INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Urgent Computing in a Grid Environment using Globus Toolkit 5 and SPRUCE

Charlotte Mach

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Urgent Computing in a Grid Environment using Globus Toolkit 5 and SPRUCE

Charlotte Mach

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Michael Schiffers
	Dr. Nils gentschen Felde
Abgabetermin:	20. Januar 2014

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 20. Januar 2014

. . . .

(Unterschrift des Kandidaten)

The demand for Urgent Computing arises from emergency situations requiring large amounts of computing power easily and quickly. Grids or Supercomputers can provide this computing power but usually there is a longer wait until access is possible. Urgent Computing allows Grid users the prioritized usage of resources in case of emergencies.

The currently only available Urgent Computing software is SPRUCE (Special PRiority and Urgent Computing Environment) which covers Globus Toolkit 4, a prominent Grid middleware. Globus has been updated to its fifth version while SPRUCE is still running with the previous release. This thesis approaches the subject of adapting SPRUCE to the current Globus Toolkit. The differences between the two versions and the general developments regarding the technologies used are examined. From the theoretical evaluation of SPRUCE and Globus can be gathered that SPRUCE is adaptable to Globus Toolkit 5, which is proven by implementing the necessary changes and documenting the setup.

Contents

List of Figures

List of Tables

Listings

1. Introduction

Large scale computing systems have long been used in the academic community to handle enormous data sets. Whether the cause was scientific research or commercial interest, at the bottom of it was some sort of problem requiring a large quantity of computing power to be solved.

Different ways to best utilize available computing resources were found and adapted to whatever demands arose. This thesis focusses on one of these solutions for solving these kind of problems, more specifically a newer type of high performance computing with its own demands and issues.

1.1. Motivation

A subsection of high performance computing, namely urgent computing, deals with giving prioritized access to computing resources to authorized users at unpredicable times of pressing need. A more thorough definition will be given in chapter 2.2.

Urgent computing is a relatively new and developing area of computer science. Currently there is one major software system implementing the idea of giving certain users the means of getting urgently needed results quicker than by using the given possibilities of waiting in a queue or reserving resources beforehand. This software, called SPRUCE (Special PRiority and urgent computing Environment), has been developed to fulfill the current need for higher priority computing. These needs include predictions regarding natural catastrophes, dealing with (imminent or happening) disastrous events or even some calculations for medical procedures. SPRUCE has so far been developed for grids and clouds, however, the grid middleware it is currently built to work with (called Globus Toolkit 4) is now no longer supported and has been widely replaced by its successor Globus Toolkit 5. These two versions of the GT differ in some aspects and SPRUCE is not been made available for the the newer version.

The importance of having a system like SPRUCE available can't be denied. There are quite a few scenarios in which urgent computing is relevant and even necessary for a positive outcome of unfolding events.

1.2. Goals

This thesis has the goal of obtaining a definitve answer to the question if and how this urgent computing software could be adapted to function again with the latest available technologies. Therefore the software itself has to be examined as well as the currently newest version of the Globus Toolkit in comparison to the previous version. If the adaption is theoretically possible a simple test implementation of SPRUCE will be set up in a GT5 environment and tested for functionality. The desired result would be to have a working implementation as well as the instructions of how to rebuild this or a similar environment in the end.

In this paper the question of whether SPRUCE is still viable and adaptable to newer standards will be answered. The preferred outcome would provide a way to get SPRUCE working with newer grid technologies, offering this urgent computing service to grid users who are in need of high priority access to their resources and to resource administrators who wish to offer that service to their users or clients. This would be done to return SPRUCE to a state of relevance, since the current state regarding grid computing is outdated and not usable with the latest technologies. This could provide a path for others to implement their own version of SPRUCE and adapt it further for their own needs.

1.3. Structure

After this introduction, the second chapter gives some background on grid computing and urgent computing and highlights the terms and technologies used in this thesis. This will be followed by the third chapter focussing on a specific grid software and the components and functions this software provides. The fourth chapter introduces urgent computing technologies, more specifically SPRUCE itself.

These chapters lay the foundations for chapter five where the theoretical possibility of an adaption is discussed. Chapter six focusses on one specific implementation of an adaption of SPRUCE to GT5. The steps for creating such a grid environment and adding SPRUCE to it are described in detail to provide the necessary information for recreating the practical parts of this thesis. In the last chapter there is a short outlook on things to come and general development possibilites for urgent computing.

2. Background

In this chapter the current state of the computational technologies and concepts relevant to this thesis are summarized, starting with grid gomputing and followed by urgent computing.

2.1. Grid Computing

This section aims to define the term grid computing and provide some insights into the typical structure and components of a grid.

2.1.1. Definition

Ian Foster, one of the computer scientists who coined the term grid computing, describes a grid simply as a *system that coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service.* [Fost 02]

These properties are partially what sets grid computing apart from most other forms of distributed computing. For one, a grid is not under centralized administrative control like a cluster but rather integrates and coordinates its resources and users from different control domains. The use of standard, generalized and open protocols and interfaces distinguishes grids from clouds and is necessary to ensure better interoperability. To fulfill the third requirement a system must allow for greater qualities of service than just the sum of its parts would.

Grid computing is for example used to solve complex scientific problems, analyze huge amounts of data or execute calculations for commercial purposes.

2.1.2. Layers

The grid architecture uses protocols since one major concern for grids is interoperability. Communication between system with different soft- and hardware, dynamic accomodation of new users and other problems have to be considered. Standard protocols offer a solution for most of these obstacles. They do not describe the implementation of components but rather define the interaction between them.

There are five conceptual layers to grids as can be observed in figure 2.1. Each consists of a number of protocols and the five of them together make up the so called *hourglass model*. The number of *Resource* and *Connectivity* protocols is significantly smaller than the numbers of the layers above and below. This just means these protocols are made to be implemented on top of many different types of resources and many different higher level protocols can be mapped onto them. This arrangement of the various protocols with larger numbers on top and on the bottom with a narrow neck in the middle led to the hourglass name.

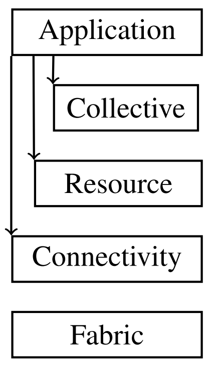


Figure 2.1.: Grid Protocol Architecture cf. [FKT 01]

The *Fabric* layer is the lowest layer and consists of physical resources e.g. storage systems, sensoric devices or computational resources and the mechanisms pertaining to these resources. These mechanisms can be enquiry functions, management of files or other data, allocation or other resource-specific functions.

The *Connectivity* layer specifies communication and authentication protocols. Communication protocols are used for interactions between the different resources, while authentication protocols determine if users or resources are actually who or what they claim to be. There are some requirements for authentification solutions [BWE+ 00]:

- Single sign on: Users should not have to authenticate more than once to have access to multiple grid resources.
- Delegation: It should be possible the give a program the access rights of its owner, so the resources are usable by the program.
- Integration: Resource providers may have their own security solutions with which the grid security solutions must be able to interact.
- User-based trust relationships: If one user uses various resources at the same time and requires interactions between these resources their providers should not have to interact in order to configure the security environment.

