Henrik Thostrup Jensen
Jesper Ryge Leth

Automatic Job Resubmission in the Nordugrid Middleware

Dat5 Project
September 2003 - January 2004

Department of Computer Science
Aalborg University
Fredrik Bajersvej 7E
DK–9220 Aalborg
DENMARK

Faculty of Engineering and Science
Aalborg University

Department of Computer Science

TITLE:
Automatic Job Resubmission
in the Nordugrid Middleware

PROJECT PERIOD:
Dat5,
1st September 2003 -
13th Januar 2004

PROJECT GROUP:
B2-201

GROUP MEMBERS:
Henrik Thostrup Jensen
Jesper Ryge Leth

SUPERVISOR:
Josva Kleist

NUMBER OF COPIES: 4

REPORT PAGES: 56

APPENDIX PAGES: 18

TOTAL PAGES: 74

SYNOPSIS:

This report describes the development of a daemon for automatic resubmission of jobs in the NorduGrid Toolkit.

The report starts by giving an introduction to grids, what they are, why they are useful, and how resource sharing is organized by creating virtual organizations. Hereafter the focus shifts to grid architectures, where a general grid model is presented, along with examples of three different grid architectures. The NorduGrid project is then described with regards to its political structure, architecture, and individual parts of the toolkit.

After discussion the NorduGrid project, the goal of the project is presented. The goal is to make automatic resubmission possible in the NorduGrid Toolkit, and use this to create automatic job resubmission on job failure. A discussion about resubmission models follows and one is chosen, and issues concerning it are discussed further.

Hereafter our implementation, called NG Proxy, is presented. The internals of NG Proxy is explained along with how it integrates with the existing NorduGrid Toolkit. Lastly a future work chapter presents our future ideas for NG Proxy.

The report concludes that the model, of having a daemon, monitor and resubmit jobs, work. Furthermore it is concluded that NG Proxy could be extended to handle other scenarios where job submission is a solution, and that NG Proxy is a step in the right direction toward a production management system.

Preface

This report has been written Henrik Thostrup Jensen and Jesper Ryge Leth at Aalborg University, Department of Computer Science; in the period from 1. September 2003 to 13. January 2004.

The report is about computational grids and the implementation of a daemon, which is able to do automatic resubmission of jobs in the NorduGrid Toolkit.

We assume that the reader has basic knowledge of operating systems, especially Unix and GNU/Linux systems. Some knowledge of distributed systems and the grid concept is also assumed. For specific knowledge of Unix/Linux concepts, we refer to [16]

Josva Kleist has been supervisor on the project and we would like thank him; especially for sending us to the 6th NorduGrid Workshop in Lund. We also like to thank the people on the mailing lists nordugrid-discuss and nordugrid-support for helping us, especially Anders Wäänänen, for his persistence in helping us installing and configuring the NorduGrid Toolkit.

The source for our implementation can be fetched from `www.cs.auc.dk/~htj/nordugrid/ngproxy-0.1.tar.gz`. The tarball includes a patch, and the instructions for patching nordugrid.

Henrik Thostrup Jensen

Jesper Ryge Leth

Contents

1 Computational Grids **9**
1.1 Why The World Need Computational Grids 10
1.2 Resource Sharing and Virtual Organizations 11

2 Grid Architectures **13**
2.1 The need for Interoperability 14
2.2 A Generic Model for a Grid Architecture 15
2.3 Grid Environments 16

3 NorduGrid **23**
3.1 The NorduGrid Components 24

4 Project Goal **35**

5 Job Resubmission **37**
5.1 Job Resubmission and scheduling 37
5.2 Models for Job Resubmission 37
5.3 Choosing a Model 39
5.4 New Job Submission Flow 42
5.5 Summary 42

6 NG Proxy **45**
6.1 Implementation 45
6.2 Integration 47
6.3 Summary 48

7 Future Work **49**

8 Conclusion **51**

A Using NG Proxy **53**

B Grid Protocols and Components **55**
B.1 GRAM 55
B.2 GRIP 55
B.3 MDS 55
B.4 GRRP 56
B.5 GRIS 56
B.6 GIIS 57

B.7 GridFTP . . . 57
B.8 GSI . . . 58
B.9 Nexus . . . 58
B.10 GASS . . . 58
B.11 DUROC . . . 59
B.12 Heart Beat Monitor . . . 59

C Installing the NorduGrid Toolkit on Debian GNU/Linux 61
C.1 Getting Debian Ready . . . 61
C.2 Installing ScalablePBS . . . 62
C.3 Installing gSOAP . . . 62
C.4 Installing gpt, globus and globus-config . . . 62
C.5 Installing NorduGrid . . . 63
C.6 Host key . . . 63
C.7 Installing certificates . . . 64
C.8 Allowing VO users . . . 64
C.9 Mimic Red Hat isms . . . 64
C.10 Configuring the software . . . 64

D The NG Proxy Todo List 67

E NorduGrid Task List 69

Chapter 1

Computational Grids

The word grid probably means something different to everybody, it has been hyped over the last couple of years, and everybody seems to have their own understanding of the grid concept. There does seem to be some consensus that is has something to do with technology that utilizes distributed and/or parallel computing. For instance when Sun Microsystems talks about grid they primarily mean clustering and load balancing [19, 20]. Likewise Oracle mainly mean distributed databases [2, 21].

The term grid was originally coined in the book by Ian Foster and Carl Kesselman "The Grid: Blueprint for a New Computing Infrastructure" [11] in 1998 to describe a new computing infrastructure. At the time the Grid was defined as:

> *"A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high end computational capabilities."* [9]

This was the beginning of the idea of "The Grid" a supercomputer made up of every computer on the Internet. An idea which still lurks in the head of a lot of people. The name grid came from an analogy to the electric power grid, where computational power was to be considered a resource similar to electric power. On a historical note we start by observing that it was not the discovery of electricity, but the invention of the power grid that really kicked off the electrical revolution and made electrical devices for everyone to own. The power grid enabled everybody to get as much power as they needed without having to pay large sums of money for their own generator.

A power grid is a dynamic heterogeneous infrastructure which consists of a network with a lot of different producers and consumers connected. The producers are power plants which vary with respect to price and contribution, from a nuclear power plant to a farmer with a single windmill in his back yard. The producers and consumers can appear and disappear without notice. When a producer disappears others take over, without the consumers noticing it. The power grid serves all types of consumers, from private households to million dollar corporations. The consumers all connect to the same power grid and use as much of the resources as needed and only pay, for the amount they have used. This is one of the key points to the success of the power grid; that even though the a power plant is very expensive, and a large investment, the power is still cheap relative to the consumer [10].

The original vision was to view computational cycles and other expensive resources such as analytical equipment as resources which everyone would have access to, by subscribing to the grid. This grid would consist of every computer connected to the

Internet, essentially acting as one supercomputer which everyone could connect to, in order to use and contribute resources. In this scenario CPU cycles other resources can be traded just like power, and consumers are being billed for what they use, and producers are being paid for what they contribute. However analogies can be a dangerous thing, and computational power is not electrical power, as they differs in several respects [10]. It is important to note this for a number of reasons. First computational cycles are a highly volatile resource and it cannot be stored for future use in the same way as electricity. This is partly true for some types of electricity as well, e.g., wind mills which produce power when it is windy, but often the production of electricity can be reduced by delaying conversion of energy (e.g., coal and oil), until it is needed. This cannot be done for CPU cycles that are lost, if they are not used immediately. This serves as a larges incitement for developing ways to trade idle computational power. The analogy to the grid is also not accurate in other aspects as there can be a number of different resources connected to the grid, not just CPU cycles, but at wide range of different equipment.

This was the vision of the people behind the idea of computational grids. Grid technology today is “a work in progress” and there are issues regarding distribution, security, scheduling, scalability, and almost every other problem imaginable when dealing with distributed systems on a large scale that has to be solved to implement “The Grid”. Today the focus have shifted from trying to create one large grid toward more specialized grids. An example of this is sharing of resources among smaller groups collaborating, but spread geographically, to try to solve a subset of the problems and make a grid that works, even though it is on a smaller scale and with a specific purpose in mind. Some predict that we will have a working grid, with these issues solved, in a matter of a few years, while others are not so optimistic.

1.1 Why The World Need Computational Grids

Under this rather pretentious heading, we will explain why we think that grids serves a purpose and are not just “hot air”. We will do this by departing from the idea of a global super computer, and look at computational grids from a more pragmatic angle.

The ordinary home user does not often need a lot of computational power and have no real need for high performance computing facilities. Most of the time, in fact 70% of the time, a typical workstation in a corporate environment are idle [10]. In some situations even the ordinary user may need access to a lot of computing power, e.g., when checking his or her stock portfolio, or a similar demanding task, a lot of CPU time is needed, e.g., to make more precise predictions of the rate of change. Here a subscription to a grid would come in handy supplying on-demand computational power. The more ordinary usage of high performance computing (HPC), would also benefit from the added computational power of a grid. In addition the grid should enable the sharing of expensive equipment between various groups and organizations, for instance scientists located around the world, and ease collaboration and lower costs (see section1.2 about virtual organizations). Below we have summarized what seems to be the main reasons for developing computational grids.

- **Bigger problems** - We have bigger problems which needs more resources to run. Even though we have bigger problems, our own resources, which are quite expensive, but are not always fully used, and they cost money to operate even when idle.

- **We need to save money** - HPC resources are expensive, and the acquisition costs can be brought down by sharing these resources in a grid. The use of grid technology also enables the users to solve problems faster than before, due to extended use of resources. If institutions with similar problems is given the ability to pool their resources they can solve a given problem faster.
- **We do not always have computational power at hand** - PDAs and other embedded devices have become more widespread during the past years, and this trend seems to continue. However PDAs still has limited computing power, and having an infrastructure that enables access to HPC resources from a PDA would be beneficial and enable pervasive use of high performance computing.

One of the requirements for these areas are the sharing of resources and the next chapter will look further into the concept of virtual organizations. A way to enable corporation and sharing of resources.

1.2 Resource Sharing and Virtual Organizations

The vision of creating an infrastructure that would bind together every computer on the Internet into a single giant supercomputer raises a number of issues regarding security and sharing of resources. Not only do we need to share our resources, but we also needs to allow foreign code to run on our machines without a chance to review it. Not everyone would grant everyone access to their computational resources, and controlling access in a single giant grid can become troublesome, due to disagreement of who will get access and to what. This concern was addressed by Ian Foster and Steven Tuecke [11] who modified the grid definition to take political and social issues into account.

> *"The sharing that we are concerned with is not primarily file exchange but rater direct access to computers, software, data, and other resources, as is required by a range of collaborative problem solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which the sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a* virtual organization.*"* [9]

This definition introduces the concept of a virtual organization, which is a fundamental concept when talking about grids and sharing of resources.

A virtual organization is a group of institutions and users, who have decided to share resources with each other. To do so they form a virtual organization, thereby establishing a relation between the users and organizations, forming a grid. The users in the organization is able to use resources shared in this virtual organization. An institution, user or resource is not necessarily tied to a single virtual organization. An institution may choose be a part of several virtual organizations, sharing some resources to one, some to another, and some resources too more than one virtual organization [11].

A running grid will have resources disappearing (e.g., hardware failure or resource withdrawal), and appearing. The situation is the same for users, that users can get access to a grid, and have it revoked as well. This means that grids will have users and resources appearing and disappearing, yielding a highly dynamic nature. Some

virtual organizations will be more dynamic than others, e.g., a group of scientists may form a grid to analyze data. Such a group could last for years and only have a few users entering and leaving. Other virtual organizations may be far more dynamic, e.g., a virtual organization could grant access to every student at a certain institute. This dynamic nature imposes certain requirements upon the infrastructure. It must support an easy and automated way of registering new users and resources, while being able to cope with disappearance of users and resources.

Virtual organizations will most certainly play an important role in the future of grids, since they concern the two of the most important things in grids: Users and the sharing of resources. Making virtual organizations easy to create and maintain are an important aspect, among many, to have in a grid infrastructure, if they are to become widely spread.

After this brief intro to computational grids, the next chapter will go into further detail. Discussing from an architectural point of view, what is needed in order to construct a working grid.

Chapter 2

Grid Architectures

This chapter discuss various grid architectures and considerations that arise when building grids. We start by looking at a general model of a grid architecture and moves on to issues that must be considered when building a grid. The chapter finishes by giving examples of existing grid environments. The discussion in this section is based on the discussion in the previous chapter along with the models and discussions presented in [10, **?**, 11, 15]. It outlines a set of basic properties and capabilities that a grid architecture must provide.

One of the things to be aware of when designing a grid architecture, is the entities in a grid environment. In principle the main entities of interest in a grid are users, resources, and jobs. These entities have characteristics and requirements which must be taken into considerations.

- **Users** - are geographically spread, they are in complex sharing relationships with organizations and other users. They require ease-of-use, authorization, authentication, trust, and needs access to several grids, assured a certain quality of service.
- **Resources** - are heterogeneous, dynamic, and geographically spread. They are not necessarily dedicated to grid jobs and requires fine grained access control.
- **Jobs** - belongs to different users and may consist of mobile code, foreign to the cluster. They may have secret content imposing security needs and need various runtime environments.

The list above is only some of the characteristics and needs, of the entities involved in a running computational grid, but they illustrate the complexity of such a system.

Looking at the definitions from the previous chapter, it is obvious that some of the key elements in a grid is sharing, decentralization of control, and heterogeneous resources. As an example of the problems and complexities when running in a grid environment, a thing as the simple operation of executing a program on a grid is nontrivial, as the input, output, and data has to be set up prior to the execution [15]. Fundamental requirements for computational grids are.

- **Scalable** - The fundamental idea behind computational grids are that they are, or at least will become very large. Thus it is important that the architecture and technology upon which grids are build scale well.

- **Robust** - As for other distributed systems the architecture must be robust. A grid must be fault tolerant and cope gracefully with network and resource failures, providing consistent and dependable quality of service.
- **Secure** - A grid architecture raises almost any security issue conceivable. Since a grid has no central control, and may span over several administrative domains, the security requirements upon the architecture are important. This is emphasized by the fact that the resource owners allows users from other trust domains to execute foreign code their resources.
- **Pervasive access** - This covers several issues of grid access. Access must be easy, with respect to user credentials (e.g., single sign on). Access must be provided from a wide range of computational devices and must be inexpensive.
- **Accounting** - There must be a reliable accounting system keeping track of resource usage, making it possible to charge the users. This includes accounting with respect to resource usage and contribution. Accounting and payment raises issues about quality of service, and a grid should facilitate a way to measure and assure this.

These are only some of the issues and list goes on, other important issues are: control, interoperability, open protocols, consistency of service, and scheduling policies.

