

HOVEDPROSJEKT:

TITTEL

Cluster and grid computing:

Accounting and banking systems

FORFATTER(E): KRISTIAN FROGNER
TARJEI MANDT
STEIN E. WETHAL

Dato: 19/5-04

Summary of the main project

| | | | |
|---|--|---|-------------------|
| Title: | Cluster an grid computing: Accounting and banking systems | Nr. : 5 | Date : 19.05.2004 |
| Participant(s): | Kristian Frogner Tarjei Mandt Stein E. Wethal | | |
| Teacher supervisor: | Erik Hjelmås | | |
| Employer: | HiG - Are Strandlie | | |
| Contact person: | Erik Hjelmås | | |
| Headwords (4) | GridBank, NorduGrid, grid computing, accounting | | |
| Pages: 59 | Appendices: 5 | Accessibility (open/confidential): open | |
| Short description of the main project: | | | |
| <p>The main goal is to survey existing grid banking tools and assess their applicability to NorduGrid. This also includes integration efforts of existing systems into the NorduGrid software or a proposal for NorduGrid's own accounting system. In addition to this we are also going to connect HiG to the NorduGrid project and look at the demands, in terms of administration and maintenance, further contribution would require.</p> | | | |

Preface

As we are finishing our last semester at HiG, we have to do a main project. This project gives 20 ECTS points, and it will last from January to 19. May.

We are three students who study bachelor of computer engineering, and has a main project which is called Cluster and grid computing. The main point of cluster and grid computing is to gather computers from all over the world to a grid where they can be used to share their resources. If someone need more resources than they have, they can use the grid and delegate tasks to other computers. When we chose this main project before Christmas 03, HiG got a request to participate in NorduGrid. But they couldn't answer before they got some information about it. We thought this could be interesting and a great task to learn more about.

Erik Hjelmås is our teaching supervisor and HiG is the employer, but we will keep in touch with the people in NorduGrid, especially Aleksandr Konstantinov, who is a software developer.

This report is meant as documentation on the work process during the project period. We will write this in English to make it easier for those who work with NorduGrid.

We wish to thank those people who have guided us through this project:

Aleksandr Konstantinov - for answering all our questions, and sharing useful information with us

Farid Ould-Saada –is the coordinator in NorduGrid and for inviting us to a meeting in Oslo

Erik Hjelmås - for guidance through the project

Are Strandlie - for helping us contacting NorduGrid

Gjøvik 19. May 2004

Kristian Frogner

Tarjei Mandt

Stein E. Wethal

Table of Contents

| | |
|--|-----------|
| Summary of the main project..... | 2 |
| Preface..... | 3 |
| Chapter 1. Introduction..... | 8 |
| 1.1 Organisation of the report..... | 8 |
| 1.2 Project description and restrictions..... | 8 |
| 1.3 Main goal..... | 9 |
| 1.4 Getting started with the project..... | 9 |
| 1.4.1 Project approach in the beginning..... | 9 |
| 1.4.2 Project environment and environment limitations..... | 9 |
| 1.5 Target audience..... | 10 |
| 1.6 The students' background..... | 10 |
| 1.7 Responsibilities..... | 11 |
| 1.8 Use of terminology..... | 11 |
| Chapter 2. Grid banking and accounting systems..... | 12 |
| 2.1 Background..... | 12 |
| 2.2 System requirements..... | 12 |
| 2.3 Introduction to the various systems..... | 13 |
| 2.4 SGAS..... | 14 |
| 2.4.1 Introduction..... | 14 |
| 2.4.1.1 Background..... | 14 |
| 2.4.1.2 Functional requirements..... | 14 |
| 2.4.2 Architecture..... | 15 |
| 2.4.2.1 Bank Service..... | 15 |
| 2.4.2.2 Logging and Usage Tracking Service..... | 16 |
| 2.4.2.3 Job Account Reservation Manager..... | 16 |
| 2.4.3 Scalability..... | 16 |
| 2.4.4 Redundancy and recovery..... | 17 |
| 2.4.5 Security..... | 17 |
| 2.4.6 Source code..... | 17 |
| 2.5 GridBank..... | 18 |
| 2.5.1 Introduction..... | 18 |
| 2.5.1.1 Background..... | 18 |
| 2.5.2 Architecture..... | 18 |
| 2.5.2.1 Overview..... | 18 |
| 2.5.2.2 Job submitting..... | 18 |
| 2.5.3 User interface..... | 19 |
| 2.5.4 Economical handling..... | 20 |
| 2.5.5 Redundancy and recovery..... | 20 |
| 2.5.6 Security..... | 20 |
| 2.5.7 Source code..... | 20 |
| 2.6 DGAS..... | 21 |
| 2.6.1 Introduction..... | 21 |
| 2.6.1.1 Background..... | 21 |

| | |
|---|-----------|
| 2.6.1.2 Definitions..... | 21 |
| 2.6.2 Architecture..... | 21 |
| 2.6.2.1 HLR and PA..... | 21 |
| 2.6.2.2 Job submitting..... | 22 |
| 2.6.3 Redundancy and recovery after failure..... | 23 |
| 2.6.4 Security..... | 23 |
| 2.6.5 Functionality and economic models..... | 23 |
| 2.6.5.1 Metrics to measure used resources..... | 23 |
| 2.6.5.2 Job payment..... | 24 |
| 2.6.5.3 Economic models..... | 24 |
| 2.6.5.4 User account administration..... | 24 |
| 2.6.6 Information logging..... | 24 |
| 2.6.7 Source code..... | 25 |
| 2.6.8 DGAS/EDG in the future..... | 25 |
| 2.7 Conclusion..... | 26 |
| Chapter 3. Usage analysis..... | 27 |
| 3.1 Why adapt GridBank into NorduGrid?..... | 27 |
| 3.2 Application dependencies..... | 27 |
| 3.2.1 SOAP - Simple Object Access Protocol..... | 27 |
| 3.2.2 Xerces - XML parser..... | 28 |
| 3.2.3 MySQL - Structured Query Language..... | 28 |
| 3.3 Open source..... | 28 |
| 3.4 Use cases..... | 29 |
| 3.4.1 Use case for how it works now..... | 29 |
| 3.4.2 Use case for how it should work..... | 30 |
| Chapter 4. Implementation..... | 31 |
| 4.1 PBS..... | 31 |
| 4.1.1 What is PBS?..... | 31 |
| 4.1.2 Which PBS version should I use?..... | 31 |
| 4.1.3 Install and configure PBS..... | 31 |
| 4.1.4 Additional information..... | 34 |
| 4.2 GridBank Implementation..... | 35 |
| 4.2.1 Implementation..... | 35 |
| 4.2.2 Additional Information..... | 45 |
| 4.2.3 Troubleshooting..... | 45 |
| 4.2.4 Patch..... | 46 |
| 4.3 How GridBank works in practice..... | 46 |
| 4.4 NorduGrid installation guide..... | 47 |
| 4.4.1 NorduGrid quick client installation guide..... | 47 |
| 4.4.2 NorduGrid quick server installation guide..... | 48 |
| Chapter 5. Evaluation and conclusion..... | 51 |
| 5.1 Administrating and maintaining a NorduGrid cluster..... | 51 |
| 5.2 Unfinished parts in GridBank..... | 52 |
| 5.3 GridBank in the future..... | 54 |
| 5.4 Different phases or increments in the project..... | 54 |
| 5.5 Problems in the project..... | 55 |
| 5.5.1 Different software and software versions..... | 55 |
| 5.5.2 Server instability..... | 56 |

| | |
|---|-----------|
| 5.5.3 Software documentation..... | 56 |
| 5.6 What we have learned in this project..... | 56 |
| Chapter 6. References..... | 57 |
| 6.1 References and bibliography..... | 57 |
| Chapter 7. Appendices..... | 60 |
| A Example XML message passed on by SOAP..... | 60 |
| B MySQL commands..... | 61 |
| C Grid monitor pictures..... | 64 |
| C.1 GridMonitor..... | 64 |
| C.2 Gjøvik gridcluster details..... | 65 |
| D Gantt-schemes..... | 66 |
| D.1 Original gantt-scheme | 66 |
| D.2 Final gantt-scheme..... | 67 |
| E Reports..... | 68 |
| E.1 Status reports..... | 68 |
| E.1.1 Status report 17/2-04..... | 68 |
| E.1.2 Status report 18/3-04..... | 69 |
| E.1.3 Status report 20/4-04..... | 70 |
| E.2 Status meeting reports..... | 71 |
| E.2.1 Status meeting report 1..... | 71 |
| E.2.2 Status meeting report 2..... | 72 |
| E.2.3 Status meeting report 3..... | 73 |
| E.2.4 Status meeting report 4..... | 74 |
| E.2.5 Status meeting report 5..... | 75 |
| E.2.6 Status meeting report 6..... | 76 |
| E.2.7 Status meeting report 7..... | 77 |
| E.3 Meeting reports..... | 78 |
| E.3.1 Meeting report 1..... | 78 |
| E.3.2 Meeting report 3..... | 79 |
| E.3.3 Meeting report 5..... | 80 |
| E.3.4 Meeting report 7..... | 81 |
| E.3.5 Meeting report 9..... | 82 |
| E.3.6 Meeting report 11..... | 83 |
| E.3.7 Meeting report 13..... | 84 |

Figures

| | |
|--|----|
| Figure 1: Information movements in SGAS..... | 15 |
| Figure 2: Information movements in GridBank (taken from GridBank document)..... | 19 |
| Figure 3: Information movements in DGAS..... | 22 |
| Figure 4: Usecase for how it works now..... | 29 |
| Figure 5: Usecase for how it should work..... | 30 |
| Figure 6: How to set up shared disk space on the front-end in NorduGrid server installation. | 49 |

Chapter 1. Introduction

1.1 Organisation of the report

Chapter 1 - Introduction

This is an introduction to the project with a description of the task, the main goal with the task, why we chose this task and our technical skills.

Chapter 2 - Grid banking and accounting systems

Chapter two contains information about the different grid banking and accounting systems that we have reviewed and a conclusion from our point of view.

Chapter 3 - Usage analysis

Here we will explain what GridBank can do for a grid solution like NorduGrid.

Chapter 4 - Implementation

Chapter four will show what we have done with the GridBank software to get it working within NorduGrid and how to use GridBank within NorduGrid and what the test results are.

Chapter 5 - Evaluation and conclusion

Evaluation and conclusion will provide the reader with our view on the whole project and what we have achieved while working with this project.

Chapter 6 - References

References to external information.

Chapter 7 - Appendices

Appendices for the project.

Layout.

This project is written in OpenOffice. Org 1.1. The document is formatted using the Times font, size 12 for normal text and Courier size 10 for system commands and code parts.

1.2 Project description and restrictions

When we said yes to look at this task, HiG had been requested to join NorduGrid and our job were to find out what to do at the school so it could be possible to join. After a while the problem description changed a bit. Since that task would be too small for a main project we needed another problem to solve. The group discussed with the teaching supervisor what the other problem could be. We looked at NorduGrid's website and found a list of tasks that should be done. We

picked one that looked interesting and mailed Aleksandr Konstantinov and asked what he thought about it, and the answer we got was positive. That task were to find an existing program to help NorduGrid with accounting and banking problems. We shall look at existing middleware and try to find something we can modify to be used with NorduGrid's middleware. So our two main goals are; get HiG on NorduGrid's grid and help them with the accounting and banking problems.

1.3 Main goal

One of the main goals is to install NorduGrid's middleware on our machines and try to get a cluster and be a part of the grid. We will also try to make an installation guide for HiG so it's easy for them to join NorduGrid if they want to. This task is of great interest for HiG because HiG will be the first college from Norway in this grid.

The other main goal with this project is to help NorduGrid finding a program which helps them to distribute and book jobs which have been done, and keep accounts over the the resources that have been used on the various jobs. We will look at existing programs and try to find something we can use. They want a system like this to get the opportunity to get more people involved in NorduGrid. Now it's only universities and other special interested peoples who is a part of NorduGrid. In the future they want companies to join, but no one gives away resources for free. And since most companies don't need extra resources, NorduGrid need a possibility to give something back e.g. real money. Or else the companies will give and give without getting anything back. Because of this it's important to use a system that check how much resources that have been used on each job, and estimates how many credits the resource owner shall get for the job.

1.4 Getting started with the project

1.4.1 Project approach in the beginning

When we started, a lot of the project was “up in the blue”. We had just heard about NorduGrid and grid computing, so in the beginning we started to read a lot. Information gathering wasn't that hard, but finding the most relevant material proved to be a little harder. A lot of the information out there is targeted at “the ones who have the money” and therefore they mostly explain grid and grid computing at a very abstract level. But little after little we gained some knowledge about grid and grid computing. We also started to look into the grid accounting/banking problematics, this also wasn't that easy, since we only had the basic knowledge of grid computing in the first place. But here we also gained knowledge little after little about different software solutions and middleware that already existed out in the world.

1.4.2 Project environment and environment limitations

We applied for three computers to use in this project and ended up getting one old computer. We had to decide how we were going to do the project in the most practical way, and we quickly

decided to work in the same room, since we already had a corner in a group room. Erik Hjelmås had some other computers for spare, so now we had a total of three old computers to use. In addition to these three old slow computers, Tarjei stepped in and brought a front-end computer for the grid to the school. The three nodes/working computers are behind a light firewall with some ports blocked, so we placed the front-end computer in an environment with a lot less restrictions so it would not interfere with the programs that needed different connections to the Internet. We also set up a primitive backup-solution that would back up our project material each night, so the project material is always copied onto a second computer in an another room in the same building each night. Due to the age of the computers used as workstations, web browsing went kind of slow. This was solved by using VNC [1] to connect to the computers we have at our home and using them for web browsing.

None of us had very much experience with Linux, so we decided to go with the Debian distribution since we primarily used this distribution last year in the “System administration” course. We started out with the 2.4.24 (unstable) kernel on the front-end computer, but later on we changed to the 2.4.19 kernel due to some stability problems. Most of the work was done using SSH to the front-end computer or VNC to an external computer. We also NFS-mounted a central working area for project documents and information that should be easily accessible by all the group members. The knowledge learned in the “System administration” course also proved to be very useful since we weren't that familiar with Linux. Google [2] also proved to be our friend in solving different problems with the Linux OS.