On the next layer, the *Resource* layer, information and management protocols are established. These protocols are used to access and control individual local resources. Information protocols are mainly for determining usage policies, configuration and other facts about a resource. For sharing a resource the access policies must be negotiated and this is where management protocols come in. They are responsible for the consistency of the protocol operations with the resource policy.

The fourth layer doesn't focus on one specific resource (contrary to the Resource layer) but on the global state and *interactions across collections of resources* [FKT 01]. For this reason it is called the *Collective* layer. On this layer Virtual Organizations (VOs) come into play. They will be discussed further in the 2.1.3. Essentially they are a temporary alliance of people and/or resources of real organizations for a specific purpose. This layer offers directory services for VO resources and their properties, monitoring and diagnostic servies, data replication services for better data access performance and many others.

The *Application* layer is the last layer and holds the user applications in a VO environment. Applications use and consist of the services of the lower layers. This layer uses so called Application Programming Interfaces (APIs) to provide a way of exchanging messages with other services and thereby using them. Some examples are resource management, data access or resource discovery.

2.1.3. Concepts

A grid usually consists of various heterogenous resources. Some are computational, others for storage or with special traits e.g. a different operating system or measuring devices. These resources are provided by companies and can be far apart when connected through other means. The grid resources can be used by submitting a job to a scheduler. A job is a program that is to be executed on the grid. It can be an actual computation, a system command or even a command to a distant machine. Applications are essentially a collection of jobs. The scheduler chooses the best resource for a given job to run on, depending on job requirements and availability of machines. A user can submit his job to a scheduler and then this scheduler will pick a resource and afterwards the user will receive his result.

The aforementioned VOs are created for the purpose of sharing resources among people for a certain time to achieve a common goal. A member of this VO can use its resources.

A software which allows users to gain access to a grid's resources and essentially is the grid is called a grid middleware. There are a couple of grid middlewares, the major ones include gLite [GM2], UNICORE [GM1] and Globus Toolkit. Globus toolkit is the middleware this paper is focussing on and will be discussed in chapter 3. The next section concentrates on a small subdivision of grid computing.

2.2. Urgent Computing

A relatively new topic in computer science is urgent computing. This section gives a short introduction into its abstract definition as well as an example of a concrete implementation.

2.2.1. Definition

For a computation to be classified as an urgent computation it needs to fulfill certain criteria that were defined by Nick Trebon [Treb 11] as follows:

1. The computation operates under a strict deadline after which the computational results may have little practical value.
2. The onset of the event that necessitates the computation is unpredictable.
3. The computation requires significant resource usage.

These requirements are not met by most standard computations. Examples for the types of computations that do meet them are some weather simulations, predictions of wild fire spreading or even medical simulations. Models of severe weather conditions (e.g. hurricanes) are a good example, because they need to yield results before the situation is over or has done a lot of damage. After a hurricane the amount of damage would be clearly visible to everyone and the results would be useless. They are only helpful before the damage is done.

Secondly, they can occur without much warning and at random times therefore the resources must be ready for input at any given time.

The last requirement is obvious given the large amount of input data and the many factors influencing the results of the computations in this situation.

2.2.2. Necessary software and resources

There are certain requirements for software and resources to be usable in an urgent computing scenario. Generally speaking, a resource should be capable of either high performance computing or some other feature which isn't readily available everywhere else. While sometimes it can be useful to have a small cluster of computers for computations or middle-scale jobs, this isn't what urgent computing is about.

In general a system must be based on top of a grid (or cloud) with a certain high performance capacity. The individual resources should comply with policies to allow urgent computing. Every resource can have a different type of policy depending on admin preferences. Several possibilities are available and also implementable [Treb 11]:

- next-to-run: The urgent computing job waits in the queue until the current job finishes and is then the next in line.
- dismissing jobs: One form of preemption is to simply cancel the currently running job and start the urgent job. This could result in quite a loss of data and there might be trouble with the owner of the cancelled job.
- checkpointing: Another form would be to write checkpoints into the program to ensure not all data is lost when a job is stopped. This might not be possible with all jobs.
- suspension: A third form of preemption would be to temporarily suspend the running job by saving the current state. This could require checkpointing and might again not be possible for most jobs.
- migration: Another possibility could be to migrate the running job to another machine, and thereby not lose much time.

Most of these options are not kind to other users which could result in discontent and loss of customers with commercial grids. A way to avoid this could be to offer certain perks to those whose jobs were cancelled or otherwise constricted by an urgent computing job, like decreased fees or increased priorities for some time.

One more necessity regarding the urgent computing software itself would be a way to create a timeframe for these computations and simplify the use of higher priority queues for users in emergency situations. One way to ensure that only certain users are allowed to submit urgent computing jobs and only during this emergency are tokens. Tokens would have a lifetime and an admin. The admin could add and remove other users as needed and if the lifetime of a token doesn't suffice he or she can apply for a second one. These tokens would be transferable, not depending on one person alone to prevent a human being the single point of failure.

2.2.3. Implementation

There are different ways to implement urgent computing environments. Some of the following might not even fully qualify but are included for the purpose of completeness of current available methods.

High priority queues to which only certain users have access are one possibility. These do require a human to confirm the necessity and to terminate the current jobs. People would have to be available

24/7 and the consequences of a mistake could be troubling. This is essentially a user calling an administrator and asking if his job(s) could receive an increased priority, bypass the queue or cancel the running jobs. In this case no additional software is needed. No computer program is necessary to authenticate a user or prioritize his or her request. The administrator would have to know if the requesting party is authorized and how to respond if there is any uncertainty. This option is not favourable, there is always the risk of human error and the staff needed to respond to many urgent computing requests at any given time would exceed the cost of implementing a computer-based solution.

Another option would be to have a supercomputing center completely free of non-urgent jobs, always available for emergencies. This is impractical, considering the huge amount of energy and the personnel needed to keep such an institution up and running. While some might see this as a worthwile option, the waste of money and perfectly fine computing power would be obscene. Given the information that there is an alternative to squandering electricity and resources this option is not the way to go either.

The third and so far most promising option is SPRUCE, the currently only available urgent computing software, which will be discussed in detail in chapter 4. This is the only option available that doesn't require much human interaction or free queues to successfully run urgent jobs. Being softwarebased, having almost no need for humans interfering with the system and being comparably cheap, SPRUCE provides all the advantages and none of the disadvantages of the other options.

3. Globus Toolkit

The first version of the Globus Toolkit was developed by a team under the lead of Ian Foster, Carl Kesselman and Steve Tuecke and then released in 1998. [RN] This toolkit is the grid middleware in question and for the successful adaption of SPRUCE to the latest version a fine-grained examination is necessary to determine the important changes between the two relevant releases. Without this part chapter 5 which is showcasing the adaption, would not have much to go on in terms of having a basic understanding of why the adaption is even necessary.

This chapter will take a detailed look at the Globus Toolkit, at first only at the latest stable release and later at the earlier versions. When looking at the earlier versions the differences between them should become obvious. Afterwards a short comparison between different grid middleware solutions is given.

3.1. Globus Toolkit 5

The following sections outline the specific components of GT5, with their abstract purposes and their exact implementations. Later a few use cases for the toolkit in real life are depicted.