The main purpose for the architecture are to conceal the heterogeneity and complexity of the underlying resources. Foster and Kesselman [10] points out that a grid architecture is first and foremost a protocol architecture and its goal is to ensure interoperability. Furthermore the architecture must facilitate fine grained access control over the sharing of resources. There is discussions [11, 15] of whether new programming models is needed and if these should be implemented in therms of basic grid protocols.

2.1 The need for Interoperability

Since users and resources can be members of several virtual organizations, there is a clear need to have common protocols [11]. Having separate protocols for each grid middleware is not a feasible option if virtual organizations are to be created quickly and maintained easily. Furthermore a user or resource could be member of several virtual organization, making interoperability almost impossible without common protocols. Additionally these protocols should be standardized such that different grid middlewares can be created, while still being able to talk together; much like how the IP protocol works today.

Writing such a middleware is not an easy task though, so most grid users will use an existing middleware to create their grid application. Some users probably want to extend their middleware, with some specialized services or access to uncommon resources. Such users should not have to create their own middleware, but rather extend existing middleware to fit their needs. This means that the code, for at least some middlewares, should be open.

Given that there will exist several middlewares for grid applications, these should be accessible in a uniform way. This means that they should provide a similar API to the application programmer. However, as described above, middlewares will differ in functionality. Due to these differences it would be impractical for all middlewares to

provide the same API. Instead they should aim to provide the basic API. This would mean that grid applications could be ported between different middlewares, without too much effort. This may not be a realistic goal since there are many different ways for solving the grid problems as we will see in the next chapter. However standards are being developed which should ensure interoperability.

2.2 A Generic Model for a Grid Architecture

To address the issues just described, a grid environment can be described as a set of abstract levels. These can roughly be divided into three parts: Core grid, services and user interface [15]. The core grid consists of the resources and applications running on the resources. The services provides a homogeneous interface to the resources, as well as facilitating discovery and resource brokering. The user interface supplies the user with a way to interact with the grid services. This interaction can be facilitated in many ways, ranging from a standard Unix shell augmented to support the functionality of the grid, to a web portal interfacing to the grid services. Furthermore a grid environment should fulfill two main functions: Provide user-side programming and control the user interaction.

We now take a look at a more detailed model of a grid architecture, and identifies the features necessary at the various levels. The model was originally presented in the article "The Anatomy of the grid" [11] and can be seen on figure 2.2. This model is

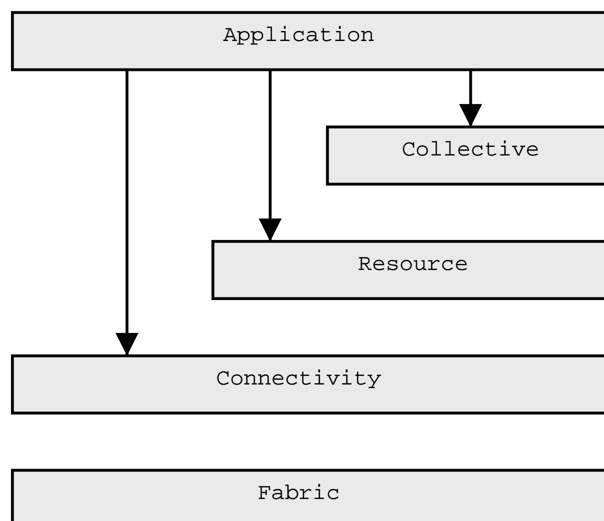

Figure 2.1: A generic model of a grid architecture. This model is originally from "The Anatomy of the Grid" [11].

made up of a set of abstraction levels as previously discussed. Starting at the bottom we have the Fabric. This is the level that interfaces with the local resources and provides shared access. The resources in this level may be either a physical or a logical entity, e.g., a cluster or distributed file system. The fabric level supports local resource specific operations which are dependent on the operations of the higher levels of the model. There is a trade off concerning functionality on this level. A richer set of functionality

may make advanced sharing functionalities available for the higher level, but at the same time making deployment of new resources more complex. As a minimum, this level should support mechanism enabling discovery of services and capabilities and a resource management mechanism delivering some control over the quality of service.

The connectivity layer defines the grid related protocols providing the necessary grid specific functionality like communication and authentication protocols. The communication protocols should enable exchange of data between fabric resources, including routing, naming, and transport. Much of this can be achieved trough existing protocols, e.g., TCP/IP, and DNS. The authentications protocols should also, due to complexity and security issues, rely on existing protocols and provide support for virtual organizations, supporting single sign on, trust relationships, and delegation. The authentication protocols should integrate well with existing protocols and systems.

The resource layer is relying on the connectivity layer supplying protocols for negotiation, monitoring, control, and accounting operations on the individual resources. The functionality of this layer can be split into two classes; information protocol and management protocol. The protocol layers form a bottleneck in the model and should therefore be kept as small and simple as possible while still supplying the needed functionality.

The collective layer are focused on the global resource view and contains services and protocols not associated with any single resource. The collective deals with relationships and interactions between resources. This layer implements services, including directory services, scheduling, monitoring, and accounting. These protocols are general in nature and rely on the services of the underlying layers to implement the needed functionality with respect to the individual resources. The functions in this layer can be implemented as services with associated protocols, or as software development kits with associated APIs. The collective can be developed for specific use (e.g., specific VO requirements) or for a more general purpose

The application layer is the applications that run within a specific VO environment and applications at this level makes use of the services of the lower levels by means of well defined protocols and APIs. We can now specify what we mean when we use the term grid, we mean:

> *An infrastructure that enables controlled sharing of computational resources across sites and trust boundaries. Between users from different organizations and institutions which may be geographically spread. The resources belongs to the institutions and users who remain in control of their own resources. The users of the grid have the possibility to securely run jobs on the shared resources and the resource owners has the ability to charge the users for usage of the resources.*

Ultimately all of these requirements are needed in order to be talking about grid, However if a fairly large subset are met, it is enough for us to be talking about grid in this report.

2.3 Grid Environments

This section examines existing types of grid toolkits that are in use today. We have selected examples among the many that exists today. The reason for selecting the ones described in this section is, that they represent some of the major different approaches

for constructing grids. For a more extensive list and a description of the grid environments available today, we refer to [15].

Before discussing the various environments, we start by describing the Open Grid Server Architecture. Even though there are much discussion about standards for computational grid protocol, there are surprisingly few. One of them is the Open Grid Service Architecture (OGSA), which is an attempt to establish a common standard for grid architectures. OGSA is based on web services and the grid services provided by OGSA follows the Open Grid Service Infrastructure (OGSI) [41], meaning that every service is a web service[1]. Many projects seems to embrace the standard and develop their toolkits accordingly [21, 27].

2.3.1 The Globus Alliance

The Globus Alliance is a research and development project with participation by several universities and large companies around the world [29]. They do not produce a grid environment, instead the main focus of the project is develop fundamental grid technologies needed to build a grid, called the Globus Toolkit 2. This toolkit should be considered a set of building blocks for creating grid middleware, meaning that it is not a complete grid solution, but rather a framework for building grid applications and solutions. Currently there exists two major versions of the Globus Toolkit; version 2 and 3. Both toolkits and the code for them are freely available on The Globus Alliance web page[2]. In the following sections both versions of the toolkit will be described.

The Globus Toolkit 2

The Globus Toolkit 2 was a continuation of version 1, however it had major revisions in several areas [27]. The toolkit defines its own set of communication protocols, meaning that it cannot easy communicate with other grid middleware. However over the past six years the Globus Toolkit 2, has evolved into becoming the de facto standard for computational grids [9]. This is likely due the large number of institutions which has build their grid solutions on the Globus Toolkit and today it is one of the most mature grid toolkits. Reasons for basing a grid solution on the Globus Toolkit are that the toolkit can be downloaded for free, and the code is freely available, making it possible to tailor it to suit your needs.

The toolkit it build of three major parts [33]: Resource management, information services and data management. Resource management is concerned with allocation and management of resources. Information services is the part that provides information about the various resources in a grid. The major component here is an information system. The last part, data management, is concerned with access and management of data. As of this writing the Globus Toolkit 2, is in version 2.4.3, which was released September 11, 2003.

The Globus Toolkit 3

The Globus Toolkit 3 is relatively new as its first release was in July 2003. This version is a major redesign of the previous Globus Toolkit. The reason for a redesign was

[1] Web services provide a standard means of inter operating between different software applications, running on a variety of platforms and/or frameworks. The interaction is made possible by using protocols and technologies such as SOAP and XML [1].

[2] Homepage at http://www.globus.org

to create an implementation compliant with OGSA [8]. The functionality of the services provided by Globus Toolkit 3 corresponds to the services provided by the Globus Toolkit 2, but they are implemented as web services in order to comply with the standard. The reason for implementing grid services as web services are extensibility and manageability. Where the services in Globus Toolkit 2, are separated and independent, meaning that it takes a significant amount of work to implement a new service or change an existing one. The Globus Toolkit 3 provides a framework for this, so existing OGSI services can be modified more easily and new services can be created faster [32]. Furthermore using and managing grid services has become uniform.

To accommodate migration from Globus Toolkit 2, several steps has been taken. Globus Toolkit 3 contains the same components as version 2, and has API compatibility. Also the same authentication is used, i.e., X.509[3] certificates [7], meaning that existing authentication and authorization mechanisms can be used.

Globus Toolkit 3, is currently in version 3.0.2. The releases since 3.0 has mostly been security and bug fixes as could be expected.

Even though Globus Toolkit 3 has been released, there is a continued development on Globus Toolkit 2. This is due to the many existing projects which has been based on version 2, and that version 3 is still relatively new. Thirdly not everyone believes that Globus Toolkit 3, is a step in the right direction, mainly because it is based on web services. It is not clear if a transition to web services will yield any advantages compared to the model of Globus 2, and Globus 3 has yet to prove its worth, competing with the six year old Globus 2. The Globus Toolkit 3, appears to have gotten a good reception [34], however only time will tell if it will replace version 2.

2.3.2 The European Data Grid

The European Datagrid is an example of a grid environment based on the Globus Toolkit 2. A brief discussion of the European Data Grid (EDG) follows. EDG is an initiative, researching in building a common European grid solution. It is funded by the European union and has a budget in the range of 10 million euro. EDG is led by CERN[4] and is a collaborative effort with several European research agencies including the European Space Agency (ESA) and national agencies from several European countries [3]. EDG is constructing a complete grid suite, not just a toolkit and are working on applications in several areas. Development is divided into four major areas: Testbed and Infrastructure, Applications, Computational and Data Grid Middleware, and Management and Dissemination. The application areas that EDG is aiming at is High Energy Physics, Biology and Medical Image processing, and Earth Observations.

EDG is based on Globus Toolkit 2, the aim for the next version, EDG 2, should be based on the OGSA standard and is going to be using web services as core technologies for creating a grid environment.

2.3.3 The NorduGrid Toolkit

The NorduGrid Toolkit will be examined in detail in the next chapter, but for the sake of completeness a short description follows. The NorduGrid Toolkit[5], is a toolkit based

[3]A widely used certificate standard.

[4]The European Organization for Nuclear Research

[5]The NorduGrid Toolkit has recently (December 2003) changed its name to Advanced Resource Connector (ARC), however we still use the old name NorduGrid Toolkit in this report.

on Globus Toolkit 2. It is the result of a collaboration between the Scandinavian countries to create and operate a Nordic computational grid. The purpose of the NorduGrid project was to create a testbed for a Nordic grid infrastructure and is a collaborative effort with participating research centers from Denmark, Norway, Sweden, and Finland. The focus was to create an infrastructure for future high energy physics experiments but this has been expanded over time and even though the main focus still is high energy physics the possibility of running other applications on NorduGrid is being examined. At the time of writing NorduGrid consists of clusters located at the participating organizations.

The NorduGrid approach is to base the toolkit on Globus Toolkit 2, reusing as many components as possible. However it has been necessary to extend some of the Globus components and replace others in order to get the desired functionality. NorduGrid is contrary to EDG not planning a move to Globus 3, although it has been discussed.

Due to the collaboration and the creation of the NorduGrid Toolkit, the local grid research centers in the Nordic countries, including Danish Center for Grid Computing (DCGC) and SweGRID, are basing their research and work on the NorduGrid toolkit. This has an impact on our choice of toolkit, making NorduGrid the natural choice, since we can get access to resources running the NorduGrid toolkit through DCGC.

After having looked at Globus and two toolkits based upon it, we move on to looking at Legion, representing a radically different approach to building grids.

2.3.4 Legion

Legion[6] is described as a world wide virtual computer. The Legion project is based at the University of Virginia the goal of the project is:

> *"Users working on their home machines see the illusion of a single computer, with access to all kinds of data and physical resources, such as digital libraries, physical simulations, cameras, linear accelerators, and video streams. Groups of users can construct shared virtual work spaces, to collaborate research and exchange information."*

This is very much in accordance with the original vision of "The Grid", making many computers act as one supercomputer. As opposed to Globus, Legion is trying to created an integrated solution and does not use preexisting services and technologies. Instead Legion are using an object oriented philosophy toward designing and constructing a grid and is build on top of a unified object model developed specifically for Legion [26]. In Legion everything is an object and Legion defines the message format and high-level protocol for object interaction, but not the programming language or the communications protocols. It is possible for users to provide their own classes, since the common services are implemented by core objects. Additionally Legion offers PVM and MPI libraries which applications can be compiled against in order for the application to take advance of the infrastructure.

The main objectives of the Legion project has a lot in common with other grid projects. The aim is to create a scalable and fault tolerant architecture that takes care of the management and exploitation of resource heterogeneity. When running Legion the user has what Legion refers to as a context space in which all applications, belonging to a user, run. That along with a virtual file system and a resource management system

[6] Homepage at http://legion.virginia.edu.

allows the user to run processes on the grid by extending the basic capabilities of the Unix shell to work with the distributed object file system [15].

The Legion project lists a set of constraints under which the software must run. These are very common for grid projects. For instance the host operating systems cannot be replaced, as well as changes to the interconnection network cannot be legislated by Legion and furthermore Legion cannot be required to run as "root". Legion works while keeping site autonomy and ensure security for users and resource owners. Legion works on top of the users operations system and negotiates security and scheduling policies with the different sites according to their policies. All of the above has to do with keeping site autonomy, maintaining, and adhering to the local policies, whether it is being security, resource or other policies.

These objectives and constraints are very similar to many of the grid projects, and when examining NorduGrid design philosophy in section 3.1 we will see that many of the same objectives and constraints are valid. The main difference is that the approach to solving the problems that differs and not the fundamental goals as many of these goals are paramount when dealing with distributed systems that cross trust boundaries.

2.3.5 Summary

We have been looking at several different grid environments and toolkits. They each use different approaches trying to solve the grid problem. These approaches each has their advantages and disadvantages. One of the main differences between Globus 2 and Legion is that Globus 2 does not have an underlying component architecture [25]. This approach yields advantages as well a some disadvantages.

