates. It is being used by a number of users, and helps users in many ways, but it also has drawbacks. For example, its client is a Java GUI, which needs to be installed on a user's desktop. It doesn't hide all the details of the grid and requires users to have some knowledge of the grid resource; it uses certificates in authentication, and doesn't support Shibboleth; users need to know where the job is executed, so that they can retrieve the output from there; it requires to use a couple of ports other than HTTP ports, which restricts users who are behind a firewall that only allows HTTP traffic. Furthermore, the administrators encountered several problems in scalability and reliability when it was put in production use. Therefore, in this project, I aim for a new approach that can improve Grisu and dramatically change the way that researchers use the Grid. To this end, Chapter 3 outlined the motivation for a national grid submission gateway, which is to simplify compute resource sharing for users from other authoritative domains, and enable interaction with other national services, such as the data storage middleware and the Shibboleth federation.

This chapter goes on to give an overview of the compute job submission gateway, called Grisu2, with details of the architecture specified in section 6.2. Section 6.3 illustrates how Grisu2 is deployed in a real environment for ARCS and how it is integrated with other services in the grid infrastructure in a generic way. Case studies and evaluation will be described in section 6.4. Section 6.5 compares our solution with other related work. Lastly, conclusion will be given in section 6.6.

6.2 The Architecture

The concept of grid has existed for many years and a number of grid middleware systems have been designed to provide a uniform interface to various local resources, which may have different hardware, operating systems, authentication systems, user accounts, file systems and so on. However, existing grid middleware systems are not developed in the same architecture, nor do they expose the same interfaces, so there is the problem of interoperability between different grid middleware systems. Even adopting the same grid middleware, different grid operators may have different configurations or different policies. Current grid middleware doesn't solve the interoperability problem effectively; it is too difficult for developers and users to use, as stated below:

1) Different user systems and user policies are adopted by different organizations. The use of a certificate partly solves the user authentication issue, although the certificate itself is not very user-friendly. Various user policies may hinder the user's ability to use the grid; for example, even if a user has access to the grid, he or she may not have the permission to use a specific application.

2) Different firewall policies are implemented. Although the same service has been installed in multiple grid resources, a user may not have access to all of them because ports are blocked.

3) Existing grid middleware, such as Globus Toolkit, is designed to provide a single interface to different local schedulers, such as PBS and SGE. But this only partly solves the issue with regard to interoperability, because the interface of Globus Toolkit is a web service interface but not a web portal, and it is not easy to use even by developers.

4) If users run jobs in various grid resources, the output will be kept in the local file system of different clusters, which is hard to manage.

The architecture of Grisu2 addresses the above problems by adopting several new technologies. Firstly, to allow users access to resources in a remote organization, I adopted the Shibboleth [23] technology. Shibboleth is emerging as the Single-Sign-On solution for users to easily access trusted systems out of their home organization. A user only needs to use a single credential from their home organization to log in to systems in a remote organization, if they are in the same federation and trust each other. The cross-organization authentication takes place behind the scenes by various Shibboleth components. By using Shibboleth, users are not required to apply for, obtain, use and renew grid certificates. From a user's point of view, this approach avoids the use of certificates, and eliminates the pain of having to remember different credentials and passwords, and they can just use the username and password of their organization.

Secondly, to enable users to more easily use different resources, I extended the idea of the Grisu framework [118] and built a service layer above the Globus middleware, using several Globus components, including the Globus Gatekeeper, Globus GridFTP and MDS [171]. The new service layer features several easy-to-use and standard interfaces, and a Grid Resource Broker that is designed to distribute jobs to a suitable compute resource. Technically, the Grid Resource Broker can submit all jobs to another meta-scheduler, such as Gridway [52]. However, connecting to Globus Gatekeeper directly from the Grid Resource Broker means the broker can make sure of some Globus-specific features, such as the notification mechanism, that is, rather than polling Globus

periodically to find out the job status, it can register a listener on Globus notification service so that Globus will notify the listener of any changes on job status.

Thirdly, to alleviate the tedious process of looking for files from different storage systems, the system automates job processing including data staging and job submission, and uses a central storage for job outputs. Normally, job output is placed in the local storage attached to the supercomputer where the job runs. If a user runs jobs in several supercomputers, the output will scatter among several sites. Hence, it is useful to move all outputs from each individual supercomputer's local file system to a central storage space, and it is easier to make sure a firewall is opened between the central storage and each supercomputer than opening firewalls between any two GridFTP servers. Grisu2 can be configured with the location of the central storage, and to stage out data automatically once a job is done. Users can also specify a different location, such as a local GridFTP server, to stage out results.

In addition, Grisu2 adopts various open standards so it can be easily integrated with other applications. For this, I considered various client APIs and remote APIs. SAGA [172, 173, 174] is one of the most popular client APIs that can connect to different grid systems and data systems. DRMAA [175] is another client API that provides a simple, unique API for accessing heterogeneous grid systems. The difference is that DRMAA only provides APIs for job submission, while SAGA also defines APIs for data access. On the other hand, OGSA BES [119] is a remote API, which offers a web service interface, abstracting basic functions that an execution service should have, and ignoring the differences, which makes it easier to develop and use. Table 6-1 shows an example of the operations related to job control in BES (a job is called an activity in BES). The

advantage is that a BES client can be written in any programming language, as long as it supports web service, and the BES service provider does not need to worry about how the client works. If adopting a client API, I may need to provide a library for any language that users may want to use, and this increases the work and requires developers to know several programming languages in order to write and maintain such an API library. Therefore, the BES specification has been chosen for simplicity and because it is maintenance-free on client side. As an OGF specification, BES is supported by a number of major grid service providers, such as CREAM [176]. To standardize the way of describing a job, I adopted JSDL [35] because it is an OGF standard and supported by a number of grid gateways and meta-schedulers and users don't need to learn a new language before they can submit a job to Grisu2.

BES-Factory Port-type	
CreateActivity	Request the creation of a new activity
GetActivityStatuses	Request the status of a set of activities
TerminateActivities	Request that a set of activities be terminated
GetActivityDocuments	Request the JSDL documents for a set of activities
GetFactoryAttributesDocument	Request XML document containing BES properties

Table 6-1 A summary of operations of BES-Factory Port-type

The architecture of Grisu2 includes three front-end interfaces and a grid resource broker, as shown in Figure 6-1.

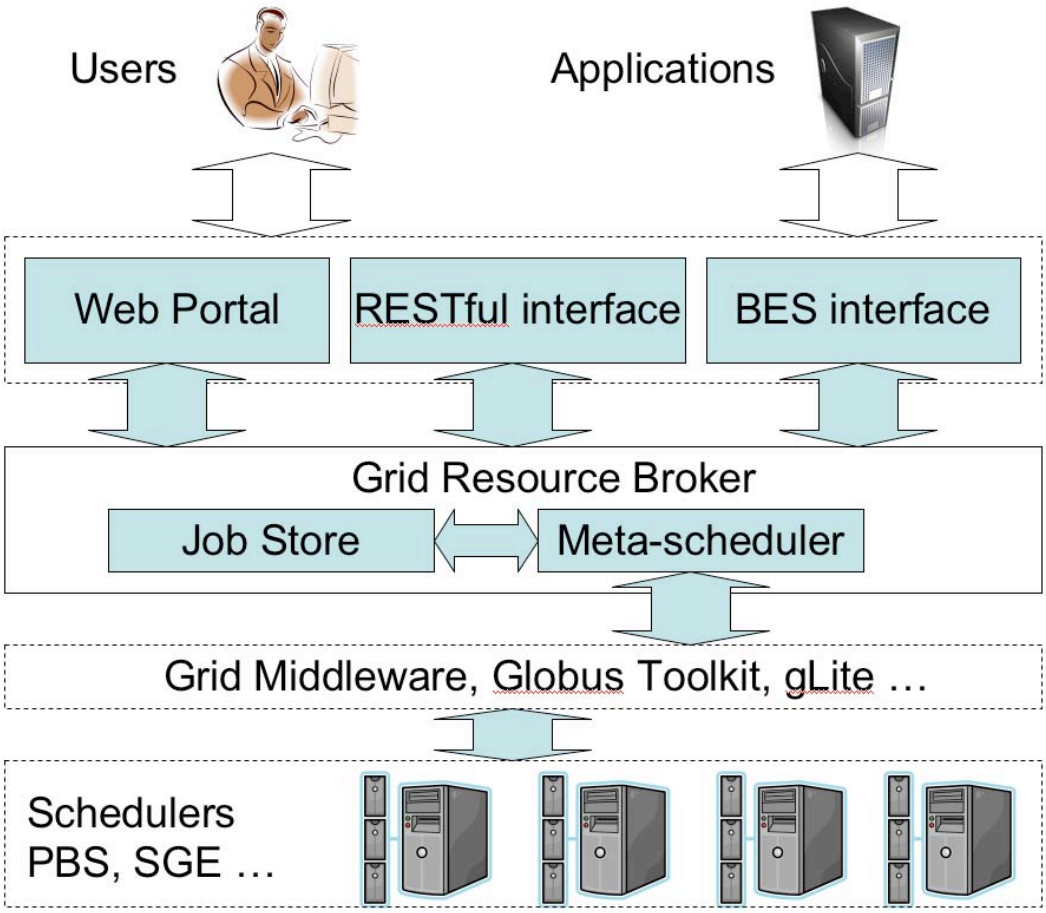

Figure 6-1 Grisu2 Architecture

6.2.1 Front-end interfaces

Grisu2 can be used to access a number of existing regional compute infrastructures that are accessible via the Globus Toolkit. However, the interface of Globus Toolkit is complicated and hard to use. In addition, each different organization has its own user policy, and firewall configuration. Although having many options, the end user may not know how the grid system is set up or where the best location is to run their job. Even knowing about that, the user may be blocked by the firewall. To solve this

problem, Grisu2 acts as an intermediate point by providing several web-based interfaces, for users with different levels of IT skills and different requirements. Users can access to these interfaces with a web browser or other tools that support HTTP access, given that web access is normally allowed in universities. Therefore, users don't need to worry about accessibility to all the grid resources; instead, only Grisu2 server needs to have direct access to all these resources.

6.2.1.1 Web portal

The web portal is a web interface based on AJAX, jQuery (a JavaScript framework) and other web 2.0 technologies. Its main usage is for users to submit and query jobs with a web browser, without needing to install additional software. The portal has a built-in user management system, which keeps a record of users who have ever used this portal, and their basic information that is retrieved from Shibboleth, such as common name and email address so an email can be sent to the users when jobs are finished.

An important feature of the portal is its job template system, which can be used by administrators to define in minutes customized web forms with widgets such that users can take advantage of these forms to enter job parameters and specify input files specific to a particular application. With this feature, the portal is able to support a number of applications, without requiring any development to customize the portal to each application. An example will be given in section 6.4.

6.2.1.2 BES web service interface

The OGSA Basic Execution Service (BES) specification [119] is an open standard defined by OGF. BES gives a standard WSDL for third-party applications to communicate with the BES service. The WSDL includes functions to create a new job, query job status, and terminate an existing job. In fact, Grisu2 conforms completely to the OGF HPC Basic Profile specification [177], which consists of not only the BES interface as the main interface to interact with a job submission gateway, but also WS-Basic Security Profile [178] as the authentication method, and JSDL [35] as the job description language. The reason for choosing BES is because it is a standard web service interface, which can be used by different programming languages, without needing a client library, and also it is widely adopted by various organizations [179, 180] and efficient [181].

6.2.1.3 RESTful interface

RESTful interfaces [182] are in wide use, due to their ease of development and use. Many websites, such as Yahoo and Flickr, have developed their RESTful interfaces for developers to hook in third-party applications. In Grisu2, the RESTful interface provides an easy additional way for users to submit jobs and query jobs. Users can write some simple scripts to invoke the functions Grisu2 offers, which required a fair bit of work by an experienced developer in the past. Job submission via the RESTful interface is as simple as posting a JSDL file to the portal, while the JSDL file complies with the

same specifications as what are used in BES interface. Section 6.4 will give examples of how this is used in a variety of use cases.

6.2.2 Grid Resource Broker

Grid Resource Broker is the core of Grisu2. It receives job requests from users and interacts with other grid services to coordinate the execution of jobs on resources in the grid. It has two key components, a Job Store and a Meta-scheduler.

6.2.2.1 Job Store

Job Store keeps job details in a relational database, which is managed by Hibernate [183], a database access framework to map database schema into Java objects. This means Grisu2 can use any Hibernate supported relational database, such as MySQL or Oracle, without changing one single line of code. Job Store saves all information about a job, such as its name, parameters, status, and the queue where the job is run, such that every query of job details may be run against the Job Store, rather than sending the request to the remote compute resource. This makes the system faster and more loosely coupled from these compute resources. Furthermore, after the job is done, it will be removed from the compute resource; however, information about the jobs will still be kept in Job Store and job files are kept in the central storage, until the user deletes it explicitly. In this sense, Job Store is a job archive to users, which enables users to look at details of all previous jobs, including links to input data and output files.

6.2.2.2 Meta-scheduler

The design of this meta-scheduler follows the principle presented by Schopf [184]. In this principle, a meta-scheduler has three phases, resource discovery, system selection and job execution. In the resource discovery phase, the meta-scheduler collects real-time information from a Grid Index Information Service, such as MDS or BDII [45]. In the system selection phase, it compares the user's hard requirements, such as the required application and its version, the number of required CPUs, and the required CPU time, with the information collected before, and generates a list of candidate LRMS job queues that are suitable to run the job. Then, a ranking algorithm is applied on this list to work out the best resource for the job.

The ranking algorithm lies in a component that can be easily changed anytime. At the moment, the purpose of the algorithm is to spread the workload to all available resources evenly. It keeps an array of all resources with the free job slots number as the initial value. Jobs will always be sent to the resource with the highest ranking; if they have the same ranking, resources will be used one by one in alphabetical order. If a job is executed, the ranking value of that resource is decreased by one. Once the value becomes zero, no job will be sent to this resource. When the values of all resources become zero, the ranking values in the array will be reset to their initial values and all this will start again. The meta-scheduler distributes jobs to the selected compute resource via Globus middleware. If anything goes wrong during staging-in or execution, the second best option in the list will be tried, and so on.