3.1.1. Components

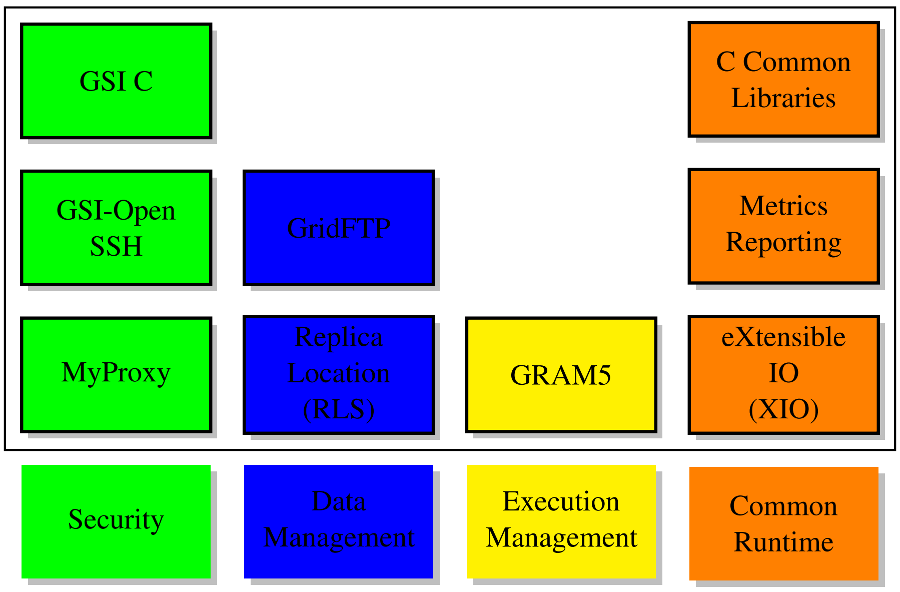


Figure 3.1.: Globus Toolkit ®Version 5 (GT5) cf. [Globus Alliance]

The functions of GT5 can be divided into four main areas (as shown by the different coloured pillars in figure 3.1).

There is software included for each of the following components:

Security This area provides components for authentication, protection of the communication, authorization and management of user credentials. All possible security issues are dealt with in this part of the Globus Toolkit. Encryption and the ensuring of the identity and correctness of the permissions a users obtains are a large share of the functions needed to secure grid roles and features [GT5]. The programs used in Grid Security are **GSI C**, **MyProxy** and **GSI-OpenSSH**.

- The Grid Security Infrastructure in C is responsible for authentication, authorization and certificate management. The authentication works with Public Key Infrastructure (PKI) technologies based on the usage of a pair of keys for cryptographic operations such as signing and encrypting.
- One addition is provided through the open source software MyProxy which also works with X.509 PKI public and private keys.
- GSI-OpenSSH is another component mostly used for the offered remote login and file transfer services which work with a single sign-on mechanism.

While security is a very interesting piece of the Globus software, it will most likely not be an issue in the adaption. The main focus will be on obtaining a functioning system, security will not play an important part in this thesis. It must be said, however, that in a real-world adaptation and implementation, security is a vital part of having a functioning, secure and therefore successful system.

Data Management One issue of grid computing lies with the transfer of bulk data between hosts. To make certain computations or execute some programs there have to be files and other data available on other hosts. If a program is to be executed on another host, the executable and any necessary files needs to be transfered to this hosts. For this purpose the network protocol FTP has been extended to provide a protocol that is more reliable, secure and offers better performance than the original. This protocol is called **GridFTP** and is only used for data movement [GT5]. A server implementation, a scriptable command line client and a set of development libraries are provided within Globus Toolkit to move data around the grid.

Another component is used for data replication and that is the Replica Location Service (RLS) which, as the name suggests, is a server providing services for registering and looking up replica information. The two services are catalog (LRC) and index (RLI) maintaining a catalog of replica information and an index of logical names for the services respectively. Users who need to find existing files located in the grid can use these tools to search for copies (or replicas) of the wanted files.

Execution Management Jobs on grid computing resources are managed through **GRAM5** (Grid Resource Allocation and Management), meaning this component is used to locate, submit, monitor and cancel jobs by communicating with Local Resource Schedulers. GRAM focusses on jobs where ”arbitrary programs, reliable operation, stateful monitoring, credential management, and file staging” [Globus Alliance Documentation] are of a certain relevancy. A Local Resource Manager (LRM) is essentially a system for access control on a resource, providing interfaces GRAM uses to execute jobs. GRAM5 consists of four service components working together to fulfill its functions which are job management, user authentication, LRM interfacing and file staging.

- The **Gatekeeper** serves as a network interface. It handles authentication of user identities and thereby acting as a literal gatekeeper preventing unauthorized access. This process is accordingly called *globus-gatekeeper*.

- The **Job Manager** is started by the *globus-gatekeeper* process. Job requests and file transfers are coordinated by the *globus-job-manager* daemon. There is one instance for the LRM and one for each job. When a user (or sometimes an application) requests information on a running or completed job or wants to cancel or clean up a job this daemon is queried and returns an answer to the user (or application).
- The **Scheduler Event Generator** translates LRM-specific data regarding job management into another, LRM-independent data format. This serves partially as an interface between the jobmanager and the job.
- The **LRM Adapter** interfaces the other GRAM5 components with the LRM. There are several adapters provided because Globus supports more than one LRM. This adapter translates the Globus job request into a format the LRM understands and responds to.

Even though GRAM5 can be used with quite a few LRM systems, it is configurable to manage jobs on its own. This way the adapter would not be necessary and there would be no need for having a LRM installed. But since Globus supports various jobmanagers and these come with their own features this function is actually of value. Supported job managers include HTCondor [HTCondor], PBS [PBS] and LSF [LSF]. Another LRM mechanism called TORQUE is also supported by GRAM5 since it is based on PBS and will become relevant in chapter 6.

Execution management (or job management) is the most important part of Globus for the purposes of this thesis. How and where the jobs are later submitted in an urgent computing environment is going to have a larger impact on this area (of job submission) than on the others.

Common Runtime Libraries needed for building the grid infrastructure and an input/output library for grid applications to use are included in this component. The set of C Common libraries make an "abstraction layer for data types, libc system calls and data structure" available for use in the toolkit [Globus Homepage].

Globus XIO, the I/O library is written in C as well and provides an API supporting several wire protocols and provides drivers which contain the implementations. The drivers are for example TCP, UDP, HTTP and GSI. These libraries serve as a basis and extension for many of the already mentioned functions of Globus.

3.1.2. Use Cases

Globus is an open-source software made available to everyone in need of a grid middleware. Many supercomputing centers offer Globus installations for their users, some researchers build their own enviroments on top of Globus to achieve their goals and sometimes the toolkit is simply used for instruction purposes to introduce others to grids.

Globus is offered at the Leibnizs Supercomputing Centre [LRZ] and several other computing centers for computational purposes to both researchers and students. While in this case GT is used as a simple job submission system sometimes Globus services are simply integrated in other architectures. Globus services can be helpful in distributed supercomputing, collaborative engingeering or high-throughput computing. Services such as security, information, fault detection, communication and resource management are given by Globus and utilized independently from each other or together.

3.2. Differences between GT5 and previous versions

While most components of GT5 were already present in a previous version, some components were replaced and others newly introduced. GridFTP, RLS, MyProxy and GSI-OpenSSH for example were available before and just updated and adapted for GT5 [GT5]. GRAM went from Web-service-GRAM4 (WS-GRAM) back to a pre-WS version (namely GRAM2). All Web service components that existed in earlier versions are no longer available in the newest release.