The main problem with the Legion approach is that every piece of software ever to run on Legion must be build specifically for Legion or at least ported or recompiled against the Legion libraries. Legion tries to accommodate this by providing special Legion enabled versions of a number of well know APIs such as MPI. However having a unified namespace and resource abstraction where everything is an object makes grid specific application development easier. It also serves as a basis for tighter integration between the applications, and the grid middleware making this approach closer to becoming “The Grid”. This closer integration should also make the construction of interactive grid applications possible. A thing that is not possible with Globus based grid environments, however this comes at the price of more complex deployment.

The Globus Toolkit however, is not without advantages. Looking at NorduGrid it it in principle a very large batch system and resource broker. This batch like behavior is more than enough for a number of applications, especially in the area of high level physics where the problems and applications can easily be distributed, e.g., parameter studies. These types of applications does not need human interaction, but computational power. Here the advantage would be that the possible overhead from the communication in tighter integrated system is reduced and limited to the transferring of jobs and job data. Additionally a lot of existing applications can be brought to run on this type of grid without modifications.

Thirdly there is the web service model as introduced by OGSA and used in Globus Toolkit 3. This model can in some way be considered the middle ground between the two other toolkit models. It offers tighter integration than Globus 2, by supplying an object model. This model is not as tightly integrated as Legion, because the interactions between the components is defined on a lower level using standard protocols (SOAP) and common data descriptions (XML). The looser integration compared to Legion is seen by, e.g., the lack of a distributed file system and lack of a uniform namespace.

It is hard to say if there are a “right” way to build grids, and we believe that the world is big enough for all three types of models, especially since they address different type of user and application needs.

Chapter 3

NorduGrid

This chapter takes a deeper look into the NorduGrid Toolkit, and examines the individual components, in order to get a better understanding of how the toolkit works.

The *Nordic Testbed for Wide Area Computing and Data Handling*, or, NorduGrid, is the grid architecture, which we have chosen as the basis for our work in this project. At the moment of writing, NorduGrid is going through reorganization of the organizational structure. The NorduGrid organization is changing name to Nordic Data Grid Facility (NDGF) and the NorduGrid Toolkit will be changing its name to Advanced Resource Connector (ARC). The primary job of NDGF, will be to coordinate connection and usage of the Nordic grid, and control the agreements with organizations who wish to use the Nordic grid. Internally NGDF coordinates the contributions to the Nordic grid by the various national grid facilities and serve as certification authority handling authentication, authorization, and accounting issues. The group maintaining and developing ARC will still be called NorduGrid [24].

The purpose of the NorduGrid project was to create a testbed for a Nordic grid infrastructure, and is a collaborative effort with participating research centers from Denmark, Norway, Sweden, and Finland. At the time of writing NorduGrid consists of clusters located at the participating organizations in the different countries. The main purpose was to create an infrastructure for future high-energy physics experiments.

The NorduGrid project was started in May 2001, in response to the ATLAS data challenge. The ATLAS data challenge is the name of the first Large Hadron Collider (LHC) application to be executed in a computational grid environment. It is a series of challenges to test the computing infrastructure. The first challenge, ATLAS DC1, was started in July 2002 and ran through the first part of 2003. During this time more than 2 TB input data was processed and more than 2.5 TB output data was produced by more than 4750 grid jobs [4]. ATLAS Data Challenge was created in order to prepare for the intaking of data from the LHC being build at CERN, when it goes into commission in April 2007. The LHC being build at CERN is the largest of it's kind and sets new standards for the amount of data generated by high-energy physics experiments. When operational the LHC is expected to generate from 100 Mb to 1 Gb/sec adding up to more than 1 Pb/year [23].

3.1 The NorduGrid Components

We start by outlining the fundamental design philosophy behind NorduGrid, as it was formulated when the project was started. NorduGrid should start out by being build on tools and technologies that actually works and proceed from there, in an effort to construct a scalable grid architecture, without single points of failure. For NorduGrid to be dynamic in creation of VOs, and to run on sites that is not dedicated to grid jobs, it is important that the owners of the respective resources retains full control over their own resources, local policies and configurations. To achieve this, the NorduGrid middleware should impose as few site requirements as possible, i.e., there should be no dictation of cluster configuration or install method. Furthermore no dependencies on particular hardware should exist and NorduGrid should reuse the existing system installation as much as possible. The computational unit of choice in NorduGrid is the cluster. Further restrictions was imposed on the design: The NorduGrid software should only be required on front end machines and the computing nodes nodes should not be required to be on a public accessible network. To summarize the goals.

- Avoid single points of failure.

- The architecture should be scalable and able to cope with a highly dynamic resource pool.

- Resource owners should retain full control over their resources.

In the initial phase of the project different existing grid middleware packages was examined. The two main candidates where Globus Toolkit 2 and EU Data Grid, which was analyzed further. Both of these found to be inadequate. The Globus Toolkit 2 did not support resource brokering, and it also lacked the middleware for staging large input and output data files. The EU Data Grid seemed to address these issues, but was at the time (early 2002) considered to premature to be the basis for NorduGrid, furthermore, it had a centralized resource broker which was seen as a bottleneck and a single point of failure.

In light of the result of the analysis it was decided to build the grid infrastructure from scratch. In practice the developers have been using the Globus Toolkit 2 as the basis of the development, addressing the various issues, and adding components that either replace, extends, or complement existing components in The Globus Toolkit.

3.1.1 Task Flow in the NorduGrid Toolkit

This section describes how the NorduGrid toolkit is usually operated, from the users perspective. This is done to give the reader a “feel” for how the toolkit works. We will go through the preparation, submission, and retrieval of jobs and job data. The first section goes quickly through the usage of the NorduGrid Toolkit, without explaining in detail how the elements work. This is done later in this chapter. Even though we will not go through the installation procedure, we will note that the installation of the entire Globus Toolkit is necessary for the NorduGrid client to work. The installation has proved difficult on flavors of Linux other than Red Hat, although, this mostly pertain to the installation of the server, and not the client. For a further discussion of the installation and installation problems see appendix C.

Job Preparation

The first thing needed in order to submit a job to the grid, is to have access to one or more resources, or be a member of a virtual organization with access to a set of resources. The access is based on a user certificate issued by the certificate authority.

In order for a job to run on a grid running the NorduGrid middleware, a job description needs to be prepared. This description supplies information to user interface, which is used to locate a cluster and execute the job. The description contains information needed to run the job on a cluster, including name of the executable, input data, location of input data, filenames, and location of output data, as well as other requirements, e.g., libraries required to run the job. The description is created in the Extended Resource Specification Language (xRSL). Since there are no screen or terminal to display input or output, when executing the job, all input and output must be redirected and the job description must state to which files the output must be redirected.

Job Submission

Before submitting the job to the grid, the user interface needs access to the proxy certificate, to be able to establish the users identity. This is done by running the program `grid-proxy-init`. Contrary to what may be suspected by the name, it does not start a program (a proxy), but generates an X.509 certificate which expires after a predefined amount of time. The proxy certificate is used to sign the job description to determine the identity, and credentials of the user. By using a time limited certificate it the severity of a compromised certificate is lessened, because it will eventually expire. This, however, does not prevent a hacker from using the certificate, to act on behalf of a user during the lifetime of the certificate. The job is submitted via the user interface, which locates a suitable cluster for the job, i.e., one where the user has the necessary privileges to execute jobs. When a cluster is found, the job is submitted, by uploading the job description using GridFTP and the user interfaces terminates.

Job Processing

Once arrived at the cluster, the Grid Manager checks every two minutes (default) for the arrival of new jobs. The grid manager submits the job to the Local Resource Management System (LRMS), and waits for it to finish. When a job is completed or failed, the user can choose to be notified by email when the job is finished, or check the status of the job manually, either through the grid monitor on the NorduGrid website, or through a program which queries the information system. When the job is completed, the grid manager does the post processing of the job data according to the job description and optionally moves the output files to a storage element for later retrieval by the user. The task flow is as follows:

1. The user creates a job description in xRSL.
2. The User Interface interprets the job description in order to perform resource brokering. It queries the information system to locate a resource to which the job can be submitted.
3. The job description is submitted to via GridFTP, to the grid manager on the chosen cluster.
4. The grid manager creates a session directory for the job data on the cluster.

5. The grid manager handles the preprocessing of the job data. If the job description states that data should be fetched from a storage element, the grid manager fetches the data, and makes them available within the session directory.

6. The job is submitted to the LRMS for execution.

7. The grid manager performs post processing of the output data. The grid manager can optionally register the data with a Replica Manager. Upon job completion the user can be notified by email.

8. The user downloads the data from the cluster, using the user interface or by GridFTP.

9. The grid manager deletes the job within a given time frame, if the user has not removed it.

On figure 3.1 task flown and the various protocols in the communications in the NorduGrid Toolkit are illustrated.

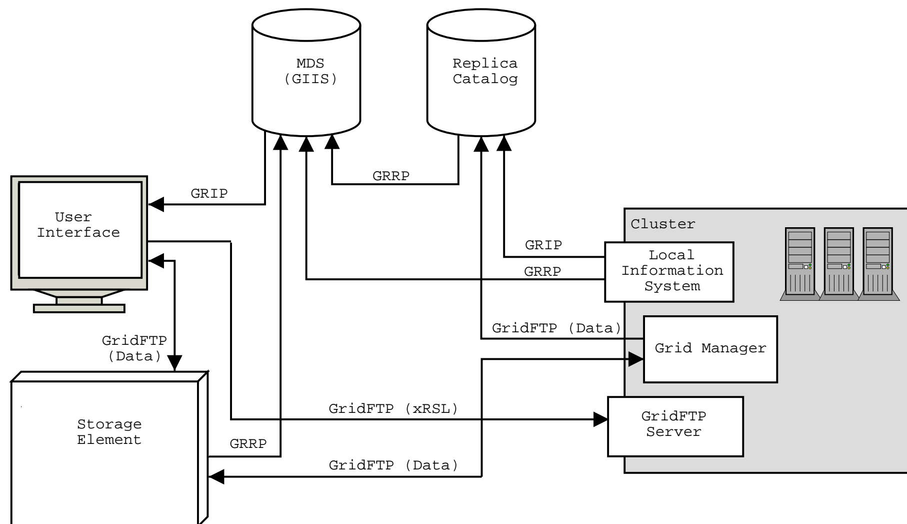

Figure 3.1: The communication and the various protocols handling the communication in NorduGrid. GRRP is the protocol used by resources for registering contact information with the information system (MDS), and GRIP is the protocol used for querying about resource information. GridFTP is used for both transfer of job data, and the job description itself.

3.1.2 NorduGrid Middleware Components

In this section the components of the NorduGrid toolkit is explained. This is done in the same order as they show up in the task flow just described. We start by looking at the extended resource specification language.

Extended Resource Specification Language

Extended Resource Specification Language (xRSL) is an extension to the Resource Specification Language supplied by Globus. It has been extended for use with NorduGrid, even though, not all Globus attributes are supported, mostly things concerning GRAM, which is replaced in the NorduGrid Toolkit with the Grid Manager. The language is divided into two levels: User-side xRSL and grid manager-side xRSL [38]. The user side part of the language is the description which is prepared by the user and send to the user interface. This part describes the job and its attributes. An example of such a description can be seen below.

```
&(executable=/bin/uname)
 (arguments=-a)
 (stdout="out.txt")
 (stderr="err.txt")
 (outputfiles=
    ("out.txt" "" )
    ("err.txt" "" ))
```

This description shows a very simple job, that prints various information about the node it is executed on. The first attribute is the name of the executable, and the second is the parameters it should be executed with. If the executable is not native to the cluster, and has to be transferred, this can also be specified. The next two attributes states where the output from the program should be redirected. The next two are the files that the user would retrieve after the job is finished, or optionally that the grid manager should upload to a storage element or register with a replica catalog. Apart from the attributes in the example, other attributes concerning disk space, runtime environment, middleware version, cluster, and many other attributes can be described.

The grid manager side of xRSL is used internally when the user interface submits the job to a cluster. It specifies things pertaining to the network, the submitting user, etc. Some of these attributes can also be specified by the user, though, this is not advised.

The User Interface

The User Interface is the major new component added. It delivers the high level functionality needed by NorduGrid but which is not supplied by Globus. The user interface the following functions: To handle resource discovery, resource brokering, job submission, and status querying. When a job is submitted through the user interface, it parses the accompanying xRSL job description in order to locate a suitable cluster to submit the job to. After retrieving a list of available clusters from the information system, the user interface queries the information system, to check if the user are allowed to submit a job to the cluster and if the cluster fits the needs of the job. In this step the user interface does some simple scheduling in selecting the cluster, based on the number of free CPUs on the clusters.

When a suitable cluster is found, the xRSL job description is stripped for user interface information, and grid manager-side information is added to the description. Hereafter the job is uploaded to the cluster by using GridFTP. Thus there is no need for additional services in order for resource brokering to work. Optionally extra data can be uploaded by the user interface, or it can be left to the Grid Manager to do this, by writing this in the job description.

Information System

For a system as complex as the NorduGrid Toolkit to work, it is important to have a robust, scalable and reliable information system, to store information about resources and jobs. The information system in NorduGrid is implemented as a distributed service, serving information to the other NorduGrid services and components. It is build upon the monitoring and discovery service (MDS), an information system framework supplied by the Globus Toolkit. The information system is essential, since the NorduGrid Toolkit relies upon it for all information related tasks. MDS is an extensible framework for creating grid information systems based on standard OpenLDAP, and and consists of the following.

- An information model described by a LDAP schema.
- Local information providers.
- Local databases.
- Soft registration mechanisms.
- Information indicies.

The information model supplied by Globus is single machine oriented, and not suited to describe clusters very well[1], so an information model was created specially for NorduGrid. The NorduGrid information model is a mirror of the architecture, and it describes the main components of the grid, i.e., clusters, jobs, and users. These elements are mapped onto an LDAP tree. This forms a hierarchical structure of queues where every user and every job has an entry. Additionally replica managers and storage elements are also described, but in a simplistic manner.

The Information System consist a dynamic set of distributed databases which are coupled to information providers residing on the clusters. In NorduGrid a single MDS service is run per resource, and the task of this service is to provide status information about the specific resource. Each resource operates its own Grid Resource Information Service (GRIS). These resources can be grouped together in order to form a virtual organization. Virtual organizations are served by a Grid Index Information Service (GIIS). For a description of GRIS and GIIS, see appendix B. This VO structure is called an MDS-tree and it is the actual mapping of the resources in the grid onto the information service.