1.5 Target audience

The target audience for the conclusions and recommendations we get up with and potentially the program we will modify will be NorduGrid. It may also be others with a grid that need an application like this, and for those this can be a guideline, but we will not think about that while we are in the process. The target audience for the guideline of how to install NorduGrid middleware and GridBank middleware will be HiG and our teaching supervisor.

The final report are mainly written for the teaching supervisor and external examiner. We have decided to focus on that because they are the ones who shall evaluate and give us a grade. We will also put parts of the report on our website in such way that the people in NorduGrid get access to it.

1.6 The students' background

Stein E. Wethal, born in 1980 and comes from Kløfta. He studied for four years at high-school, three of them to be an agronomist and the fourth year to get his papers to start at college, before he left for one year basic training in the army. There he was in the engineering troop, which has nothing to do with this engineering education. When he came to Gjøvik in 2001, he started at three-semester course. He has no working experience related to computers, but has been interested in computers for a while.

Cluster and grid computing

Kristian Frogner, born in 1981, comes from Hamar. He studied electro/electronics at high-school for two years and the third year to get his papers needed for studying at college. Started with three-semester course at HiG in 2001 and is fulfilling the main project for engineering education. Has been interested in computers and new technology since middle the 1990s.

Tarjei Mandt, born in 1981 in Harstad. Studied for three years at Elverum high-school. He was one year in the army for basic training. After that, in 2001, he started at HiG, and now he is fulfilling the last year of the engineering education.

1.7 Responsibilities

Kristian Frogner is the group leader and contact person.

Tarjei Mandt is responsible for updating the website and take backup of the project material.

Stein E. Wethal is responsible to write meeting reports and log the work.

1.8 Use of terminology

| | |
|-------------|--|
| CP | Copy |
| DN | Distinguisd Name |
| Globus | Globus Toolkit, a tool for grid computing used in various middleware as NorduGrid, EDG, GridBank |
| GridCredits | A currency used to describe the value of computational energy in the grid |
| LRMS | Local Resource Management System |
| Middleware | Are the programs used between the front-end machines in the grid |
| NOK | Norwegian kroner |
| Open source | The source code from open source software can be used by anybody |
| OS | Operating system |
| RCP | Remote copy |
| Resource | Computational energy needed for executing a job |
| SCP | Secure copy |
| Ts | Timeshared |
| VNC | Virtual Network Computing, a program that can be used to use a computer from an another computer |
| VO | Virtual Organization, an organization that administratively groups a set of users and/or resources |

Chapter 2. Grid banking and accounting systems

2.1 Background

A grid is a Internet-based network of computers which are gathered for parallel and distributed computing. In a grid there are many computers from around the world that share their resources for computation of a great amount of data. Each of the organizations in the grid can make their own cluster if they have several computers in the grid. They can use one of the computers as a front-end computer and the rest will be back-end computers. The front-end is the one which communicates with the grid while the back-end computers only communicates with the front-end computer.

Grids were originally used in physics research environments to compute all the data they got from the research. It was a huge amount of data to compute so they needed many computers to get the resources that was required. Accounting and banking was not a problem then, but now it's getting more important since they want organisations and enterprises with their resources. The problem is that most of the organisations have no use of the grid, and they don't bother to share the extra resources they have without getting something back. It's an advantage to store used resources in non-commercial grids too, just to have an overview over used resources, but it's when you start to think commercial it will be important to store used resources. In the future there will be more and more commercial grids out there, and they have to decide what kind of resources they have to pay for and what they don't have to pay for.

NorduGrid [3] was started in May 2001 under the name "Nordic Testbed for Wide Area Computing and Data Handling". They tried to build a Grid infrastructure for use in production-level research tasks. That project was a success, and the testbed was set up in May 2002 and has been kept in continuous operation since August 2002.

The goal for NorduGrid is to: "deliver a robust, scalable, portable and fully featured solution for a global computational and data Grid system."

The NorduGrid testbed project was started by physics groups in Nordic universities, but now it's open for every committed member around the world.

2.2 System requirements

We got a list of wishes from NorduGrid to a banking/accounting system for use in NorduGrid. None of these are requirements, but it is preferable that most of it are implemented to get the product as good as possible. The list below is copied from the e-mail we got from Aleksandr Konstantinov.

a) It should work.

Cluster and grid computing

- b)** It is good if it is easy to install (NG aims at non-intrusive easily-manageable Grid).
- c)** Should be manageable by VO administrators or Resource administrators (or both) and require as little effort from system administrator as possible (same reason). It still should be useable in case of single user or small resource owner (down-scalable).
- d)** System should be scalable(up), both regarding resources and users. And redundant or to be able to recover after failure.
- e)** It should be secure. Preferably using PKI because that makes it compatible with current trends in Grid computing development (Globus).
- f)** Good if it could operate with loans (for users) and reservations (for resources).
- h)** System should be reliable in sense that neither the resource provider (cluster) nor the user could misuse the system.
- i)** At least simplest usage unit (CPU time, with varying price per cluster or maybe even per day time) should be supported.
- j)** Users should be able to obtain information about their status (in secure way) and possibly to control some features of their account (freeze, limit amount to be spent per day, ...)

2.3 Introduction to the various systems

When we started to look after different systems we could use, we got a list from Aleksandr Konstantinov with different systems that could be interesting to look at. Two of them were SGAS and GridBank. But we wanted to search more at the Internet after systems that we could use. We didn't find many systems out there, but we found three that were interesting for us. Those three systems are SGAS, GridBank and DGAS. The criteria we searched for was how it was built, which programs it used, is it easy to modify, is it easy to install and so on. We tried to compare our requirements with what we read about the systems.

We found a few others too, but they were mostly "on the paper systems".

We will go deeper into these three systems because we believe they could be usable by NorduGrid, and we will use the system that we find most suitable for NorduGrid's existing software.

2.4 SGAS

2.4.1 Introduction

2.4.1.1 Background

The SweGrid Accounting System (SGAS) [4], started in August 2003, is, as implied, a coordinated accounting system for SweGrid clusters. It is designed to be non-intrusive on site deployment and has a uniform architecture, built on open standard-based grid protocols and existing toolkits. An alpha release is scheduled to be ready in March 2004. As SGAS is still very much in early development (current version is 0.0.1a), the information below is mostly based on the available SGAS documentation.

2.4.1.2 Functional requirements

(as defined in architecture proposal)

From resource perspective

- Provide cost information to users
- Grant/deny users access based on account balance
- Get a guarantee that user funds will be available to pay for resource usage
- Track resource usage
- Charge the user for resource usage
- Get resource usage information for each submitted job

From user perspective

- Specify which VO/project should be charged for the resource usage
- Get the cost associated with using a resource
- Get account balance
- Get information on completed transactions
- Get usage information on completed jobs

2.4.2 Architecture

The major components in the SGAS architecture [5] consists of VO (User and Bank service), Resource broker, Resource (Account and Job Manager), Logging and Usage Tracking Service (LUTS). Figure 1 shows how the different components interacts with each other in SGAS.

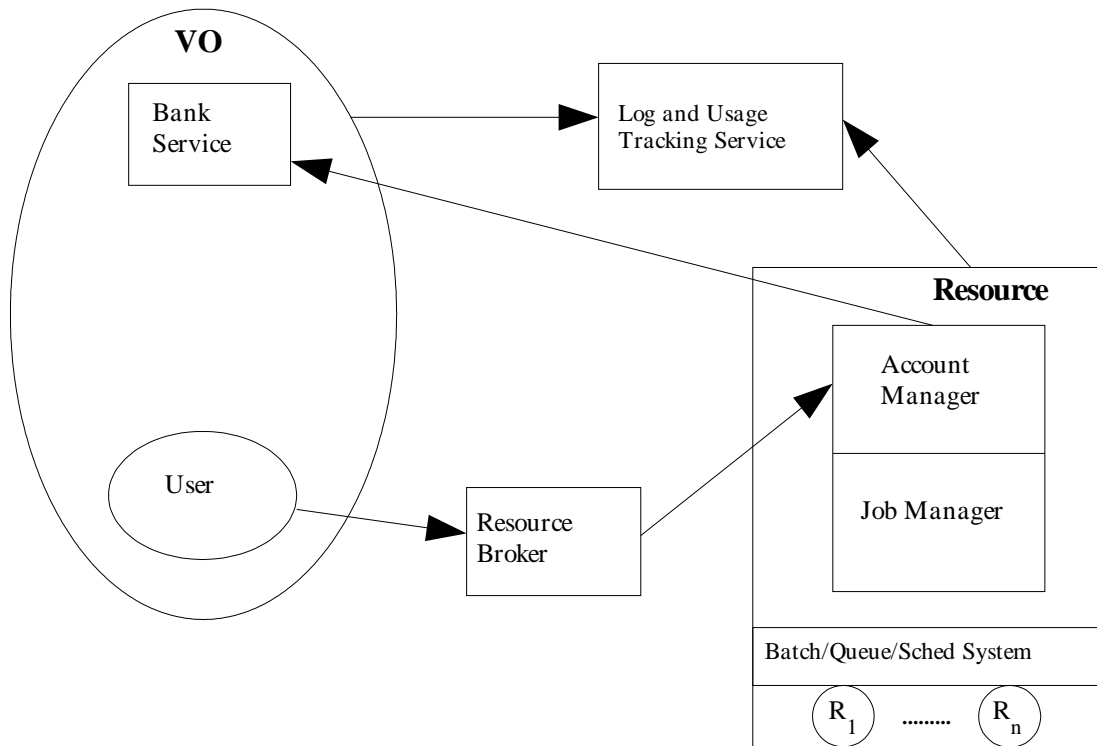


Figure 1: Information movements in SGAS

2.4.2.1 Bank Service

The primary purpose of the SGAS Bank [6] is to keep track of the resource usage for the individual projects/users. The Bank component is based on OGS (Open Grid Services Infrastructure). All bank related information is stored in an XML database as Xindice, and is already incorporated into Globus Toolkit 3. Features of the banking component include bank account administration, chargeable account (multiple account holders can withdraw/create holds from the same account), transaction history, logging service (retrieve detailed info about account entries), security based on delegated user credentials, and soft state account holds.

In SweGrid the bank accounts are assigned to projects and not to the individual project members, even though the members have individual user accounts on the Grid. Within each VO, a user might participate in multiple projects (with different bank accounts). A user can also be a member of projects in several VOs.

The bank account supports a dynamic set of account holders that are modifiable at runtime. The account is only accessible by the account holders and each bank client is authenticated and authorized before access to the account is granted. Also, the account holders have different privileges, controlling their access rights.

Cluster and grid computing

Account holders are able to request time-limited reservations (holds) on account funds. A client may reset the expiry time of a hold after its creation. On hold expiry, the reserved amount is returned to the account.

A transaction history is maintained for each account and account holders are able to pose non-trivial queries against the transaction log.

All transactions are performed using grid credits (unit less currency). While only node hours are accounted for in the initial SweGrid setting, additional types of resource usage can be incorporated into the accounting system (by using a function that converts the desired resource into grid credits).

2.4.2.2 Logging and Usage Tracking Service

LUTS [7] allow resources to log usage data and users to query the data in a consistent way. It is standardized around the GGF Usage Record XML format and uses standard Xpath-based Query languages. It includes batch publication, federated databases, and embedded local database support which together adds scalability and performance. When users submit their jobs, the job broker sends the job onward to an available resource for execution. When the job is done, the resource can push the usage record into (based on GGF UR standard) a local database (XML) or use the LUTS to publish the data in a service on a remote host. In both scenarios the users talk to a well-known LUTS service (that may be specified when the job is submitted) to query the log and usage data. The usage records can be pushed to the service in batch mode for better scalability, and there is hence no time guarantees on when the records will be available in LUTS.

2.4.2.3 Job Account Reservation Manager

Job Account Reservation Manager (JARM) [8] provides an integration point for the SGAS system into various Grid middleware (i.e. NorduGrid) with minimal intrusion. It interacts with the job submission infrastructure and creates a hold (soft-state fund reservation), or a reservation of a certain amount of available funds. A hold will either be granted or denied depending on the availability of requested funds. The calculation of the expected allocation and the actual allocation required to run the job is customizable, and based on run time properties as well as the client job specification. The account to be charged can be specified in the job submission (RSL), otherwise JARM will attempt to find the targeted user in the bank. In both cases, the security proxy of the user submitting the job must be used when authenticating and authorizing with the bank account. Once the job has completed, JARM calculates the cost and charges it against the hold that was used to reserve the allocation. It is possible to extend the reservation, if the job has not completed within the estimated run time.

2.4.3 Scalability

The SGAS architecture does not assume any centralized account handling entity. Each VO has an associated bank that handles the accounts of the VO users. Should the VO grow too large (thus affecting scalability), the VO can either be split up into two VOs with two separate bank services, or deploy a second bank. While in current practice, users are either mapped 1:1 or n:1 to local resource accounts, SGAS is aiming for a more dynamic creation of user accounts (so-called

template accounts) at submission time.

2.4.4 Redundancy and recovery

The system should run smoothly even after component failure. In the case of a bank being unreachable, the user should still be able to access a resource, but with less strict consistency guarantees. Replication will be included. The system will try to mask failures by proceeding as if nothing happened, resolving inconsistencies at a later time.

All information such as account state and transaction logs is stored in an XML database, to assure recoverability in the event of a server crash. It also allows users to query banking information.

2.4.5 Security

Authentication is based on the Public Key Infrastructure (PKI), and SSL like handshakes over SOAP in compliance with WS-Security, WS-SecureConversation, XML-Encryption, XML-Signature, and GSS-APIs (to allow future integration of Kerberos)[9].

Authorization comprises two parts: policy management and policy enforcement. Management is done through a WDSL interface while policy enforcement is implemented as a call out from a message handler to deny or permit a request.

Accounting information is only exchanged between trusted entities and accessed by trusted entities.

WS-SecureConversation handshakes ensure that account reserves as well as the account charge operations are secured against replay attacks and snooping.