The following section will show how each component of the current GT developed over time, starting with Globus 2.0 and working its way up to GT5 (focussing on GT4 versions 4.0 and 4.2 and GT5 versions 5.0 and 5.2). These assessments are based on the Globus Toolkit release notes [RN].

GT Version $\rightarrow$ $\downarrow$ Component	2	3	4.0	4.2	5.0	5.2
GRAM	o	+	++	+	++	+
GridFTP	o	+	+	+	+	+
RLS	o	+	+	++	+	+
GSI C	o	++	++	+	+	+
MyProxy	-	-	o	+	+	+
C Common Libs	a	a	+	+	+	+
XIO	a	++	+	+	+	+

Table 3.1.: Development of Globus Components over time

A shorter version of the following parts can be seen in table3.1, which simply describes if a component has been newly introduced (- $\rightarrow$ o) , updated (+), is available in a different form (a) or has changed in a significant manner (++).

Execution Management GRAM, the part of Globus responsible for job management, has been a part of the Globus Toolkit from the start. In GT2 GRAM was updated to version 1.5 and some additional features were included. GT3 already split some of its services into pre-WS and WS versions. Several improvements were made to GRAM between GT2 and GT3. New WS GRAM features in GT4 included the selection of which account a job will be run under if a user's grid credential corresponds to multiple accounts. This can be specified in the job description and might be of note for the adaption.

In 4.0 GRAM the completely reworked WSRF-compliant GRAM4 caused this GRAM to be non-backwards-compatible to any previous version of it. This version, GRAM4, only stayed on until 5.0, when it was replaced by GRAM5, a service no longer relying on Web services and based on GRAM2, the GRAM version before WS GRAM was introduced. The new GRAM5 is based on GRAM2 but offers better scalability than it and is more reliable than the WS GRAM.

There have been many minor changes to improve user accessibility and generally create a user-friendly system. Of course the unavoidable bugfixes are also included in these changes. [RN]

Data Management Both GridFTP and RLS have been available in this form in the toolkit. The changes in GridFTP have not been bigger than the obligatory bugfixes, some updates and minor alterations that didn't affect the interfaces. The RLS changes are mostly minor, too; a couple of libraries and commandline tools have been added. The biggest alteration was the embedding of a database into the RLS server, providing a simpler and user-friendlier version of the installation.

Before this version the database would have to be pre-installed or otherwise preconfigured by a user, afterwards the GT took complete care of this.

Security The Grid Security Infrastructure was added in the first version of Globus. The most development in this particular area was done in the first two version upgrades, after that it decreased to the usual fixes and updates.

MyProxy (as well as GSI-OpenSSH) was first included in GT4. The updates in MyProxy are not included since it is an external service and the usage commands have not changed in appearance or function. Security was adapted where necessary but no major differences exists between GT4 and GT5.

Common Runtime Components Libraries were obviously available from the beginning, the C Common Libs were only named like they are now after GT5. Similar functions were provided before but by a different library.

XIO's functions were previously fulfilled by *globus_io* (until the third release). In GT3 Globus switched to another library, XIO. In GT4 some drivers were added and API changes were made. From then on practically no changes were made in this particular library. [RN]

Other alterations were made around these components. Globus has developed farther than these short descriptions let on. Many components and features were deprecated or compacted into others. Globus offers extensive information on 24 components of GT4 but GT5 only has nine components itself. On the one hand a huge part of GT4 were the now defunct Web service parts, while on the other hand other components were merged or removed completely. The biggest alterations in Globus happend before GT5, setting up some challenges for adapting any service from version 4.0 to 5.0.

Globus Toolkit differs from most other grid middlewares in one aspect. It offers a variety of basic tools that can be used by third parties to construct their own grid systems while other middlewares provide a more tighly coupled approach also considering the client side. Globus focusses mainly on the different parts and services, literally offering a toolkit.

4. SPRUCE - Special PRiority and Urgent Computing Environment

SPRUCE has been developed by the University of Chicago and Argonne National Laboratory as a TeraGrid [Teragrid] project to support urgent computing needs. These require an architecture which can provide sessions, allows user interference and allows resources to still use different policies. Additionally, it must be possible for a user to quickly transfer his or her permissions to others and team leaders must be able to grant their team similar access rights. SPRUCE uses a token-based architecture to meet these requirements. In this case a ”right-of-way” token is a 16 character long alphanumeric string serving as a simple code for authorizing urgent computing. It can be easily transferred from one person to another, which adds to flexibility in emergencies.

Tokens are given out before emergencies occur and are needed to initiate an urgent computing session. The token is simply input into it's activation field on a Web-based SPRUCE portal and thereby starts a time period of at most 24 hours (which is a time restriction given by SPRUCE and can be substituted through virtually any duration of time by a SPRUCE admin) during which priority access is granted. Users must have the permissions to access the resources, and job submissions must be marked as urgent. The following sections will outline what components SPRUCE is made of, what functions they serve, how SPRUCE fits into grids and how the different files work with each other.

4.1. Components

This section outlines the three main components [BNTB 06, p.3] of SPRUCE:

1. User workflow and client-side job submission tools
2. A Web service-based user interface to manage tokens
3. Local resource provider agents for responding to priority access requests

The first two of these three can be comprised into one component. Implemented in one centralized server and a database (for token information and such) which is also hosting Web services and the user and administrative portal. The Web service packages are shown in table 4.1. They can be divided into user services, allowing all kinds of requests for information from the portal and the submitfilter which is taking care of authentication requests during urgent computing phases.

The submission tools are scripts that change the behaviour of different parts of Globus job management and LRMs. The second component would be the part of the portal responsible for providing access to authorized users and allowing them to manage their tokens. These tokens are what permit the high priority access to a resource later on. The final component is composed of a Web service and a script on the resource. These can communicate and the Web service can authenticate a user requesting urgent computing power against the database. This view of the components shows mostly the user side, which is the predominant and most sensible way of looking at SPRUCE.

Package	Service	Description
User Services	getTokenInfo()	Return attributes of a token (for token managers only).
	addUserToToken()	Add user to token (for token managers only).
	removeUserFromToken()	Remove user from token (for token managers only).
	activateToken()	Start an urgent computing session (for token managers only).
	checkTokenTime()	Return time remaining on an active urgent computing session (token managers only).
	getUserInfo()	Return information on any activated urgent computing sessions for a given user.
Submit Filter	authenticate()	Authenticate a request for an urgent computing task.

Table 4.1.: Web service packages and their functionality cf. [Treb 11, p.19]

Other Web services could be implemented for administrative purposes and adding more functions to the portals by imitating the existing services. The available Web services provide enough functionality for a simple setup and for a user to activate a token, add other users and then submit jobs. These three Web services are the most important ones.