The NorduGrid information providers are small programs that generates LDAP entries in the database upon search request. The NorduGrid Toolkit provides its own information providers. They serve as interface to local systems, collecting information about the status of a job from the clusters LRMS and the grid manager. This information can be used to find information about the resource; available CPUs, disk space, and effective queue length. NorduGrid provides access to two queues: NorduGrid-authuser and NorduGrid-jobs. Authuser contains information about the CPUs available for the user, disk space and effective queue length. The job queue describes the jobs submitted to the cluster, i.e., status, job id, certificate, and owner. An example of an MDS tree containing information about users and jobs queues is depicted on figure 3.2.

The information is gathered via simple LDAP queries, or through the NorduGrid Web Interface, located at http://www.nordugrid.org. This information is used to serve

[1] The EDG information model was also considered, because it was better at describing clusters, but there were doubts about its practical use.

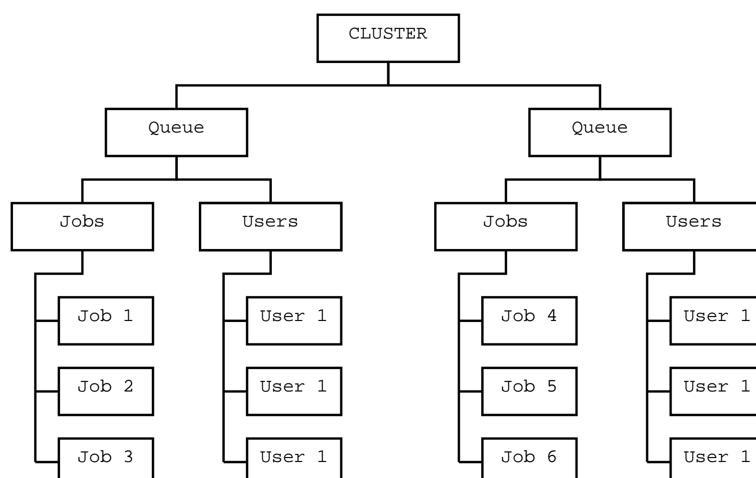

Figure 3.2: The organization of a LDAP MDS-subtree for a cluster, showing the queues of the cluster.

the necessary information to the brokering functions of the user interface. The requested information is generated locally on the resources, but it can optionally be cached for subsequent use. The local databases implements first level caching of the providers output and answers queries, making use of GRIS.

NorduGrid has an indexing service, used to maintain information indicies in the form dynamic lists of available resources. These lists is used to get the contact information for the resources in the grid. The Globus Toolkit provides higher level caching though the GIIS (for a description of the protocols mentioned see appendix B) protocol. But this function is not used in the NorduGrid Toolkit, where the indexing service consists of simple dynamic link catalogs. The main function of these lists is to reduce the overall load, and the indexing service is only used for storing contact information. The resource information is ordered, in a topology, according to their national and geographical locations. This structure is depicted on figure 3.3.

The last part of the information system is the soft state registration mechanism it is used by the local resources to register their contact information. Soft state is necessary because the amount of resources are not constant and thus no constant database of resources can exist. The resources must register themselves with the service as they appear on the grid, and they must subsequently keep registering themselves continually, otherwise the monitoring system purges the contact information.

Grid Manager

The Grid Manager runs on the front-end of a cluster where it controls the jobs and handles the interaction with the local resource management system. The Grid Manager replaces the Globus Resource Allocation Manager (GRAM). It provides job and data pre- and post-staging functionality that are not supported by GRAM. The Grid Manager is implemented as a layer above the Globus toolkit. Mainly because the Globus Toolkit did not meet the requirements, i.e., integrated support for Replica Catalogs, sharing of cached files among users,and staging functionality of data files.

The main responsibility of the grid manager is to process input and output data files.

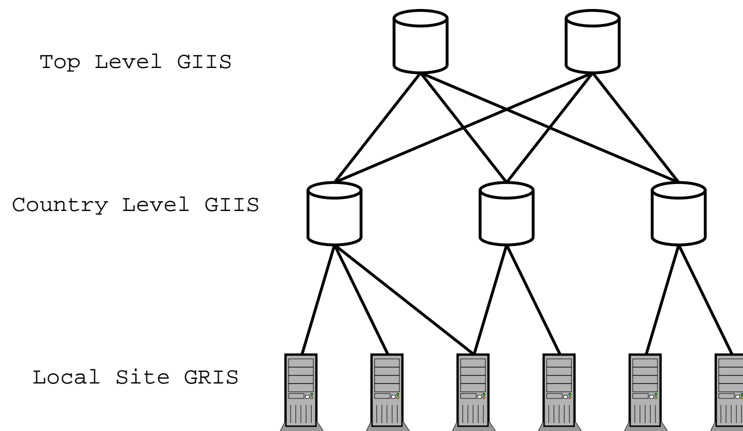

Figure 3.3: The structure of the GIIS topology in the NorduGrid Toolkit. The local information providers (GRIS) on the clusters registers with the country level indexing service (GIIS) which in turn registers the information with the top level indexing services.

When a job is submitted to a cluster, a session directory is created. This directory holds all the files associated with the job. The grid manager checks the session directories, at a certain interval, to see if new jobs have been uploaded. If new jobs has arrived the job description is parsed to see if any additional data is needed. It is the responsibility of the grid manager to gather all the data necessary for the job to be executed. This data can be uploaded by the user or downloaded from a storage element, by the grid manager. When needed data is downloaded, it is placed in the session directory for the job. Optionally any downloaded data can be registered with a replica catalog. Figure 3.4 shows the interaction between the grid manager and the cluster software.

When all input data is collected the grid manager submits the job to the LRMS running on the cluster[2]. Once submitted, the grid manager, periodically checks to see if the job has finished. When the job is finished, the grid manager collects the output data. The user is notified by email if this is stated in the job description, otherwise the job status can be monitored by using the web interface. Furthermore data can automatically be uploaded to a storage element and registered with a replica catalog. When a job is submitted to a cluster it can be in one of several states [18].

- **Accepted** - The job has been submitted and accepted, but no processing have been done.
- **Preparing** - The Grid Manager is collecting the data needed for the job to run.
- **Submitting** - The job is being submitted to the local batch system.
- **InLRMS** - The job is submitted to the Local Resource Management System. The status when in LRMS can be: queued or running.
- **Finishing** - The output data is being processed and optionally moved to a storage element or perhaps registered with a replica catalog.

[2]Currently the NorduGrid Toolkit only supports the batch systems Open PBS, Scalable PBS, and PBS Pro, but support for others are being planned.

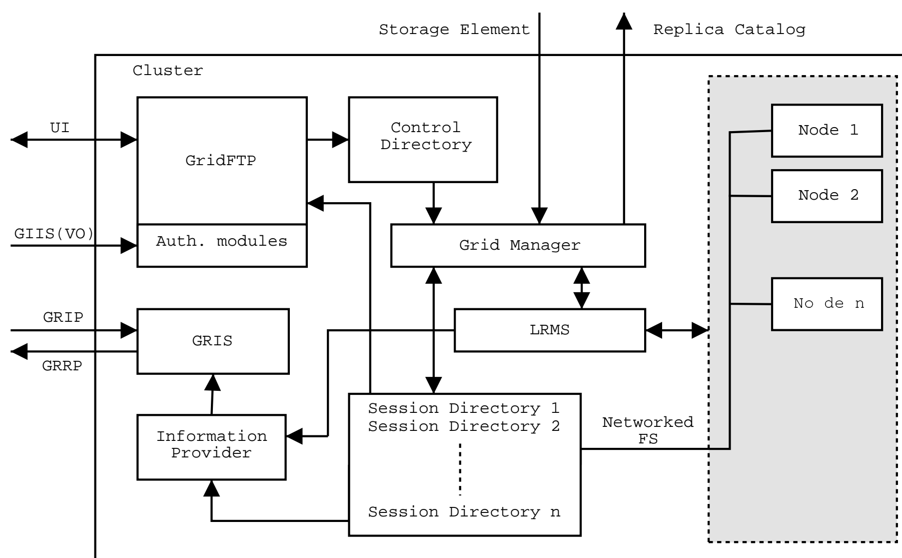

Figure 3.4: Closeup of the front end and nodes of a cluster, and how the middleware interacts. When the job description is uploaded by GridFTP, it is placed in the control directory. The grid manager reads the job description, an creates a session directory, and downloads any data needed. The job is then submitted to the LRMS and executed on the nodes, which have access to the session directory on a distributed file system. The information providers collects information, about the jobs, from the LRMS and session directories.

- **Finished** - The job is finished and the user can download the data. The data are deleted after a certain amount of time.

The Grid Manager handles jobs by creating a separate directory where it stores the input files. There is no single point where all jobs in the grid has to pass and thus no single point of failure.

GridFTP

GridFTP is used for data transfer within the grid, and for submitting jobs. It uses a special NorduGrid implementation of the GridFTP software, that differs from the GridFTP packaged with The Globus Toolkit (for an in depth explanation of the GridFTP protocol see appendix B). The server has been extended for better interoperability with NorduGrid, e.g., it can handle user access based on NorduGrid certificates. The main differences compared to Globus GridFTP are:

- Virtual directory tree: A virtual directory tree is configured per user.
- Access control is based on the user identity supplied with the NorduGrid user certificate.
- Local access is implemented through plug-ins. These come in two types: A local files system access plug-in and a job submission plug-in that has an interface for submission.

Computational Cluster

A cluster is the main computational element in the NorduGrid architecture, even though other computational resources can run on the grid, as log as they run a batch system. A cluster consist of several machines, where one is the front-end dividing the work between the other computational nodes. In NorduGrid all clusters run some flavor of Linux[3].

In order to add a cluster to the grid, the NorduGrid software has to be installed on the front end machine and have permission to interact with the local batch system. For further description of the installation see Appendix C. It is not necessary for the other nodes to run the NorduGrid software, but there must exist some form of distributed file system within the cluster in order for the nodes to get the job data. This goes a long way of making NorduGrid an add-on system and it respects local security and configuration policies.

Storage Element

A Storage Element is, as the name states, an element that stores data in the grid. A job description can specify that some data for the job should be fetched from a storage element. It is at present not fully developed in the NorduGrid software, and it is implemented as a GridFTP server. Either as the GridFTP server delivered as part of The Globus Toolkit or as the GridFTP server supplied by NorduGrid.

At the moment, a smart storage element is being developed. This type of storage element can take care of data replication, automatic registration of incoming content, automatic up and download of data, and failure recovery [17].

Replica Catalog

The replica catalog is used for registering and locating data sources. NorduGrid uses the replica catalog supplied by the Globus Toolkit, but with minor changes to improve functionality. The information contained in the replica catalog is primarily used and maintained by the grid managers, but it can also used by the user interface for the purpose of resource brokering. The replica catalog is based on OpenLDAP and is used without modifications, other than patching it to better cope with transferring of large files and adding the possibility to perform securely authenticated connections based on the Globus Security Infrastructure.

3.1.3 The Future of the NorduGrid Toolkit

The task list, in Appendix E, is a list of things that the developers of the NorduGrid toolkit would like to see implemented. Several of these tasks are already being developed while others are just suggestions. From this list it is obvious that there is still a lot of work to be done in order for the NorduGrid toolkit to be “complete”, even though, it is considered ready for production by the developers. It has been used with success in the first ATLAS data challenge, where it completed up to 15% of the total jobs completed, even though the NorduGrid data facilities only had 4.7% of the total CPU time in the data challenge [22]. One of the main lesson learned during DC1 was that a lot of time was spent babysitting jobs and resubmitting failed jobs to ensure that all the

[3]Support for other Unixes is underway.

DC1 jobs completed correctly. This was done manually by the developers and physicists. This leads to the conclusion that the NorduGrid Toolkit does not yet scale to the level needed by high energy physics. It also demonstrates the need for an automated production system for all the grids contributing to the ATLAS Data Challenge.

At the 6th NorduGrid Workshop, a talk was given by Brian Vinter about the future usage of NorduGrid. A discussion of how to get users to the embrace the toolkit was initiated. At the moment there are initiatives to get applications other that HPC applications to run on the toolkit. These are applications in the areas of biology, chemistry, and visualizations. Even if the applications are developed to run on the grid, one of the challenges NorduGrid faces, is the problem of getting users to adapt the middleware. There are several reasons for this:

- The middleware is not necessarily compatible with the users existing systems.
- At the moment the users are not getting anything from using the NorduGrid middleware, which they do not already have. And why install some middleware which could have an impact on the stability of their production system. A lightweight front end is being developed for non Linux/PBS systems to accommodate this problem.
- There is no real HPC resources available on the grid run by NGDF. Even though there are two former top 500 machines on the grid, inspection reveals that only a limited number of CPUs are available to NorduGrid.
- It is only possible to run the grid manager as a root user, and each job runs as the same local user; having an impact on both security and and secrecy.
- The uploading of data files is relatively difficult.
- Accounting, Accommodation, and Authentication, are stated as the three most important requirements in order to get new users to the toolkit. Accounting are still missing, so the grid is basically “paid” for by the resource owners who contribute it.

To get users interested in grid, the plan is to “gridify” some applications which could benefit from computational grids. Examples of applications being ported to the NorduGrid Toolkit are: Dalton a chemistry tool creating huge jobs. The Povray ray tracer as an example of something that is easy to explain to people not knowing anything about grid and HPC in general. Finally BLAST, a genome sequencing application, which has a lot of security requirements as it is dealing with datasets that is worth a lot of money and thus secrecy is a must. These examples is not only relevant in a NorduGrid context, but for most of the grid research today.

Chapter 4

Project Goal

In section 3.1.3, it was mentioned that one of the apparent needs was a automated production management system. This need came from the experiences of ATLAS DC1, where a lot of job babysitting was done, especially resubmitting jobs [22]. A solution to this, would be a system with the ability to resubmit jobs automatically, saving the user from monitoring the job and resubmitting it manually. It is easy to imagine that such a system could also be extended to be able move jobs that has not yet started to execute, by canceling them and submitting them once again to another cluster. It could even "move" jobs by terminating a currently running job and submitting it to another computing element, thereby enabling some forms of user side scheduling.

A case where job resubmission would be solution are, if a job fails, due using to much CPU time, lack of runtime environments, or OS peculiars on the cluster. Another case where resubmission would be useful, is that if a job has been submitted to a cluster, but are in queue[1], while local jobs arrive, and the job is pushed back into the queue, thereby extending the amount of time, before execution starts. These problems are symptoms of the dynamic nature of grids and their heterogeneity. This cannot be changed. Instead one must try to recover from the job failure, by executing the job elsewhere. Therefore it would be desirable to have the NorduGrid Toolkit to support automatic job resubmitting.

We have chosen to focus on automatic job resubmission, in this project. Developing automatic job resubmission would the first steps in creating a production system as previously described. Basically the system should have the ability to automatically submit a job, and have "handlers" for certain different scenarios, acting on behalf of the user, where resubmission is one of the key functionalities.