The meta-scheduler controls every job with a finite state machine. Figure 6-2 shows the state-transition diagram of Grisu2. The states are utilized by Grisu2 for workflow control, and also shown to users so that users know what is going on. The design of this state machine is mostly influenced by the BES specification. It is a specialization of the state model in the BES specification, and expands one of the BES simple states into several complex states, so clients that understand basic BES states can interact with it. Applying a standard also means the BES interface can make use of it seamlessly, without needing to do a mapping. All states starting with Running are sub-states of Running in the simple BES state machine. Other states exist in the BES state machine except New.

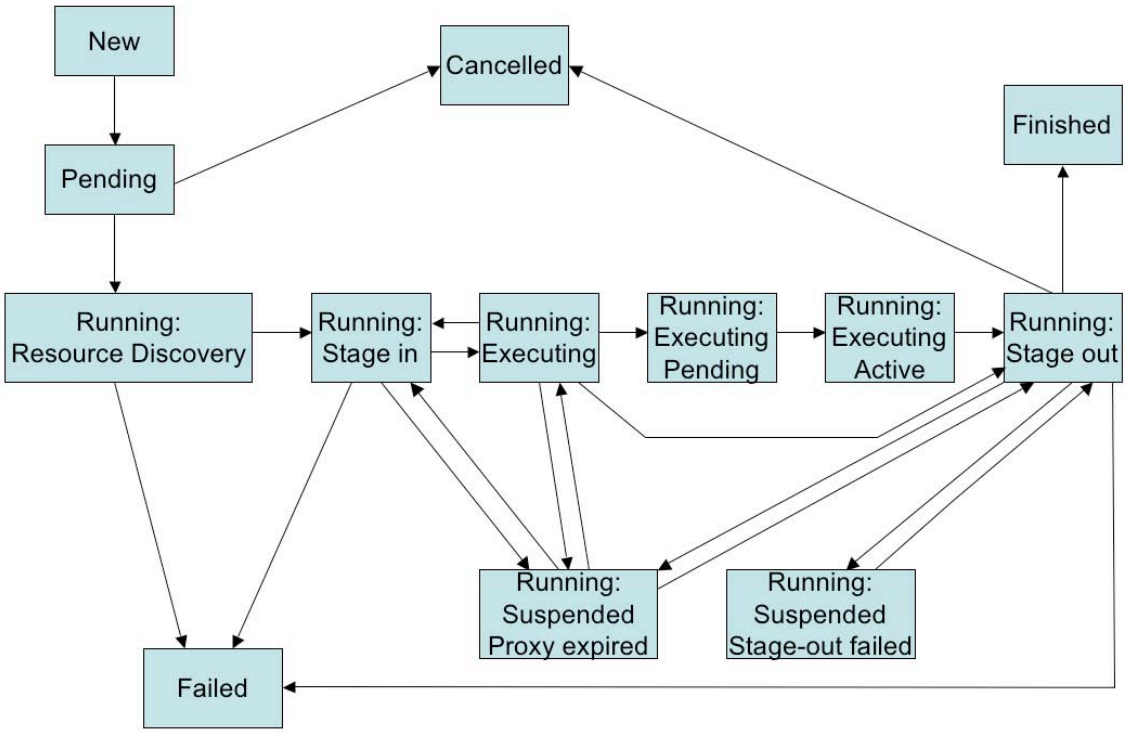

Figure 6-2 Grisu2 state machine diagram

The first state, New, is the default state when a job object is initiated. When the job is submitted to Grisu2, its state is changed to *Pending*, meaning that the job is waiting for Grisu2 to process. It remains *Pending* in a situation where Grisu2 server does not have enough resource to process a new job. Once the job is started, the state is changed to *Running: Resource Discovery*, which refers to the phases of resource discovery and system selection. By the end of this state, the meta-scheduler should have a list of candidate queues with ranking values; otherwise the whole job execution process will stop. The following state is *Running: Stage in*, which means the meta-scheduler sends input data to the target queue; if it fails, the second queue in the list will be tried, until all queues are tried; if none of them works, the whole process will stop and the job will be

tagged as *Failed*. If input files are staged in successfully, *Running: Executing* is the next state, when jobs are sent to the remote Globus gatekeeper for execution. If the destination Globus gatekeeper can send back notifications, the state may change accordingly to *Running: Executing Pending*, meaning pending in Globus, and *Running: Executing Active*, meaning active in Globus. If no notification is received from the Globus gatekeeper, the job's status will still be *Running: Executing*. In this case, a background thread will check its status in Globus periodically until the job is done or failed. Then Grisu2 will move the job to the next state: *Running: Stage out*, which transfers output data back to the central storage, and then the whole process will finish. In the state machine there are two *Running: Suspended* states, which indicate the job is suspended due to some reason but is able to continue if the situation changes. For instance, *Running: Suspended Proxy Expired* happens when the cached proxy is expired, and the user hasn't logged in to Grisu2 for a long time. To fix that, the user just needs to log in to Grisu2, and a proxy will be automatically cached, then the user can resume these suspended jobs. *Running: Suspended Stage-out failed* happens when Grisu2 cannot stage out results from the compute element to the destination due to problems at either end, or the network connecting them. Grisu2 will try a pre-configured number of times should this happen, and suspend the job if no success. In this case, the user has to wait until the problem is fixed, then resume the job so that Grisu2 can continue to stage out data and do the rest of work.

6.3 System Integration and Deployment

The system is dependent on a number of other grid services. This section outlines the integration, and gives an example of how it has been deployed on the ARCS Grid, a national grid service in Australia. To keep the system generic, all customization during deployment has been done as system configurations, while the integration has been achieved with the use of other standard/generic components. Figure 6-3 shows the interactions between Grisu2 and other standard grid services that are used by the ARCS Grid.

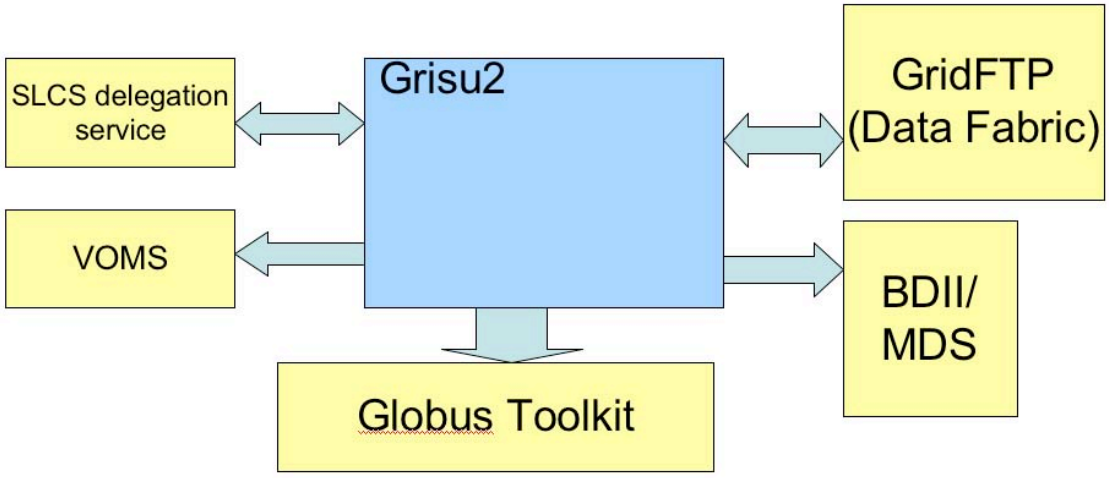

Figure 6-3 Grisu2 and other ARCS services

6.3.1 Authentication and authorization

Globus accepts grid certificates as the only way of authentication but they are not easy to obtain and use. To lower the barrier to user registration and hide the complexity

for users to use the grid, Shibboleth has been chosen as the main authentication method by the portal. The Grisu2 portal is configured as a Shibboleth service provider. Users can then use their credentials for their own organization, without the need to remember a different set of usernames and passwords or manage a grid certificate. As the grid systems require GSI authentication and Shibboleth does not involve any certificates but only username and password pairs, Short Live Certificate Service [82] (SLCS) is employed to issue short-lived X.509 certificates to users for accessing Globus services based on successful Shibboleth authentication. To issue SLCS certificates to other Shibbolized services on behalf of the requesting user, the SLCS delegation service, an add-on to SLCS service, is exploited for Grisu2 to receive certificates so that Grisu2 can use the users' certificates to access grid services. Authorization is achieved by the use of Virtual Organization Membership Service (VOMS) [43], a generic service that manages Virtual Organizations (VO) and VO groups. ARCS requires users to register before using its service, where users have to sign a user agreement. Once registered, users will be placed into a VO. With the given SLCS certificate, Grisu2 queries against VOMS to check whether the user has registered. If yes, the user is then allowed to use the portal.

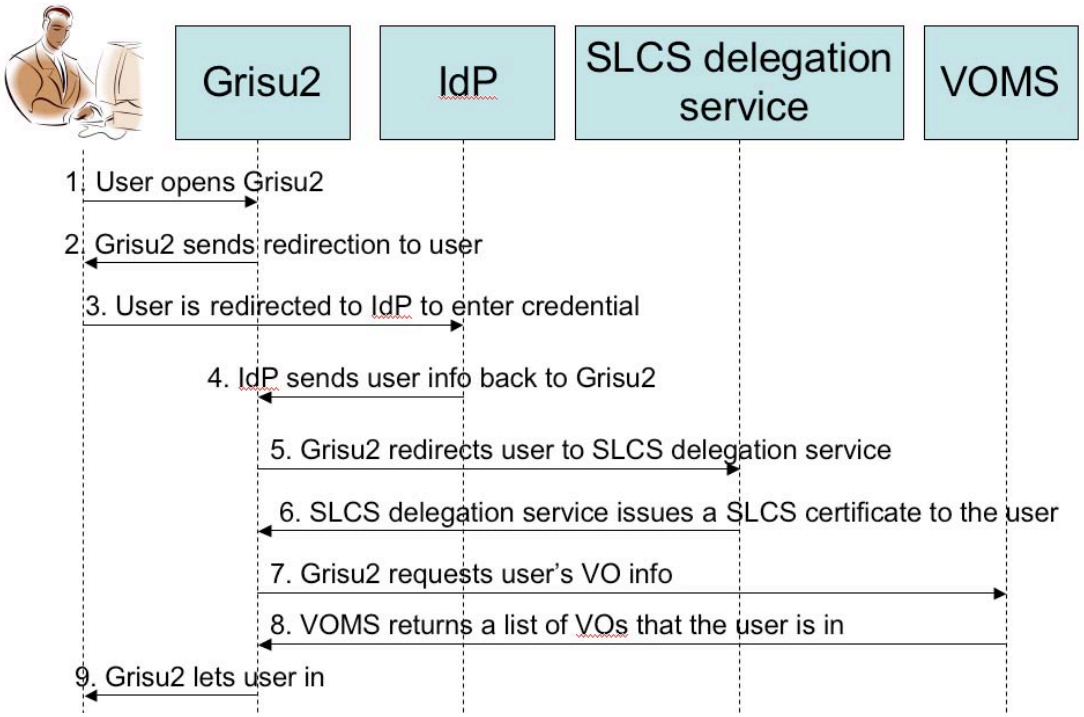

Figure 6-4 Authentication with Shibboleth and SLCS delegation service

Figure 6-4 shows the login procedure in detail. In fact, not all of the steps are required. For instance, if the user has already logged in with Shibboleth and the Shibboleth session remains valid, step 2, 3 and 4 will be skipped; if the user already has a proxy cached in Grisu2, step 5 and 6 will be skipped. The cached SLCS certificate will be valid for 10 days, so the user does not need to get a new certificate from SLCS delegation service every time.

6.3.2 Integration with Data Fabric

The integration of the Data Fabric makes it possible for Grisu2 to make use of the storage provided by the Data Fabric, which is based on a virtual file system, iRODS [18], with Davis [102], discussed in Chapter 4, and Griffin [103], discussed in Chapter 5. A higher-level description of how the Gateway fits into the whole infrastructure can be found in Chapter 3. The Grisu2 web portal has embedded the ARCS Data Fabric's web interface. Because both the Grisu2 portal and Davis are Shibboleth-protected, users do not have to log in twice when using the embedded Davis interface, e.g. to share their job output to other researchers. The GridFTP interface of the Data Fabric is heavily used by Grisu2 in data staging. To some extent, if you consider Grisu2 as the gatekeeper of all compute elements in the grid, the Data Fabric is its default storage resource. Input data from the user's desktop will be placed in the Data Fabric when submitting a job from the web portal. After the job is done, output data will be kept in the Data Fabric. Grisu2 uses the GridFTP protocol to interact with the Data Fabric, which means users can also specify other GridFTP resources, such as a local GridFTP server, to store data.

Figure 6-5 shows the relationship among Grisu2, the HPC cluster and the Data Fabric. Normally, each HPC cluster (with a Globus interface) is associated with a GridFTP server. Both the HPC cluster and its associated GridFTP server can access a shared area of the file system. Input data will be transferred into the shared storage space via GridFTP, then the HPC cluster can process this data. Thanks to GridFTP's third-party transfer feature, when transmitting data between an HPC cluster's GridFTP server and

the Data Fabric, data flows directly without going through Grisu2, which only needs to establish a control channel to each of them to control and monitor the transfer.