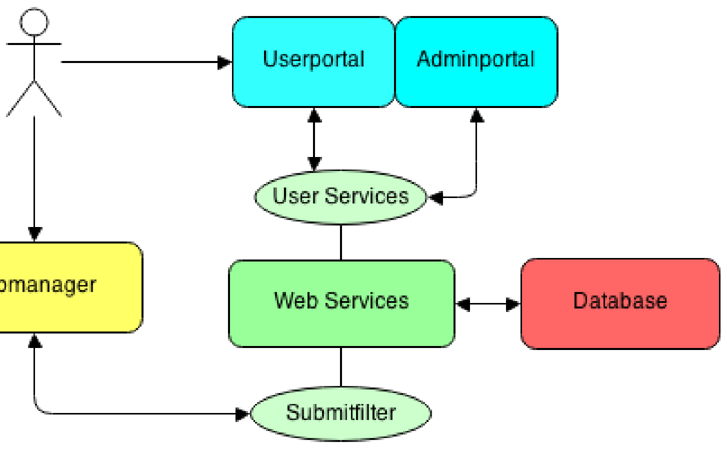


Figure 4.1.: SPRUCE components

As seen in figure 4.1 the workflow is quite simple. A user first has to check with the userportal if he or she has been added to an active token, if not they would have to either activate a token or ask to be added to one. Then the user can submit jobs via Globus. If the job description is written correctly and an urgency parameter is present in the job description the SPRUCE jobmanager would check first if the user is authorized to submit this job. This works by calling the submitfilter part of the Web services which then query the database for information on the user. If the query returns the right answer the job is successfully submitted to a high priority queue.

4.1.1. Portal

The SPRUCE portal contains two access points, one for resource administrators, the other for urgent computing users. The administrative portal allows you to issue tokens for your resources, view already created tokens and every (successful and unsuccessful) attempt to log onto your resource via SPRUCE. Every resource admin participating in SPRUCE can log into the portal and control his or her resources from there. The second portal was created for users, using the Web services from table 4.1. It delivers all the needed information to a user, including their available tokens and resources. The portal is needed for the activation of a token and thereby starting the urgent computing session. Afterwards users can check the remaining time on their token and add or remove users. The information is being kept on a seperate database-server, only accessible through the Web services.

Setup

The portal is PHP-code communicating with the webservices via SOAP. While the user services are called via Javascript and PHP, the submit filter is using Java as the more direct approach. Since the authentication function is only used internally, when a job is submitted and doesn't need to appear on the website, the obfuscation through PHP and Javascript is not necessary, since it doesn't endanger the server. In both cases the XML-String is just passed to the Web service in SOAP-Notation, processed and the following SOAP response is then parsed and handled. If no exception is thrown, the response should contain the necessary information for processing, either displayed on the website or used to handle a SPRUCE-job request.

4.1.2. Resources

There is a part of SPRUCE consisting of a collection of scripts which is installed on every resource where SPRUCE is supposed to be used. These files are described in table 4.2.

Script	Description
install-spruce	Responsible for adding all the necessary files to the resource
spruce.rvf	Resource verification file, used for globus submissions
spruce.pm	Globus job plugin (LRM adapter)
token_authentication_ws.sh	Checks user permissions with Web services call
token_authentication	Encoder/Decoder for ws-call results
token_authentication_ws.class	Java file for token check
spruce_sub	Build version of script to submit jobs into SPRUCE queue
jobmanager-spruce	Jobmanager, resource contact
<scheduler>_submitfiler	Script for checking qsub jobs

Table 4.2.: Scripts for the local resource provider agents

These communicate with each other to ensure the proper use of a token is given, that a users is authorized to use the token and of course, that the token is valid and active. The submitfilter talks to the submitfilter Web service, spruce_sub is a wrapper for TORQUE's qsub for submitting jobs, spruce.rvf is needed to add the urgent computing parameter to job descriptions and spruce.pm replaces the current adapter between Globus and the installed LRM as the interface.

As can be seen in figure 4.2 the most important components of SPRUCE are the jobmanager-spruce which is replacing the current GRAM jobmanager contact on a resource and the spruce.pm, the adapter file for connecting GRAM and the LRM. The picture shows a simple setup where there is one metascheduler where the job submission first come in contact with globus. The jobs are sent to one of the two corresponding resources (depending on the contact) which have a Globus installation on top of a local resource manager. (In this case we have TORQUE, but the principle is the same for

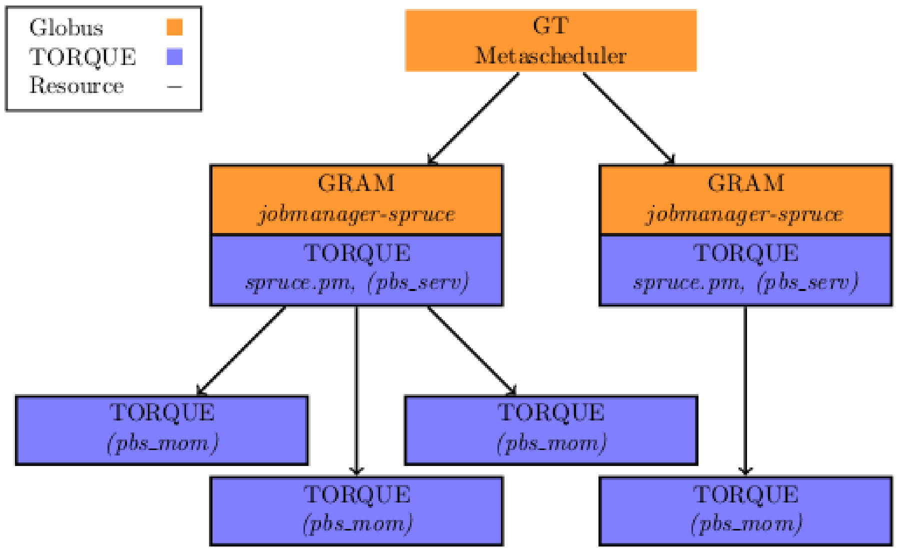


Figure 4.2.: SPRUCE files in a grid

any scheduler). The TORQUE server (*pbs_serv*) communicates with the TORQUE clients (*pbs_mom*) on the other nodes and submits the jobs according to their specifications and policies.

4.2. Workflows

This section gives a closer look into the filestructure of SPRUCE as well as some workflows.

User View Generally a GT user submits a job using *globusrun*, *globus-job-submit* or *globus-job-run*. *Globusrun* uses an RSL-file as input, while the second and third function use command-line options. The difference between *globus-job-submit* and *globus-job-run* is, that the former is used for batch jobs running in the background while the latter submits interactive jobs with the output being redirected to the client.

So when a user specifies the urgency-parameter in his or her RSL (depending on the command either in a file, or as a parameter with *-x*), the job should, if the user is authorized, run according to the urgency, the resource policy and the user's permissions. Let's say a user specifies the urgency as red, on a resource the user has permissions to use, with a next-to-run policy. Then this job should be next in line to run on that specific resource (if there are no other urgent jobs waiting).

Workflow (*globusrun* with *jobmanager-spruce*) First the *jobmanager-spruce* script is called, which works similarly to *jobmanager-pbs* (and others). The only difference is the specified type and an additional Scheduler-Event-Generator module called *spruce-seg_module*.

GT5 then calls a Perl script, named after the jobmanager, in this case *spruce.pm* but it could also be *jobmanager.pm* or *pbs.pm*. *Spruce.pm* is the adapter talked about in chapter 3. The adapter parses the RSL-file given to it by *globusrun* and creates a job description in a text file which then is submitted to the local resource manager.