Creating a system that could handle automatic job resubmission of failed jobs, would serve as a prototype for building such a production system. Therefore we have chosen to develop a system that will enable automatic job resubmission on failure. This problem still contains interesting problems like, what should submit the jobs, and how failure can be handled automatically. Is also catches the essence, i.e., it makes automatic submission possible and uses it for resubmitting failed jobs.

The solution for creating automatic job resubmission in the NorduGrid Toolkit is to find models for doing resubmission, analyze them to find the most suited, and implement it in the NorduGrid Toolkit. The definition of our goal is the following.

[1] Remember that clusters often use batch systems, and that a job will often spend time in queue before starting.

> *"Find a suitable model for automatic job submission in the Nordugrid Toolkit, implement it; and make the implementation do automatic job resubmission in the case of a job failure."*

Since we are extending an existing project, the current design and code base must be respected, meaning that the system cannot be designed from scratch. Therefore we have focused on integration and coherency with NorduGrid Toolkit.

The first part of our problem is to find a suited model for doing automatic job resubmission in the NorduGrid Toolkit, which is exactly what will be handled in the next chapter.

Chapter 5

Job Resubmission

This chapter continues the discussion of automatic job resubmission. It starts by defining what job resubmission is, and how scheduling relates to it. Hereafter is describes different resubmission models, along with their strengths and drawbacks. Then the different model are discussed in a larger context, and one is chosen for implementation. Finally it describes how the job flow changes, after introducing the model into the NorduGrid Toolkit.

5.1 Job Resubmission and scheduling

In the strictest sense job resubmission is the ability to submit a previously submitted job, and nothing more. If the resubmission is to happen automatically, it should also be decided when or if the resubmission must happen and how to submit it, (e.g. should the previously used computing element be used again) since these actions is normally decided by the user. These decisions are scheduling of job resubmission.

For this scheduling to be useful it requires that a proper infrastructure for job resubmission exist, so that it is possible to resubmit jobs. Even though job resubmission and the scheduling of it could be viewed as two distinct topics, they intermix heavily. Therefore the two topics are described together as one, in the following.

As discussed in chapter 4, reasons for resubmitting jobs could be job failure or extended waiting time for job starting. Other reasons for doing job resubmission could be that the computing element was being revoked from the grid, or that the cluster crashed or disappeared from the grid. Other resources could appear on the grid which was better suited for the job. Some of the last reasons also include very non-trivial scheduling decisions, about when a job should be resubmitted.

All of the above reasons reflects the prime reasons for doing job resubmission: We want the job solved, and we want it solved as soon as possible. Wanting the job solved usually means resubmitting a job when it has failed, where wanting it solved as fast as possible includes much monitoring and doing some difficult scheduling decisions.

5.2 Models for Job Resubmission

In this section we describe different models for doing job resubmissions. The strengths and weaknesses of each models are discussed. Even though the models are abstract

they are discussed with respect to the NorduGrid Toolkit, since it is in this software that job resubmission is to be implemented in. Since we focus on making automatic resubmission we have by intention left out the "Let the user resubmit the job" model.

5.2.1 Letting the Computing Element Resubmit the Job

If a cluster cannot execute a job for some reason, or execute it early enough, it could submit the job to another cluster. This model is very convenient for the user since the job is automatically resubmitted to another computing element, without needing any user side intervention. The model is depicted on figure 5.3. However this model breaks the security model in the NorduGrid Toolkit, since the cluster is allocating resources on behalf of the user. The model is depicted on figure 5.1.

One could imagine that a cluster which has low accounting prices, would cancel all their incoming jobs and submit them to a far more expensive cluster. Having such cluster in a VO is unlikely though; however, having clusters allocating resources on behalf of users is not a good idea, since it breaks the existing security model.

Doing job resubmission on the server side should in general be avoided, since it requires delegation to the computing element to allow it to submit job on behalf of the user. This means that job resubmission should be done on the client side, which is where the resubmission occurs in the following models.

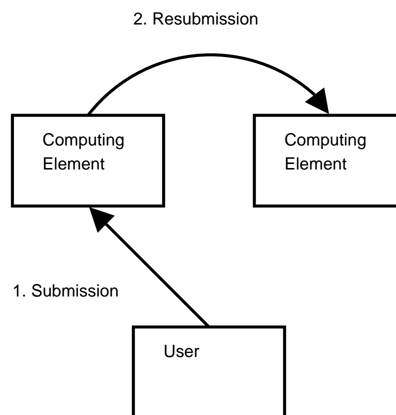

Figure 5.1: Resubmission by Computing Element.

5.2.2 Adding Job Redundancy

If the focus is on not having jobs fail or having jobs start as soon as possible, the job could be submitted to several computing elements at the same time. The model is illustrated on figure 5.2. It raises several issues: When one of the jobs has started and have been running for a number of minutes (to make sure that the job does not start by failing) the other jobs should be canceled to minimize wasted cycles, minimizing load and avoid being accounted for the same job on several clusters.

This cancellation should be done from the client, preferably without user interaction, since it could take some time before any of the jobs starts. This would require a daemon monitoring the jobs from the client machine, which would have access to the proxy certificate of the user , would cancel the job. This model does not require any

delegation, since it runs local on the users system, thereby having access to the proxy certificate.

This daemon would often have to do some hard scheduling problems, like "OK, the job have started on this slow machine, but could soon be started on this super fast machine, what action shall be taken?". Additionally jobs does not always fail when starting, but could fail anytime. This is especially a problem when a job requires more time to run, than expected and the cluster does not allow the job to run for this amount time. There is also a minor problem if the client crashes after job submission, since no jobs would be canceled If job submission redundancy would become the norm, several of the computing elements in a grid, could very well start accounting for submission, and not just used cycles, which would add to the cost when using redundancy, and users are usually not interested in paying more than necessary.

A side effect of job redundancy would be that the load on the grid becomes artificially high due to the same job being in different queues. This would also make scheduling more difficult, since it makes it harder to estimate where the job could be executed as soon as possible.

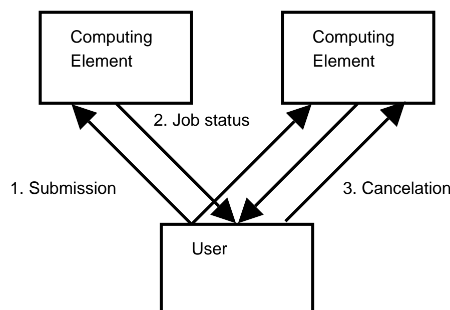

Figure 5.2: Job Redundancy when Submitting.

5.2.3 Letting a Daemon Resubmit the Job

In the previous model a daemon for canceling jobs was described. Instead of using this daemon for canceling jobs, it could instead resubmit any jobs that fails. This means that only failed jobs will be attempted to executed more than once.

Such a daemon could be created along with the users proxy certificate, so the user could use the daemon without hardly noticing. This model incorporates nicely into the existing security model in the NorduGrid Toolkit, since no delegation is necessary. The daemon just uses the proxy certificate which must also normally be created when submitting jobs. A client side requirement is introduced by using this daemon though. For resubmission to work, the daemon must be running. This means that the user must have access to a host which can remain turned on while the job is running.

5.3 Choosing a Model

In the previous section a list of models for automatic resubmission was presented. Each of these models were described, including their strengths and weaknesses. In this section we will describe and discuss the criteria for selecting a model, then discuss the

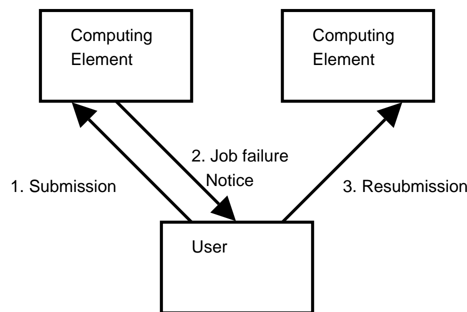

Figure 5.3: Letting a Daemon Resubmit the Job.

models further, and compare them to each other. Finally a model will be selected for our implementation.

When choosing a model for automatic job resubmission, several aspects must be taken into account. One is that we cannot design the implementation from scratch, but must integrate with an existing code base. This means we should change as little in the code as possible, while still using its functionality to avoid code duplication. Besides integration, the coherency of the existing code must also be respected, i.e., one should respect the original design, solve problems like it is already done.

Another aspect is the aesthetic of the model. Often some model will seem more right that other, even though their is no clear argument for it. This "feeling" about the model should not be ignored, since it reflects the programmers experience, about how the model will work in real life, how the model will be to implement and maintain[1] Yet another aspect it the performance of the model. Performance is often ambiguously defined, and is usually context dependent. In a grid context performance is often meant as scalability, network performance, and how quick the job will be finished.

We start by discussing the first model presented, i.e., letting the computing element resubmit the job. This model changes some code, but not a necessarily a lot. The code however must be in the grid manager which usually runs as a privileged user[2], where the amount of code should be minimized. Also the coherency in the model is quite bad since it mixes the user interface part of the toolkit with the grid manager, making the grid manager act as a user. The delegation that is necessary for the scheme to work, is also bad for overall coherency in the architecture.

The second model, adding job redundancy, works on the client side, meaning that it no extra code is run as a privileged user. Such a solution could be implemented in the user interface without changing much, while still being able to reuse a lot of code. Sending the same job out to several computing elements at the same time, seems wrong, however this is mostly an aesthetic view. Perhaps the worst thing about job redundancy is that it makes the behavior of the computing elements much harder to predict for the scheduler, since the load appears artificially high. This makes scheduling harder to do, since the queues on the server does not reflect the actual queue. This makes the model less coherent, since it changes how the load is reflected. The accounting problems described are constructed[3], but are not unrealistic.

[1]Although this can also backlash due to pride, religion, etc.

[2]It should be possible to run the grid manager as a non root user, but it is not yet the default, and problems must be expected.

[3]Accounting does not yet exist in the NorduGrid Toolkit, but work is underway.

The last model described, letting a daemon resubmit the job, also works on the client side. Like the previous it integrates well into the user interface. Since jobs are only resubmitted in the case of failure, it acts much like a user - although a user would probably investigate the cause of failure first. For this model to work it requires that the user has created a proxy certificate, since this is a requirement for job submission. Such a certificate is only valid for twelve hours pr. default, meaning that the daemon will usually only work this period of time. The user can prolong the valid time of the proxy certificate or create a certificate valid for longer periods of time if needed though.

For resubmission model in our implementation we selected the last described: A daemon which can automatically resubmit a failed job on behalf of the user. The reason for this is that such a solution fits nicely into the existing security model and it can be created with only some small changes in the infrastructure. It also seems more correct to only resubmit the job, if it fails and instead of adding redundancy.

5.3.1 Model issues

Using this model also raises some issues. An apparent one is that the daemon must get the xrsl file content and arguments from the submission, since the xrsl file send to the server has been stripped for some information, e.g., cluster choice. This means that some changes must be done in the user interface part of the NorduGrid Toolkit, to make the daemon aware of new jobs and any extra command line choices.

Another issue is how the daemon should become aware of job failures. There are two general methods to consider: Push and Pull. Using push means that the daemon must be told that the job has failed. If a job has failed, the grid manager could contact the daemon. This means that the daemon must supply a method to contacting it, e.g., an open socket on a routable IP address. This introduces an extra requirement for using the daemon compared to using the normal interface for submitting jobs. This indicates that a push model is not what we are looking for, since the daemon must have a way of being contacted, imposing a requirement, which we may not always be able to fulfill. This method would also mean that the grid manager should be changed, giving a bad integration due to new network requirements and changes in the grid manager. Fortunately there is a better way.

Instead of a push model, a pull model should be used, meaning that the daemon should fetch the status of the job. This should be done at regular intervals since the status of a job changes over time (until it has finished). The place to check for job status should be the MDS information system, since this already contains the information about the current status of a job, giving values like “RUNNING” and “FINISHED”, as described in 3.1.2. In the case of job failure the information also provides information about the job failure, e.g., error code of the job. Since the MDS information system is already used when submitting a job, and that the information needed is already there, no new requirements are imposed on the client using a pull model. This is also more coherent with the existing design, since job information is pulled through MDS. It is possible that a push model could give more information about why the job failed. However using such a model imposes new requirements, and would require changes in the grid manager, gives that it should not be used.

5.4 New Job Submission Flow

Introducing this model into the NorduGrid Toolkit changes the flow of how a job is submitted. On figure 5.4 the old job flow is depicted. It is seen how ngsub queries a GIIS for a resource list. Each resource in then queried, where after ngsub finds a suitable cluster to submit to. Finally the job is submitted to cluster. The arrows on this picture and the next displays information flow.

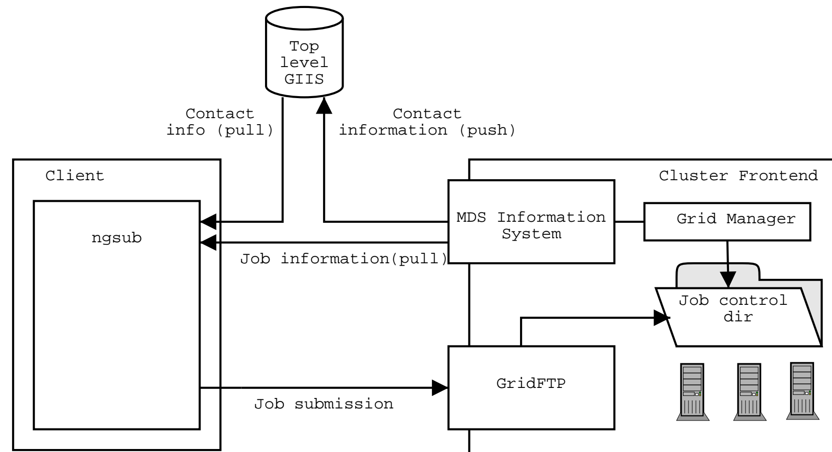

Figure 5.4: Old job flow.

With the daemon introduced the job flow changes slightly. The new flow is illustrated on figure 5.5. The exact same queries are performed, and from an external point of view nothing is changed. Internal in the client, the structure has changed. Instead of using ngsub, a submitter which sends the xrsl job file to the NG Proxy daemon. The daemon now acts as the client program, instead of the program invoked by the user. The daemon does the same actions as ngsub, i.e., it contacts a GIIS, queries the resources, does brokering and submits the job on behalf of the user. The daemon keeps a list of the current jobs in an internal data structure, so that it can query for status of jobs, and resubmit them if necessary.

5.5 Summary

By selecting a model that is coherent with the NorduGrid Toolkit, we should be able to reuse a large part of the existing code, and avoid having to change or rewrite components essential to the toolkit. Another advantage of using a model running on locally on the client should enable us to test it more easily, on the “real” grid. Which would not have been possible if we had chosen a model which required server side changes. Having selected a model, we are now ready to move on to the implementation, which is what the next chapter is about.