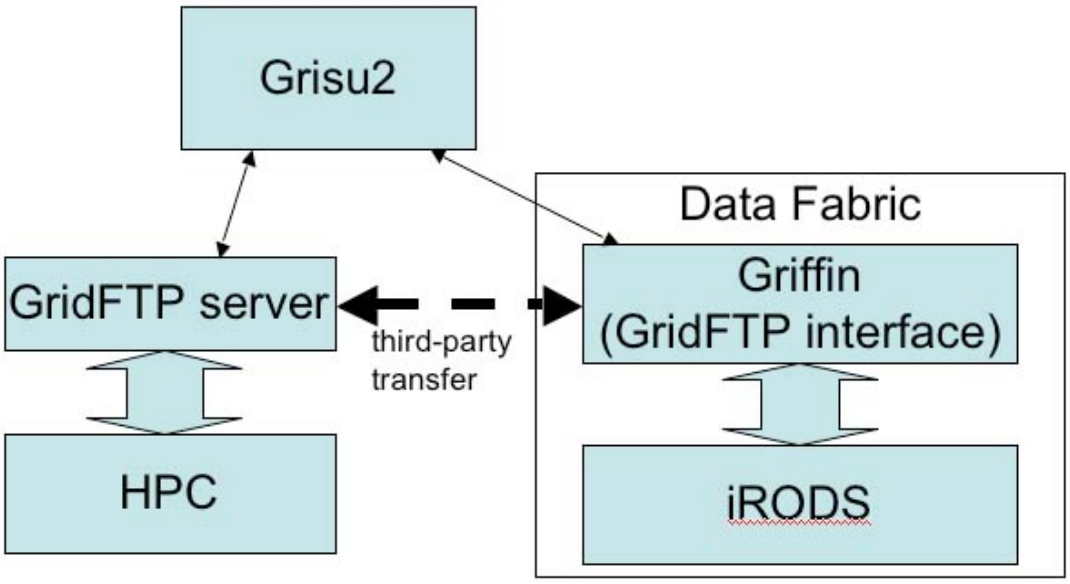

Figure 6-5 Integrating Grisu2 with Data Fabric

6.3.3 Job execution in a national grid environment

Grisu2 views a typical job execution as involving a number of steps, and several components, as shown in Figure 6-6. This is implemented with the use of an open-source framework, Spring Batch [185], which is a batch framework aiming to provide reusable functions and automate job processing. It suits this situation very well, because it is designed to handle a job in simple steps with an XML configuration file, for the case where a workflow system is too heavy to use. The whole job submission process is divided into steps, while each step represents one action in the process, with pre-defined

exit statuses. Spring Batch will decide what the next step will be based on the exit status, or may just terminate the job. Since each action is wrapped in a step, it is easy to change the workflow, or replace one action with a different action in the future. In addition, Spring Batch provides a mechanism to re-do a step if it fails until it gets to a state that satisfies the job submitter. Spring Batch maintains a pool of running jobs, whose number can be configured according to the hardware specification of the server, so as to ensure the stability of Spring Batch and Grisu2, and prevent it from running out of resources. When the number of running jobs reaches the limit, Spring Batch will put new jobs on hold until any existing job in the pool is finished. Scalability can easily be achieved by Spring Batch. If there are more jobs than one instance can handle, more Spring Batch instances, which are Grisu2 instances in our case, can be set up to share the load, as described in the next section.

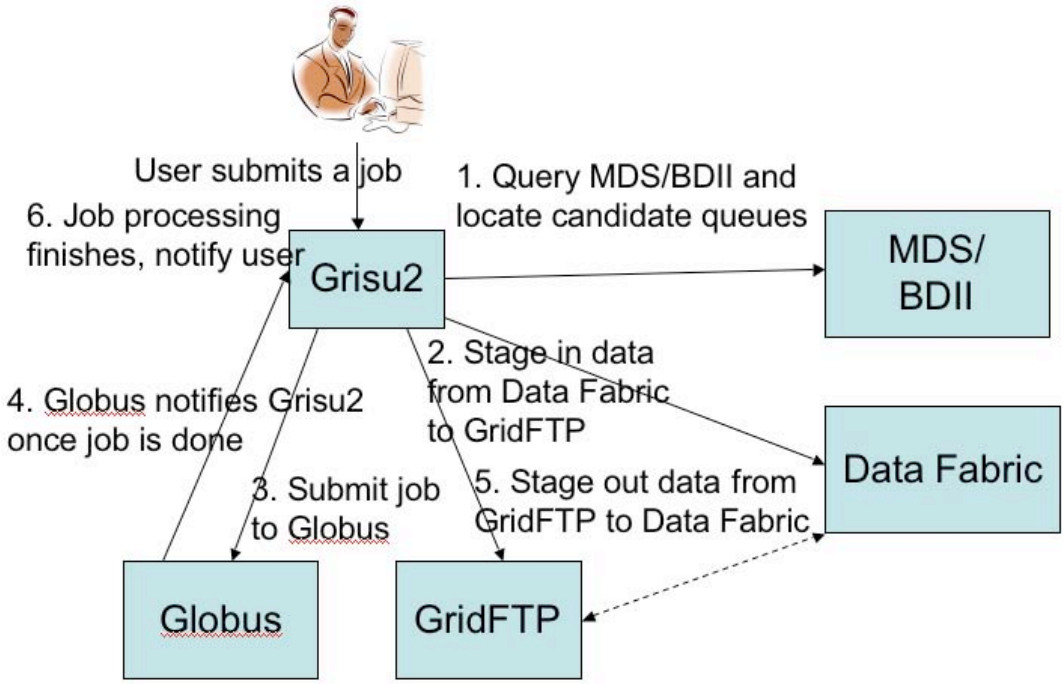

Figure 6-6 Steps of job processing in Grisu2

Figure 6-6 shows all of the steps in job processing. Jobs can be submitted via any of the three interfaces, and monitored via any of them. When submitting jobs from the web interface, users can upload input files from their desktop to the Data Fabric, so when the job is processed, input data can be staged in from the Data Fabric, rather than the users' desktop. This makes it possible for users to create a large number of jobs in a short time, without having to wait for input data to upload. After a job is submitted, it is pending in Grisu2. If the Spring Batch job pool is not full, Spring Batch will start the job, which proceeds thus:

1. Grisu2 queries MDS or BDII, based on job parameters, such as application name, application version, number of CPUs and CPU time, to find out a list of available queues that have the required application installed and available job submission

resource. A ranking value will be attached to each queue, and used as the priority in step 2. The calculation of each ranking, discussed earlier in section 6.2.2.2, is currently based on several factors, such as the number of jobs the queue has run, and the number of its free job slots published in MDS. The queue with a larger ranking value will be given a higher priority. The list is then saved into Grisu2's Job Store for later use.

2. Grisu2 tries the items from the candidate queue list in the order from the highest ranking to the lowest ranking, unless the user specifies a desired one. In this step, Grisu2 tests each queue in the list to see whether its associated GridFTP is accessible, readable and writable. If yes, a working directory will be created and input data will be staged in. If not, try next queue according to the priority.
3. Submit this job to the selected queue through Globus gatekeeper. If the job submission fails, it will go back to step 2 to try the next queue in the list, and do stage-in again.
4. Globus sends back notification if status is changed. Grisu2 will also query Globus every 5 minutes if no notification has come back. This makes sure Grisu2 is aware of any events. Once Grisu2 receives a notification of job finish or failure, step 5 will be triggered.
5. Grisu2 copies all files from the working directory in the compute resource to the Data Fabric, and removes them after this. Because the cluster can only use limited storage space, this makes sure space will be freed up after each job process and makes room for the next job process.

6. This step sets all necessary flags and statuses in Grisu2 to mark the job to finished, or failed. Grisu2 also notifies the user by sending an email. The email notification is useful because the user does not need to keep an eye on the job but will get a message when it is done, so he/she can focus on other research work during this time.

6.3.4 Scalable deployment

The component design of Grisu2 enables the separation of the portal and submission component in production system deployment. The production system structure is shown in Figure 6-7.

In this deployment, web interfaces and submission nodes are in different hosts, and the communication between them is via the Job store, which is a standalone database instance. Users take advantage of the web interfaces to create jobs, query jobs and delete jobs. Once new jobs are created, they will be stored in the Job Store, while each submission node monitors any changes in the Job Store. If a new job is found, a Submission Node will claim ownership, thus other submission nodes will not process the same job again. Each Submission Node is configured with a limit of concurrent jobs. If the limit is reached, it cannot retrieve more jobs and has to wait until its own jobs to finish. If there are more jobs to run, it is feasible to run more submission nodes. If there are fewer jobs, likewise, redundant submission nodes can be shutdown to free up resources. This structure is very suitable for the infrastructure with Virtual Machine provisioning to run submission nodes according to the demand.

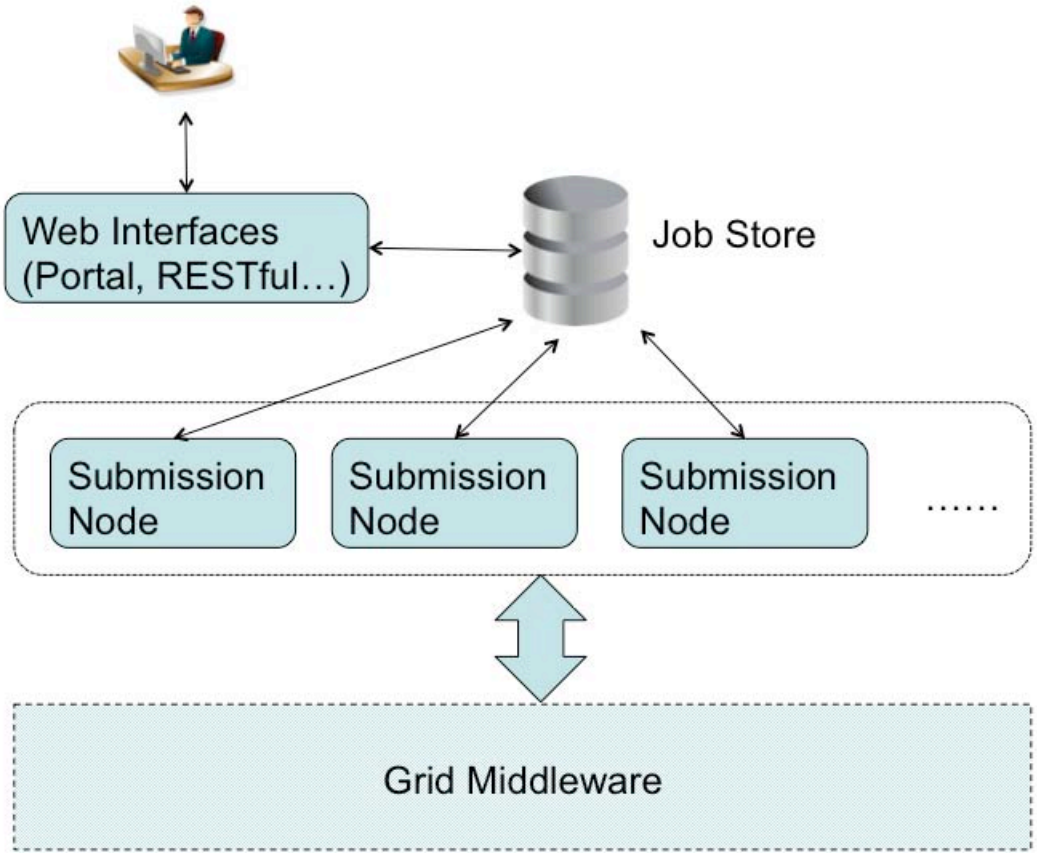

Figure 6-7 Grisu2 production system deployment

6.4 Use cases

Users would not be interested in Grisu2 if it did not support specific applications they want to use. This section briefly outlines how Grisu2 fits into users' everyday work and some examples of how users have taken advantage of the capabilities of Grisu2 in their research.

6.4.1 Support of various applications

One key feature of the system is its support of different applications, rather than being a portal for a particular application. This feature comes from several design decisions. Firstly, the system allows users to specify the application name and application version when they submit a new job, and this information will be stored in Job Store. Secondly, the adoption of the JSDL specification enables users to specify the application name and application version if they use the RESTful or BES interface. Thirdly, the web interface is equipped with a template system, which can be used to customize the web form for each application, given that each of them may have a different set of job parameters or input files. Lastly, Grisu2 retrieves information from MDS, to where all regional grid operators publish information about what applications have been installed and what versions are available. To submit a job, the user only needs to select an application and they will get a web form, customized for that application, to fill in. After a job is submitted to Grisu2, the Grid Resource Broker will first query MDS to get a list of queues that have the required application installed, and then one of them will be chosen based on ranking values, as described in section 6.2.2.2.

A template is a property file defining a job submission form with widgets. The job submission form may contain widgets such as wall time input, CPU number input, input file upload, etc. This makes it easy for system administrators to create a form for a new application without knowing how to program. In the first version, Grisu rendered the template with Java Swing components. In Grisu2, templates are shown in HTML format as it is a web-based system. Figure 6-8 is the web form for the LAMMPS [186]

application, which has widgets to upload input files and enter parameters. Figure 6-9 is the web form for the Underworld [187] application. As can be seen, different widgets are displayed for different applications. Figure 6-9 also has the advanced option panel expanded, where users can choose a specific VO, a specific version and/or a specific queue to run the job, rather than letting Grisu2 to make the decision. This feature is useful to advanced users who have more understanding of the Grid.