With SPRUCE the job description includes an urgency parameter specifying the urgency and at the same time declaring it as an urgent job submission. If no urgent parameter is given in the RSL (and therefore not in the job description) the job is simply forwarded to a non-urgent queue and submitted. This way it isn't necessary to submit urgent and non-urgent jobs to different jobmanagers; *jobmanager-spruce* takes care of distinguishing between the two and acts accordingly.

4.3. Use Cases

SPRUCE has a few use cases most of which regarding the prediction and prevention of damage from natural disasters. Natural disasters fit the urgent computing criteria well, they are not (long-term) predictable, usually happen during a short period of time and create massive amounts of data. When a hurricane forms and happens to hit a city or another populated area the results can be deadly. In general, this is a case SPRUCE could be used for to evacuate nearby areas in time (when the data reveals the direction the hurricane is going), predict the results of similar disasters and help build better models for the next event. SPRUCE can strongly support disaster management this way.

There are three actual projects SPRUCE has been used in.

LEAD Linked Environments for Atmospheric Discovery [DCC$^+$ 04] is a cyberinfrastructure of a meteorological nature. Events such as floods, tornadoes or lightning fall into the area of this project. LEAD is supposed to provide a dynamic framework in which meteoroloical analysis tools and models can use grid resources to observe, model and predict changes in weather.

SCOOP The SURA Coastal Ocean Observing Program [SCOOP] has similar goals to LEAD. SCOOP's purpose is mainly the prediction of the occurence of hurricanes. Sensor are used to gather data which is then used for hurricane forecasts. Other target areas include floods, tornadoes, tsunamis and rising sea-levels. These are all possible hazards that can occur in coastal regions.

GENIUS Contrary to the other two projects, Grid Enabled Neurosurgical Imaging Using Simulation [GENIUS] does not concern itself with the weather. GENIUS was created to examine cerebral fluid flow and improve surgery outcomes. By simulating a patient's blood flow in real-time during surgery, some insights and guidance can be given to the surgeons.

This doesn't technically fall into the area of urgent computing, since the time of a surgery is often predictable, but sometimes emergency situations can cause the sudden need for resources.

5. Adaptability of SPRUCE to GT5

This chapter shows how the SPRUCE source code and parts of the grid infrastructure need to be modified to work with the newer version of the GT middleware. The next section describes the parts themselves and the alterations that are required.

5.1. Theoretical causes for modifications

This section details the modifications which are required when adapting SPRUCE to Globus Toolkit 5. The parts which may cause the need for modifying SPRUCE are the Globus Toolkit itself and the Local Resource Manager(s), both of which could possibly get in the way of SPRUCE working properly with GT5.

Figure 5.1 shows Globus (more specifically GRAM) components and SPRUCE components on top of a resource and their interactions. This provides a complete overview of SPUCE integrated in a grid.

A user can access the user portal which is built on Web services. An administrator could use the admin portal to access the same services and control his or her resource options. When a user has an active token, he or she can submit a job using the Globus Gatekeeper (with either *globusrun*, *globus-job-submit* or *globus-job-run*). This action creates a new instance of a Globus Jobmanager which is then responsible for handling status requests, monitoring or cancelling this job. The jobmanager delegate the job execution to the LRM via the LRM-specific adapter provides by Globus.

The difference between an ordinary job and a SPRUCE job lies in this adapter. If it is an ordinary job, the black route is taken from the user down to the local scheduler. If it is a SPRUCE job, the red route is taken, the LRM-adapter is provided by SPRUCE (based on the orginal adapter and patched). This adapter then submits the job to the correct high priority queue (in this case the queue called ”spruce”). Before the job is executed the LRM parses if a submitfilter is available. SPRUCE provides this submitfilter to check if a user is indeed allowed to use urgent computing. It does this by checking the database for a connection between the user and an active token. This is done via the submitfilter Web service.

These two components, the LRM adapter and the submitfilter, are the ones we need to pay attention to when adapting SPRUCE. Changes in GRAM or the LRMs can affect how SPRUCE functions and those two components are the only ones connecting SPRUCE and thereby the only ones that can be affected.

5.1.1. GT

As seen in 3.2, there are several differences between the various versions of Globus Toolkit components. But since SPRUCE only interferes with the job submission and the scheduling GRAM is the only part affected. Changes in other components should not affect the interactions between SPRUCE and Globus. Therefore this section simply concerns itself with the area that could need adapting by

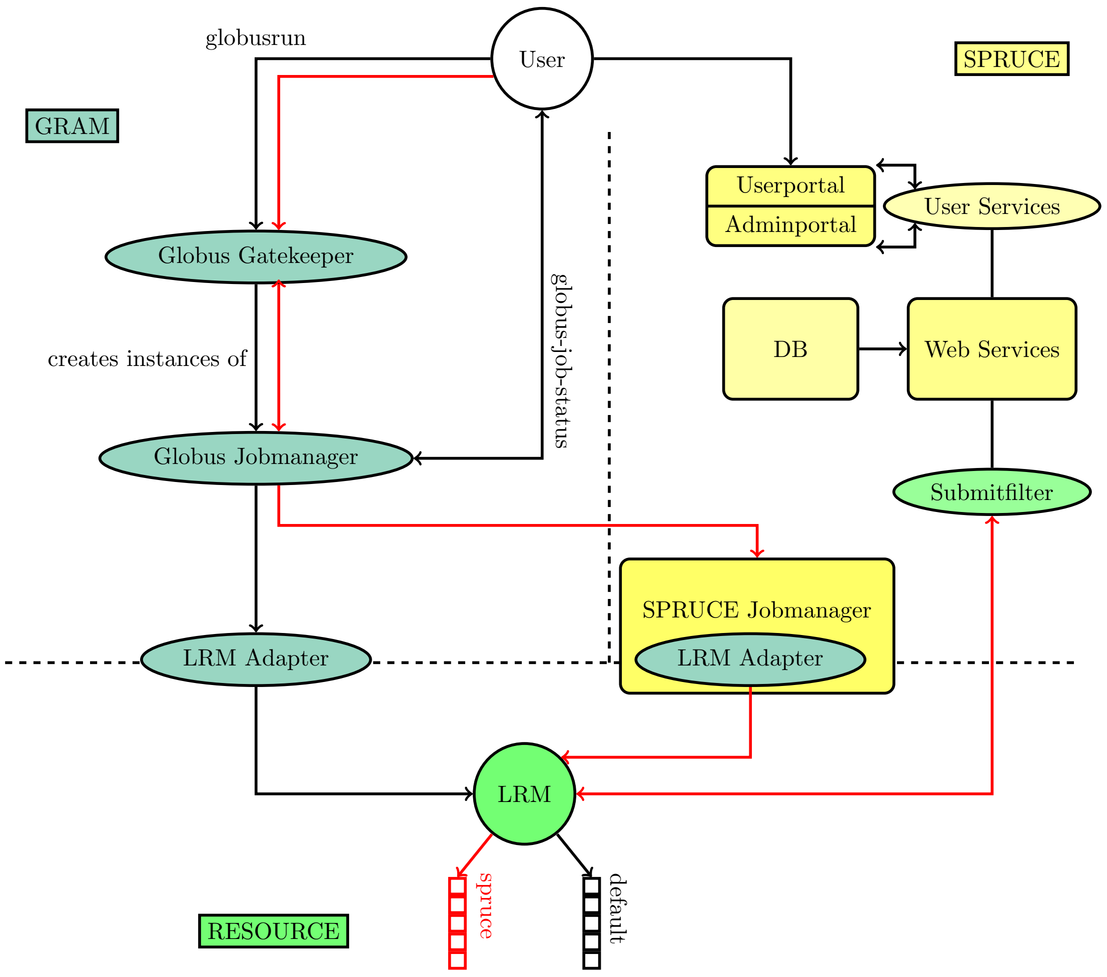


Figure 5.1.: Intersection points between Globus and SPRUCE components

looking at the specific changes in GRAM from version 4.x to version 5.2. GT 4.0 offers both WS GRAM (Web Service Grid Resource Allocation and Management) and Pre-WS GRAM which is basically the same as the GT 3.2 version (also known as GRAM2). The WS GRAM component is "a set of services and clients for communicating with a range of different batch/cluster job schedulers using a common protocol"[WS GRAM, p.1], offering the same functions and features the regular GRAM does by using different means. As the name suggests, WS GRAM uses Web services to implement the job management functions for grids.