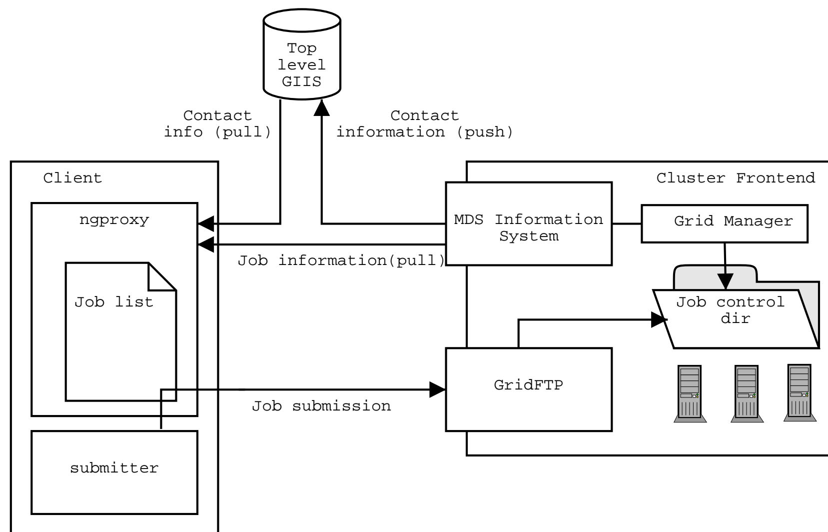

Figure 5.5: New job flow.

Chapter 6

NG Proxy

For the name of our resubmissions daemon, we selected NG Proxy[1]. NG for NorduGrid, and Proxy, since the daemon stands between the user the and the grid, where it acts on behalf of the user. The primary goal of NG Proxy is to add a way of doing automatic resubmission, and resubmit a failed job automatically. As stated in 4, this was our goal and is a prerequisite for doing smarter handling of failed jobs.

This chapter describes what NG Proxy is and what it is able to do. Secondly it describes in details how NG Proxy is constructed and how it works. Furthermore it is explained how NG Proxy integrates with the rest of the Nordugrid Toolkit.

NG Proxy is able to automatically resubmit jobs that have failed, however it does not handle it in an "intelligent" way. The job is simply submitted to another cluster, than the one on which the job failed to finish on. It does not try to guess the reason for job failure, or whether the job can finish successfully on another cluster. This means that the errors that can be recovered will usually be related to runtime environments, differences in operating systems, or time constraints.

NG Proxy will act on behalf of the user, having to the proxy certificate of the user. It will submit jobs the usual way by querying the MDS information system, and submitting through GridFTP, just like `ngsub`. Furthermore it will be monitoring the state of the jobs continuously, by pulling job information from MDS information service.

In the next section the construction and the inner workings of NG Proxy will be explained.

6.1 Implementation

NG Proxy is implemented as a daemon that runs in the background on the client host on behalf of the user. The user must submit jobs through this daemon to make it aware of the existence of new jobs. The reason for doing it this way, was described in section 5.2, and was good coherency with the existing NorduGrid architecture and no delegation was necessary.

Pulling a job list from MDS would complicate things though; if the user is running NG Proxy on several machines (there is nothing preventing this), they would have to coordinate how to handle job failures. Getting access to the `xrsl` file would also

[1]This might change when the Nordugrid Toolkit changes name

be troublesome. However this scheme would also mean that the daemon would be decentralized, and thereby more failsafe. Implementing this decentralization is major task though, and is out of the scope of our problem definition and this project.

Instead of using `ngsub` for job submission, the user sends the xrsl file through a Unix Domain Socket (UDS), located at `${HOME}/.ngproxy/xrsl`. A UDS works much like a regular TCP socket, except that it does not listen on a port for connections, but on a local file. This file has the Unix permissions 600, meaning that only the user who started the daemon, can read and write to it, giving a simple, yet effective way of controlling who submits jobs to the daemon, while still being secure and well proved. This security scheme is exactly as strong as the existing one, where the proxy certificate is guarded by the exact same mechanism. Using a socket to communicate, also makes it easier for NG Proxy to grow, since the communication could also happen over over the network, instead of local file system.

The daemon has two major parts, one waiting for new connections on the socket and one pulling info about the current jobs from MDS services of the clusters. This lead to the design of having the main process and an thread. The main process handles what is called the "main loop" which updates information on the jobs. If the state of the job checked has been updated to "FINISHED", it will be checked if there was any errors during the job run. If not, the job will removed from the list of current jobs. If an error is present, the daemon will submit the job again to another cluster, using the existing API in the NorduGrid Toolkit user interface.

The thread listens for new connections on the socket. When a new connection is created, the thread reads everything from the socket and creates an xrsl object from this. A sanity check is performed on the object to ensure that the string received is a valid xRSL job specification. If not, nothing will be done. After receiving the xrsl object is put into a vector. The main loop takes the xrsl object out of the vector and submits it. The vector access is of course done with mutual exclusion in mind.

After the job has been submitted, an object of the type `JobStatus` is instantiated. This object contains the xrsl object so that it can be reused if the job fails. Furthermore it contains the number of resubmission attempts, since a job should only be resubmitted a finite number of times. It also contains the current job id and a list of the clusters where the job has failed on, so that these can be excluded from any future resubmission. This object is updated accordingly when a job gets resubmitted, and deleted when a job finishes correctly. An overview over NG Proxy and its parts is depicted on figure 6.1. The two parts of the program has been illustrated by surrounding them with dotted line boxes.

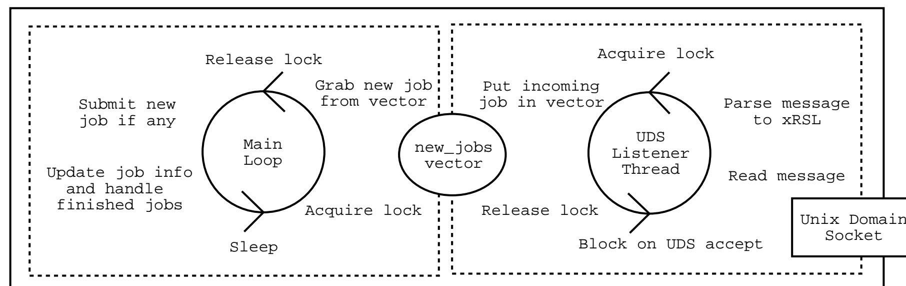

Figure 6.1: NG Proxy overview, showing the two major parts, and their shared data structure.

The resubmission scheduling in NG Proxy is relatively simple. If a job has failed a decision must be taken: Should the job be resubmitted? In the current implementation, a job is given a maximum number of resubmissions attempts. If this number has not yet been reached the job is resubmitted, otherwise the job is given up. If the job is to be resubmitted a second decision must be taken, i.e., which cluster should the job be resubmitted to. On a resubmission NG Proxy, submits the job like any ordinary job submission except that the previous cluster is set the `oldcluster` parameter in the `JobSubmission` function. Unfortunately the `oldcluster` parameter is a string, and not a vector of strings, so it currently not possible to exclude more than one cluster from a submission. Converting the `JobSubmission` function to take a vector would improve this, but one of our focuses has been not to change existing code in the NorduGrid Toolkit.

NG Proxy reports every action to a log file. The log file is called `ngproxy.log` and is located in the `.ngproxy` directory of the user. Every action which changes some internal structure in the daemon is logged to this file. Examples of such actions could be new job submission, resubmission, or that a job finished correctly.

During development the various subparts of NG Proxy was also tested, and finally the daemon was tested, by submitting good and bad jobs to clusters in the NorduGrid testbed, and monitoring the behavior of the daemon. Some problems concerning management of internal data structures was found, but was quickly corrected. Thereafter no problems where found and NG Proxy appears to be working correctly. It resubmits any failed jobs, but only a limited number of times, where after it gets removed from the data structure. Successfully finished job also gets removed, so no queries about them will be made.

6.2 Integration

As stated in chapter 4 the current design and code base should be respected, and our implementation should integrate well into and be coherent with the existing Nordugrid Software. By integration we mean how well it fits into the code base and how it changes it; by coherency we mean how well it "fits" together with the existing implementation, e.g., is the problems solved in a consistently manner. Another of the criteria for the model, was whether it adds any new requirements for running the program, compared to other similar components in the software.

The changes that must be made in the code base for making NG Proxy run are very small. Only two small changes are required. The first is the addition of a method in `MdsQuery`, which returns the error string of a job. The method body consist of one statement. The other change is a slight change to the function `JobSubmission`, where an additional argument has been added. This extra argument is a string pointer which is set to the job id (which is normally only outputted to std out and a file). The argument defaults to NULL, if not set, and is not set by the function then. This allows existing code to continue working without any modification. Currently NG Proxy also requires changes to the makefile, since NG Proxy does not yet follow the plug in structure in the user interface. It is prepared for it though.

NG Proxy uses the same code for submitting jobs and for pulling info through MDS, as the rest of the user interface, e.g., ngsub. This means that NG Proxy does not impose any new system requirements for using it, i.e., if you can use ngsub, you can also use NG Proxy.

6.3 Summary

In this chapter we have presented the implementation of our resubmission daemon, which is called NG Proxy. Our expectations to the model, has been fulfilled since the daemon integrates well into the NorduGrid toolkit, is coherent to it, since a lot of code has been reused for submitting jobs and for querying the information system. The design for the daemon, having two different parts for each their functions have also proved to be a good choice.

NG Proxy is still a work in progress, and in appendix D a to do list for making NG Proxy better is presented. The list does not contain future features; these are described in the next chapter.

Chapter 7

Future Work

This chapter presents some new ideas for improving the functionality of NG Proxy. The suggestions in this chapter, is different than the ones in appendix D, which merely presented small improvements, where as the ones presented in this chapter are on higher level, looking at future functionality and possibilites.

Each users options for NG Proxy is likely to differ, and starting the daemon with these options every time will become an annoyance. Therefore NG Proxy should be able to take these options from a configuration file, containing the default number of resubmissions attempts and other ordinary user interface options.

Besides a configuration file, a file containing rules for resubmissions would be desirable as well. Such a rule file would contain specifications for what to do on certain types of failure, e.g., if a certain binary was not found on the system, the job should be submitted to a type of system where this binary was known to exist. This could be abstracted even further for handling other things than just resubmission, i.e., if this happens, take this or these actions. Such a rule file, would be quite a task to create, but would lead to more "intelligent" handling of a failed jobs.

Currently no real protocol exists between the submitter and NG Proxy. If the submitter must send options along to the daemon as mentioned in appendix D. A protocol should therefore be developed. However, developing a protocol can be a daunting task, so an existing framework, e.g., the Jabber protocol [14], should be used.

When submitting a job, it is possible to specify a storage element on which the resulting files should be transferred to after the job has completed. However, it is not possible to get the files transferred to the users local machine (unless it happens to be a storage element, which is rather unlikely). The only way for the user to this is to fetch them via a GridFTP client. Instead of this, NG Proxy could on a successful job completion fetch the output files to client host on which it is running. Since all that is necessary to login over GridFTP is the proxy certificate and that NG Proxy already has access to this, no extra delegation of credentials to NG Proxy is needed.

As mentioned in 6.1, each major action that NG Proxy does is written to a log file. The end user, however, is unlikely to inspect this on regular intervals, which means that the user should be notified by some other means. A way of doing this, is to send the user a mail, that a job has been completed, or it has been resubmitted due to failure. Such functionality could be integrated into the daemon itself, or as a separate program, which parses the log file at regular intervals.

Running several NG Proxys on different hosts are possible, but they know nothing of each other, and they only care about the jobs submitted through them. This means if

an NG Proxy disappears, the jobs which it monitors will not get resubmitted, if any of them fail, even though other NG Proxys are running. If the daemons were able to share information, about the jobs and cooperate about job resubmissions, the single point of failure would be removed from using NG Proxy.

In [22] a wish for an automatic production management system was expressed. NG Proxy could be the first step in such a direction. By implementing more of the features described in this chapter, like rule file and fetching of output files, would bring it another step forward. However creating such a system is a big undertaking, but could improve the reliability and usuability of the NorduGrid Toolkit.

Chapter 8

Conclusion

This report documents this process and the development of NG Proxy, a daemon for automatic resubmission of jobs in the NorduGrid Toolkit. One of the purposes of this project has been to uncover the concept of grid, to locate and solve problems found. A significant amount of time was spent understanding the concept, technologies, and toolkits about grid, and the possibilities and limitations that arises. We chose to work with the NorduGrid toolkit, because it is the grid technology employed by the Danish Center for Grid Computing.

We started by examining the concept of grids, along with expectations of what a grid should be able to do, and why grids are needed. Furthermore we looked into the concept of virtual organization, and how they enable resource sharing. Hereafter the focus shifted toward different grid architectures. A general model for grid computing was investigated, along with the key elements and requirements for a grid. Going from abstract to concrete, the grid projects Globus 2, Globus 3 and Legion was explained. NorduGrid and EDG which produces grid solutions based on existing grid middlewares where also presented. Looking at the different approaches to solving the issues gave us valuable insight into problems concerning grids, since each of the projects solves the problems in different ways and with different outcome.

The organization of NorduGrid was described, along with an explanation of why we chose NorduGrid. From here the point of view changed to a more technical one, where the NorduGrid Toolkit was presented, and the individual parts of the toolkit was described. After having presented the NorduGrid project, the focus shifted toward defining the goal of the project. This included looking at the relevant problems, concerning the NorduGrid Toolkit.

A brief discussion with Brian Vinter, indicated the need for job resubmission. This need was confirmed during our participation in the 6th NorduGrid Workshop, where the need for an automatic production management system was expressed, herein a way of resubmitting jobs, as a large portion of time during the first ATLAS data challenge, was used for babysitting jobs and resubmitting them. Also when the number jobs grew, manual resubmission becomes increasingly tedious. Therefore we decided to focus on automatic job resubmission, and try to accommodate this problem.

We started looking at models for automatic job resubmission. Finding such one turned out to be one of the challenges in the project. Three different models where discussed, and their strengths and weaknesses where uncovered. The models were compared and one was chosen. The selected model was where a daemon on the client side would submit the job, monitor it, and take action accordingly if the job failed.

Issues with the model was also discussed, and it was presented how the job flow would change with such a model.

To get hands on experience with the NorduGrid Toolkit we started by installing it on a single machine, which was later to be used as a front end for the cluster located at the Department of Computer Science at Aalborg University. The installation proved to be difficult because we where installing it on Debian and it was originally developed to work with Red Hat or another RPM based distribution. Another problem was that we had to install the NorduGrid Toolkit from source since we needed to be able to compile it, in order to do any development. Luckily and thanks to the people on the nordugrid mailing lists we got it installed, and gained a better and more practical knowledge about the NorduGrid Toolkit, than by reading about. We also discovered several discrepancies between the articles about the toolkit, and the actual implementation. This later helped us, when developing NG Proxy.