2.4.6 Source code

The source code is available via AFS[10] and the cell [11]. You can also download a nightly checkout of the code from this link [12].

2.5 GridBank

2.5.1 Introduction

2.5.1.1 Background

GridBank (GB) [13], developed in Australia by Gridbus [14], has been designed to meet new grid requirements as an accounting and payment handling system. This makes it possible for the resource owners to profit from the resources that are used by other people. Until now, those who had free resources had to make a local account for anyone who needed resource access. In the future, this will be impossible because it will be thousands of computers in the same grid. GridBank has a different infrastructure. Everyone has to register themselves on a central server; hence the resource owners don't have to make accounts for every resource user.

2.5.2 Architecture

2.5.2.1 Overview

Maintenance and modifications is easy in this system. GB's server architecture consists of many modules which can be modified or replaced without disturbing the other modules. These are organized in three levels: the account layer, payment protocol layer and security layer. The account layer has the responsibility for databases and operations towards the accounts. The payment protocol layer defines payment schemes, message formats and communication protocols. It's easy to add payment systems in the future without modifying DB accounts and DB security modules. The security layer ensures that everyone is authorized and everything is authenticated. GB's structure makes it easy to have many users. In other middleware technologies such as Globus, the local resource owner has to create and manage local accounts for each user. This will be impossible if thousands of computers are connected to the grid. When we use GB, a negotiator connects the buyer and the seller together.

2.5.2.2 Job submitting

The Grid Service Provider (GSP) and Grid Service Consumer (GSC) opens an account in the GB server. The user (GRC) submits a job with the requirements it has to the resources, costs, deadline and so on. This is the QoS-requirements. Grid Resource Broker (GRB) get this information and communicate with GSP's Grid Trading Service (GTS) to find a GSP which has the optimal resources available for this job. GRB negotiates with GTS and Grid Agent (GA) get the responsibility to make a GSP computer ready and download the job. But before the job can start it has to be a local account on the GSP. The GSP's have some accounts in a pool which is not dedicated to any users. If the GSC get authorized, one of the accounts will be used by the GSP. GSC's certificate name connects temporary to the account to indicate the relationship between the account and the user. GRB keeps contact with GridBank Payment Module (GBPM) to manage the use of available

funds to help GSC avoid overspending. GBPM forwards payment details to GridBank Charging Module (GBCM).

GSP's Grid Resource Meter (GRM) gathers information about which resources that has been used for processing the job when it's finished. GRM forwards this to GSP's GBCM which estimates the total cost. They find out which GSC who has the account, and then the relationship get removed and put the account back to the pool, and GBCM forwards the payment details together with Resource Usage Record (RUR) to GB server with a request to charge the user account. The total cost estimates out of CPU-time, network load and other metrics. Figure 2 shows how the different components interacts with each other in GridBank.

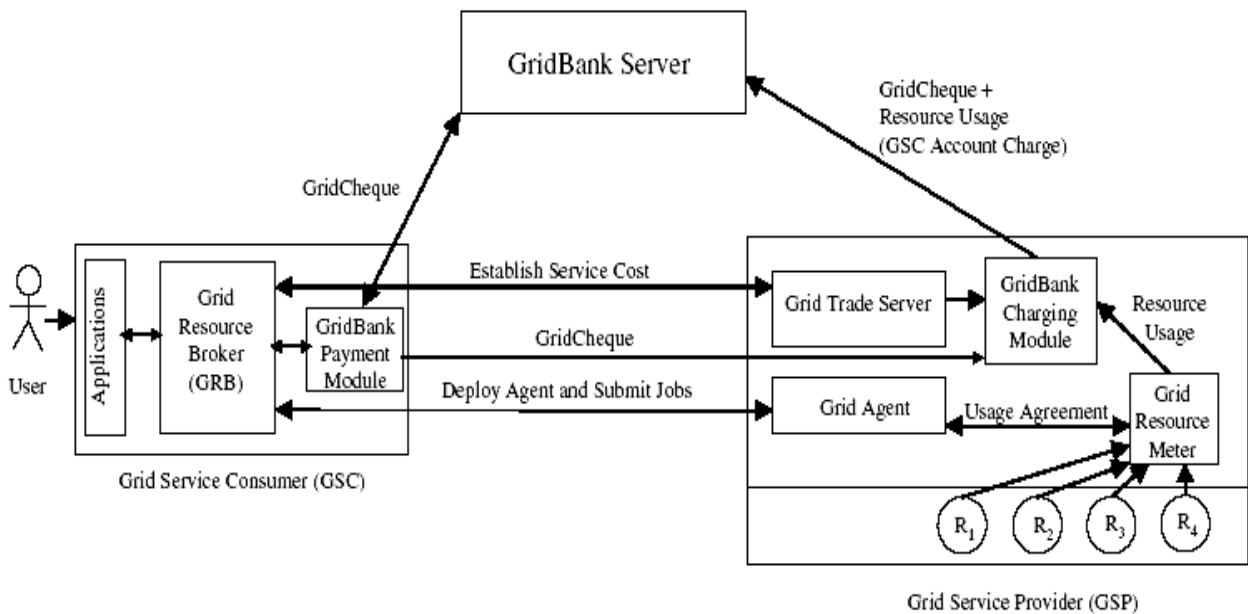


Figure 2: Information movements in GridBank (taken from GridBank document)

When GRM shall pick up information about used resources, it will communicate with the local systems, e.g. cluster scheduler. It filters relevant information and forwards it to the conversion unit, which generates a RUR, which is a standard XML-document. When we use different protocols it can be necessary with different stats for the resource usage. R₁ – R₄ in the figure forwards individual stats over the resource usage GRM. GB saves this information so they have proof that the job has been done. RUR is forwarded to GBCM, which get information about from GTS to estimate the cost. Every chargeable item in GTS must have a corresponding item in RUR. This will avoid cheating in terms of getting it cheaper or more expensive.

2.5.3 User interface

Before we can install GB, Globus toolkit with Globus server bundle 2 or later must be installed. We also need MySQL. The users can check the state of the account, and the administrators can add new administrators, open and close accounts and deposit and withdraw funds from the accounts. You can also use a grid monitor, a web-based GUI to control your account [15].

2.5.4 Economical handling

It's the owner of the resources (GSP) who decides the price. The users can choose what resource they want to use, but if it's too expensive, they will probably look elsewhere. The price will be regulated as economic systems in real life. If the demand is high the price will be high, with low demand the price will be low. GSP can send a description of the resources to GB which estimates a price from similar jobs. This makes a reasonable price for the job.

To avoid overspending, we can use a budget. But if you pay with a credit card after the job is done, you can overspend. This is possible even if a credit limit is set. However, you can also pay prior to a job submission, which will limit the usage to the amount paid.

To be sure that a user can pay for the jobs when he uses grid check, GB can lock some funds before the check is validated. Every GSP gets a check with some locked funds. All payment handling happens through secure connections. It's the administrators who handle payments, change the credit limit, close accounts and other things regarding accounts and payments. They transfer real money, but here it's possible to automate things in the future with e.g. PayPal.

GB supports the use of grid credits as payment. Co-operative groups that use resources from each other can pay with credits instead of real money. It's important that everybody gives and takes resources or some few get all the credits and the rest of them have none.

2.5.5 Redundancy and recovery

If we want to have a redundant system we need to take backup of the GB server. How often this is done, is up to the administrators. Though, it is very important since everything happens through GB. All the grid user accounts are there, so none can use the grid if the GB server is down. It can be necessary to have an extra server if the first one breaks down. In this system they use MySQL which makes data recoverable.

2.5.6 Security

Security is important in every grid because we don't want anyone to break into the grid, nor do we want the grid users to abuse the system. It's the security protocol which authorizes and authenticates the users of the grid. The Globus toolkit is based on PKI which use x509v3 certificates. The users have to be listed in a central file to get access to the grid. If they aren't listed, they won't be able to submit any jobs. This method of authorization and authentication prevents DoS-attacks.

When you have logged in, you don't have to type the password any more. The system uses the certificates to authenticate you when it has to. It's difficult for GSC and GSP to abuse the system because when a job is done, used resources have to match the requirements. Additionally, DB saves the RUR document to have proof of what has been used.

2.5.7 Source code

The source code for GridBank can be downloaded from their website [16].

2.6 DGAS

2.6.1 Introduction

2.6.1.1 Background

DGAS was developed for use in the European DataGrid project [17], which have been in the works since 2001 and up to now, March 2004. With DGAS [18] they have tried to make an approach to a more economical brokering in data grids, in contradiction to the other brokering methods used a lot today. The idea is that users who contribute with resources should get something back, in this case they will earn GridCredits when they contribute with resources. These GridCredits can be spent later by using other resources on the grid. Initially, DGAS can only handle GridCredits, but for the future there is a possibility that these economic grids could have a way of transforming GridCredits into other currencies. Small companies that need computational power can then buy the resources they need and resource-providers can benefit from giving others access to the resources.

2.6.1.2 Definitions

| | |
|-----|---|
| EDG | European DataGrid, the project group that have been developing DGAS as a system for the larger EDG project. |
| HLR | Home Location Register, a part of DGAS that contains the user database, their resources, and takes care of economic transactions between different HLRs. |
| IS | Information Service, a part of EDG that contains the information concerning the resource clusters and their attributes. |
| JDL | Job Description Language, a language used to define the requirements required by a job to run on a node. |
| PA | Price Authority, a part of DGAS that decides the price of the resources at different clusters. |
| RB | Resource Broker, a part of the WMS that collect information and decides which clusters are best suited for executing a specified job. |
| WMS | Workload Management System, a system that receives a job and decides if it can be executed, where it should be executed and sends the job to the cluster. |

2.6.2 Architecture

2.6.2.1 HLR and PA

HLR – Home Location Register

The HLR is a database that contains the different user accounts on the grid, and their account information. It also takes care of the communication between different HLRs and credits/debits the

different user/resources owners for the amount that their job uses. EDG has suggested a model where each VO has their own HLR for their own members.

PA – Price Authority

The PA is a component in DGAS that decides the prices for all the different resources in the grid. The PA can assign different prices to all the different clusters, based on predetermined factors.

2.6.2.2 Job submitting

When a user wants a job executed, the job goes through different phases in the system. Here we will look very short on how the accounting system works in general.

At first, the user submits the job to a central WMS, using a language for describing the requirements from the system that will execute the job, JDL. The RB will then decide which places that are appropriate to execute the job on; the places that fulfil all the requirements are defined in the JDL message from the user. The information needed for taking this decision is gathered from the IS.

At this stage, the RB should gather an estimate of the price, but this is not implemented yet, due to the difficulty of estimating the running cost of the job. A “cost estimation algorithm” could then optimize the cost of the job based the the prerequisites of the job.

The RB then checks the user's bank account, and reserves an amount of money if the user is able to pay for the job. If the user can't pay for the job, e.g. negative balance on the account, the RB will not send the job any further. If the user is able to pay, the job will be submitted to the cluster decided to do the job.

When the job is finished, the cluster will message the job submitter's HLR and the HLR will calculate the job cost based on the resources used. Then it will communicate with the resource owner's HLR and they transfer the GridCredits. Figure 3 shows how the different components interacts with each other in DGAS. Ideally, the system should be monitored while running the job. For instance, if the job is taking a lot longer time than expected, it could be suspended if the job owner runs out of GridCredits. This is not implemented now since there is no way of monitoring the resources used before the job is finished at the moment.

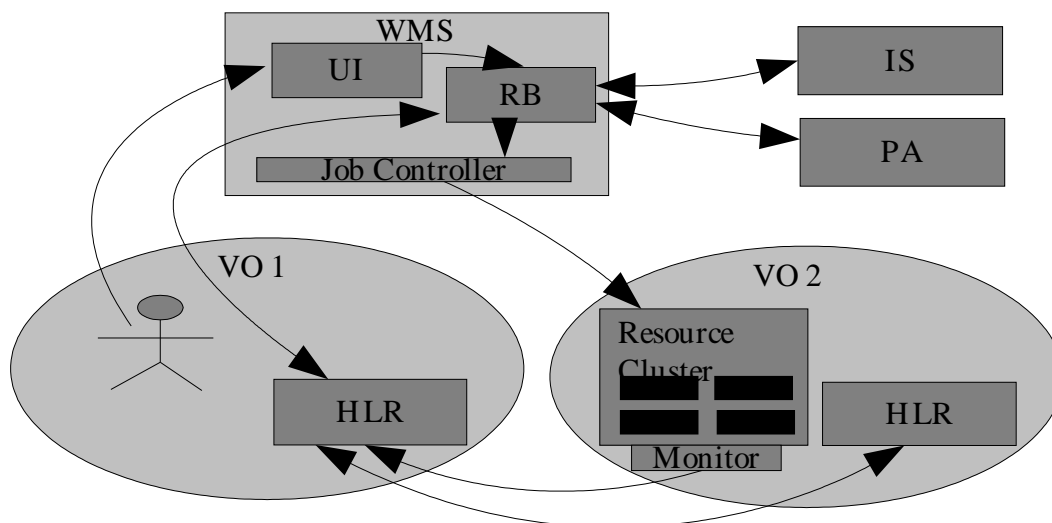


Figure 3: Information movements in DGAS

Even if NorduGrid and EDG base their systems on the same software at the lower level, the Globus toolkit, there are 2 major differences in their structure. NorduGrid uses a decentralized resource brokering system, where each resource broker resides in the job submitter's computer. In EDG, however, they have a central resource broker. The DGAS will have to be modified to handle this difference, but if you look at DGAS itself, it is very much like NorduGrid, decentralized with each VO having its own HLR. DGAS itself should be scalable to larger system with the “bank-branch” like structure, spreading the user accounts out over a large net rather than having them all on one place.

With all these HLRs spread out on the net, there must be some kind centralized place to list all the approved HLRs, so users can only use these approved HLRs.

2.6.3 Redundancy and recovery after failure

None of the redundancy and recovery after failure issues is specially mentioned in the DGAS architecture, but the possibility of having only one PA could be a problem. This could, as mentioned in the DGAS architecture, be divided into several physical instances, that share a common dataset, making the system more redundant if one of the PAs should fail. Since there are several HLRs in the network, a job will only be dependant on the two HLRs involved in the job (The resource consumer's HLR and the resource provider's HLR). Recovery after system failure will have to be addressed in the database used for storage, e.g. MySQL.