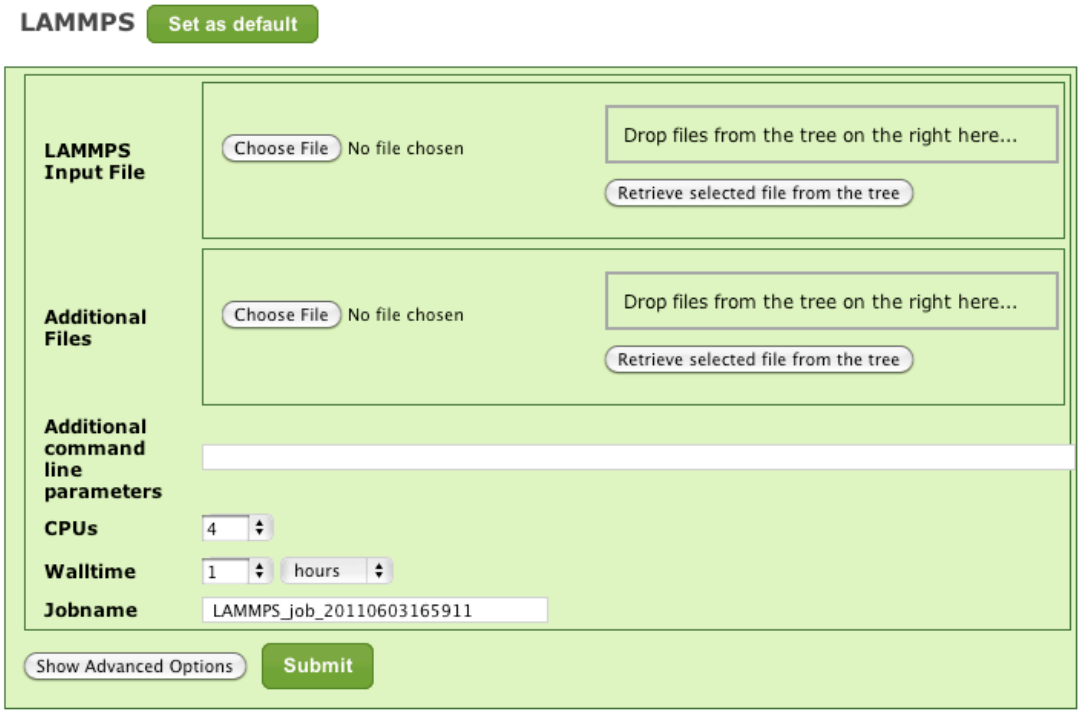

Figure 6-8 A UI template for LAMMPS

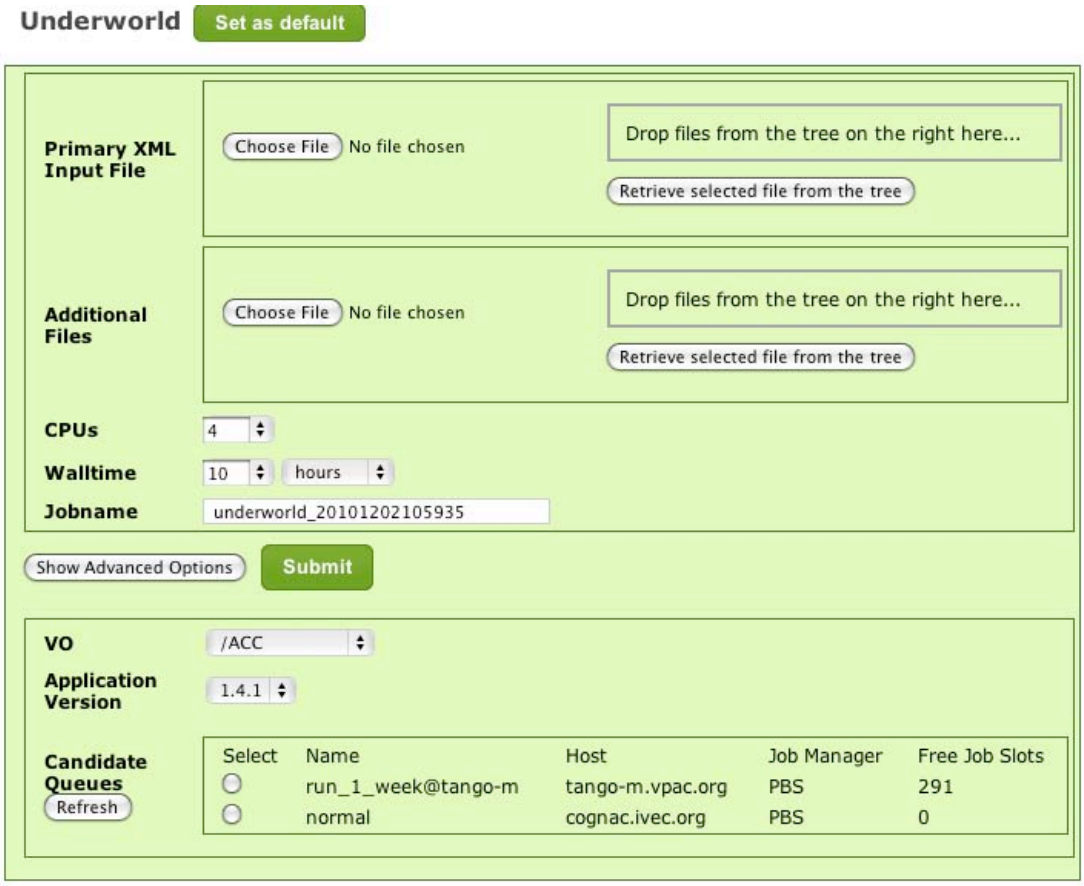

Figure 6-9 Job template for Underworld in Grisu2

Figure 6-10 is the template file of Underworld. In the beginning of the template, a user can designate the application name, the application version and the command line to run this job. Followed are five widgets, which are a text box widget to enter job name, a combo box widget to select or enter CPU number, a combo box widget to select or enter wall time, a single file widget for the main input file and a multi file widget for additional input files. Users can select a file from local file system or from the remote central storage, Data Fabric. There are some common parameters for the widgets, such as its size, its title, which are useful when displaying the widget.

```
commandline = Underworld ${file}
application = Underworld
applicationVersion = 1.4.1
= Generic =
[jobname]
type = Jobname
defaultValue = underworld
title = Jobname
size = 2000x70
[cpus]
type = Cpus
title = CPUs
size = 100x100
defaultValue = 4
[walltime]
type = Walltime
title = Walltime
defaultAmount = 10
defaultUnit = hours
size = 200x100
[file]
type = SingleInputFile
title = Primary XML Input File
size = 2000x70
[file2]
type = MultipleInputFiles
title = Additional Files
size = 2000x210
```

Figure 6-10 Underworld template configuration in Grisu2

To date, about 20 applications are supported for various disciplines, such as BEAST, BLAST and MrBayes for Bioinformatics, LAMMPS for Computational Chemistry, Underworld for Earth science, CalculiX and Meep for Engineering, Octave and R for Mathematics and statistics, as well as ESyS-Particle, POV-Ray, GrADS, and user-developed applications, such as Java and Python. Apart from these, Grisu2 also provides a generic template so that users can enter an arbitrary executable and arguments

just in case the application they want to use is not in the template list, or they want to do something special.

Figure 6-11 shows the main page, which displays a full job list of all applications in the upper half of the screen. Users can also filter a particular application using the advanced search option. If the user clicks a job, details of this job will be displayed in the lower half of the screen. In job details section, there are five tabs. Job properties will be shown as the first tab by default. Other tabs include logs, a list of input files, candidate queues and files that are associated with this job. For instance, if Candidate Queues is clicked, it will show a list of queue information gathered from MDS/BDII. It only has what is interesting to Grisu2, such as the queue name, gatekeeper contact string, default GridFTP URL, module name of the application. It also shows which queue has been chosen to run the job. In the Files tab, users can preview a file, before downloading it. In the current version, users can preview images, text files and video files. When a job is running, the Files tab shows files in the cluster's local file system, so users can have an idea of what outcome has been produced so far from the execution. After the job finishes, the Files tab shows the files in Data Fabric, where the output has been moved to, and will be kept as an archive.

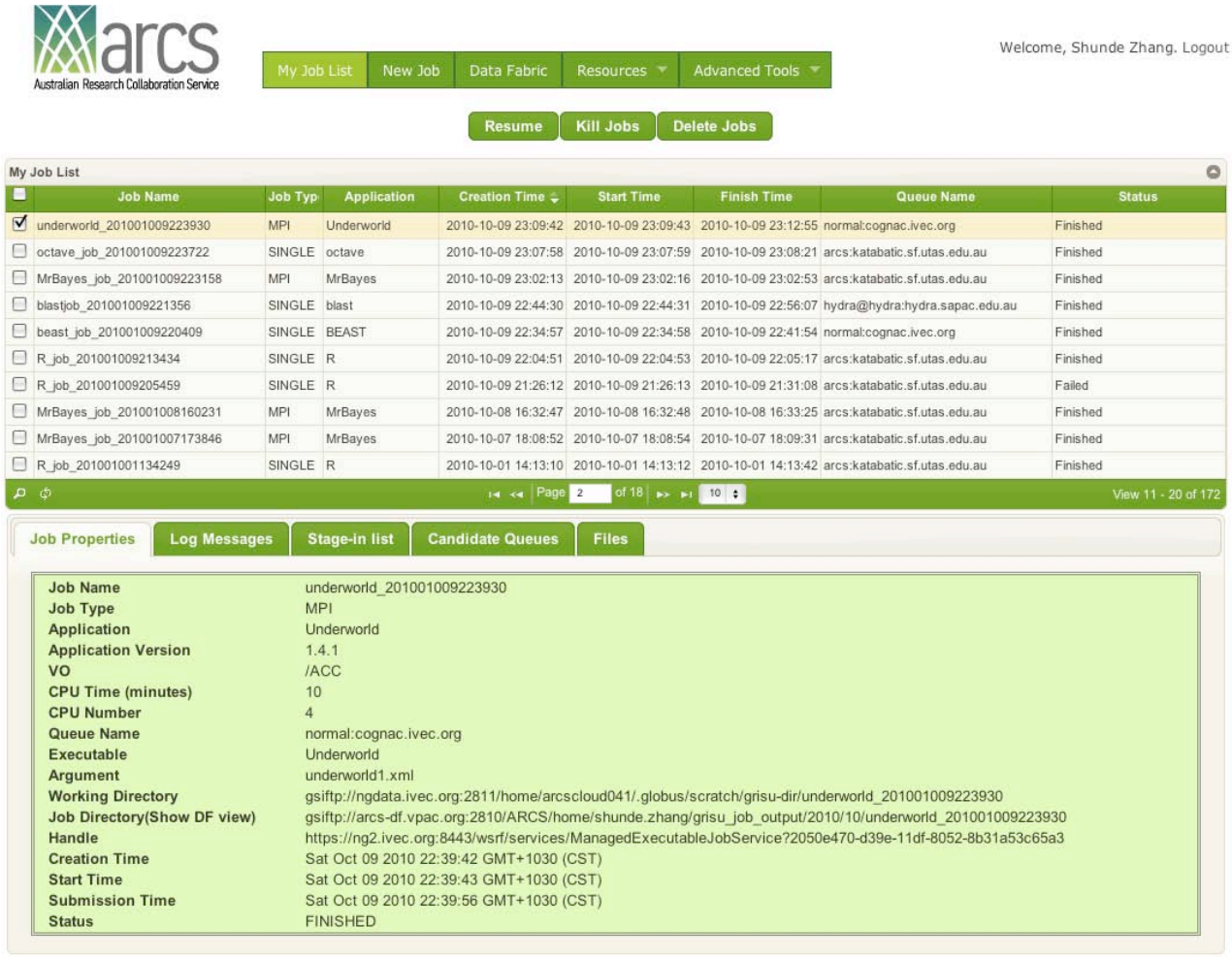

Figure 6-11 Grisu2 front page - Job list and job details

Apart from general features, special features are also supported. Figure 6-12 shows a special tool for the application Underworld. The output of Underworld job includes a table of numbers. With using a JavaScript plotting widget, the web interface can visually display the table of numbers as a line chart.

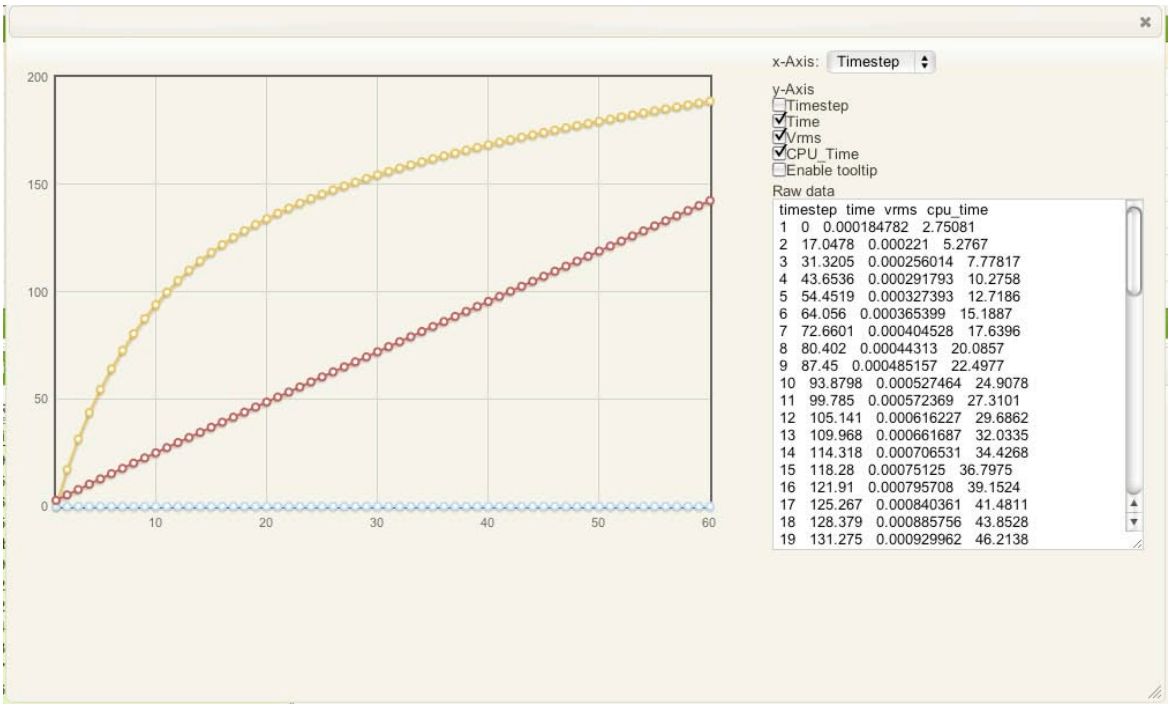

Figure 6-12 Underworld data analysis chart in Grisu2's web interface

6.4.2 Support for a large number of concurrent jobs

Support for a large amount of jobs is achieved by three main features. Firstly, Grisu2 is running in an asynchronous manner, that is, jobs that are submitted via the web frontend will be saved into Job Store, which only requires to do some database insertions, while another service running in the background processes these jobs in a different host, which may include doing some time-consuming tasks such as communicating with Globus Gatekeepers and GridFTP servers. Because local database operations are much faster than submitting jobs to Globus or querying an external service, creating thousands of jobs can be accomplished in a short period of time. Once all jobs are created in Job Store, the Grisu2 Grid Resource Broker will process them in steps that are described in

Section 6.3.3. Moreover, Grisu2 implements JSDL Parameter Sweep Specification [188], which allows users to create multiple jobs in one submission, and decreases the number of requests sent from a client, hence shortens the time used in job creation. Lastly, the RESTful interface and BES interface are the best and only way to send a large number of jobs as it is not realistic to create thousands of jobs by manually filling in web forms. In the current version, the RESTful interface provides a few simple but essential functions, including submitting, querying, and deleting a job, as in Figure 6-13.