Having found a suitable model for the implementation, and installed NorduGrid, we started reading the source code for the NorduGrid Toolkit. This was quite an interesting experience as the quality of the code was varying quality, mainly due lack of comments and documentation of internals.

After getting familiar with the code and the design of the NorduGrid Toolkit, the implementation was relatively straight forward. No major problems where found during testing. One of the reasons for this was the time invested in getting acquainted the code, and having installed the toolkit, had substantially increased our understanding of the NorduGrid Toolkit.

One of the main problems with the software developed is the fact that it runs on a single machine, leaving a single point of failure in the user side of the job control. This can be solved by expanding NG Proxy, with the possibility to run several proxies working together.

By creating a NG Proxy, we have demonstrated that it is possible to create another and more advanced user interface to the NorduGrid Toolkit, automating job resubmission.

This avenue should be explored further giving the user possibility of having more advanced, intelligent, and diverse control over the jobs running on the grid.

As discussed in chapter 7, NG Proxy should in the future be able to provide the user with advanced functionality when interacting with the grid, while concealing and automating trivial tasks. Examples of advanced functionality, could be a more rich interface to the grid, and ability to schedule jobs, while the retrieval of output data would happen automatically. This serves two purposes. The automation and hiding of trivial task would make the grid accessible to a larger set of users. More advanced functionality gives users with special needs, the ability to tailor the use of the grid to his or her need, thus befitting more from using NorduGrid.

Getting NG Proxy into a state, where it could provide functionality described above, would require substantial development. But none the less it is a feasible task for a later project.

In conclusion, experiences from the ATLAS DC1, have demonstrated the need for a production management system in the NorduGrid Toolkit. A part of such a production system, is a general unspecified form of job submission, with support for various handlers. By choosing a substantial example of such a handler, the resubmission of failed jobs, we have demonstrated that our model for resubmission does in fact work. This allows for further work to be done, developing handlers for cases where automatic job submission is needed. It should be possible to develop NG Proxy into such production system, or to be an important component in one.

Appendix A

Using NG Proxy

This appendix explains how NG Proxy is used. To use NG Proxy, one must patch the NorduGrid Toolkit code with the ngproxy patch. Instructions for doing so, should be included along with the patch. After compiling and installing, one should have a new binary called `ngproxy`, which is the command for starting NG Proxy. Before starting though, the proxy certificate must be created, by:

```
grid-proxy-init
```

Where after NG Proxy can be started by:

```
ngproxy
```

And NG Proxy should be running. For submitting jobs to NG Proxy, the xrsl job description file must be copied to the xrsl UDS in the users `.ngproxy` directory. For this we use a program called `uds_sendfile`. The code for this program is included in the NG Proxy tarball, and can be compiled using:

```
gcc uds_sendfile.cpp -o uds_sendfile
```

Here after a binary called `uds_sendfile` should exist in the current directory. The binary is used like this:

```
uds_sendfile myjob.xrsl ${HOME}/.ngproxy/xrsl
```

The contents of the `myjob.xrsl` file will be copied to the UDS. Hereafter NG Proxy will submit the job and start monitoring it, as described in 6.1. If one wishes to follow the actions of NG Proxy, one can follow the output of the log file by:

```
tail -f ${HOME}/.ngproxy/ngproxy.log
```

Which outputs the last part of the log file, and follows it, i.e., outputs any appended data to the file, to the terminal.

Appendix B

Grid Protocols and Components

The purpose of this appendix is to provide a short overview of the protocols / components used in the Globus Toolkit version 2 and 3, and the Nordugrid software.

B.1 GRAM

The Grid Resource Access Manager is the lowest level of the Globus Toolkit resource management architecture. GRAM is responsible for:

- Parsing and processing the Resource Specification Language (RSL) specifications that outline job requests. The request specifies resource selection, job process creation, and job control. This is accomplished by either denying the request or creating one or more processes (jobs) to satisfy the request.
- Enabling remote monitoring and managing of jobs already created.

However the NorduGrid does not make use of GRAM as its resource manager (see section 3.1.2), and will therefore not be described in further details.

References: [31, 5]

B.2 GRIP

Grid Resource Information Protocol is used to obtain information from a provider about a resource. Since an information provider in Globus can hold information about several resources the GRIP supports both discovery and inquiry. Discovery is basically a search on a provider to get the resources that match a given criteria. Inquiry is a lookup of information for a given resource. GRIP is based on LDAP as the protocol for GRIP.
References: [5]

B.3 MDS

The *Monitoring and Discovery Service* (MDS) from the Globus Toolkit is an extensible framework for creating grid information systems. MDS is based on standard OpenLDAP and uses schemas to describe the information model. A MDS information system consists of the following elements.

- An information model, described as a LDAP schema.
- Local information providers.
- Local databases.
- Soft registration mechanisms.
- Information indicies.

B.4 GRRP

The Grid Resource Registration Protocol complements the functionality delivered by GRIP. The GRRP supplies a notification mechanism, that allows a service component to push simple information about its existence to another element of the information service architecture. This information is used but by an information provider to notify aggregate directories of its availability, either for indexing or to invite an information provider to join a VO.

It is implemented through a soft-state protocol, meaning that states established at remote notifications can be discarded. Therefor the service components must refresh the information by a stream of subsequent notifications.

The message send by a service provider contains an URL where service can be reached, the name of the service provider, the type of message and a time stamp which determines for how long the information should be considered valid.

The protocol is designed to run over unreliable transports but it does not define the underlying transport. Therefore it can be used with both reliable and unreliable transports. This also has the implication that the protocol implements an unreliable failure detector that cannot distinguish between resource and network failure.

References: [5]

B.5 GRIS

The Grid Resources Information Service is a standard, configurable information provider framework. It is implemented as an OpenLDAP server that can be customized by plugging in different information sources. In Globus 2 information sources for static host information (architecture, number of processors, etc.) and dynamic information (queues, load, etc.), storage (Space available, total space etc.) and network related information (bandwidth, latency etc.).

GRIS handles requests from GRIP it authenticates them and dispatches them to the local information providers. The communication with the information providers is done via an API, that has two forms, either shell scripts or pluggable modules. Each providers results can be cached for a configurable period of time, to reduce the number of provider invocations. The caching period is determined by a TTL specified on a per provider basis. All results returned by the provider is filtered by the GRIS to match the clients search. This is not for performance, but to ensure the protocol's search semantics are implemented correctly.

References: [5]

B.6 GIIS

The Grid Index Information Service is back end for the registries developed by Globus. It is a standard configurable information provider framework and provides a hierarchical structure of resources and virtual organizations in the grid. The indexing service accepts GRRP messages from child GRIS and GIIS sources and merges these information sources into an unified information space.

The GIIS consists of three major components: GRRP handling, pluggable index construction and pluggable search handling. It is implemented as a LDAP back end and uses the same API as uses for interfacing GRIS to information sources. The GRRP messages delivered to the LDAP front end is decoded and the backed constructs GIIS indicies, i.e. a list of active providers, though this construction can be more complex.

GIIS is configured so GRRP can be used for both registration and invitation. A GRIS joins a VO by registering itself with a directory. When a GRIS is invited by use of GRRP, it must explicit use the GRRP for registering itself with the directory upon receiving the invitation.

In a performance perspective caching of data within GIIS is preferable, but in the absence of delegation only resources with anonymous access can be cached, otherwise the client is referred by use of standard LDAP referral.

References: [5]

B.7 GridFTP

The protocol used for transferring data files in grids. It is based on the widely used FTP[6] protocol. It extends this protocol, with several new features, making it more suitable for grid environments. These extensions are:

- **Parallel data transferring** Fetching data over multiple TCP streams will often improve bandwidth utilization, even when fetching from a single source. Multiple sources for the same data (striping)is also supported.
- **Negotiation of TCP window size** GridFTP can auto-negotiation the size of the TCP window size, since setting it manually is often prone to errors. Though it does support setting the window size manually.
- **Third party control** Since based on FTP, GridFTP supports transferring data from one site to another while controlling the transfer from a third site.
- **Partial transfers** GridFTP support fetching partial files. This comes in handy when dealing with large files where only a part of the dataset is needed.
- **Recovery mechanisms** Handling stopped transfers, failures and other crashes, are not defined in the FTP protocol and must be dealt with manually. GridFTP allows the user to specify fail over servers, number of restart attempts and other failure recovery options.
- **Security** GridFTP support anonymous and GSS login, using X.509 certificates. Transferring data can happen both encrypted and unencrypted.

References: [36]

B.8 GSI

GSI is not a protocol, but in infrastructure within the Globus toolkit. It stands for Grid Security Infrastructure, and provides a basic security interface for grid protocols and applications to use.

Below GSI sits the Generic Security Services, (or GSS-API), which provides a uniform way of using specific security technology, making it transparent for the user. The GSS-API can use plain text (mostly for testing), X.509 certificates and kerberos for authorization and authentication. Other protocols can then use the GSS-API to access resources in a uniform and secure way.

References:[13]

B.9 Nexus

The main communication library in the Globus Toolkit 2 is based on the Nexus library[12]. The Nexus library is a communication middleware which mediates between language communication extensions (like the MPI) and low level communication methods. Low communication method could be UDP, TCP or a cluster communication library. Nexus can be configured so that it handles messages differently depending on where they go. E.g. internally in a cluster it may use a specific library to send messages between nodes, but use SSL over TCP when sending messages to computers outside the cluster, encapsulating how and where the messages are send.

Nexus works by creating a communication link between a start point (sender) and an endpoint (receiver). Several start points can be connected to one endpoint, multiplexing the information into one channel for the receiver. After establishing a communication link a start point can be moved to another process. Endpoints cannot be moved. A process can have several start and end points.

The Globus Toolkit extends the Nexus library with several new capabilities. Some of these are: Secure Process Creation which makes it easy for processes to start new processes on other hosts. Transparent management of multiple security mechanisms, providing the same abstraction to the programmer even though several security mechanisms are used, e.g., SSL and Kerberos. Also a way to transfer communication links is provided along with security extensions to existing communication libraries.

References:[12]

B.10 GASS

GASS stands for Global Access to Secondary Storage and is a library/utility to access storage resources that are not available through normal grid protocols. These storage resources are files located locally on disk, ftp or on an x-gass server.

GASS provides APIs to access these files, so they can be used in applications. Furthermore GASS provides the x-gass server with API to provide access to files for other GASS clients. GASS also provides an API to manipulate local file cache.

References:[35]

B.11 DUROC

The The Dynamically-Updated Request Online Coallocator (DUROC), handles allocation of several resources. Individual resources are usually handled be a specific protocol, such as GRAM in the Globus Toolkit. However such protocols only provide interface to a single specific resources. A job might need several resources spread across different sites to complete. This is what DUROC handles, by finding and allocating the needed resources before the job starts. Besides handling allocation of resources before job start, DUROC also provides an API, which application can use to acquire or release resources at runtime.

References:[28]

B.12 Heart Beat Monitor

The Heart Beat Monitor (HBM) is a mechanism for monitoring the status of processes on a host. The HBM consists of three parts: Client Library (CL), Local Monitor (LM) and Data Collector (DC). Each host (which can be used as a computational resource) runs a LM. The processes running on that host can then use the CL to register itself to the LM. The reason for this scheme is that only process that say they are interesting are monitored; the rest are left alone. The LM regularly checks the registrated processes for their status, which at sends to one or more DC. The list of DC to send to is specified in the registration the local process does to registrate itself it the LM.

The LM monitors and reports about different states of the process, e.g., running, unavailable, exit with failure or success. The local process also has the option of sending a string to the LM, which the LM then forwards to DC. The main method for LM to DC communication is UDP, which is used due to its simpleness and low overhead, compared to connection oriented communication, i.e., TCP. All in all the HBM allows the user of DC to make qualified guess about the state of running processes and act accordingly.

References: [30]

Appendix C

Installing the NorduGrid Toolkit on Debian GNU/Linux

This appendix describes how to install the NorduGrid on the Debian GNU/Linux operating system. The current NorduGrid installation mechanisms is very Red Hat centric, and installing in on a non-rpm distribution can be quite a challenge. We hope that this document will be useful to other people, saving them some of the time that we have used installing it. In this guide gpt, globus and globus-config are installed from binary packages, where as the NorduGrid Toolkit is installed from source. However some thinking must be done when following this guide, since some of the commands are specific to our system.

C.1 Getting Debian Ready

After installing Debian some things are missing to be able to install NorduGrid. This is mostly development libraries and such. Before installing these, Debian should be upgraded to testing, since a newer version of libtool is required. It has been fixed in Cvs (as bug #198), but no release has been made since the fix. Upgrade to testing by adding the following into `/etc/apt/sources.list`:

```
deb ftp://ftp.de.debian.org/debian woody main contrib non-free
deb ftp://ftp.uk.debian.org/debian woody main contrib non-free
```

And do a:

```
apt-get update && apt-get upgrade
```

Hereafter development libaries should be installed:

```
apt-get install ssh gcc libc6-dev perl-modules automake \
patch screen file vim make wgetlibmysqlclient10-dev \
libstdc++2.10-dev bison flex libnet-ldap-perl time \
dnsutils alien less libtool libxml-dev libxml2-dev zsh \
bzip2 hdparm
```

This also installs some other good stuff that can come in handy. Then do a:

```
hdparm -d1 -c3 /dev/hda
```

To make sure the hard disk runs at an acceptable speed.

C.2 Installing ScalablePBS

Now the LRMS should be installed. Currently the only supported by NorduGrid is PBS, which exists in three versions: OpenPBS [39], ScalablePBS [37], PBS Pro [40]. We like ScalablePBS[1], since its free, the code is available and is under active development. Unpack it and install it like this:

```
./configure --set-default-server=HOSTNAME --disable-gui
make
make install
cd /usr/local/
sbin/pbs_server -t create
sbin/pbs_sched
sbin/pbs_mom
echo '\$clienthost HOSTNAME' > /usr/spool/PBS/mom_priv/config
echo 'HOSTNAME' > /usr/spool/PBS/server_priv/nodes
qmgr < /some/path/server.conf
```

If job are submitted from another machine, you will have to put the submitting machine into `/etc/hosts.equiv` file.

C.3 Installing gSOAP

gSOAP [42] is a library used by the NorduGrid Toolkit. It is not provided in Debian, so it must be compiled itself. Fortunately this is quite simple. Get the source from the homepage[2] and unpack it, then:

```
./configure
make
make install
```

And thats is.