2.6.4 Security

DGAS is based on the Globus Grid Security Infrastructure (GSI) [19], a system that is already in use in NorduGrid. GSI uses X.509 PKI certificates and communicates using SSL. Authentication is mutual between hosts to ensure that both parts really are talking to the one they think they are talking to, preventing spoofing. Since DGAS bases its communication around XML messages, they use a unique identification of different XML messages to prevent repetitive registration of XML message and replay/playback of messages. A list of approved HLRs has to be created to ensure that no one can create his own fake HLR to obtain false GridCredits for use in the grid.

2.6.5 Functionality and economic models

2.6.5.1 Metrics to measure used resources

A good system should have the capability to use many different metrics for measuring used resources while processing the job. DGAS only supports used CPU-time and used wall-clock-time now, with CPU-time mandatory and wall-clock-time optional. Other measurable metrics like used memory, network traffic, and used disk space should be implemented at some time, but is not currently supported by DGAS. The total cost of the job will then be calculated from these factors.

2.6.5.2 Job payment

The payment of a job is only done when the job is finished. If the job terminates or crashes the resource owner will not get paid. While this encourages resource owner to have stable systems, it doesn't really help them if the job owner terminates the job before it is finished. The resource owner will not get paid even if the job occupied his resources while running. A scheme with payment “as you go” here will solve this, with some kind of sensor telling the bank not to pay if the cluster goes down. (This is of course not implemented)

2.6.5.3 Economic models

The DGAS system does not focus on any specified economic model. The system has been designed with an eye on the fact that they do not really know what economic model they want to use, therefore they have used the system with PAs. With the PA, the way resources are priced can be decided later. Implementation of the price scheme can be implemented in a specified file on the PA server. At the moment the PA decides the resource price based on the different queues on the different clusters, but this algorithm can easily be changed into other pricing schemes.

By regulating prices on different clusters, you can partially do job brokering on an economical level, since user tend to lean towards the cheaper places if they know they get the same service. Regulating prices this way, and at the same time regulating grid load, is the basics behind “Economic Brokering”.

Currently, there is no way of reserving resources ahead of time. These will have to base themselves on the queues at the local clusters.

2.6.5.4 User account administration

Account administration is at a very primitive level at the moment. You can create accounts and give them GridCredits, but the user cannot control the account in any way, e.g. limit how much is spent each day. A user can overdraw his account, but this really depends on what kind of economic policy you want on the accounts. By enforcing a strict economic policy, users might be allowed to overdraw the account only if they already have positive funds and if the job is estimated to use less than what the user actually have in his account. A less strict policy could allow users to submit jobs regardless of their funds.

2.6.6 Information logging

Since all the executed jobs go through the central WMS server in EDG, logging of everything is fairly simple. With NorduGrid you would have to decide if all the logged information about job execution should reside on each HLR or if it should be collected to a central server and stored there. For the information to be gathered on a centralized server, there has to be some system for either collecting it at given intervals or sending the information when the job is finished. This information is useful for analysing data about the grid, and on local HLRs users would probably like to have a log of their activity on the grid.

2.6.7 Source code

All the source code from the DGAS system and the rest of the EDG project is available at the EDG website [20] and can be freely used and modified as long as you state that parts of the code come from the EDG project.

2.6.8 DGAS/EDG in the future

The EDG project is scheduled for final evaluation in March 2004. A new European grid-project is already moving, the EGEE [21]. According to their plans, they are supposed to take advantage of the experience gained in other grid projects and take these plans further. As they have stated, this will also include the EDG project. This might open a possibility for further development of DGAS, but there is no guarantee for this, they might even decide to ditch the whole system.

2.7 Conclusion

Out of all the three system we have looked at, GridBank is the only complete and finished system. DGAS is “finished” in the sense that the EDG project is finished, but the software itself have many things that are proposed, but not yet implemented. SGAS is still in early development and as of now only nightly builds are available (no announced releases).

While DGAS's distributed model with HLRs is very similar to NorduGrid's structure, SGAS is also aiming for a decentralized bank-structure with several banks, maybe one for each VO. GridBank hasn't migrated to a distributed structure yet, but as stated in their architecture, this might come in the future. Apart from this, GridBank is a very thoroughly planned structure with a hard focus on economic structure and economic brokering. While DGAS and GridBank have a more traditional approach to bank accounts, SGAS have a system which makes it possible to define accounts to different projects, and then assign users to these projects, thus making it possible to have many users on a single project, then all these users can access the resources assigned to the project. SGAS also seem to aim a little more against universities and similar institutes, with their model, they don't seem to have that much focus on the cost brokering and market values.

Payment of resources is also very thoroughly addressed in GridBank with different payment schemes, based on both GridCredits and real money.

Implementation will probably require some work and modification with GridBank, while SGAS is already being developed for use with the NorduGrid software. DGAS is using somewhat the same structure as SGAS, but it would still have to be modified to be usable in NorduGrid. GridBank already support a large variety of different measurable metrics for calculating job cost, things like used network traffic, used memory, used cpu/wallclock-time have their own price, so they will be included in the final resource price. At the moment NorduGrid only provides information about used time, but in the future in a full scale production grid you may want to calculate the price with a background from many different factors.

We believe that GridBank, when properly adopted and modified, would pose as the overall best system for NorduGrid with commercial economical brokering in our mind. If the grid is more focused against scientific applications used by non-commercial institutes, SGAS also seem to be a good scheme, as an overall simpler system, with less focus on the economic brokering.

Chapter 3. Usage analysis

3.1 Why adapt GridBank into NorduGrid?

The way grids work today, most users have access to unlimited resources as long as they have access to the grid. As grids are starting to go more commercial these days, the problem with “who should have access to what and how much” is more and more becoming an issue. When someone contribute with resources and others exploit it for pure economical reasons, why shouldn't they have to pay for it? This problem is pushing forward the need for some system that can keep track of resource usage and access-based usage based on that possibility that you can pay for it.

With GridBank we are trying to get a simple solution of this working in a grid. If you have an organization that have a project that needs a lot of computational power, you could either apply for credits or pay for the credits needed. Then you would get a certain amount of credits and this could be used for the project. However, you would be limited to the amount available to you since you cant freely use the grid as much as you want and the resource providers would get something back as a compensation for sharing their resources. When resource providers get paid/compensated for the resources they provide this will also increase the resources in the grid, since people are more likely to join the grid when they get something back.

GridBank claim to support a lot of the features that could be needed by a solution like this, so by integrating GridBank into NorduGrid we hope it can give us a view on how the system should be and maybe even be a usable solution in itself. Even if we get it working and it does not really fit into NorduGrid it might give some leads as to how such a system could be developed and used in the future.

3.2 Application dependencies

GridBank itself is dependent on some external software to be present in order for it to compile. Using different modules for different tasks makes the development of GridBank easier and faster, since others who have a lot more knowledge in these areas have written modules that can be reused by others. The downside is that the GridBank software requires these models to work at all. This means there is more to install and more to get working, which again points to the fact that there is more to do when installing it.

3.2.1 SOAP - Simple Object Access Protocol

GridBank uses SOAP for communication between the different modules, the server module, the gbpm (GridBank payment module) and the gbcm (GridBank charge module). SOAP is a

framework for sending messages between different computers connected to a network. It is specialized for use in a light distributed computing environment, used to send messages between different functions in programs using a XML format on the messages, allowing it to use port 80 on the network, making it compatible and easy to use on most networks since port 80 is used by normal web pages and then very seldom blocked in the network. [Appendix A] shows an XML message passed on by the payment module to the server for pre-paying a job.

3.2.2 Xerces - XML parser

Xerces-C is used for parsing XML messages and obtaining input from XML messages in a C++ program. Xerces can both read and write data from/to XML messages easily from within a C++ program, making a lot easier to create and modify XML messages in a clean efficient way.

3.2.3 MySQL - Structured Query Language

The well-known database MySQL is used for maintaining the user and account information in the system, and logging the job and account history. The database can be accessed in a very basic way with the java UI or using SQL commands. [Appendix B] shows different information in the test database.

3.3 Open source

Open source [22] means that the source code is free to use, modify and distribute for anybody. There are several criteria for the distribution terms:

1. Free distribution

The license cannot restrict any part from selling or giving away software which is a part of a software distribution that contain programs from different sources. The license cannot require fees for a sale like that.

2. Source code

The program must contain the source code, and distribution of both source code and in compiled form is allowed. If a product doesn't get distributed with the source code, it must be possible to obtain it from the Internet or elsewhere. It's not allowed to mess up the source code if the intention is just to mess it up.

3. Modified applications

The license must allow modifications, and must allow these to be distributed under the same terms as the license in the original software.

The main idea behind open source is quite simple. When the programmers can read, modify and distribute the source code again and again, the program will go through an evolution. Peoples make it better, it will be easier to use, and this will be done much faster than through conventional

development.

NorduGrid's middleware is open source and is freely available from the NorduGrid website [3]. The software is distributed under the GNU General Public License [23]. GridBank is also open source, as mentioned earlier.

3.4 Use cases

3.4.1 Use case for how it works now

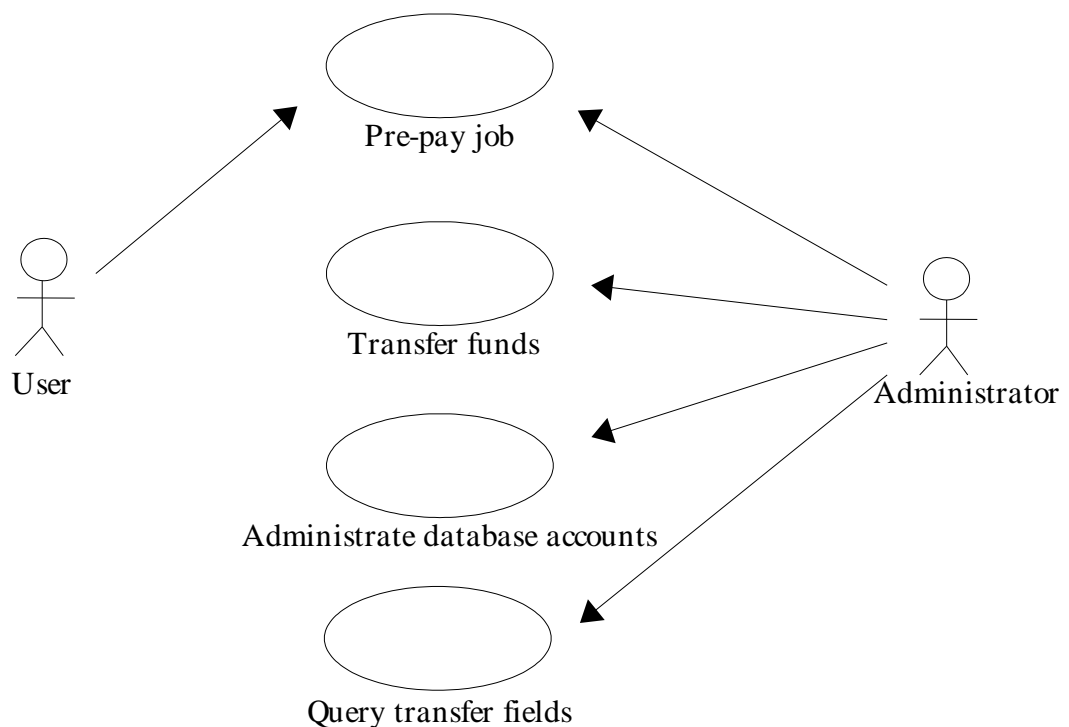


Figure 4: Usecase for how it works now

The use case in figure 4 shows how the GridBank system works today. The user can only pre-pay jobs. We think that this isn't good enough. It would be better if it worked like the use case in figure 5. There the user can not only pre-pay, but also look at the transfer funds and account transactions. He/she can also administrate accounts.