API Method: submit a new job
Request: /grisu/jobs?type=jsdl
HTTP method: POST
Request body: JSDL content
Response: <id>*id*</id>

API Method: get job details
Request: /grisu/jobs/*id*.format
HTTP method: GET
Format: xml, json
Response: Job details in JSON or XML format

API Method: delete a job
Request: /grisu/jobs/*id*
HTTP method: DELETE
Response: {“message”:”success/failed”}

API Method: resume or kill a job
Request: /grisu/jobs/*id*?action=action_name
HTTP method: POST
Parameters: action_name: resume, kill
Response: {“message”:”success/failed”}

Figure 6-13 RESTful API of Grisu2

With the RESTful interface, users can write a simple script to submit a job to Grisu2, without needing to install a client application. Figure 6-14 is an example of a Python script, and Figure 6-15 is a version written in shell script. They both submit a job in JSDL format via the RESTful interface. This is a useful feature to integrate Grisu2 with other applications. For example, Open Office supports macros that can be written in Python. It is easy to write a macro so people can submit a large amount of jobs from Open Office's spreadsheet.

```
f = open(jsdlfile, 'r')
data = f.read()
f.close()

auth = 'Basic ' + string.strip(base64.encodestring(username + ':'
+ password))
conn = httplib.HTTPS("grisu-dev.arcs.org.au")
base64string = base64.encodestring('%s:%s' % (username,
password))[:-1]
conn.putrequest("POST", "/grisu/jobs?type=jsdl")
conn.putheader("Authorization", auth)
conn.putheader("Content-type", "text/xml")
conn.putheader('Content-length', str(len(data)))
conn.endheaders()
conn.send(data)

# get the response
statuscode, statusmessage, header = conn.getreply()
print "Response: ", statuscode, statusmessage
print "headers: ", header
res = conn.getfile().read()
print res

conn.close()
```

Figure 6-14 Script snippet in Python

```
curl -H "content-type:text/xml" -XPOST -k -u username:password -d "`cat myjob.jsdl`"
https://grisu-dev.arcs.org.au/grisu/jobs?type=jsdl
```

Figure 6-15 Job submission with curl to Grisu2

According to our experience, it takes 2-3 seconds to submit a single job using JSDL via RESTful interface. This is acceptable if the user only submits a couple of jobs, however, when it comes to hundreds or thousands of jobs, this is too slow. If these jobs have some degree of similarity, such as using the same executable, or processing the same input file, Parameter Sweep job [189] scheduling may be suitable as a means of facilitating more rapid completion of the suite of concurrent jobs. A Parameter Sweep is a job that internally defines a collection of jobs running multiple iterations of the same command but with different input values and generating different outputs. It is widely used by Grid systems such as Nimrod [190]. Part of the definition of a Parameter Sweep is a range of values delineating the possibly multi-dimensional space to be explored – one job is generated for each point of the defined range. Often, the range is expressed as an array of values, or the combination of a start value, an increment value and an end value constituting an iterative definition of the space explored (nested iterations define a multi-dimensional space). The JSDL Parameter Sweep Extension defines a schema to append Parameter Sweep definitions to a normal JSDL document, such that when the JSDL document is processed, a collection of jobs are generated and one or more parameters in the JSDL can change between each successive job. Figure 6-16 shows an example, which

is a portion taken from a Parameter Sweep job written in JSDL, and the complete version can be found in Appendix A.

```
……
<jsdl:Application>
               <jsdl:ApplicationName>Mira</jsdl:ApplicationName>
            <jsdl-hpcpa:HPCProfileApplication >
                <jsdl-hpcpa:Executable>mira</jsdl-hpcpa:Executable>
                <jsdl-hpcpa:Argument>--project=</jsdl-hpcpa:Argument>
                <jsdl-hpcpa:Argument>--parameterfile=miraIntersect.txt</jsdl-
hpcpa:Argument>
                <jsdl-hpcpa:Argument>-MI:sonfs=no</jsdl-hpcpa:Argument>
            </jsdl-hpcpa:HPCProfileApplication>
        </jsdl:Application>
……
            <jsdl:DataStaging>
               <jsdl:FileName>any</jsdl:FileName>
               <jsdl:CreationFlag>override</jsdl:CreationFlag>
               <jsdl:Source>
                       <jsdl:URI>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/public/AATest/MiraAutoSelect-46-66/miraIntersect.txt</jsdl:URI>
               </jsdl:Source>
            </jsdl:DataStaging>
……
<sweep:Sweep>
               <sweep:Assignment>
               <sweep:DocumentNode>
               <sweep:NamespaceBinding ns="http://schemas.ggf.org/jsdl/2006/07/jsdl-hpcpa"
prefix="jsdl-hpcpa" />
                     <sweep:Match>
                        /*//jsdl-hpcpa:Argument[1]
                     </sweep:Match>
                 </sweep:DocumentNode>
                 <sweepfunc:Values>
                     <sweepfunc:Value>--project=Kukri19</sweepfunc:Value>
                     <sweepfunc:Value>--project=Kukri20</sweepfunc:Value>
                     <sweepfunc:Value>--project=Kukri21</sweepfunc:Value>
                 </sweepfunc:Values>
        </sweep:Assignment>
               <sweep:Assignment>
                <sweep:DocumentNode>
               <sweep:NamespaceBinding ns="http://schemas.ggf.org/jsdl/2005/11/jsdl"
prefix="jsdl" />
                     <sweep:Match>
                        /*//jsdl:DataStaging[2]/jsdl:Source/jsdl:URI
                     </sweep:Match>
                 </sweep:DocumentNode>
                 <sweepfunc:Values>
                     <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri19_in.454.fasta</sweepfunc:Value>
                     <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri20_in.454.fasta</sweepfunc:Value>
                     <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri21_in.454.fasta</sweepfunc:Value>
```

```
                </sweepfunc:Values>
        </sweep:Assignment>
……
```

Figure 6-16 A portion of a sample parameter sweep job in JSDL

This JSDL file will actually generate three jobs. Each is a Mira job with three input files. The first section is a normal JSDL file, used as a template. The second section is swept parameters. In this case, it replaces the first argument and three input files; each of them will be replaced by three different values, hence results in three final JSDL files, therefore three jobs. Using Parameter Sweep is an easy way to submit multiple jobs.

In a real user scenario, the gateway was used to submit thousands of jobs running Mira [191] for a bioinformatics project. The goal of the project is to build a Wheat transcriptome, which is the sequence data of an organism's mRNA population. mRNA is transcribed from the gene sequence on the chromosome and is later translated into the amino acid sequence that forms the protein product. The individual reads from the sequencers in this instance, called Expressed Sequence Tags (ESTs), are short, and multiple ESTs are assembled to produce the final mRNA transcript sequence. Consequently, MIRA takes short DNA sequencing reads and assembles them into longer sequences that represent the actual transcripts.

The Wheat genome is a hexaploid - there are six copies of each chromosome - so there is a huge pool of EST sequences produced. MIRA could technically take all the EST reads into one job and spit out all the individual transcripts, the problem is that it is memory intensive, in the order of around 1TB would be required in this case.

So the users needed to use other programs to cluster the ESTs into separate MIRA jobs. These programs, such as Velvet [192], are also assemblers but are built for speed and are not as accurate as MIRA. So each MIRA job is a subset of ESTs that are broadly similar in sequence, MIRA then goes to work and produce an accurate transcript. In total, they have about 30 thousand DNA clusters for MIRA to work on. Therefore, a total of 30 thousand jobs are submitted to the Grisu2 portal and then the portal distributes them to several underlying Globus gatekeepers that have the Mira application installed on the associated HPC resources.

The final transcriptome, the results of these 30 thousand jobs, can be used to build an EST library that is a platform for an enormous range of wet and bioinformatic experiments. One possible form is to build a BLAST database so it can be used by other BLAST jobs.

The user uses OpenOffice Calc (SpreadSheet) for job submission. As OpenOffice Calc supports Python extensions, the user has written a python script and attached them to Calc's spreadsheet as a function. The spreadsheet keeps track of all jobs with one cell storing the parameter of one job, and places a flag on those that have been submitted, such that it knows which ones have been submitted and which ones are not. The python script assembles jobs into JSDL format and sends them to Grisu2's RESTful interface. To submit a new job, the user only needs to call the associated function on the related cells. The MIRA application has been installed on five regional compute resources, so Grisu2 can distribute all jobs to all five compute resources to share the load, and that takes much less time than executing all jobs in one supercomputer. Moreover, the local compute

resource where the user has access to is usually busy; if submitting to the local resource, jobs will often wait in the queue for some time before being processed.

6.5 Related work

There are a number of grid portals developed over the past few years, such as the CMS web portal intending to be a gateway to CMS job submission and management [115], a computational portal for processing astrophysical and high energy physics data based on GridSphere and Gridbus broker [193], the VIVE portal for 3D medical image processing and visualization [194], the CSEO portal for multi-scale modeling of complex reacting systems in science and engineering [195]. There are other generic grid portals. The Pegasus portal is a tool to generate abstract workflows based on some metadata description, and submit these workflows to the Grid using Pegasus [196]. The GENIUS portal, with job submission and data management functions, allows easy access to grid resources provided by INFN [197]. There are other user community portals aiming to support a particular discipline. For example, the Earth System Grid [111] is designed to enable community access to climate simulation results in DOE and NSF supercomputers, with rich features including data query, data analysis, data transfer. The Southern California Earthquake Center (SCEC) portal is an entrance to its digital library that contains results of large-scale earthquake simulations executed in TeraGrid and provides tools for querying and visualization [198].

These portals, however, mostly rely on a specific environment, and are developed to meet the requirements of a particular research group or discipline area. GridSphere

[199], based on the Portlet technology, aims to be a generic framework to build grid portals, but it doesn't provide a mechanism to build other interfaces, such as web service interface and RESTful interface that are needed in this project. Moreover, Portlets are not easy to develop compared to web 2.0 technologies based on AJAX and JavaScript or simple templates as provided by Grisu2. The Open Grid Computing Environment (OGCE) project [200] has developed a REST interface for job submission, a number of reusable Portlet-based components and Java programming abstractions for the grid. It includes a portal, a number of web services and client libraries, to enable access to all sorts of grid services, such as information, data and execution services. However, the RESTful interface provided by OGCE doesn't support JSDL, and OGCE doesn't support BES either. Andreozzi and Marzolla have proposed a REST mapping of the BES interface [201]; however, their work is still at the conceptual stage, and they have not developed any prototype system for that.

A lot of effort has been put into the research and design of grid gateways, grid resource brokers and meta-schedulers. They normally provide API-level interfaces, such as a web service interface or remote APIs, but only some of them provide a BES interface. They don't provide a simple RESTful interface, or a GUI for end-users, such as a comprehensive web interface like Grisu2 has, let alone the ability of customization for different application and the support of Shibboleth. On the other hand, it is easy to add a component to Grisu2 so that Grisu2 can distribute jobs to these systems; Grisu2 can already distribute jobs to Globus, and one of our future works may be to support other middleware like GridWay. Globus is a grid gateway with WSRF-based web service interface for local schedulers. It provides uniform authentication mechanism and enables

remote access to local schedulers. GridWay [52] is a popular framework based on Globus and allows easy and efficient execution of jobs on dynamic grid environment. It aims to provide a grid resource broker, which Globus lacks, and offers its own API and a standard DRMAA API. CREAM [176, 180] is another job execution and management system that distributes jobs to remote and heterogeneous HPC systems, with a proprietary interface and a BES interface. Gridbus [202] has a grid resource broker that aims to provide access to the most efficient and economical grid resources. Nimrod/G [190] aggregates various grid resources on the backend and presents a single entry to these resources to users, with focus on resource management and scheduling for parameter sweep applications.

6.6 Conclusion

In summary, I have evaluated existing solutions for bringing to users an easy-to-use and customized interface to access a grid of distributed HPC resources. Since existing solutions are neither providing all required features, nor easy to customize, I developed Grisu2, which adopts open standards and is very easy to use and builds on existing systems. It provides a grid resource broker with a built-in meta-scheduler and various interfaces. The web portal attracts users for its simplicity and efficiency in job submission, so they can spend more time on their research work. It has been Shibbolized, giving users a simple way to authenticate in order to access the system. The portal also features a template system that is being used by administrators to define templates for various applications. The BES web service interface and the RESTful interface are good

resources for developers and advanced users who want to hook a third-party application into the grid. These different interfaces improve interaction and usability. Additionally, the system has been integrated with our national file system via GridFTP, so that data produced during job execution can be well managed. The system is now being used by many users in a variety of disciplines and with various use cases from single job submission via web interface to parameter sweep jobs via scripts.

Chapter Seven: Conclusion and Future Work

7.1 Conclusion

This thesis began with an introduction to e-Science and the grid. Grid technology is the backbone of e-Science, providing fundamental infrastructure that supports research activities via grid services. Generally speaking, there are both data grids and computational grids, focusing on sharing storage resources and compute resources respectively. Many grid middleware systems have been developed over the past several years. This thesis mainly focuses on data grid middleware, computational grid middleware and data transfer services. Authentication mechanisms, especially web-based authentication methods, have also been studied. The literature review gave a comprehensive description of the above, with many existing systems as examples. A few major grid projects in the world were examined, from their goals to their implementations.

Although these existing grid projects are successful in development and operating in production, it is not always easy to migrate a system to a different environment; if a use case or user model is even slightly changed, migration can become quite difficult. Other generic middleware systems, such as Globus Toolkit, aim at middleware level and lack a user-friendly interface, thus they are not easy for end-users to use. A new grid infrastructure differentiates from existing systems in many ways, such as how data is processed and manipulated according to their use cases. If the whole infrastructure is divided into layers as in Figure 3-1, customization can happen in two levels: at the

middleware level, the system should be easily tailored, e.g. using some configuration to change its behaviour, without needing to modify the source code; at the user interface level, users should be offered a standard and easy-to-use interface that they can easily get familiar with, without a long learning curve. These are the main motivations of my work in this thesis, and the strategy in accomplishing these goals is to take an adaptive middleware, and build a standard and user-friendly interface on top of that.

My contributions in this project include the research and evaluation of existing data grid middleware and compute grid middleware, which led to the development of three new independent components:

- *Davis*, the WebDAV and web interface for iRODS;
- *Griffin*, a generic GridFTP interface; and
- *Grisu2*, a universal grid job submission gateway.

Each of them can work as a standalone system; but they can also work together in a grid environment, especially a national grid infrastructure like the Australian National Grid, which has all three components deployed and running in production.