WS GRAM functions are named after their predecessors, only with a suffix (e.g. *globusrun-ws, globus-jobs-submit-ws, globus-job-clean-ws*) and return the same results as their counterparts would.

GRAM2, the Pre-WS GRAM uses custom HTTP as its access protocol, while GRAM4 (the WS GRAM) uses the already mentionen Web services for this task. Another difference between the two is the amount of jobs that can run on one machine at the same time. GRAM2 uses polling (with each poll calling a query regarding a jobs status) which uses a LRMs status request repeatedly to track the state changes in a job. This causes a tremendous workload on the nodes. GRAM4 on the other hand uses a different mechanism, namely a Scheduler Event Generator (SEG). This SEG is a daemon running on a node (only one on each), reporting back whenever a job changes its status. This is a server-sided solution, as opposed to for example limiting the number of jobs that can be submitted to one host. GRAM2 could only support approximately 250 concurrent jobs this way while GRAM4 allowed for 32,000 jobs. [FFM 07]

GRAM5 supports both polling and the SEG; SPRUCE only supports polling so far. If the polling mechanism had differed much between the two versions, it could have been a wise decision to adapt SPRUCE to the SEG. It would bring definite advantages in scalability. But the mechanism hasn't changed enough to require a switch to another method for this implementation.

As it turned out later SPRUCE wasn't built for WS GRAM, but for the pre-WS GRAM which was mainly included in GT 4.0 for legacy purposes. The plan for the pre-WS GRAM was to deprecate it and keep only the WS GRAM. This development didn't occur and later on GRAM5 would not be build on Web services anymore. This makes it a lot easier to adapt SPRUCE to GRAM, since no major differences exist between the non-WS versions. There are some minor differences in the GRAM jobmanager (specifically in the adapter file *pbs.pm*) which SPRUCE patches to *spruce.pm*. The changes are addressed in chapter 6, where an actual implementation highlights the process of the necessary adaptions.

5.1.2. LRM

Globus works with a number of local schedulers, of which quite a few were supported by SPRUCE. These include TORQUE with Moab [Adaptive Computing], PBS Pro [PBS], Loadleveler and LSF [LSF]. The only non-commercial schedulers are Cobalt and TORQUE. Moab however, is a commercial software, but the same company offers Maui as an alternative. The TORQUE version has changed since the SPRUCE adapter has been written for it. The differences between TORQUE 2.0 (the version previously supported by SPRUCE) and TORQUE 4.2.5 (the current version) are not very significant, most are enhancements regarding scalability, bug fixes and other improvements irrelevant to the structure of TORQUE and therefore not relevant for the adaption. The only relevant file would be the *torque_submitfilter*, a file SPRUCE needs to influence the scheduler to activate the authentication with tokens. This feature, which allows adding any kind of filter, is still supported in the newest version of TORQUE.

SPRUCE supports Moab 4.0.x, the question is whether the current version of Maui (3.3.1) will suffice or if some tweaking is necessary. Moab is the commercial version of Maui, allowing for support of large scale grids, easier job submission and quite a few improvements in the system itself. In regards to a proper setup, not just a small scale testing environment Moab would definitely be the better choice.

The differences between Moab and Maui can be summarized into naming conventions, less options and less customer service with Maui and better and more intricate policies regarding scheduling, deadlines and prioritizing with Moab. [Maui] Maui does provide preemption policies and differently prioritized queues which is almost the only requirement by SPRUCE. There might be less possibilities to configure a solution but Maui offers enough options for regulation towards the desired result.

5.2. SPRUCE

Since SPRUCE is not given as a ready made solution – the portal and Web service code is not publicly available– some parts will have to be written from scratch and others will need to be adapted to this implementation [Treb 11]. There is one install script for the resource side of SPRUCE, which has to be changed to accomodate the different folder structure.

Another part of the resources side is the *spruce.pm*, which is based on the *<LRM>.pm* and patched up with the necessary code. The *<LRM>.pm* has changed (at least from that TORQUE version to the newest) and needs a few extra lines of code. The components that are almost completely missing are the two portals and the interfaces between the the portals and the Web services as well as between the jobmanager and another Web service.

Apart from the already mentioned components which integrate SPRUCE into the grid (the submit-filter and the adapter) there does not seem to be anything else to consider. Of course there could be problems with SPRUCE containing outdated software components or using deprecated packages. These problems will be considered in the next chapter.

5.3. Problems

Problems could arise from these differences, a newer version could still be incompatible. Some differences are discouraging, while others don't seem to provide much of a challenge. *Globusrun* no longer supports Message Passing Interface (MPI) jobs, which isn't necessarily required but still restricts our options. Another problem is Maui, since it has never been used with SPRUCE before. Maui doesn't offer as many functions and options as Moab and has only limited support. If Maui can't provide the features SPRUCE needs, a commercial solution would have to be tried.

6. Implementation

This chapter presents the actual implementation of SPRUCE in a GT5 environment. These are simply three virtual machines showing that the adaption is working; on a larger scale the mechanisms would work the same, which is why this small-scale example suffices. How this implementation came to be will be described through the following sections detailing the installation and configuration. One of the virtual machines is used as the database server and the Webserver for parts of SPRUCE (with its operating system being Debian 6.0.7 (squeeze) while the other two virtual machines, who make up the grid, operate on Debian 7.1 (wheezy). Debian was used since GT only runs on Unix platforms and it provides packages for the Globus components.

6.1. Setup of the Grid Infrastructure

The Grid is set up on two virtual machines, with the respective hostnames *projekt24.pub.lab.nm.ifi.lmu.de* and *projekt15.pub.lab.nm.ifi.lmu.de*. We will call them proj24 and proj15. They each have grid middleware and local resource managers installed.

6.1.1. Middleware

The Globus Toolkit version used in this implementation is 5.2.5, the (at the time of the writing of this thesis) latest stable release [Globus Alliance Documentation]. From the GT components GSI, GridFTP and GRAM were installed and for reasons regarding simplicity a local Globus Simple CA is used.