3.4.2 Use case for how it should work

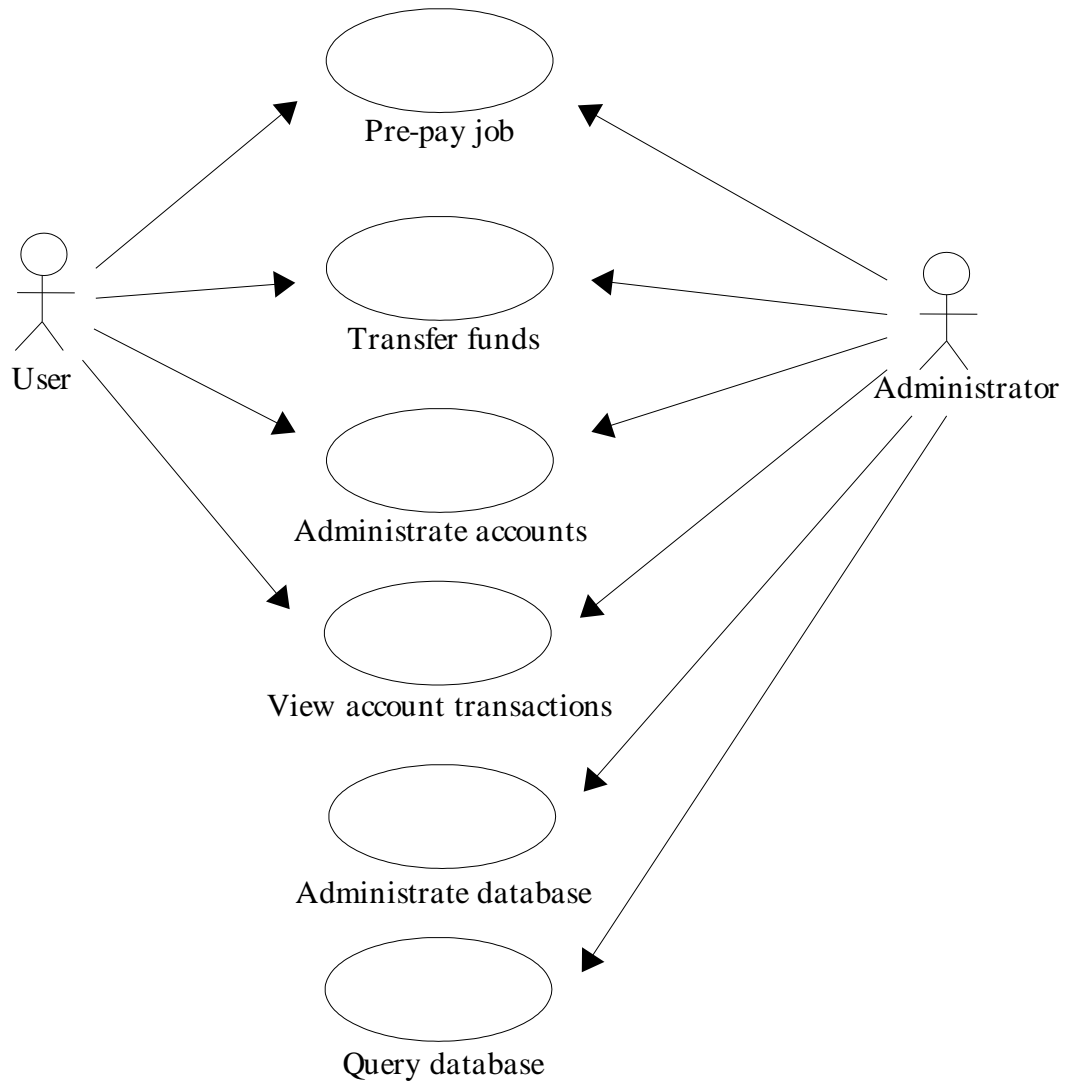


Figure 5: Usecase for how it should work

Chapter 4. Implementation

4.1 PBS

4.1.1 What is PBS?

PBS (Portable Batch System) is the leading workload management solution for HPC systems and Linux clusters. It was originally designed for NASA in the mid-nineties as the existing resource management systems did not meet the requirements of modern parallel/distributed computers and clusters. PBS enabled the establishment of a common queuing environment across heterogeneous systems. Its purpose is to provide additional controls over initiating or scheduling execution of batch jobs. The batch system lets site owners define and implement policy as to what types of resources and how much of each resource jobs may use.

4.1.2 Which PBS version should I use?

In the latest NorduGrid middleware distribution you are free to use any OpenPBS [24] based LRMS. You may also use Condor [25], but it's not recommended at this point. Both the front-end and the computation nodes will need an LRMS installed, although this is not needed on storage elements.

Which one you use depends on how scalable you want your cluster to be. While Open PBS only works well with up to 32 nodes with 100s of jobs, PBS Pro [26] is in production use at sites with thousands of CPUs, 30,000+ jobs. Open PBS doesn't scale well above 32 nodes due to fault tolerant problems. Torque (Tera-scale Open-source Resource and Queue manager) [27], however, is based on Open PBS v.2.3.12 and adds the lacking scalability and fault tolerance. Additionally, it features extension patches provided by NCSA (National Center for Supercomputing Applications), OSC (Ohio Supercomputer Center), and many other leading HPC organizations.

If you want to set up a small cluster (consisting of 20-30 cpus), it doesn't really matter whether you choose Open PBS or Torque. Keep in mind, however, that Open PBS is rather old and currently unsupported. For larger clusters, it's recommended that you either use Torque or PBS Pro (license costs \$125 - \$500 per CPU, discounts on educational and academic usage).

4.1.3 Install and configure PBS

Installing Torque and Open PBS is very straight forward. Upon compiling the latter from source in Debian, you might stumble across errors like:

```
*** No rule to make target 'built-in', needed by 'attr_atomic.o'.
```

Cluster and grid computing

This may be fixed by adding the following lines to the `buildutils/exclude_script` file:

```
/ \<built-in>//d
/ \<command>//d
```

After install (configure, make, make install), files are by default put in `/usr/spool/PBS`. If you're setting up a node, you need to create the file `/usr/spool/PBS/mom_priv/config` and edit it with the following:

```
$clienthost <hostname of pbs server>
$logevent 255
```

Normally, after jobs have completed, files will be sent back to the front-end through RCP/SCP. In the event of rcp connections not being allowed between the nodes and the front-end, you will have to use regular `cp` instead. This means you also have to mount the grid area, to which the jobs are sent, on each individual node. You also want to let the mom know that you want to use `cp` instead of `rcp` by adding something similar to the following line to the mom config:

```
$usecp p02081.hig.no:/scratch/grid/ /scratch/grid/
```

There are four ways of setting up a mom on a node:

1. You do a full install of PBS. While this works, it's a bit wasteful.
2. Rerun configure with the following options: `--disable-server --set-sched=no`.
3. Do an install of just the mom on each system. In the PBS source tree, `cd src/mom` or `src/resmom` (depending on pbs type/version) and `make install` as root. Commands may be installed as well, but are not required (`/cmds`).
4. The last option requires access of the PBS daemons and commands which may reside on a network file system. If `{target_tree}` is accessible on the host, run the following commands:

```
sh {target_tree}/buildutils/pbs_mkdirs [-d new_directory] mom
sh {target_tree}/buildutils/pbs_mkdirs [-d new_directory] aux
sh {target_tree}/buildutils/pbs_mkdirs [-d new_directory] default
```

Also, add the server hostname to `<PBS path>/server_name` and make sure that the server has access rights to the node (`/etc/hosts.allow`). Once this is done, you can launch `pbs_mom` (`<sbin dir>/pbs_mom`).

Once the mom is up and running, you want it added to the server node file. It must be created as `<PBS path>/server_priv/nodes` and should contain one line per node.

```
node_name[:ts] [property ...] [np=NUMBER]
```

An example would be

```
frontend.hig.no   P4      np=1
grid1.hig.no      PII     np=1
grid2.hig.no      PII     np=1
grid3.hig.no      PII     np=1
```

Cluster and grid computing

Ts (timeshared) nodes are currently unsupported and may not be used. Zero or more properties may be set to uniquely identify a set of nodes, example you may group Pentium and Athlon processors together. In this case, the property value is used for identifying processor type. Np defines the number of processors of the given node (default = 1).

To set up the server and a default queue, create a file (e.g. /etc/pbs.conf) with the following input:

```
set server log_events = 511
set server mail_from = adm
set server scheduler_iteration = 60
set server default_queue = slow
set server query_other_jobs = true
set server node_pack=false
set server scheduling=true

create queue slow
set queue slow queue_type = Execution
set queue slow enabled = True
set queue slow started = True
set queue slow Priority = 100
set queue slow resources_default.neednodes = PII
set queue slow resources_max.cput = 02:00:00
set queue slow resources_default.cput = 01:00:00
set queue slow max_user_run = 3
set queue slow max_running = 3
set queue slow max_queuable = 40

create queue fast
set queue fast queue_type = Execution
set queue fast enabled = True
set queue fast started = True
set queue fast Priority = 100
set queue fast resources_default.neednodes = P4
set queue fast resources_default.cput = 01:00:00
set queue fast max_user_run = 1
set queue fast max_running = 1
set queue fast max_queuable = 20
```

Here we have two queues set up with the properties PII and P4. The idea behind this is that some jobs may require more processing than other jobs, so while light weight jobs are passed on to the PII machines, heavier tasks will be processed by the P4. Notice that the resources_max.cput attribute is set in the slow queue. This is because we don't want tasks taking up more than 120 minutes CPU time to be processed on the PII's.

When starting pbs_server for the first time, use the -t create parameter to initialize various files. This is only required the first time you start pbs_server unless you want to clear the database and start over.

```
pbs_server -t create
qmgr < /etc/pbs.conf
```

Cluster and grid computing

To restart the server, a quick and easy way is:

```
qterm -t quick
pbs_server
```

Now that the server is up and running, status of nodes may be checked with:

```
<PBS installation path>/bin/pbsnodes -a
```

To check parameters on a specific queue, use:

```
<PBS installation path>/bin/qstat -f -Q queue_name
```

If nodes don't show up (status=down), make sure to check the pbs log files. These are located in `mom_logs` on the nodes and in `server_logs` on the front-end.

Also, notice that PBS relies on hostnames and not IP addresses. If a node has an IP address with no DNS record (must resolve both ways), it will not qualify as a valid resource.

The `qmgr` command provides an interface to the batch system. Batch systems may be administrated both locally and remotely depending on the administration settings used. The `qmgr` enables you to directly pass commands to the batch system with immediate effect, hence no restart of the server is necessary. For instance, if you add another node to the cluster, you would update the nodes file. Instead of restarting, however, you can do the following in `qmgr`:

```
create node node_name np=1,ntype=cluster,properties="P4"
```

You can also delete a node directly in `qmgr` if needed,

```
delete node node_name
```

Finally, launch the scheduler,

```
pbs_sched
```

Note: You may replace the PBS scheduler with a more sophisticated scheduler like Maui [28] which is acknowledged by many to be the most advanced scheduler in the world. Maui is capable of optimizing scheduling and node allocation decisions, improving the manageability and efficiency of machines. It is currently in use at hundreds of leading government, academic, and commercial sites throughout the world.

4.1.4 Additional information

You can find additional information about NorduGrid PBS configuration and OpenPBS manual from [29] and [30].

4.2 GridBank Implementation

4.2.1 Implementation

The following implementation routine was tested on Debian 3.0 using kernel 2.4.19.

Requirements:

NorduGrid Globus Toolkit distribution (2.4.3) [31]

GridBank source code [16]

NorduGrid ARC grid middleware source code (0.4.1 or later) [31]

J2SE 1.4.1+ SDK [32]

gSOAP (v2.1.6 bundled with GridBank; don't replace with newer version) [33]

Xerces C++ Parser (v2.1.0 bundled with GridBank; don't replace with newer version) [34]

Please refer to the NorduGrid documentation [35] on how to install the Globus toolkit. You will need the Globus Packaging Tools (GPT), Globus Toolkit and the Globus config. All files are available from the NorduGrid FTP site [31].

Once the Globus toolkit is in place (normally installed into `/opt/globus`), download the NorduGrid middleware source code. Make sure the variables `$GLOBUS_LOCATION` and `$GPT_LOCATION` are set to the location of Globus (usually `/opt/globus`) and GPT (usually `/opt/GPT`).

1. Set up the Globus Toolkit for use with GridBank.

Generate the header files and libraries of Globus I/O module.

```
$GLOBUS_LOCATION/sbin/globus-makefile-header -flavor=gcc32dbg globus_io
```

This step is not required if you use the pre-compiled NorduGrid Globus distribution.

2. Modify the NorduGrid middleware source code

GridBank depends on generated request and resource usage files on the cluster front-end. In order for these to be created, you must make the following changes to the middleware source code (this task may also be accomplished through the use of plugins). Note that the modifications provided below differs from the changes suggested in the GridBank documentation.

FILE: `<nordugrid_source_path>/grid-manager/jobs/states.cc`

After the include statements, add the following:

```
//***** GRIDBANK MODIFICATION *****  
//Variables for recording resource usage information  
struct rusage          resource_usage;
```

Cluster and grid computing

```
FILE*          resource_usage_file;
char*          resource_usage_filename;
char*          rur_globus_id = GLOBUS_NULL;
char*          char_ptr = NULL;
char*          slashch = NULL;
time_t         wallclock_time;
FILE*          job_request_file;//for multiple jobs
char*          job_request_filename = NULL;
int            rur_seq_no = 0;
int            request_no = 0;
char           request_buffer[100];
//***** GRIDBANK MODIFICATION END *****
```

This will set up the variables required by GridBank for each job. Continue onward and search for the following string:

```
job_desc->sessiondir=i->session_dir;
i->local=job_desc;
```

Below it, insert the following code:

```
//***** GRIDBANK MODIFICATION *****

memset(request_buffer,0,sizeof(request_buffer));

wallclock_time = time(&wallclock_time);//record wallclock time

//Resource Usage Record: job request acknowledgement

string tmps;
tmps=i->local->DN;
make_escaped_string(tmps, '');

rur_globus_id = (char*)malloc(tmps.length());
memset(rur_globus_id, 0,tmps.length());
strcpy(rur_globus_id, tmps.c_str());

//replace '/' and ' ' (space) characters with '_' for filenames

char_ptr = rur_globus_id;

while(char_ptr != '\0') {
    slashch = strchr(char_ptr, '/');
    if(slashch==NULL)
        break;
    *slashch = '_';
    char_ptr = slashch+1;
}

char_ptr = rur_globus_id;

while(char_ptr != '\0') {
    slashch = strchr(char_ptr, ' ');
    if(slashch==NULL)
        break;
    *slashch = '_';
    char_ptr = slashch+1;
}
```

Cluster and grid computing

```
}

//create job request filename

job_request_filename = (char*)malloc(strlen(rur_globus_id)+15);
memset(job_request_filename,0,strlen(rur_globus_id)+15);
strcpy(job_request_filename,"/tmp/");
strcat(job_request_filename,rur_globus_id);
strcat(job_request_filename,".requests");

//create job request file

job_request_file = fopen(job_request_filename,"r+");
if(job_request_file!=NULL) {
    while(fgets(request_buffer, sizeof(request_buffer),
job_request_file)){
        char_ptr = request_buffer;
        while(*char_ptr != ':')
            char_ptr++;
        *char_ptr = 0;
        request_no = atoi(request_buffer);
        if(request_no > rur_seq_no)
            rur_seq_no = request_no;
    }
    rur_seq_no++;

    //assign the job a unique sequence number
    i->rur_seq_no = rur_seq_no;

    job_request_file = freopen(job_request_filename, "a",
job_request_file);
    fprintf(job_request_file,"%i:%s:%s", i->rur_seq_no, i->job_id.c_str(),
asctime(gmtime(&wallclock_time)));
    fflush(job_request_file);
    fclose(job_request_file);
}
else //file does not exist
{
    job_request_file = fopen(job_request_filename,"w");
    fprintf(job_request_file,"1:%s:%s", i->job_id.c_str(), asctime(gmtime
(&wallclock_time)));
    rur_seq_no = 1;
    i->rur_seq_no = rur_seq_no;
    fflush(job_request_file);
    fclose(job_request_file);
}

//end of job request acknowledgement for resource usage record
//***** GRIDBANK MODIFICATION END *****
```

Once a job has reached accepted state, this modification will create a requests file holding the job id and timestamp of each job submitted by a user in a queue. The requests file uses the format DN.requests where DN is the subject of the user certificate (spaces and slashes are replaced with an underscore). E.g. `_O=Grid_O=NorduGrid_OU=hig.no_CN=Tarjei_Mandt.requests`. Should a user submit multiple jobs, the requests file will contain multiple entries, one line per job submitted.