Davis offers a WebDAV-compliant interface so that users can use any standard WebDAV clients to connect to a data grid that is built on the iRODS middleware. iRODS is a data grid system that provides a virtual view of geographically distributed storage resources, with several interfaces provided by default. I surveyed almost all available clients of iRODS and its ancestor, SRB, as in section 4.3, which compares the authentication methods, supported operating systems, user interfaces and the development language. This survey shows that none of these clients is generic or easy to use. This observation revealed a gap between a well-designed middleware system and

users being able to use it easily, and that was the motivation to implement a user-friendly and easy-to-use interface for end-users. For example, Mac Finder and Windows Explorer allow users to mount a WebDAV resource as a local drive, which integrates the data grid with users' desktop seamlessly so users can easily drag-and-drop files between their desktop and the data grid. As WebDAV is an extension of the HTTP protocol, Davis is able to run on most commodity HTTP servers, such as Apache. Users can benefit from the features they provide with no development cost. For example, Apache can be configured to use SSL to protect data transfer if data security is essential in the user environment. Davis also features a comprehensive web interface that enables permission and metadata management as well as sharing from a web browser, and eliminates the need to set up a client application. A RESTful interface is also provided to programmers to write simple scripts to interact with iRODS via Davis, which hides the complexity of setting up iRODS clients.

The development of Griffin, a generic GridFTP interface, is in fact a by-product of a study on data transfer services, for which the motivation was to enable data transfer in an iRODS-based data grid, as iRODS does not have a standard interface that can interact with other data services. After studying several data transfer services, my idea was to turn an existing data source that exposes a proprietary interface into a standard GridFTP-compliant data service, but not to duplicate current work and develop another data transfer service, in order to obtain a simple and clean system architecture and benefit from existing data transfer services. This is achieved by deploying Griffin as the entry point, which translates GridFTP protocol to any protocol used by the underlying data source. My evaluation shows that the overhead generated by Griffin is acceptably low,

and in many cases is close to zero. Being compliant with the GridFTP protocol, Griffin inherits useful features of GridFTP, such as the ability to transfer data in parallel streams, transfer partial data, and transfer data with or without encryption. With an extensible structure, new plugins can be developed to support more data sources; as a specific example, a developer from another research group has developed a plugin for MongoDB. Further on, a special plugin is being developed to support striping transfer, as in the ongoing work of Griffin Conductor.

Grisu2 is a job submission gateway aiming to simplify compute job submission and offer an easy-to-use job submission platform to users. Most traditional job submission clients require installation of client applications, and understanding of various IT technologies, such as how to use a X.509 certificate, which is far too complex for users with minimal IT knowledge. Therefore, the gateway I developed offers a web-based interface so users can use a web browser to submit compute jobs, without knowing any details of how the grid is set up, and how many compute resources they can use. This automation is achieved via a built-in resource broker that is able to work out the best compute resource to execute the job and the ability to distribute jobs to a remote grid resource gatekeeper, such as Globus GRAM. Moreover, the web interface is protected by Shibboleth, so users can use their home organization's accounts to log in and use the gateway, without the need to use a grid certificate or register extra accounts. It also features a RESTful interface and a web-service interface for advanced users to write scripts and applications to submit massive jobs.

In a typical setup such as the Australian National Grid, these components are all used to deliver a good user experience. At the middleware layer, iRODS is chosen as the

data grid middleware due to its flexibility in customization; at the user interface layer are Davis, offering a quick and easy way for users to access data from anywhere, and Griffin, targeting large volume data movements and data exchange with grid systems. On the computational grid side, Globus Toolkit is used as the gatekeeper of HPC clusters, while Grisu2 is the job submission gateway for users to easily get access to all available HPC clusters via Globus Toolkit. The interaction between the computational grid part and the data grid part is via Griffin, through which data travels seamlessly. Job results, on the other hand, can be viewed and shared via the web interface, Davis. My work on all these components makes it easier for users to use both data grid and compute grid. To date, the Australia National Grid has attracted more than 2000 users. What's more, Davis and Griffin are also being used in production in a number of projects in the US, Europe, New Zealand and Japan. Having a generic design, they are installed and deployed in a different environment only with some configuration work. Programmatically, these systems use standard APIs for iRODS and Globus GridFTP, so they will work with future versions of those as long as they are backward compatible, which they usually are. Moreover, all these systems are open source projects hosted in Google Code. They are well designed and comments are inserted in source code for other people to easily understand. In fact, developers from some user groups have already contributed valuable additions to the original software I developed. These developers are the good candidates to take over the projects if I no longer work on them.

7.2 Future work

Having delivered generic components to the community, I learnt from this project that simplicity is in great demand in modern computer systems. Being exposed to a great number of resources, users expect a unified and easy-to-use interface rather than having to deal with each resource separately. This means users should see the same data no matter what system they are accessing, or what device they are accessing from. The future of my research is driven by the lesson learnt in this project, and there are several things that can be done to further improve user experiences.

For Davis, more functions can be added to the web interface. When sharing data, users may want to share with other Davis users, or just a random person. Sharing with other Davis users is easy because that can be handled by the data grid middleware, iRODS; but sharing with a random person, such as a colleague from overseas who does not hold a credential from a local authority, needs a way to allow temporary access to the user's data with some kind of access token. Managing replicas is required when there are multiple storage resources, as users may want to keep backups of their data in a dedicated or arbitrary location. Integration with other web portals will be useful to some third-party developers who want to store their system's data in Davis. Currently, a RESTful API is available but more functions can be added if required. Web page level integration may also be interesting. Web widgets written in JavaScript can be provided so other web systems can easily embed Davis in their web pages.

For Griffin, UDT [33] support will be investigated, as it is supported by GridFTP server and client in Globus Toolkit 5. Several checksum mechanisms will be implemented to verify copied files. In addition, multiple streams from Griffin to the data source will be investigated in the case where the data source supports multiple streams.

This will definitely add complexity to the whole system, and will possibly not lead to significant performance improvement. The current implementation of Griffin Conductor is half finished, and only the upload function is working now. More work needs to be put in to finish the download function. When the Griffin Conductor is completed, it will provide striping download to users.

The job submission gateway requires enhancement on data staging, especially when staging in or out large datasets, because the current implementation cannot restart a failed transfer from where it fails; when failed, the gateway can only restart the whole transfer from the beginning, which may waste bandwidth if it fails right before the end of transferring a huge file. In this case, it may need to make use of a data transfer service to manage the transfer so as to make data transfer more efficient and controllable. More work will be done on the meta-scheduler, so that more sophisticated scheduling policies can be defined based upon user requirements. Job submission will be extended to support other grid middleware, such as gLite and GridWay, or even the cloud, by implementing a mechanism to dynamically start or shut down VMs in the cloud. As a further research topic, it would be interesting to find out an easy way to start and stop a whole cluster in the cloud, as well as dynamically add or remove nodes from this cluster. There will be definitely a mixture of grid and cloud in my future research, and that might be the future of e-Science too.

REFERENCES

[1] (2009). LHC Homepage. [Online]. Available: http://lhc.web.cern.ch/lhc/.

[2] A. J. G. Hey and A. E. Trefethen, "The UK e-Science Core Programme and the Grid," *Future Generation Computer Systems,* vol. 18, pp. 1017-1031, 2002.

[3] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*: Morgan Kaufmann, 1999.

[4] I. Foster, et al., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *The International Journal of Supercomputer Applications,* vol. 15, 2001.

[5] I. Foster and C. Kesselman, "Computational Grids," in *Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure"*: Morgan-Kaufman, 1999.

[6] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today,* vol. 55, pp. 42-47, 2002.

[7] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The International Journal of Supercomputer Applications,* vol. 12, pp. 115-128, 1997.

[8] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," in *IFIP Int Conf on Network and Parallel Computing*, 2006.

[9] EGEE. (2010). gLite. [Online]. Available: http://glite.web.cern.ch/glite/.

[10] D. E. A. Streit, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder, "Unicore — From project results to production grids," *Advances in Parallel Computing,* vol. 14, 2005.

[11] M. Litzkow, et al., "Condor - A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, 1998.

[12] OGF. (2011). The Open Grid Forum. [Online]. Available: http://www.ogf.org/.

[13] W. Allcock, "GridFTP: Protocol Extensions to FTP for the Grid," *Global Grid Forum GFD-R.0201*, 2003.

[14] Computation Institute, University of Chicago, Argonne. (2010). Globus Online. [Online]. Available: http://www.globusonline.org/.

[15] A. Chervenak, et al., "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, vol. 23, pp. 187-200, 2001.

[16] E. Laure, et al., "Middleware for the next generation Grid infrastructure," in *Computing in High Energy Physics and Nuclear Physics 2004*, Interlaken, Switzerland, 2004.

[17] C. Baru, et al., "The SDSC Storage Resource Broker," in *Proc of the Centre for Advanced Studies on Collaborative Research (CASCON)*, 1998.

[18] A. Rajasekar, et al., "A Prototype Rule-based Distributed Data Management System," in *workshop on "Next Generation Distributed Data Management" of High Performance Parallel and Distributed Computing (HPDC)*, Paris, France, 2006.

[19] M. Ernst, et al., "dCache, a distributed data storage caching system," in *Computing in High Energy and nuclear Physics (CHEP2001)*, Beijing, 2001.

[20] M. Lamanna, "The LHC computing grid project at CERN," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment,* vol. 534, pp. 1-6, 2004.

[21] P. H. Beckman, "Building the TeraGrid," *Phil. Trans. R. Soc. A,* vol. 363, pp. 1715-1728, 2005.

[22] V. Welch, et al., "Security for Grid Services," in *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, 2003.

[23] R. L. C. Morgan, Scott; Carmody, Steven; Hoehn, Walter; Klingenstein, Ken, "Federated Security: The Shibboleth Approach," *EDUCAUSE Quarterly*, vol. 27, pp. 12-17, 2004.

[24] CCLRC. (2006). e-Science Centre Annual Report 2005-2006. [Online]. Available: http://www.e-science.stfc.ac.uk/publications/pdf/annual-report-05-061919.pdf.

[25] Science and Technology Facilities Council. (2011). Guide to e-Science. [Online]. Available: http://www.e-science.stfc.ac.uk/guide/index.html.

[26] W. Gentzsch, "Sun Grid Engine: towards creating a compute power grid," in *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, 2001, pp. 35-36.

[27] Altair Engineering Inc. (2011). PBS Professional. [Online]. Available: http://www.pbsworks.com/Product.aspx?id=1.

[28] Platform Computing™ Corporation. (2011). Platform LSF. [Online]. Available: http://www.platform.com/workload-management/high-performance-computing.

[29] I. Foster, "What is the Grid? A Three Point Checklist," *GRIDToday,* 2002.

[30] R. Buyya and S. Venugopal, "A Gentle Introduction to Grid Computing and Technologies," *CSI Communications*, vol. 29, pp. 9-19, 2005.

[31] D. P. Anderson, et al., "SETI@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, pp. 56-61, 2002.

[32] M. Feller, et al., "GT4 GRAM: A Functionality and Performance Study," in *Teragrid 2007*, 2007.

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

[34] Globus Alliance. GT 4.0 WS GRAM: Job Description Schema Doc. [Online]. Available: http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html.

[35] A. Anjomshoaa, et al., "Job Submission Description Language (JSDL) Specification, Version 1.0 (GFD-R.136)," *Open Grid Forum*, 2008.

[36] Z. Xuehai and J. M. Schopf, "Performance analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2," in *Performance, Computing, and Communications, 2004 IEEE International Conference on*, 2004, pp. 843-849.

[37] Globus. GT 4.0 WS MDS WebMDS. [Online]. Available: http://www.globus.org/toolkit/docs/4.0/info/webmds/.

[38] W3C, "XSL Transformations (XSLT) Version 1.0," 1999. Available: http://www.w3.org/TR/xslt.

[39] V. Welch, et al., "Security for Grid services," in *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, 2003, pp. 48-57.

[40] J. Novotny, et al., "An online credential repository for the Grid: MyProxy," in *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, 2001, pp. 104-111.

[41] E. Laure, et al., "Programming the Grid with gLite," *Computational Methods in Science and Technology*, vol. 12, 2006.

[42] EGEE. (2010). Enabling Grids for E-science. [Online]. Available: http://public.eu-egee.org/.

[43] F. Fernández Rivera, et al., "VOMS, an Authorization System for Virtual Organizations," in *Grid Computing*, vol. 2970: Springer Berlin / Heidelberg, 2004, pp. 33-40.

[44] G. Carcassi, et al., "A Scalable Grid User Management System for Large Virtual Organization," in *Computing in High Energy Physics and Nuclear Physics 2004*, Interlaken, Switzerland, 2004.

[45] CERN. (2010). Berkeley Database Information Index V5. [Online]. Available: https://twiki.cern.ch/twiki//bin/view/EGEE/BDII.

[46] P. Andreetto and et al., "The gLite workload management system," *Journal of Physics: Conference Series*, vol. 119, p. 062007, 2008.

[47] C. Aiftimiei, et al., "Design and implementation of the gLite CREAM job management service," *Future Generation Computer Systems*, vol. 26, pp. 654-667, 2010.

[48] A Roy on behalf of the OSG consortium, "Building and testing a production quality grid software distribution for the Open Science Grid," *Journal of Physics: Conference Series*, vol. 180, p. 012052, 2009.

[49] NSF and DOE. (2010). Open Science Grid. [Online]. Available: http://www.opensciencegrid.org/.

[50] D. Thain, et al., "Distributed computing in practice: the Condor experience.," *Concurrency - Practice and Experience*, vol. 17, pp. 323-356, 2005.

[51] J. Frey, et al., "Condor-G: A Computation Management Agent for Multi-Institutional Grids," in *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, San Francisco, California, 2001, pp. 7-9.

[52] E. Huedo, et al., "The GridWay Framework for Adaptive Scheduling and Execution on Grids," *Scalable Computing - Practice and Experience*, vol. 6, pp. 1-8, 2005.

[53] D. Abramson, et al., "High performance parametric modeling with Nimrod/G: killer application for the global grid?," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, 2000, pp. 520-528.

[54] P. J. Braam and Others, "The Lustre storage architecture," *White Paper, Cluster File Systems, Inc., Oct*, vol. 23, 2003.

[55] J. Walgenbach, et al., "Enabling Lustre WAN for Production Use on the TeraGrid: A Lightweight UID Mapping Scheme," in *Proceedings of the 2010 TeraGrid Conference*, 2010.

[56] S. C. Simms, et al., "Wide Area Filesystem Performance using Lustre on the TeraGrid," in *Proceedings of the TeraGrid 2007 Conference*, 2007.