C.4 Installing gpt, globus and globus-config

Start by download the newest Debian rpms from `ftp.nordugrid.org`. Then convert them to debs using alien; like this:

```
alien PACKAGE.rpm
```

Now symlink `/etc/rc.d/init.d` to point at `/etc/init.d`, i.e.:

```
ln -s /etc/init.d /etc/rc.d/init.d
```

[1] Can be found at `http://www.supercluster.org/downloads/spbs/`.
[2] Found at `http://gsoap2.sourceforge.net/`.

So start/stop scripts will be installed correctly. Now the packages can be installed using dpkg:

```
dpkg -i PACKAGE.deb
```

They must be installed in the order gpt, globus, and globus-config.

Now some post install steps has to be done. Start by creating environments for the scripts.

```
export GPT_LOCATION=/opt/gpt/
export GLOBUS_LOCATION=/opt/globus/
```

Then run the post install scripts.

```
/opt/gpt/sbin/gpt-postinstall
/opt/globus/setup/globus-post-install-script
```

Some of these scripts will probably complain a bit. Do not worry.

C.5 Installing NorduGrid

Get the source tarball from `ftp.nordugrid.org`, and unpack it.

```
./bootstrap
./configure --with-nordugrid-location=/opt/nordugrid \
  --with-globus-location=/opt/globus/ \
  --with-gpt-location=/opt/gpt
make
make install
```

And thats is.

C.6 Host key

To run the NorduGrid Toolkit, a host certificate is needed. This certificate is used by other entities in the grid to ensure the identity of the machine. To request certificates the package nordugrid-certrequest-config is needed. This can be fetched from `ftp.nordugrid.org`. Filename will be something like `ca_NorduGrid-certrequest-config-0.1-1.src.tar.gz`. Unpack it and do the following.

```
mkdir /etc/grid-security/certificates
cp *.HASH* /etc/grid-security/certificates/  (HASH was 1f0e8352)
cd  /etc/grid-security/
ln -sf certificates/globus-host-ssl.conf.1f0e8352 globus-host-ssl.conf
ln -sf certificates/globus-user-ssl.conf.1f0e8352 globus-user-ssl.conf
ln -sf certificates/grid-security.conf.1f0e8352 grid-security.conf
```

One can now request a host key by issuing the following commands.

```
grid-cert-request -host FQDN -dir `pwd`
cat hostcert_request.pem |mail ca@nbi.dk
```

The last command mails the public key of the certificate to danish certification authority. FQDN stands for fully qualified domain name, e.g., `benedict.auc.dk`.

C.7 Installing certificates

After receiving the host certificates they should be installed. Copy the files `hostkey.pem` and `hostcert.pem` to `/etc/grid-security/`. Permissions should be as this:

```
-r--r--r--  hostcert.pem
-r--------  hostkey.pem
```

C.8 Allowing VO users

To allow the users from the VO to use ones cluster, the package ca_nordugrid is needed. The file will be called something like `ca_NorduGrid-0.19-1.src.tar.gz`. Fetch it, unpack it, and do the following.

```
cp -p 1f0e8352.0 /etc/grid-security/certificates/
cp -p 1f0e8352.signing_policy /etc/grid-security/certificates/
cp -p 1f0e8352.crl_url /etc/grid-security/certificates/
```

However this only installs the certificates, it does not create the user list of who has access. This must either be done manually be editing the file:

```
/etc/grid-security/grid-mapfile
```

One can maintain the file manually by editing it, or setting a cron job to fetch the user list on a regular basis. If one chooses to maintain it manually the syntax is as following.

```
"/O=Grid/O=NorduGrid/OU=ORGANISATION/CN=FULL_NAME" GRID_USER
```

Where organization could be, e.g., `cs.auc.dk` and full name could be `Henrik Thostrup Jensen`.

C.9 Mimic Red Hat isms

As mentioned in the beginning of this chapter, NorduGrid has many Red Hat isms in it. The easiest way of overcoming the problems that arise from this is to mimic the needed Red Hat features[3]. One ism is the file `/etc/init.d/functions`, which only exist on Red Hat like distros. We got a version which was ported to Debian kindly provided by Åke Sandgren. Finally the directory `/var/lock/subsys` must be created, since this is needed, and does not exist per default in Debian.

C.10 Configuring the software

Globus and nordugrid need some configuration files to run. Fortunately templates are supplied in the NorduGrid distribution. Copy the templates.

```
cp /opt/nordugrid/share/doc/nordugrid.conf.template \
 /etc/nordugrid.conf
cp /opt/nordugrid/share/doc/nordugrid-globus.conf.template \
 /etc/globus.conf
cp /opt/nordugrid/share/doc/grid-manager.conf \
  /etc/grid-manager.conf
```

[3]Although this is not really a long term solution

And edit them accordingly.
As a last step, some scripts must be edited, to make the function correctly. Edit the file `$NORDUGRID_LOCATION/sbin/gridftp-server` and set the location for PBS, NorduGrid Toolkit and certificates correctly. You may also have to edit the su command to preserve the environment (-m). Also make the gridmap file location correctly.
When installing from source the jobstatus directory is not created. So do a

```
mkdir -p /var/spool/nordugrid/jobstatus
```

Now the NorduGrid Toolkit should be ready to start; so do.

```
/etc/init.d/gridftp-server start && \
/etc/init.d/globus-mds start && \
/etc/init.d/grid-manager start
```

Good luck.

Appendix D

The NG Proxy Todo List

This section presents a list improvements to NG Proxy. It is mostly work that will improve NG Proxy, so that it behaves more in the way that one wills expect it to work. It does not describe any new features, since these are described in 7.

- Instead of having incoming jobs submitted sequentially, a thread could be created for each new job. This would get the job submitting out of the main loop, and make the daemon able to submit several jobs at the same time. Thus it would scale better with regards to the number of incoming jobs, since the submission of a job would not have to wait for the previous one to be submitted. This only matters when dealing with large quantities of jobs over a relative short amount of time.

- The client currently used to send jobs to the daemon is currently very simple. All it does is to copy a file to the socket. It does not support any other parameters like the other commands in the user interface. The client should support the same set of parameters as, e.g., `ngsub`. A solution to this would be to include the client into the plug-in structure of the user interface and reuse the parsing from there. Another solution would be to reuse the parameter parsing from the user interface to get the parameters. Unfortunately this is rather hard since the parsing is not well separated from the rest of the user interface and therefore not easily reused.

- When submitting a job with `ngsub` information is displayed on the terminal, e.g., which cluster the job gets submitted to. When using the daemon this information should be send to the client over the socket, so that the same information is displayed to the user. A requirement for this to work as expected, is that a thread is created for doing job submission, so the user should not wait for any other job submissions as described in the previously. Unfortunately this is not the hardest part. In the current user interface, all the information which is displayed to the user is simply send to standard out. This could perhaps be redirected by closing file descriptor number 2, and opening it to the socket, but this is not really something worth chasing, since this is a quite ugly way of solving the problem. Instead a more flexible solution using, e.g., C++ streams should be created, however that would require a lot of rewrite in the GUI layer.

- In the user interface each command usually takes some arguments, e.g., dryrun and dumpxrsl. In NG Proxy these settings are currently set per daemon, making

them global for all jobs, thus losing flexibility. These settings should be made per job, so that each job can be treated differently like when using the normal user interface commands. A requirement for this to be possible is that the client submitter can parse its command line for arguments, and send them to the daemon, like described earlier. Besides the regular user interface options, resubmissions options could be specified as well, e.g., number of resubmission attempts. Besides specifying number of resubmissions options on the command line, they should also be possible to specify in the xrsl file as well.

Appendix E

NorduGrid Task List

The NorduGrid Task list can be found at http://www.nordugrid.org/documents/tasklist.php

Area	Task
Indexing service for stored files	- Requirement collection and review of existing tools and services - Service design: reliable, decentralized, interfaced to storage facilities, intelligent, fast Indexing service proposal draft - Data Management System design draft - User- and group-based access control - Management tools (copy, move, delete, rename, replicate etc) - Disk SE interface - Interface to mass storage system(s)
Storage element	- Disk-based SE concept - "Smart" and "Stupid" SE testing - Mass storage support - User- and group-based space control, quotas - Role-, user- and group-based access rights and permissions - GACL tests, user guide, migration - "Stupid Storage Element" configuration
Information system	- Authorized access to information - Requirements collection and design for the information indexing service - Fail-safe topology - Fast / scalable response, performance studies - QoS control, registration authorisation - Storage Element information - Better support for schedulers (like Maui)
Logging, bookkeeping and accounting	- Authorized access - Full job history / provenance data - System performance statistics - Resource usage account per user and per group (CPU, memory, disk space, bandwidth etc) - Web interface to logs

Area	Task
	- Accounting plugins
User- and group-based management (authorisation and resource allocation)	- User and group policies - Set up a security group - CA Web page: certificate request on-line, public keys etc - Extended/non-ambiguous information associated to a personal certificate - Replace existing VO server with VOMS one - Task-driven public key downloads - Proxy storage/delegation service (MyProxy?)
Grid Manager	- Replacing GridFTP with a more sophisticated protocol(s) - Quotas for session directories - Support for clusters without shared file systems - Automatic registration of cached files into the indexing service - Support for interactive tasks - Parallel input file download from different sources - Automatic compressing/decompressing input/output files
Workload management / brokering	- Respect for local policies (User- and group-based time allocation etc) - Cost evaluation (data transfer, storage, execution) - Benchmark-based load balancing - Re-scheduling, re-submission, recovery - User- and group-based resource discovery - Cross-cluster parallelism
Installation	- Installer/uninstaller script(s) and/or procedures - Configuration tool - Clean up external packages, sort out dependencies (INSTALL file etc) - Build-on-demand
User-specific software installation and support	- ATLAS EventFilter - ATLAS Production system interface - Interoperability with POOL
Other	- New technologies: GT3 - Different systems (Solaris, Mac, Windows) - Interface to Condor, ? both GM and IS - Performance tests: resources, users, jobs - Setting up a user support system - Graphical user interface

Bibliography

[1] World Wide Web Consortium. Web services architecture. http://www.w3.org/TR/ws-arch/, August 2003.

[2] Oracle Corporation. Oracle grid computing. http://www.oracle.com/solutions/grid.

[3] The European Datagrid. The datagrid project. http://eu-datagrid.web.cern.ch/eu-datagrid.

[4] P. Eerola, T. Ekelof, M. Ellert, J. R. Hansen, S. Hellman, A. Konstantinov, B. Konya, T. Myklebust, J. L. Nielsen, F. Ould-Saada, O. Smirnova, and A. Waananen. Atlas data-challenge 1 on nordugrid. *CHEP'03*, 2003.

[5] P. Eerola, B. Konya, O. Smirnova, T. Eklof, M. Ellert, J. R. Hansen, J.L. Nielsen, A. Waananen, A. Konstantinov, and F. Ould-Saada. The nordugrid architecture and tools. *CHEP'93*, March 2003.

[6] Internet Engineering Task Force. File transfer protocol (ftp). http://www.ietf.org/rfc/rfc0959.txt, October 1985.

[7] Internet Engineering Task Force. Internet x.509 public key infrastructure. http://www.ietf.org/rfc/rfc2459.txt, January 1999.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[9] Ian Foster. What is the grid? a three point checklist, July 2002.

[10] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2. Morgan Kaufmann, 1998.

[11] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2001.

[12] Ian T. Foster, Nicholas T. Karonis, Carl Kesselman, Greg Koenig, and Steven Tuecke. A secure communications infrastructure for high-performance distributed computing. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 125–136. IEEE Computer Society Press, 1997.

[13] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[14] Jabber Software Foundation. Jabber :: Protocol. http://www.jabber.org/protocol/.

[15] G. Fox, M. Pierce, D. Gannon, and M. Thomas. Overview of grid computing environments, Febuary 2003.

[16] Michael K. Johnson and Erik W. Troan. *Linux Application Development*. Addison Wesley, 1998.

[17] A. Konstantinov. The http and soap framework. http://www.nordugrid.org/documents/HTTP_SOAP.pdf, October 2003.

[18] A. Konstantinov. The nordugrid grid manager and gridftp server - description and administrators manual. http://www.nordugrid.org/documents/GM.pdf, July 2003.

[19] Sun Microsystems. Grid computing solutions. http://wwws.sun.com/software/grid.

[20] Sun Microsystems. What is grid computing? http://www.sun.com/2003-1118/feature/grid.html.

[21] Oracle Technological Network. Oracle grid computing technologies. http://otn.oracle.com/products/oracle9i/grid_computing/index.html.

[22] Jakob Nielsen and Oxana Smirnova. Nordugrid / data challenges. http://www.nordugrid.org/slides/20031127-jakob.ppt, November 2003.

[23] Nordugrid. Nordic testbed for wide area computing and data handling (nordugrid), September 2001.

[24] Farid Ould-Saada. Nordugrid sg meeting. http://www.nordugrid.org/slides/20031127-farid.sxi, November 2003.

[25] Legion Project. Legion a world wide virtual computer. http://legion.virginia.edu/.

[26] Legion Project. Legion: Frequently asked questions. http://legion.virginia.edu/FAQ.html.

[27] The Globus Project. About the globus toolkit. http://www-unix.globus.org/toolkit/about.html.

[28] The Globus Project. The dynamically-updated request online coallocator. http://www.globus.org/duroc/frames.html.

[29] The Globus Project. Globus collaborators. http://www.globus.org/about/collaborators.html.

[30] The Globus Project. The globus heartbeat monitor specification v1.0. http://www.globus.org/hbm/heartbeat_spec.html.

[31] The Globus Project. Globus resource allocation manager. http://www-unix.globus.org/api/c-globus-2.2/globus_gram_documentation.

[32] The Globus Project. Globus toolkit 3.0 fact sheet. http://www.globus.org/toolkit/gt3-factsheet.html.

[33] The Globus Project. Globus toolkit™2.4 overview. http://www.globus.org/gt2.4/overview.html.

[34] The Globus Project. Industry, research leaders hail globus toolkit 3.0. http://www.globus.org/about/news/prGT3quotes.html.

[35] The Globus Project. Global access to secondary storage (gass). http://www.globus.org/gass/, April 1999.

[36] The Globus Project. Gridftp universal data transfer for the grid, September 2000.

[37] Supercluster Research and Development Group. Scalable openpbs resource manager. http://www.supercluster.org/projects/pbs/.

[38] O. Smirnova. Extended resource specification language. http://www.nordugrid.org/documents/xrsl.pdf, October 2003.

[39] Altair Grid Technologies. Openpbs. http://www.openpbs.org.

[40] Altair Grid Technologies. Pbs pro home. http://www.pbspro.com.

[41] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling. Open grid services infrastructure (ogsi) version 1.0, 2003.

[42] Robert A. van Engelen. gsoap: Soap c++ web services. http://gsoap2.sourceforge.net/.