Find the string:

```
if(i->job_state == JOB_STATE_FINISHED) {
```

Below, insert:

```
//***** GRIDBANK MODIFICATION *****
//*****RECORD RESOURCE USAGE*****

resource_usage_filename = (char*)malloc(strlen(rur_globus_id)+50);
memset(resource_usage_filename,0,strlen(rur_globus_id)+50);
strcpy(resource_usage_filename,"/tmp/");

if( strcmp(rur_globus_id,"unknown_globusid")==0 )
    strcat(resource_usage_filename, "Unauthenticated_client.rur");
else {
    strcat(resource_usage_filename,rur_globus_id);
    strcat(resource_usage_filename, ".rur.");
    memset(request_buffer,0,sizeof(request_buffer));
    sprintf(request_buffer,"%i",i->rur_seq_no);
    strcat(resource_usage_filename,request_buffer);
}

if((resource_usage_file=fopen(resource_usage_filename,"a"))==NULL) {
    i->AddFailure("JM: failed to open resource usage file\n");
}
if(getrusage(RUSAGE_CHILDREN, &resource_usage) ) {
    i->AddFailure("JM: failed to get resource usage file\n");
    if(resource_usage_file != NULL)
        fprintf(resource_usage_file, "getrusage failed\n");
}
else {
    string tmps;
    tmps=i->local->DN;
    make_escaped_string(tmps, '');

    hostent * hostnn;
    char hostn[128];
    gethostname(hostn, 128);
    hostnn = gethostbyname(hostn);

    fprintf(resource_usage_file,"Resource Usage Record (Globus v2)");
    fprintf(resource_usage_file,"\nclient certificate subject name:%s",tmps.c_str());
    fprintf(resource_usage_file,"\nclient contact:%s",i->local->clientname.c_str());
    fprintf(resource_usage_file,"\nresource certificate subject
name:nordugrid-cluster-name=p02081.hig.no,Mds-Vo-name=local,o=grid");
    fprintf(resource_usage_file,"\nresource address:p02081.hig.no");
    fprintf(resource_usage_file,"\nresource manufacturer:HiG");
    fprintf(resource_usage_file,"\nresource cpu type:i386");
    fprintf(resource_usage_file,"\nresource os:Linux - Debian");
    fprintf(resource_usage_file,"\nresource os version:2.4.19");
    fprintf(resource_usage_file,"\nresource domain name:hig.no");
    fprintf(resource_usage_file,"\nresource condor os:NA");
    fprintf(resource_usage_file,"\nresource org domain name:hig.no");
```

Cluster and grid computing

```
fprintf(resource_usage_file, "\nresource gate host:%s", *hostnn);
if(i->job_state == JOB_STATE_FINISHED)
    fprintf(resource_usage_file, "\njob status:done");
else
    fprintf(resource_usage_file, "\njob status:failed : %s", i-
>failure_reason.c_str());
    fprintf(resource_usage_file, "\nexecutable:%s", i->local->jobname.c_str());
    fprintf(resource_usage_file, "\nstart date:%s", asctime(gmtime
(&wallclock_time)));
    wallclock_time = time(&wallclock_time); //get current time
    fprintf(resource_usage_file, "end date:%s", asctime(gmtime
(&wallclock_time)));
    fprintf(resource_usage_file, "jobmanager job id:%s", i->job_id.c_str());
    fprintf(resource_usage_file, "\nnumber of processes:%s", "todo");
    fprintf(resource_usage_file, "\nproject:%s", "todo");
    fprintf(resource_usage_file, "\nuser time used (s):%i", (int)
resource_usage.ru_utime.tv_sec);
    fprintf(resource_usage_file, "\nuser time used (us):%i", (int)
resource_usage.ru_utime.tv_usec);
    fprintf(resource_usage_file, "\nsystem time used (s):%
i", resource_usage.ru_stime.tv_sec);
    fprintf(resource_usage_file, "\nsystem time used (us):%
i", resource_usage.ru_stime.tv_usec);
    fprintf(resource_usage_file, "\nmaximum resident set size:%i",
resource_usage.ru_maxrss);

    fprintf(resource_usage_file, "\nintegral shared memory size:%i",
resource_usage.ru_ixrss);
    fprintf(resource_usage_file, "\nintegral unshared data size:%i",
resource_usage.ru_idrss);
    fprintf(resource_usage_file, "\nintegral unshared stack size:%i",
resource_usage.ru_isrss);
    fprintf(resource_usage_file, "\npage reclaims:%i",
resource_usage.ru_minflt);
    fprintf(resource_usage_file, "\npage faults:%i", resource_usage.ru_majflt);
    fprintf(resource_usage_file, "\nswaps:%i", resource_usage.ru_nswap);
    fprintf(resource_usage_file, "\nblock input operations:%i",
resource_usage.ru_inblock);
    fprintf(resource_usage_file, "\nblock output operations:%i",
resource_usage.ru_oublock);
    fprintf(resource_usage_file, "\nmessages sent:%i",
resource_usage.ru_msgsnd);
    fprintf(resource_usage_file, "\nmessages received:%i",
resource_usage.ru_msgrcv);
    fprintf(resource_usage_file, "\nsignals received:%i",
resource_usage.ru_nsignals);
    fprintf(resource_usage_file, "\nvoluntary context switches:%i",
resource_usage.ru_nvcsw);
    fprintf(resource_usage_file, "\n involuntary context switches:%i\n",
resource_usage.ru_nivcsw);
fclose(resource_usage_file);
}

globus_libc_free(resource_usage_filename);

//***** GRIDBANK MODIFICATION END *****
```

Note: The hardcoded property values should be altered to represent your setup.

FILE: <nordugrid_source_path>/grid-manager/jobs/job.h

In the JobDescription class, under private, add the following member:

```
/* rur sequence number */
int rur_seq_no;
```

This is needed in the event of multiple job submissions taking place. It assigns each job in a queue a unique sequence number so that wallclock time is calculated correctly.

Alternatively, you may use plugins instead. This requires you to compile the code above into an executable, runnable from console. The idea is to have it generate the needed files at the different job stages. Example, in nordugrid.conf in [grid-manager], you would put something like:

```
authplugin="SUBMIT 100 /opt/gridbank/gb-make-request %C %I"
authplugin="FINISHED 100 /opt/gridbank/gb-make-rur %C %I %S %U"
```

This completes the NorduGrid middleware source modification. To compile it, use the following commands:

```
./configure --with-nordugrid-path=/opt/nordugrid
make
make install
```

Additionally, you must restart the grid-manager if you already had it running.

```
/etc/init.d/grid-manager restart
```

The next steps involves changes to the GridBank source code

3. Modify GridBank Server

FILE: <gridbank_source_path>/server/protocols/ServerMain.cc

Find the function:

```
globus_bool_t gridbank_secure_authorization_callback(
    void *                arg,
    globus_io_handle_t *  handle,
    globus_result_t       result,
    char *                identity,
    gss_ctx_id_t *        context_handle);
```

Change it to:

```
globus_bool_t gridbank_secure_authorization_callback(
    void *                arg,
    globus_io_handle_t *  handle,
    globus_result_t       result,
    char *                identity,
    gss_ctx_id_t         context_handle);
```

Cluster and grid computing

The only change here is the pointer in the last variable declaration. While the pointer was present in Globus Toolkit 2.0, it was removed in later versions. Also note that the function is present in two different parts of ServerMain.cc and needs to be changed both places.

4. Modify GridBank makefiles

Go through `_all_` makefiles and remove all instances of `-lglobus_ssl_utils_gcc32dbg` as this library no longer exist in Globus Toolkit 2.4.3. Also, set the correct paths for java (step 15) and Globus.

5. Compile soapcpp

```
cd /gridbank/soapcpp-linux-2.1.6
make
```

Read the install or readme if you need more information.

6. Build Xerces-C++

```
export XERCESCROOT=<path to xerces>
cd $XERCESCROOT/src/xerces
autoconf
runConfigure -plinux -cgcc -xg++ -minmem -nsocket -tnative -rpthread
make
```

Please note that as of now, GridBank only works with the bundled version of Xerces (2.1).

7. Compile the GB server.

```
cd <gridbank path>/server
sh install
```

Note: If you receive undefined reference errors to Globus libraries, a quick work-around is to add your globus library path to `LD_LIBRARY_PATH`:

```
export LD_LIBRARY_PATH=$GLOBUS_LOCATION/lib
```

Another way is to edit the makefile with the following:

```
-Wl,-rpath,$(globus_location)/lib
```

Place it in the line where you specify the other Globus locations.

8. Install mySQL

If you haven't already installed mySQL, `mysql-3.23.51.tar.gz`, is included in the gridbank installation bundle. The following installation instructions will install mysql in `/usr` which is preferred at this time. You may change the path as long as you change the corresponding paths in the GridBank makefiles as well.

Cluster and grid computing

```
./configure --prefix=/usr
```

For further installation instructions, please see the available MySQL documentation [36].

9. Create the GridBank database

```
cd gridbank/server/accounting
mv Makefile MakefileNormal
mv MakefileCreateDB Makefile
make
su
./createdb -admin <Administrator Certificate Name> -bank <bank number>
          -branch <branch number>
```

example: `./createdb -admin "/O=Grid/O=NorduGrid/OU=hig.no/CN=Tarjei Mandt" -bank 1 -branch 1.`

This creates the mysql database `gridbank1_1` with the given DN set at administrator.

```
exit
mv Makefile MakefileCreateDB
mv MakefileNormal Makefile
rm createdb
rm *.o
```

GridBank account numbers consist of the following: 2 digits for bank number, 4 digits for branch number, and 8 digits for account number, eg. 01-0001-00000001. The branch number has to be unique.

10.Run the GridBank server

A root, start the GridBank server with the following parameters.

```
./gbserver [-port 2500] -dbname gridbank<bank no.>_<branch no.>
```

example: `./gbserver -dbname gridbank1_1`

The port parameter is optional (default = 2500)

11.Modify GridBank Charging Module

FILE: `/gridbank/gbcm/secureapi/GBCMMain.cc`

Find the function:

```
globus_bool_t gbcm_authorization_callback(
  void *          arg,
  globus_io_handle_t * handle,
  globus_result_t result,
  char *         identity,
  gss_ctx_id_t * context_handle);
```

Cluster and grid computing

Change it to:

```
globus_bool_t gbcm_authorization_callback(  
    void *                arg,  
    globus_io_handle_t *  handle,  
    globus_result_t       result,  
    char *                identity,  
    gss_ctx_id_t          context_handle);
```

Like in step 3, this function appears twice and needs to be changed both places.

12.Edit GridBank Template Module

FILE: /gridbank/gbcm/template_accounts/template_accounts.cc.

Find:

```
for(int i=0; i < index; i++)  
    fputs(accounts_pool[i],accounts_file);
```

Below (not part of for-loop), insert:

```
fflush(accounts_file);
```

Without this change/fix, the template-accounts file may be left empty after allocation of template accounts.

13.Edit GBCM GridMeter files

FILE: /gridbank/gbcm/gridmeter/GridMeter.h

comment out the line:

```
// #include <linux/quota.h>
```

FILE: /gridbank/gbcm/gridmeter/Gridmeter.cc

comment out both occurrences of resource_usage_filename:

```
// char* resource_usage_filename;  
// free(resource_usage_filename);
```

14.Compile the GridBank Charging Module

Edit gbcm.conf and make sure it has the correct values. Compile and run as root.

```
sh install  
./gbcm
```

Cluster and grid computing

15. Install Java SDK and set paths for GBPM makefiles

Download and install the latest J2SE SDK (1.4.x).

```
./j2sdk-1_4_2_<version>-linux-i586.bin
```

Once completed, update the java paths in the /gridbank/gbpm makefiles.

16. Compile the GridBank Payment Module

Edit gbpm.conf and make sure it has the correct values.

```
cd /gridbank/gbpm
sh install
```

17. Start the account management GUI

```
grid-proxy-init
Java GBClient
```

Note: If you receive the error:

Exception in thread "main" java.lang.UnsatisfiedLinkError: no gbapi in java.library.path , add gridbank/gbpm path and globus lib path to LD_LIBRARY_PATH.

18. Create an account

In the java GUI, click on “open new account”, and enter in the required information. You might want to set up an account for yourself first, as users need a destination account to transfer credits. This should create account xx-yyyy-00000001. The x's represent bank number (gridbank = 01), while the y's represent branch number. These values will correspond to the values you set earlier when creating the database.

Once the first account has been set up, continue by setting up accounts for your resource users.

You may want to check if your users have been created correctly in mysql (replace 1_1 with bank and branch number, no leading zero's):

```
mysql -> use gridbank1_1 -> select * from accounts;
```

19. Finalizing

Clean your grid-mapfile (!) and add the host DN of the gserver (mapped to any account... not sure what this does but GBCM doesn't authorize the gserver itself if it isn't included).

```
"/O=Grid/O=NorduGrid/CN=host/p02081.hig.no" grid
```

This will restrict access to anyone who hasn't paid. Alternatively, you may add users in the grid-mapfile who are not eligible to pay as GridBank will automatically authorize users already

Cluster and grid computing

present in this file.

Edit `/etc/grid-security/template-accounts` and add all local system accounts (seperated by new lines) you want users to be able to allocate for grid jobs, e.g. 'grid'.

20. Test your setup

Everything should now hopefully function to some extent. The easiest way to test it is to compile the GridBank payment module on another machine, initialize grid proxy and run `gp-pre-pay`.