[57] IBM. (2011). General Parallel File System. [Online]. Available: http://www-03.ibm.com/systems/software/gpfs/.

[58] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, 2002, pp. 231-244.

[59] P. Andrews, et al., "Exploring the hyper-grid idea with grand challenge applications: the DEISA-TeraGrid interoperability demonstration," in *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, 2006, pp. 43-52.

[60] S. Weil, et al., "Ceph: A Scalable, High-Performance Distributed File System," in *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*, 2006.

[61] F. Hupfeld, et al., "XtreemFS – a case for object-based storage in Grid data management," in *Proceedings of 33th International Conference on Very Large Data Bases (VLDB) Workshops*, 2007.

[62] P. Fuhrmann and V. Gülzow, "dCache, Storage System for the Future," in *Euro-Par 2006 Parallel Processing*, vol. 4128, W. Nagel, et al., Eds.: Springer Berlin / Heidelberg, 2006, pp. 1106-1113.

[63] P. Fuhrmann, "A Perfectly Normal Namespace for the DESY Open Storage Manager," in *Computing in High-energy Physics (CHEP 97)*, Berlin, Germany, 1997.

[64] A. Shoshani, et al., "Storage Resource Managers: Middleware Components for Grid Storage," in *proc of the 19th IEEE Symposium on Mass Storage Systems, MSS'02*, 2002.

[65] DICE. (2009). SRB – The DICE Storage Resource Broker. [Online]. Available: http://www.sdsc.edu/srb/index.php/Main_Page.

[66] DICE. (2010). IRODS:Data Grids, Digital Libraries, Persistent Archives, and Real-time Data Systems. [Online]. Available: http://www.irods.org/.

[67] R. W. Moore, "Managing Large Distributed Data Sets Using the Storage Resource Broker," *ITEA Journal of Test and Evaluation*, 2007.

[68] S. M. Sadjadi, "A Survey of Adaptive Middleware," Computer Science and Engineering, Michigan State University 2003.

[69] P. Kunszt, et al., "The gLite File Transfer Service," in *Proceedings of the 5th International Conference on Computing In High Energy and Nuclear Physics*, Mumbai, India, 2006.

[70] R. Egeland, et al., "Data transfer infrastructure for CMS data taking," in *Advanced Computing and Analysis Techniques in Physics Research (ACAT08)*, Erice, Italy, 2008.

[71] T. Kosar and M. Balmana, "A new paradigm: Data-aware scheduling in grid computing " *Future Generation Computer Systems*, vol. 25, pp. 406-413, 2009.

[72] I. Foster, "Globus Online: Accelerating and Democratizing Science through Cloud-Based Services," *Internet Computing, IEEE*, vol. 15, pp. 70-73, 2011.

[73] B. Allen, et al., "Globus Online: Radical Simplification of Data Movement via SasS," ANL/MCS-P1905-0611, June 2011. Available: http://www.mcs.anl.gov/publications/paper_detail.php?id=1633.

[74] GO development team. (2011). User Stories of Globus Online. [Online]. Available: https://http://www.globusonline.org/stories/.

[75] Internet2. (2007). Shibboleth Project - Internet 2 Middleware. [Online]. Available: http://shibboleth.internet2.edu/.

[76] SAML working group. (2009). SAML Specifications. [Online]. Available: http://saml.xml.org/saml-specifications.

[77] Australian Access Federation. [Online]. Available: http://www.aaf.edu.au/.

[78] InCommon. [Online]. Available: http://www.incommonfederation.org/.

[79] T. Spreng, et al., "Implementation of an Authentication and Authorization Architecture for Mobile Internet Users," *Exchange Organizational Behavior Teaching Journal*, vol. 7, 2004.

[80] R. Sinnott, et al., "Shibboleth-based Access to and Usage of Grid Resources," in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, 2006, pp. 136-143.

[81] D. Spence, et al., "Shibgrid: Shibboleth access for the uk national grid service," in *e-science06*, 2006.

[82] SWITCH. (2008). Short Lived Credential Service (SLCS). [Online]. Available: http://www.switch.ch/grid/slcs/.

[83] T. Barton, et al., "Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy," in *5th Annual PKI R&D Workshop*, 2006.

[84] (2009). GridShib. [Online]. Available: http://gridshib.globus.org/.

[85] (2011). CILogon. [Online]. Available: http://www.cilogon.org/.

[86] E. E. Hammer-Lahav. (2010). The OAuth 1.0 Protocol. [Online]. Available: http://tools.ietf.org/html/rfc5849.

[87] N. Wilkins-Diehr, et al., "TeraGrid Science Gateways and Their Impact on Science," *Computer*, vol. 41, pp. 32-41, 2008.

[88] M. Altunay, et al., "A Science Driven Production Cyberinfrastructure—the Open Science Grid," *Journal of grid computing*, vol. 9, pp. 201-218, 2011.

[89] J. Shiers, "The Worldwide LHC Computing Grid (worldwide LCG)," *Computer Physics Communications*, vol. 177, pp. 219-223, 2007.

[90] F. Gagliardi, et al., "Building an infrastructure for scientific Grid computing: status and goals of the EGEE project," *Phil Trans R Soc A*, vol. 363, pp. 1729-1742, 2005.

[91] N. Geddes, "The National Grid Service of the UK," in *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, 2006, p. 94.

[92] X. Yang, et al., "A Web Portal for the National Grid Service," in *UK e-Science AHM 2005*, 2005.

[93] A. Michael, et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50-58.

[94] K. Keahey, et al., "Virtual Workspaces in the Grid," in *Europar 2005*, Lisbon, Portugal, 2005.

[95] J. Brodkin. (2011). $1,279-per-hour, 30,000-core cluster built on Amazon EC2 cloud. [Online]. Available: http://arstechnica.com/business/news/2011/09/30000-core-cluster-built-on-amazon-ec2-cloud.ars.

[96] D. Baran. (2008). Cloud Computing Basics. [Online]. Available: http://www.webguild.org/20080729/cloud-computing-basics.

[97] B. P. Rimal, et al., "A Taxonomy and Survey of Cloud Computing Systems," in *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, 2009, pp. 44-51.

[98] S. L. Garfinkel, "An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS," *Applied Sciences,* 2007.

[99] K. Keahey and T. Freeman, "Contextualization: Providing One-Click Virtual Clusters," in *2008 Fourth IEEE International Conference on eScience*, 2008.

[100] D. Nurmi, et al., "The Eucalyptus Open-Source Cloud-Computing System," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 124-131.

[101] Google. (2011). Google App Engine. [Online]. Available: http://code.google.com/appengine/.

[102] S. Zhang, et al., "Davis: A Generic Interface for iRODS and SRB," in *the proceeding of the 10th ACM/IEEE International Conference on Grid Computing*, Banff, Canada, 2009.

[103] S. Zhang, et al., "Connecting arbitrary data resources to the grid," in *the proceeding of the ACM/IEEE International Conference on Grid Computing*, Brussels, Belgium, 2010.

[104] W. Leinberger and V. Kumar, "Information power grid: The new frontier in parallel computing?," *Concurrency, IEEE*, vol. 7, pp. 75-84, 1999.

[105] I. Foster, et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," in *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

[106] F. Berman, et al., "The GrADS project: Software support for high-level grid application development," *International Journal of High Performance Computing Applications*, vol. 15, pp. 327-344, 2001.

[107] Myricom. (2009). Myrinet Overview. [Online]. Available: http://www.myricom.com/myrinet/overview/.

[108] InfiniBand Trade Associasion. (2010). InfiniBand. [Online]. Available: http://www.infinibandta.org.

[109] SGI. Data Migration Facility. [Online]. Available: http://www.sgi.com/products/storage/software/dmf.html.

[110] J. Knobloch and L. Robertson. (2005). LHC computing Grid technical design report. [Online]. Available: http://lcg.web.cern.ch/LCG/tdr/.

[111] D. Bernholdt, et al., "The Earth System Grid: Supporting the Next Generation of Climate Modeling Research," *Proceedings of the IEEE*, vol. 93, p. 485, 2005.

[112] W.-D. Nancy, "TeraGrid Science Gateways and Their Impact on Science," *Computer*, vol. 41, pp. 32-41, 2008.

[113] Altair Engineering. (2011). PBS Works. [Online]. Available: http://www.pbsworks.com/.

[114] Globus. (2010). Globus Toolkit. [Online]. Available: http://www.globus.org/toolkit/.

[115] N. David Braun and Norbert, "A Web portal for CMS Grid job submission and management," *Journal of Physics: Conference Series,* vol. 219, p. 072012, 2009.

[116] B. Plale, et al., "Computational Science – ICCS 2005 Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD," *Lecture notes in computer science,* vol. 3515, p. 624, 2005.

[117] J. Novotny, et al., "GridSphere: a portal framework for building collaborations," *Concurrency and Computation: Practice & Experience - Middleware for Grid Computing,* vol. 16, 2004.

[118] C. Altinay, et al., "High-Performance Scientific Computing for the Masses: Developing Secure Grid Portals for Scientific Workflows," in *Proceedings of the Sixth IEEE International Conference on e–Science*, Brisbane, Queensland, Australia, 2010.

[119] I. Foster, et al., "OGSA Basic Execution Service (GFD-R.108)," *Open Grid Forum,* 2008.

[120] R. Moor, et al., "Towards the interoperability of web, database, and mass storage technologies for petabyte databases," in *In Proceedings of the Fifth NASA GSFC Conference on Mass Storage Systems and Technologies (College Park)*, 1996.

[121] H. Mark, et al., "Curation and Preservation of Research Data in an iRODS Data Grid," in *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, 2007.

[122] T. Mortimer-Jones. (2008). SRB and iRODS work at the STFC and National Grid Service. [Online]. Available: http://www.nesc.ac.uk/action/esi/download.cfm?index=3731.

[123] J. Whitehead. (2008). WebDAV Resources. [Online]. Available: http://www.webdav.org/.

[124] C. Baru, et al., "The SDSC Storage Resource Broker," in *In Proceedings of CASCON*, 1998.

[125] A. Rajasekar, et al., "A Prototype Rule-based Distributed Data Management System," in *HPDC*, Paris, France, 2006.

[126] The_Sakai_Foundation. (2008). Sakai Project. [Online]. Available: http://www.sakaiproject.org/portal.

[127] A. Young. (2008). Guanxi Project. [Online]. Available: http://www.guanxi.uhi.ac.uk/index.php/Guanxi.

[128] R. Groeper, et al., "An Architecture for Authorization in Grids using Shibboleth and VOMS," in *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, 2007.

[129] SDSC. (2008). iRODS website. [Online]. Available: http://www.irods.org.

[130] SDSC. (2008). SRB Website. [Online]. Available: http://www.sdsc.edu/srb/index.php.

[131] G. v. Laszewski. (2006). Java CoG Kit SRB Integration. [Online]. Available: http://wiki.cogkit.org/index.php/Java_CoG_Kit_SRB_Integration.

[132] ARCS and JCU. (2008). Hermes. [Online]. Available: http://projects.arcs.org.au/trac/commons-vfs-grid/.

[133] (2009). ARCHER project. [Online]. Available: http://archer.edu.au.

[134] Apache. (2008). Commons Virtual File System. [Online]. Available: http://commons.apache.org/vfs/.

[135] CCIN2P3. (2008). JUX (Java Universal eXplorer). [Online]. Available: http://cc.in2p3.fr/docenligne/821.

[136] IN2P3_Computer_Center. (2008). JSAGA. [Online]. Available: http://grid.in2p3.fr/jsaga/.

[137] S. D. Olabarriaga, et al., "Virtual Lab for fMRI: Bridging the Usability Gap," in *Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, 2006.

[138] T. Glatard, et al., "Workflow Integration in VL-e Medical," in *21st IEEE International Symposium on Computer-Based Medical Systems*, 2008.

[139] M. J. Wyatt, et al., "YourSRB: A cross platform interface for SRB and Digital Libraries," in *In Proc. Fifth Australasian Symposium on Grid Computing and e-Research*, Ballarat, Australia, 2007.

[140] JCU. (2008). SRB Plone Product. [Online]. Available: https://eresearch.jcu.edu.au/wiki/wiki/SrbPloneProduct.

[141] Globus. (2008). GT 4.0 GridFTP: Storage Resource Broker (SRB). [Online]. Available: http://www.globus.org/toolkit/docs/4.0/data/gridftp/GridFTP_SRB.html.

[142] Microsoft. (2008). CIFS Protocol. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa302240.aspx.

[143] The_Internet_Society. (1999). HTTP Extensions for Distributed Authoring -- WEBDAV. [Online]. Available: http://www.ietf.org/rfc/rfc2518.txt.

[144] (2009). HANDLE.NET - The Handle System. [Online]. Available: http://www.handle.net.

[145] R. Pandya, et al., "Finding and using data in educational digital libraries," in *Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*, 2003.

[146] A. Woolf, et al., "A Web Service Model for Climate Data Access on the Grid," *International Journal of High Performance Computing Applications*, vol. 17, pp. 281-295, 2003.

[147] Google. (2008). Using Image Overlays and 3D Models [Online]. Available: http://earth.google.com/userguide/v4/ug_imageoverlays.html.

[148] M. Litzkow, et al., "Condor - A Hunter of Idle Workstations," in *the 8th Int Conf of Distributed Computing Systems*, 1988.

[149] Globus. (2009). Reliable File Transfer (RFT) Service: User's Guide. [Online]. Available: http://www.globus.org/toolkit/docs/4.0/data/rft/user-index.html.

[150] C. Hines, et al. (2009). Developing Parallel Applications. [Online]. Available: http://www.ivec.org/pages/tiki-download_file.php?fileId=194.

[151] IN2P3. (2009). JSAGA. [Online]. Available: http://grid.in2p3.fr/jsaga/.

[152] Apache. (2008). Apache Commons Virtual File System. [Online]. Available: http://commons.apache.org/vfs/.

[153] P. Calvat. (2010). JUX (Java Universal eXplorer). [Online]. Available: http://cc.in2p3.fr/docenligne/821.

[154] J. Ou, et al., "International Workshop on Web-Based Internet Computing for Science and Engineering (ICSE 2006) - A Web-Based System for Adaptive Data Transfer in Grid," *LECTURE NOTES IN COMPUTER SCIENCE,* vol. 3842, pp. 803-810, 2006.

[155] W. Allcock, et al., "The Globus Striped GridFTP Framework and Server," in *Proc of Super Computing 2005 (SC05),* 2005.