Make sure you edit `gbpm.conf` with the user account information (the one that is to be charged on pre payment).

```
./gb-pre-pay -amount 200 -destacc 01-0001-00000001 p02081.hig.no
```

If you receive a 'sucessfully allocated' message, it means you are good to go and may submit your job to the cluster. In any other case, please see the troubleshooting section provided below.

```
ngtest 1 -c p02081.hig.no
```

4.2.2 Additional Information

pre pay usage:

```
gb-pre-pay -amount <Amount> -destacc <Account> <Host>
```

Transfers `<Amount>` of currency from client account (set in `gbpm.conf` configuration file) to resource provider `<Account>`. `<Host>` is the internet address of provider where confirmation will be sent. `<Amount>` is a float (e.g. 123.456), which specifies how much currency to transfer into `<Account>`, which is the GridBank account number (e.g. 01-0001-00000001) of the resource.

4.2.3 Troubleshooting

gbserver: Permission denied

Make sure `mysqld` is started:

```
mysqld_safe &
```

gbserver: error while loading shared libraries: libglobus_io_gcc32dbg.so.0

Set `LD_LIBRARY_PATH` to globus library directory

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/globus/lib
```

gb-pre-pay: Can't connect: a system call failed (Connection refused)

Make sure your services are running on the proper ports:

```
netstat -tapn | grep gb
```

gb-pre-pay: segmentation fault

Again, make sure the services (gbserver and gbcm) are running

gbserver: Failed to accept/authorize connection

Make sure your certificate is initialized and that you are added to the account database. Note that adding a simple record to the account table in MySQL will not work as the account relies on other information as well.

SOAP FAULT: SOAP-ENV:Client

‘Method 'gb:confirmPayment' not implemented’

This means something is wrong with the client or server side xml templates. Normally, this shouldn't happen unless you mess around with the soap xml files. Compile stdsoap2.c with -DDEBUG to generate soap logs (RECV.LOG, SENT.LOG, TEST.LOG).

gbserver: globus_memory.c:165: globus_memory_pop_node: Assertion `s_mem_info != 0' failed.

Critical bug/error (due possibly to ng globus 2.4?)! Invalid gb-pre-pay parameters may cause the gbserver to crash. This includes spending more credits than currently available, paying to a non-existing account, etc.

4.2.4 Patch

At the moment there is no patch available for applying these modifications to the NorduGrid/GridBank code. We discussed the possibility of making a patch, but due to the rapid development of the NorduGrid software, a patch would be useless in a very short time.

4.3 How GridBank works in practice

The cluster administrator adds the user who would like access to the cluster by opening a new account in GridBank. The user will then pay for the use of the cluster by purchasing grid credits with real money. When the payment has been received/confirmed, the administrator deposits the amount paid to the user's account (in form of grid credits).

GridBank is based on pre payment and therefore requires users to pay in advance (using grid credits) before submitting new jobs. Users pay to the cluster targeted for the job and is granted access once the payment has been authorized. Authorization is granted if the user is present in the local GridBank account database (MySQL based). Each cluster must maintain their own GridBank database, in which also administrators and transfer records are stored. To administer the database, GridBank provides a java client, in which you may open new accounts, make deposits/withdrawals, add administrators and perform database transaction queries. Once user authorization has taken place, the GridBank meter charges the account with the amount paid.

When a user has pre-paid, GridBank allocates an available local system account from the template file (/etc/grid-security/template-accounts) and maps the user DN (fetched from the database) to the system account in the grid-mapfile. The template-accounts file contains a list of local system accounts that GridBank may allocate. The grid-mapfile is originally empty, denying all users who

haven't paid for jobs (the gridmap-file may also contain users who are not eligible to pay). The GridBank charging module will now begin to monitor the local /tmp directory for a requests file. This file contains all jobs (sequence number, job id and wallclock time of job start; one job per line) submitted by the user in a given queue. E.g. a user may pay once and then submit 4 consecutive jobs. As long as there is a job running, a user may submit additional jobs. The record is added to the requests file when the job has reached ACCEPT state (it would be better if SUBMIT state was used instead as wallclock calculation would be more accurate...) GridBank will then scan through it and start looking for RUR (resource usage record) files at regular intervals. These files are made at job completion and contain all information relevant to the job such as system time used, number of processes, job executable, etc. The charging module of the resource provider reads the file and converts the data into an XML-based resource usage record (which is transferred to GridBank). When all RUR files have been processed and stored in the database, the template account is deallocated. If the user wishes to submit another job, he/she will have to pre pay again.

4.4 NorduGrid installation guide

4.4.1 NorduGrid quick client installation guide

The aim of this document is to provide a quick and easy way of installing the NorduGrid client software onto a clean (no previous NorduGrid installation) Linux system. The following procedure has been tested on a Linux Debian 3.0 distribution, using kernel version 2.4.19.

1. Download and install the NorduGrid stand-alone client [31] in the desired path. Normally, you would just extract it in your home directory. If you use a Linux distribution that is not listed, you must compile it yourself, or ask the NorduGrid developers if they will be kind enough to do it for you.
2. Execute *source setup.sh* (or *setup.csh*) in the directory where installed. Alternatively, you may use a *.* (dot) instead of the source command. This script will export the proper user environment variables such as `NORDUGRID_LOCATION` and `GLOBUS_LOCATION`.
3. Run: `<installation directory>/bin/grid-cert-request -int -ca`
In this process you will be required to select the CA you want to sign your certificate. Participants of Nordic countries should here choose 'NorduGrid'. Furthermore, you will be required to provide a private key which will be used whenever you want to gain access to the grid. Leave the Organization Level 0 and 1 Name default, set your domain to the domain you intend to be accessing the grid from, and, last, specify your name and email address.
4. Send the generated `usercert_request.pem` file (located in `~/globus/`) to the address specified upon generation (or in most cases `ca@nordugrid.org` or `ca@nbi.dk`) along with the subject line:
`/O=Grid/O=NorduGrid/OU=domain.com/CN=Your Name/Email=youraname.com`
If everything is in place, your certificate will be signed in 1-2 working days.
5. In the email returned by the CA, copy the part that begins with `----BEGIN CERTIFICATE----`. This information is your signed key, and should be placed in the empty file `usercert.pem` (located

in `~/globus/`). Once done, make sure all the `.pem` files are owned by the user who will access them (`chown user:user *.pem`).

6. In order to submit jobs, you also require authorization to the individual grid resources. This is done through the NorduGrid VO [37], which basically consists of groups of people authorized to run grid jobs on a set of grid resources. In order to get added, send a request to `nordugrid-support@nordugrid.org`.
7. Once added, it will take a short period of time before the individual clusters are updated. You can check which resources you have access to from the NorduGrid VO [38].
8. You should now be able to access the grid by running '`<installation directory>/bin/grid-proxy-init`'. Issue a test by running '`ngtest 1`' which will echo "Hello, Grid!". Your jobs can be accessed with '`ngcat -a`' and retrieved with '`ngget -a`'. To specify a cluster for the job, use '`-c <hostname>`'.

You can find additional information from [39] and [40].

4.4.2 NorduGrid quick server installation guide

The aim of this document is to provide a quick and easy way of installing the NorduGrid middleware onto a clean (no previous NorduGrid installation) Linux system. The following procedure has been tested on a Linux Debian 3.0 distribution, using kernel version 2.4.19.

All downloadable packages can be downloaded from the NorduGrid FTP site [31].

1. Download and install `globus`, `globus-config`, and `gpt` packages. Extracting these from / (root) using `tar -xvfz <package>` will place the files in the default location (`/opt/globus` and `/opt/gpt`).
2. Create unix accounts dedicated for use on the grid. If you are going to use `cp` instead of `rcp/scp` (see PBS guide) for moving completed jobs from nodes to the front-end, you must create grid accounts both on the nodes and the front-end using the same UID. If this requirement isn't met, the front-end will make the directory for the completed job with its local grid account UID. The node that has the grid account with a different UID will then see the mounted grid area as:

```
drwx-----  2 1002  1002  4096 Apr 20 23:55 302901082498137230085981
```

This will result in a "permission denied" when attempting to move the completed job files. To change the userid of a user, see '`man usermod`'.

3. Set up the shared disk space on the front-end.

| <i>Function</i> | <i>Description</i> | <i>Example</i> |
|------------------------------|---|--------------------------------|
| grid area (required) | Session directories for grid jobs | /scratch/grid |
| Cache directory | Shared input files of grid jobs | /scratch/cache |
| Runtime environment scripts | Initialization scripts for pre-installed software | /SOFTWARE/runtime |
| Control directory (required) | Internal control files for grid-manager | /var/spool/nordugrid/jobstatus |

Figure 6: How to set up shared disk space on the front-end in NorduGrid server installation

You must also export these directories (except the control directory) so they may be accessed on the nodes. Here follows an example of what you may put in your `/etc/exports` file on the front-end.

```
/scratch/grid          grid*.hig.no(rw, sync, root_squash)
/scratch/cache        grid*.hig.no(rw, sync, root_squash)
/SOFTWARE/runtime     grid*.hig.no(rw, sync, root_squash)
```

4. Set up and configure the PBS (see PBS guide chapter 4.1).
5. Download and install the NorduGrid middleware packages. Nordugrid-server is required, while nordugrid-client, nordugrid-devel and nordugrid-doc are optional but recommended.
6. In order for your cluster to be able to function, you must request a host certificate from the CA (Certificate Authority), and also preferably an LDAP certificate. These must be placed in `/etc/grid-security` (host certificate) and `/etc/grid-security/ldap` (ldap certificate) on the frontend. To generate a host- and an LDAP certificate, use the following commands:

```
grid-cert-request -host <my.host.fqdn>
grid-cert-request -service ldap -host <my.host.fqdn>
```

`<my.host.fqdn>` is your fully qualified domain name (consists of host, domain name, and top-level domain). Send the generated keys by e-mail to the corresponding CA for signing.

Once signed, fill the empty certificate files (`hostcert.pem` and `ldapcert.pem`) with the signed certificates and make sure that all files are owned by root (`chown root:root *.pem`). The private keys should only be readable by root and none of the files should be executable.

7. Download and install all packages with the public keys of each individual CA you wish to authorize on your cluster. By default, you should get the CA of which the cluster is a member of (see host certificate request).
8. Download and install `nordugrid-ca-utils` from the NorduGrid downloads area to periodically check and maintain the list of revoked personal and service certificates (CRL). Outdated CRL will render your site unusable.

Cluster and grid computing

9. Define the groups and local unix accounts (step 2) you wish for each group to use in `/etc/grid-security/nordugridmap.conf` . Assuming you also want to join NorduGrid, download and install `nordugrid-gridmap-utils`, a set of tools which will automatically update `/etc/grid-security/gridmap-file` with the NorduGrid userbase mapped to the local accounts set in `nordugridmap.conf`.
10. Create and configure `/etc/nordugrid.conf` using the template on the NorduGrid CVS server[41].
11. Create and configure `/etc/globus.conf` using the template on the NorduGrid CVS server[42].
12. Start the required services from root:

```
/etc/init.d/gridftp start  
  
/etc/init.d/globus-mds start (or /etc/rc.d/init.d/globus-mds start)  
  
/etc/init.d/grid-manager start
```

13. If errors occurred while starting the services, make sure to check the following log files:

```
/var/log/infoproviders – Information System  
/var/log/globus-mds – Information System  
/var/log/gridftp.log – Gridftp server  
/var/log/grid-manager.log – Grid Manager  
/var/log/gm-jobs.log – Job logging informations
```

We strongly recommend that you also read and study the documentation [35 | 43 | 44] provided by NorduGrid when setting up a cluster.

Chapter 5. Evaluation and conclusion

5.1 Administrating and maintaining a NorduGrid cluster

How much time is required to administer a cluster? The answer to this question depends on your current knowledge of system administration and grid computing. While we were faced with several challenges throughout the project, both in terms of understanding grid computing and dealing with Linux related issues, we can safely conclude with that once a cluster is properly set up, it is actually quite easy to maintain and administer. The time you end up spending is actually very little, and sometimes you may even forget its there.. However, there are a few things you need to be aware of.

First, you need to make sure the list of revoked personal and service certificates is updated regularly. By installing the nordugrid-ca-utils package, this will be done automatically. If the CRL (Certificate Revocation List) is outdated it will render your site unusable (CRL errors will occur on job posting). Since you also probably would want NorduGrid members gain access to your cluster, you need to make sure your gridmap file is up-to-date. This can also be done automatically by installing the nordugrid-gridmap-utils package. It will periodically, through the use of cron jobs, update your gridmap file.

You must ensure that the PBS node file is up-to-date. In this context, it may be advisable to check the nodes now and then to make sure they are up and running (type 'pbsnodes -a' in console). If you want to add new nodes or remove nodes, you simply edit the PBS node file. However, for PBS to update the nodes list you are required to restart the pbs_server, and if you have running jobs on your cluster it's generally a bad idea to do this. Therefore it may be wise to add nodes by using the qmgr in addition to placing it in the node file. This is explained more thoroughly in the PBS chapter, and even more in-depth in the PBS manual (although this is over 100 pages and it's quite easy to get lost).

Every once in a while you may also want to upgrade the NorduGrid server software. While the existing versions today are considered reasonably stable (0.4.1+), it's always recommended to use the latest release. Upgrading is considered a simple task as NorduGrid provide pre-compiled binaries to the most common Linux distributions (Debian, Red Hat, Suse, Mandrake etc). Extracting these from root will place the binaries into the proper directories (unless your NorduGrid installation is not in default location). After that, only a restart of the NorduGrid services (gridftpd, globus-mds, and grid-manager) is required. In case you use a non-listed Linux version, you can always download the source and compile it yourself (although this should be done on a reasonably fast computer as it may take a while...). You won't have to worry about upgrading the nodes as they only rely on the PBS connectivity. However, if you should decide on changing the PBS flavour (for instance, switch from OpenPBS to Torque), you are required to do this on every node as well.

It may also be requested/required that you install certain runtime environment scripts from time to

time. These are simple bash scripts executed to help jobs benefit from installed applications by setting paths and variables. For instance, certain jobs will require that you have the Atlas software kit installed. The runtime environment scripts are easily manageable and placed into a predefined directory set in the NorduGrid configuration file. Runtime environments are made available on the worker nodes as well, hence they only need to be placed on the front-end.

5.2 Unfinished parts in GridBank

After some testing and discussion with Aleksandr Konstantinov in Oslo we got some of his points of his view on the GridBank software as it is now. In contrast to the SGAS software, GridBank is somewhat a much lighter software than the SGAS system. A much lighter software will be easier to install on a new system and is easier to maintain while in use, and usually uses less resources.

When you open an account in the GridBank, you get your own personal account with an account number, e.g. 01-0001-00000004. Only you have access to this account, but in a grid environment users should have access to community accounts as well. If a user uses resources he is most likely to be working on a project with, then the usage for the job should be debited to an account belonging to the project, not the user itself. Before the project is started, it would have access to limited funds, and the users attached to the project would have access to use funds from the project/community account. This is at the moment not supported in with GridBank.

When we visited Aleksandr at the Department of Physics (UiO) in Oslo and presented him with the somewhat working parts of GridBank, Aleksandr pointed out something very important to us. After job submission, when jobs are completed, the resource user's name is removed from the gridmap file, thus denying the user access to download his output files. This have to be worked around in some way. A possible solution could be to have several gridmap files, so that GridBank could have its own access list for users. Another possibility could be to change the access policy. All users with positive account balance could be allowed to post jobs, and if all users with positive account balance are listed in the gridmap file, the users could submit and retrieve job output as long as they don't overdraw their account.

Aleksandr also suggested the possibility of using account names, not account numbers, which would make it easier for users to remember the account name. The DN field in the X.509 certificates could for example be used as an alias for account numbers, allowing users to define accounts by using the DN of a grid user. By implementing this into the user interface for NorduGrid, it would also solve another problem. Currently, you have to specify what cluster to pre-pay to, and with NorduGrid's combined user interface and broker this causes a problem because you don't know what cluster the job will be submitted to. If you don't know what cluster the job will be submitted to, you can't pre-pay to it, but with the pre-pay integrated into the user interface, the user interface can pre-pay the job between resource brokering and job submission. It first decides where to submit the job, pre-pays it, and then submits the job to the cluster.

The pre-pay function also doesn't seem to be limited to pre-pay to resource owners. Tests showed that you could pre-pay to anyone with an account in GridBank and still get access to the resources. A check should be performed, and the cluster itself should have an account that users would have

Cluster and grid computing

to pay to. This could work in conjunction with the community accounts, so other users can directly spend the earned credits, if the project/community account also is the provider of the resources. A community account administrator can then also transfer funds to other private accounts on a per-user basis.

When the job is pre-paid and after the job is finished running on the node, the GridBank charging module picks up the .rur file and processes it. If the job cost is less than what the user has pre-paid, the rest of the funds should be returned to the users account. At the moment the GridBank charging module will charge the user with all the money that was pre-paid, hence overcharging the user compared to what resources he actually used.

The modulation of GridBank will make it easier to implement new features and functionality later because if we alter the behaviour of one of the modules, it will still have to communicate in the same way with the other modules in order to work.

Batch/Fork jobs

Aleksandr also mentioned that it might seem like that the GridBank system right now is more suitable for fork jobs, not batch processing like in NorduGrid. When jobs are executed as a forked job, they are running on the same computer as the job is submitted to. With batch processing jobs can be distributed on a several different nodes for execution, and if GridBank should be used in a system like this, there would have to be some kind of scripts that generate the rur files on the nodes itself and transfer them to the front-end node.

List of implemented functions in GridBank

As of now, only the most vital and simplest functions are implemented into GridBank.

- add administrator
- add account
- withdraw from account
- deposit to account
- query transaction records
- pre pay
- request gridhash (pay-as-you-go, however, not fully implemented)
- lock funds (avoid overspending by transferring funds from available balance to locked balance)

Not implemented

- metering of data storage resources (i.e. metric GB*hour)
- community/shared accounts
- cheque (request/redeem)
- cancel account
- overspent money is not refunded
- cancel transfer (no action taken on overspending, although warning is given)
- change credit limit
- redeem gridhash
- user-friendly account names

5.3 GridBank in the future

At the moment GridBank is somewhat working, but not usable for the public crowd. A lot of things need to be fixed and the unfinished parts will have to be finished for it to be a usable solution for NorduGrid. We tried to contact Alexander Barmouta, one of the developers of GridBank to ask him some questions, but we did not get any answer from him. Rajkumar Buyya, another developer of GridBank, did however answer us, telling us that Alexander might take up the development later on. Quote from the email from Buyya: "Alex who is the main developer of GridBank is currently finishing his Masters thesis (almost written) and he is interested continuing with the development on his spare time as he has taken up job now". Obviously, we won't have time to do any more right now, but further development is possible, perhaps by students working in cooperation with Barmouta.

If further development is going to take place, chances are it might not be pointed specifically towards NorduGrid, but more in the direction of a general accounting/banking system for grid computing. This is again a decision to be taken by anyone that want to work on a solution in the future. The problem with working further on with this as a student project is the fact that there is a lot you need to know before you can really start to do things. Institutions today even offer extensive grid computing courses to people interested in working with grid computing. During the course of our project, a lot of time was used for reading and preparation.

5.4 Different phases or increments in the project

When we started we didn't have much knowledge about grid computing and our task wasn't very defined. After some time we knew a little better what we had ahead, but we were still very uncertain about how to approach the project. In the beginning we knew what we had to do, but when we started to implement GridBank into NorduGrid it was very hard to follow any real development model, simply because we were not sure what had to be done. We didn't really know what parts that would work out from the box and what parts that needed a lot of work. Therefore the development was very "loose", as most work consisted of finding parts to do, and then do it if it needed work.

The project can roughly be divided into different phases:

- Define project goals
- Gather information
- Make a conclusion from the information gathering
- Make the solution working

Alongside with these different phases, the work of setting up the grid internally/externally was always "running". Different versions being released and testing of different programs and systems caused this work to always be there. These different phases could be looked at as different increments in the project. Especially the 3 first phases, since each one of them had to be completed in order to move on to the next phase.

Comments on the milestones in the project

| | |
|--|-------------------|
| Hand in the pre-engineering project This went like planned with no problems. | 29.01.2004 |
| Decide what application to integrate into NorduGrid This wasn't really finished before mid-march. Information gathering and analysis did take a little longer than anticipated. The search for usable systems to implement especially did take a little longer, since we at the same time had to read a lot about NorduGrid and grid in general. | 01.03.2004 |
| Finish the integration of the chosen system into NorduGrid This phase pretty much went on up until the deadline. We got the software up and running to some extent and then focused on testing to see what parts that were missing and didn't work in the solution. | 04.05.2004 |
| Hand in the main project report Not completed yet, but so far we are on schedule. | 14.05.2004 |
| Present the project Not completed yet. | 26.05.2004 |

5.5 Problems in the project

When we started the project we were really uncertain on what to do and how to approach it. Some communication with the NorduGrid developers helped us out on how to “get started” with the project and how to approach the problem. Still we were very much stumbling in the dark, since the grid computing field is very big, and a lot of different technologies had to be looked into.

5.5.1 Different software and software versions

When we first started to install and compile different software modules on the our computers, we soon discovered the “hell” of having different programs dependent on other different programs in different versions needing different compiler versions. (Did I mention that the different compiler versions are dependent on different libraries in different versions?). This used up a lot of our time when trying out new software, but at the same time we learned a lot more about the Linux OS, as these problems required a lot of poking around in the OS. A big thank you goes out to Google for helping us out a lot of the times here.

5.5.2 Server instability

Early in the project we discovered some instability in our front-end node. We used some time to try to fix it, but nothing worked out really good. For some time we kept the server running by rebooting it every two days or so. This is by no means a good solution, but at the time it seemed like easiest way to not waste a lot of time on reinstalling everything. Later on in the project when NorduGrid released the 0.3.6 release and we tried to compile this, the computer would totally lock up in the middle of the compiling. Trying several things did not work out, so we decided to reinstall everything from scratch on a new disk. To our disappointment this didn't work either, the computer would still lock up in the middle of the compiling. Even poking around with the IRQ settings didn't solve the problem. As a last resort we stripped the computer for unnecessary hardware and disks, and finally, this seemed to do the trick. Everything now went smoothly with no instability issues. Even though we solved the problem, it bothered us and, for a while, used up a lot of our time.

5.5.3 Software documentation

When we started the software installation in February, the installation documentation seemed a little out of date. This made things a little harder to install since it required some guessing to finally get it going. Later on, in April when NorduGrid released their first official release (NorduGrid ARC 0.4), the documentation was more up to date due to the official release.

5.6 What we have learned in this project

We hope this report has reflected some of the technical aspects we have faced and dealt with during this project. This includes a whole lot of information about grid computing, different grid projects in the world, and cluster computing around on different sites. We realized how important computational power is to different organizations like Cern and biochemical companies. In addition to the knowledge acquired on grid computing, we have also learned about new technologies, such as the SOAP structure and uses of the Xerces parser system. Even though we had the “system administration” course before Christmas, we have also learned a lot more about the Linux OS and gained much more experience with it. One thing we quickly discovered was the fact that about nothing works out the way it should, before tweaked out to some extent. This, of course, again turns in to the fact that simple things might end up taking a long time to get done, because of that little thing that refuses to work.

We didn't really get the “project feeling” of working with this project, since the project frame was very loose and flexible. The reason for this was the fact that we weren't really sure what would take time and what would go smooth and fast, or what task that had to be done in what order.

Chapter 6. References

6.1 References and bibliografy

[1] VNC's website

[\[www.realvnc.com\]](http://www.realvnc.com)

[2] Link to Google

[\[www.google.com\]](http://www.google.com)

[3] NorduGrid's website

<http://www.nordugrid.org>

[4] SGAS design documents

<http://www.pdc.kth.se/grid/sgas/>

[5] SweGrid accounting system architectural proposal

<http://www.pdc.kth.se/grid/sgas/docs/SGAS-0.1.3.pdf>

[6] SweGrid accounting system bank design

<http://www.pdc.kth.se/grid/sgas/docs/SGAS-BANK-DD-0.1.pdf>

[7] SweGrid logging and usage tracking service design

<http://www.pdc.kth.se/grid/sgas/docs/SGAS-LUTS-DD-0.1.pdf>

[8] Job account reservation manager design

<http://www.pdc.kth.se/grid/sgas/docs/SGAS-JARM-DD-0.1.pdf>

[9] SweGrid accounting system security design

<http://www.pdc.kth.se/grid/sgas/docs/SGAS-SEC-DD-0.1.pdf>

[10] AFS system

<http://www.openafs.org>

[11] SweGrid website

<http://www.pdc.kth.se>

[12] Nightly checkout of the SweGrid code

<http://www.pdc.kth.se/grid/sgas/source/>

[13] GridBank design document

<http://www.gridbus.org/papers/gridbank.pdf>

Cluster and grid computing

[14] GridBank's homepage

<http://www.gridbus.org/>

[15] G-monitor, web portal for monitoring and steering application execution

<http://www.gridbus.org/papers/gmonitor.pdf>

[16] GridBank source code

<http://www.csse.uwa.edu.au/~barmouta/gbinstallation.html>

[17] EU-DataGrid project website

<http://eu-datagrid.web.cern.ch/eu-datagrid/>

[18] DataGrid Accounting System Architecture

http://server11.infn.it/workload-grid/docs/DataGrid-01-TED-0126-1_0.pdf

[19] Globus Toolkit Grid Security Infrastructure

<http://www-unix.globus.org/security/>

[20] DGAS source code

<http://datagrid.in2p3.fr/cvsweb/workload/>

[21] EGEE: Enabling Grids for E-science in Europe

<http://www.eu-egee.org>

[22] Open Sources: Voices from the Open Source Revolution

<http://www.oreilly.com/catalog/opensources/book/toc.html>

[23] GNU license

<http://www.nordugrid.org/middleware/license.html>

[24] Open PBS (open source, no longer supported)

<http://www.openpbs.org/>

[25] Condor (open source)

<http://www.cs.wisc.edu/condor/>

[26] Torque (open source)

<http://www.supercluster.org/projects/torque>

[27] PBS Pro (commercial)

<http://www.pbspro.com/>

[28] Maui

<http://www.supercluster.org/maui/>

[29] NorduGrid PBS configuration

<http://www.nordugrid.org/documents/pbs-config.html>

Cluster and grid computing

[30] Open PBS manual (also available from openpbs.org; requires registration)
<http://hovedprosjekter.hig.no/v2004/data/gruppe05/files/openpbs.pdf>

[31] NorduGrid Globus Toolkit distribution (2.4.3)
<http://ftp.nordugrid.org/download>

[32] J2SE 1.4.1+ SDK
<http://java.sun.com/j2se/>

[33] gSOAP (v2.1.6 bundled with GridBank; don't replace with newer version)
<http://www.cs.fsu.edu/~engelen/soap.html>

[34] Xerces C++ Parser (v2.1.0 bundled with GridBank; don't replace with newer version)
<http://xml.apache.org/xerces-c/>

[35] NorduGrid Documents
<http://www.nordugrid.org/documents/ng-server-install.html>

[36] MySQL Documentation
<http://dev.mysql.com/doc/mysql/en/Installing.html>

[37] NorduGrid VO description
<http://www.nordugrid.org/NorduGridVO/vo-description.html>

[38] NorduGrid VO resources
<http://www.nordugrid.org/NorduGridVO/vo.php>

[39] NorduGrid Documents
<http://www.nordugrid.org/documents/ng-client-install.html>

[40] NorduGrid user guide
<http://www.nordugrid.org/documents/userguide.pdf>

[41] NorduGrid CVS template for `/etc/nordugrid.conf`
<http://grid.uio.no/cvs/cvsweb.cgi/nordugrid/doc/nordugrid.conf.template>

[42] Template on the NorduGrid CVS server for `/etc/globus.conf`
<http://grid.uio.no/cvs/cvsweb.cgi/nordugrid/doc/nordugrid-globus.conf.template>

[43] NorduGrid ARC middleware build procedure
<http://www.nordugrid.org/middleware/build.html>

[44] Grid Certificate Mini How-to
http://www.nordugrid.org/documents/certificate_howto.html