[156] R. Kettimuthu, et al., "Globus Data Storage Interface (DSI) - Enabling Easy Access to Grid Datasets," in *First DIALOGUE Workshop: Applications-Driven Issues in Data Grid,* 2005.

[157] W. Allcock, et al., "The Globus eXtensible Input/Output System (XIO): A protocol independent IO system for the Grid," in *Proc of the Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models held in conjunction with Int Parallel and Distributed Processing Symposium (IPDPS 2005),* 2005.

[158] I. Mandrichenko, "GridFTP Protocol Improvements," *Global Grid Forum GFD-21,* 2003.

[159] I. Mandrichenko, et al., "GridFTP v2 Protocol Description," *Global Grid Forum GFD-47,* 2005.

[160] J. Postel and J. Reynolds. (1985). RFC959 - File Transfer Protocol. [Online]. Available: http://www.faqs.org/rfcs/rfc959.html.

[161] M. Horowitz and S. Lunt. (1997). FTP Security Extensions. [Online]. Available: http://www.ietf.org/rfc/rfc2228.txt.

[162] Globus. (2010). Overview of the Grid Security Infrastructure. [Online]. Available: http://www.globus.org/security/overview.html.

[163] M. Yang, et al., "Research on GridFTP Traffic Features," in *Proc of the 2009 WRI Int Conf on Communications and Mobile Computing*, 2009, vol. 3.

[164] J. Bresnahan, et al., "Globus GridFTP: What's New in 2007 (Invited Paper)," in *Proc of the First Int Conf on Networks for Grid Applications (GridNets 2007)*, 2007.

[165] J. Arthur and S. Azadegan, "Spring Framework for rapid open source J2EE Web Application Development: A case study," in *Proc of the Sixth Int Conf on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing(SNPD'05)*, 2005.

[166] G. v. Laszewski, et al., "A Java Commodity Grid Toolkit," *Concurrency: Practice and Experience*, vol. 13, 2001.

[167] DICE. (2010). Jargon, A Java client API for the DataGrid. [Online]. Available: https://http://www.irods.org/index.php/Jargon.

[168] M. Fowler. (2004). Inversion of Control Containers and the Dependency Injection pattern. [Online]. Available: http://www.martinfowler.com/articles/injection.html.

[169] G. Kloss. (2010). MataNui - Building a Grid Data Infrastructure that doesn't suck! [Online]. Available: http://www.eresearch.org.nz/content/matanui-building-grid-data-infrastructure-doesnt-suck.

[170] MongoDB. GridFS. [Online]. Available: http://www.mongodb.org/display/DOCS/GridFS.

[171] K. Czajkowski, et al., "Grid Information Services for Distributed Resource Sharing," in *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, 2001.

[172] J. Shantenu, et al., "Developing Scientific Applications with Loosely-Coupled Sub-tasks," in *Proceedings of the 9th International Conference on Computational Science: Part I*, Baton Rouge, LA, 2009.

[173] G. Iwai, et al., "SAGA-based user environment for distributed computing resources: A universal Grid solution over multi-middleware infrastructures," in *International Conference on Computational Science, ICCS 2010*, 2010.

[174] T. Goodale, et al., "A Simple API for Grid Applications (SAGA) (GFD-R-P.90)," *Open Grid Forum*, 2008.

[175] T. Peter, et al., "Standardization of an API for Distributed Resource Management Systems," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, 2007.

[176] C. Aiftimiei and et al., "Job submission and management through web services: the experience with the CREAM service," *Journal of Physics: Conference Series*, vol. 119, p. 062004, 2008.

[177] B. Dillaway, et al., "HPC Basic Profile, Version 1.0 (GFD-R-P.114)," *Open Grid Forum*, 2007.

[178] M. McIntosh, et al. (2007). Basic Security Profile Version 1.0. [Online]. Available: http://www.ws-i.org/profiles/basicsecurityprofile-1.0.html.

[179] A. R. Alvarez, et al., "BES++: HPC Profile Open Source C Implementation," in *proceeding of the 9th IEEE/ACM International Conference in Grid Computing*, Tsukuba, Japan, 2008.

[180] P. Andreetto, et al., "Cream: A simple, grid-accessible, job management system for local computational resources," in *Proceedings of CHEP'06*, Mumbay, 2006.

[181] H. Fredrik, et al., "Benchmarking of Integrated OGSA-BES with the Grid Middleware," in *Euro-Par 2008 Workshops - Parallel Processing*, C. Eduardo, et al., Eds.: Springer-Verlag, 2009, pp. 113-122.

[182] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Doctoral dissertation thesis, University of California, Irvine, 2000.

[183] Hibernate. [Online]. Available: http://www.hibernate.org/.

[184] J. M. Schopf, "A General Architecture for Scheduling on the Grid," *JPDC, Special Issue on Grid Computing*, April 2002.

[185] Spring Batch. [Online]. Available: http://static.springsource.org/spring-batch/.

[186] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *J Comp Phys*, vol. 117, pp. 1-19, 1995.

[187] L. Moresi, et al., "Computational approaches to studying non-linear dynamics of the crust and mantle," *Physics of the Earth and Planetary Interiors*, vol. 163, pp. 69-82, 2007.

[188] M. Drescher, et al., "JSDL Parameter Sweep Job Extension," *Open Grid Forum*, 2009.

[189] H. Casanova, et al., "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," in *Proceedings of the 9th Heterogeneous Computing Workshop*, 2000.

[190] R. Buyya, et al., "Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid," in *the Proceedings of High Performance Computing in the Asia-Pacific Region*, 2000, vol. 1, pp. 283-289 vol.281.

[191] B. Chevreux, "MIRA: An Automated Genome and EST Assembler," PhD thesis, Department of Molecular Biophysics, German Cancer Research Center Heidelberg 2005.

[192] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research*, vol. 18, pp. 821-829, May 1, 2008 2008.

[193] B. Brett, et al., "A portal for grid-enabled physics," in *Proceedings of the 2005 Australasian workshop on Grid computing and e-research - Volume 44*, Newcastle, New South Wales, Australia, 2005.

[194] B. Marović and Z. Jovanović, "Web-based grid-enabled interaction with 3D medical data," *Future Gener. Comput. Syst.*, vol. 22, pp. 385-392, 2006.

[195] T. N. Truong, "An Integrated Web-based Grid-Computing Environment for Research and Education in Computational Science and Engineering," in *Proceedings of the 37th annual symposium on Simulation*, Virginia, 2004.

[196] S. Gurmeet, et al., "The Pegasus portal: web based grid computing," in *Proceedings of the 2005 ACM symposium on Applied computing*, Santa Fe, New Mexico, 2005.

[197] G. Andronico, et al., "The GENIUS Web portal: Grid computing made easy," in *Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. International Conference on*, 2003, pp. 425-431.

[198] M. Faerman, et al., "Managing Large Scale Data for Earthquake Simulations," *Journal of grid computing*, vol. 5, p. 295, 2007.

[199] R. Wyrzykowski, et al., "The Grid Portlets Web Application: A Grid Portal Framework," in *Parallel Processing and Applied Mathematics*, vol. 3911: Springer Berlin / Heidelberg, 2006, pp. 691-698.

[200] M. Pierce, et al., "Open Grid Computing Environments," in *TeraGrid 2009*, Arlington VA, 2009.

[201] S. Andreozzi and M. Marzolla, "A RESTful Approach to the OGSA Basic Execution Service Specification," in *Fourth International Conference on Internet and Web Applications and Services, ICIW '09.* , 2009.

[202] R. Buyya and S. Venugopal, "The Gridbus toolkit for service oriented grid and utility computing: an overview and status report," in *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, 2004, pp. 19-66.

APPENDIX A: A SAMPLE PARAMETER SWEEP JOB

This is the complete version of the sample parameter sweep job I gave in Figure 6-16.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns:sweepfunc="http://schemas.ogf.org/jsdl/2009/03/sweep/functions"
                      xmlns:sweep="http://schemas.ogf.org/jsdl/2009/03/sweep"
                      xmlns:file-sweep="http://schemas.ogf.org/jsdl/2009/03/file-sweep"
                      xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-hpcpa"
                      xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl">
    <jsdl:JobDescription>
        <jsdl:JobIdentification>
            <jsdl:JobName>mira_param_sweep_job</jsdl:JobName>
        </jsdl:JobIdentification>
        <jsdl:Application>
               <jsdl:ApplicationName>Mira</jsdl:ApplicationName>
            <jsdl-hpcpa:HPCProfileApplication >
                <jsdl-hpcpa:Executable>mira</jsdl-hpcpa:Executable>
                <jsdl-hpcpa:Argument>--project=</jsdl-hpcpa:Argument>
                <jsdl-hpcpa:Argument>--parameterfile=miraIntersect.txt</jsdl-
hpcpa:Argument>
                <jsdl-hpcpa:Argument>-MI:sonfs=no</jsdl-hpcpa:Argument>
            </jsdl-hpcpa:HPCProfileApplication>
        </jsdl:Application>
        <jsdl:Resources>
               <!--
               <jsdl:CandidateHosts>
                      <jsdl:HostName></jsdl:HostName>
               </jsdl:CandidateHosts>
               -->
            <jsdl:TotalCPUCount>
                <jsdl:Exact>1</jsdl:Exact>
            </jsdl:TotalCPUCount>
            <jsdl:TotalCPUTime>
                <jsdl:Exact>60</jsdl:Exact>
            </jsdl:TotalCPUTime>
            <jsdl:TotalPhysicalMemory>
               <jsdl:Exact>5000</jsdl:Exact>
            </jsdl:TotalPhysicalMemory>
        </jsdl:Resources>
           <jsdl:DataStaging>
               <jsdl:FileName>any</jsdl:FileName>
               <jsdl:CreationFlag>override</jsdl:CreationFlag>
               <jsdl:Source>
                      <jsdl:URI>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/miraIntersect.txt</jsdl:URI>
               </jsdl:Source>
           </jsdl:DataStaging>
           <jsdl:DataStaging>
               <jsdl:FileName>any</jsdl:FileName>
               <jsdl:CreationFlag>override</jsdl:CreationFlag>
               <jsdl:Source>
                      <jsdl:URI>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/public/AATest/MiraAutoSelect-46-66/miraIntersect.txt</jsdl:URI>
               </jsdl:Source>
```

```
            </jsdl:DataStaging>
            <jsdl:DataStaging>
                <jsdl:FileName>any</jsdl:FileName>
                <jsdl:CreationFlag>override</jsdl:CreationFlag>
                <jsdl:Source>
                        <jsdl:URI>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/public/AATest/MiraAutoSelect-46-66/miraIntersect.txt</jsdl:URI>
                </jsdl:Source>
            </jsdl:DataStaging>
            <jsdl:DataStaging>
                <jsdl:FileName>any</jsdl:FileName>
                <jsdl:CreationFlag>override</jsdl:CreationFlag>
                <jsdl:Source>
                        <jsdl:URI>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/public/AATest/MiraAutoSelect-46-66/miraIntersect.txt</jsdl:URI>
                </jsdl:Source>
            </jsdl:DataStaging>
            <jsdl:DataStaging>
                <jsdl:FileName>any</jsdl:FileName>
                <jsdl:CreationFlag>override</jsdl:CreationFlag>
                <jsdl:Target>
                </jsdl:Target>
            </jsdl:DataStaging>
    </jsdl:JobDescription>
    <sweep:Sweep>
                <sweep:Assignment>
                <sweep:DocumentNode>
                <sweep:NamespaceBinding ns="http://schemas.ggf.org/jsdl/2006/07/jsdl-hpcpa"
prefix="jsdl-hpcpa" />
                    <sweep:Match>
                       /*//jsdl-hpcpa:Argument[1]
                    </sweep:Match>
                  </sweep:DocumentNode>
                  <sweepfunc:Values>
                     <sweepfunc:Value>--project=Kukri19</sweepfunc:Value>
                     <sweepfunc:Value>--project=Kukri20</sweepfunc:Value>
                     <sweepfunc:Value>--project=Kukri21</sweepfunc:Value>
                  </sweepfunc:Values>
        </sweep:Assignment>
                <sweep:Assignment>
                 <sweep:DocumentNode>
                <sweep:NamespaceBinding ns="http://schemas.ggf.org/jsdl/2005/11/jsdl"
prefix="jsdl" />
                    <sweep:Match>
                       /*//jsdl:DataStaging[2]/jsdl:Source/jsdl:URI
                    </sweep:Match>
                  </sweep:DocumentNode>
                  <sweepfunc:Values>
                     <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri19_in.454.fasta</sweepfunc:Value>
                     <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri20_in.454.fasta</sweepfunc:Value>
                     <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri21_in.454.fasta</sweepfunc:Value>
                  </sweepfunc:Values>
        </sweep:Assignment>
                <sweep:Assignment>
                 <sweep:DocumentNode>
```

```
               <sweep:NamespaceBinding ns="http://schemas.ggf.org/jsdl/2005/11/jsdl"
prefix="jsdl" />
                    <sweep:Match>
                       /*//jsdl:DataStaging[3]/jsdl:Source/jsdl:URI
                    </sweep:Match>
                 </sweep:DocumentNode>
                 <sweepfunc:Values>
                    <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri19_in.454.fasta.qual</sweepfunc:Value
>
                    <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri20_in.454.fasta.qual</sweepfunc:Value
>
                    <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri21_in.454.fasta.qual</sweepfunc:Value
>
                 </sweepfunc:Values>
        </sweep:Assignment>
                <sweep:Assignment>
                 <sweep:DocumentNode>
                <sweep:NamespaceBinding ns="http://schemas.ggf.org/jsdl/2005/11/jsdl"
prefix="jsdl" />
                    <sweep:Match>
                       /*//jsdl:DataStaging[4]/jsdl:Source/jsdl:URI
                    </sweep:Match>
                 </sweep:DocumentNode>
                 <sweepfunc:Values>
                    <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri19_in.solexa.fastq</sweepfunc:Value>
                    <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri20_in.solexa.fastq</sweepfunc:Value>
                    <sweepfunc:Value>gsiftp://arcs-
df.vpac.org:2810/ARCS/home/Cloud_Input_Examples/Kukri21_in.solexa.fastq</sweepfunc:Value>
                 </sweepfunc:Values>
        </sweep:Assignment>
    </sweep:Sweep>
      </jsdl:JobDefinition>
```