6.1.2. Local Resource Manager

Proj24 and proj15 both use TORQUE Resource Manager (4.2.5) with the Maui Cluster Scheduler (3.3.1). TORQUE[Adaptive Computing], which is short for Terascale Open-Source Resource and QUEue Manager, is one of only two non-commercial resource managers being supported by Globus as well as SPRUCE. Based on Portable Batch System (PBS)[PBS] TORQUE still uses tools with names like *pbs_server* or *pbs_mom*. Another version of PBS supported by SPRUCE would be PBS Professional (commercial, but contrary to OpenPBS still actively developed). Essentially TORQUE is providing functions to control jobs on computational resources. These control functions allow users to start, hold, cancel and monitor job. While there is the built-in scheduler *pbs_sched* TORQUE is generally used as just a resource manager with a different scheduler making requests. In our case this scheduler is Maui. Maui is a cluster and supercomputer scheduler maintained by Adaptive Computing, Inc.[**?**]. This company offers a similar commercial scheduler called Moab which is basically the new and improved Maui. SPRUCE currently only supports Moab, our implementation uses Maui. SPRUCE needs queues to be configurable as high- and low-priority, which is where Maui comes in very handy. By simply adding a few lines to the *maui.cfg*6.1, the queues are allowing jobs in a high priority queue to preempt other jobs. This example shows the necessary definitions and attributes so that the jobs in the queue called ”spruce” always run before the jobs in other queues and that they can, if necessary, suspend them.

```
PREEMPT POLICY                SUSPEND

NODEALLOCATIONPROLICY         PRIORITY

CLASSWEIGHT                   1

CLASSCFG[DEFAULT]             FLAGS=PREEMPTEE PRIORITY=0
CLASSCFG[spruce]              FLAGS=PREEMPTOR PRIORITY=100000
```

Figure 6.1.: Added lines in maui.cfg

6.2. Setup of SPRUCE

6.2.1. Portal and Database Server

The Web frontend and most pieces of the portal are written in PHP. After gathering some of the source code and a few Javascript libraries from the pre-existing portal[SPRUCE], some pieces to connect the PHP-files to the Web services had to be written. Examples for how these can be written are attached in A.1. They serve as an interface between the server and the Web services; one example shows how to retrieve the data associated with a token when given the token string through the portal. The PHP-script uses GET to save the token id into a variable and then creates a SOAP client. This client packages the token id from XML into the proper SOAP syntax to call the Web service function which queries the database for the corresponding information. This information is returned to the script in a SOAP envelope, which then is converted into simple JSON syntax.

For example if our token id would be *4444-4444-4444-4444*, the XML would be embedded into the SOAP message as shown in figure A.2. The reply for this case is shown in figure A.3. The return message contains data regarding the lifespan of the token, several dates, information about the priorities, allowed resources and the assigned users. Our token is at the time of the reply active, has a maximum urgency of red (the highest urgency), isn’t a test token and allows several test users high priority access to resources proj15 and proj24.

For simplicity the database has been recreated on the same server the Web services and the portal are on. From a dump provided by a former member of the SPRUCE team the MySQL database could be reconstructed by a simple set of commands. After creating a new database called ”db” in MySQL with `create database db;`, the tables are created in this database with `use db; source sprucedb.sql;`, where *sprucedb.sql* is the database dump. An example of how a few of these tables look is provided in A.4. The database was later filled with some information for testing purposes. Not all tables were used, the ones which were needed and their relations to each other are depicted in figure 6.2. The first word in each box represents the table name, the following the attributes belonging to it. The underlined words represent the primary keys, which are referenced in other tables, where they are depicted in italics.

Most tables are self-explanatory (e.g. **users** contains every person known to SPRUCE along with their email address and their identity (in this case identity stands for a Distinguished Name (DN) to identify the user)); the table **token_perms** contains information on which token has rights on which resource, **token_user** returns the tokens with each of their users and **authentications** (which has a

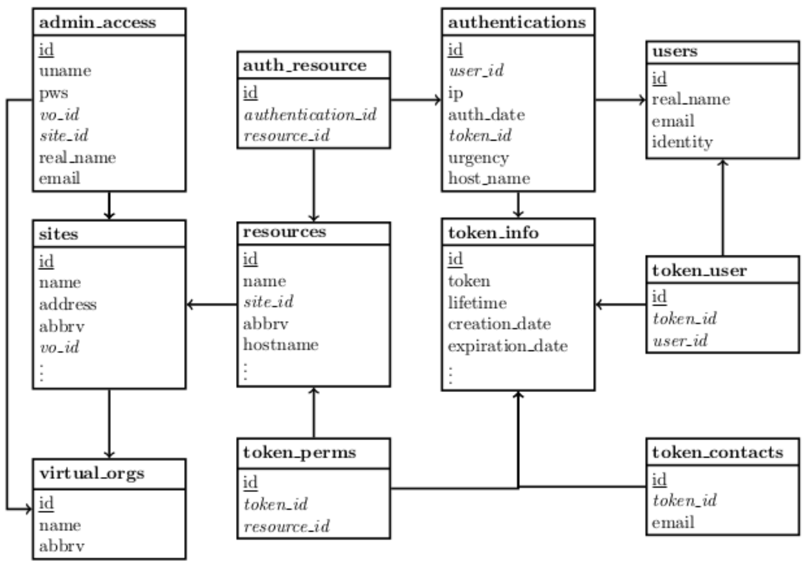


Figure 6.2.: SPRUCE database structure

sibling **failed_auths** for analog use cases) logs the times a user sucessfully used the urgent computing feature with an active token. **Admin_access** is for the administrators of the resources, which usually belong to a virtual organisation. These tables provide all the data needed for both the admin and the user side of managing and utilizing resources with SPRUCE.

While some tables have to be edited by hand (e.g. **admin_access**), most are filled automatically or via the Web services through the portals. The tables keeping track of the authentications are edited when the submitfilter confirms or rejects a proposed authentication, the user tables and some token information can also be changed with functions available on the portal.

6.2.2. SPRUCE on Resources

Some adapations have to be made in the *spruce.pm* and in the *install-spruce* script. The install script requires a few extra lines of code, mainly adding the lib-files to the right folder and changing the environment variables. The *spruce.pm* is bit trickier, but not very difficult. It is written in Perl. The fixed patches have been added in A.5. They have to be added at the corresponding spots to where the installation guide points.

The differences stem from the alterations made when updating GRAM to a newer version. The Globus commands remain unaffected since one can simply specify the desired jobmanager and therefore avoid changing anything in *globusrun, globus-job-submit* and *globus-job-run*. If Globus didn't provide the option to choose a jobmanager beforehand the job submission commands would have to be altered to call the SPRUCE jobmanager.

The other component interfacing SPRUCE with the grid is the submitfilter. These files are still functional and don't need to be adapted.

7. Conclusion and future works

This last chapter will sched some light on the outcome of this thesis. The results of the implementation will be discussed in the first section and some information about what could come in the time ahead in the field of urgent computing.

7.1. Outcome

The implementation has shown that it is indeed possible to adapt SPRUCE to GT5. The implementation was only tested with short jobs and one type of LRM, but should be just as easily adaptable to other schedulers. The alterations made in this thesis are mostly minor corrections to adapt SPRUCE to newer versions of the used technologies.

Other tests could show how SPRUCE works on a larger scale, with a real grid and more schedulers. Scalability might prove to be difficult with the used LRM and with polling. Polling should be substituted by SEG and Maui should not be used. The adaptions for other schedulers should not differ too much from the ones shown in this thesis.

PBS should be the easiest to adapt, since it is closely related to TORQUE, but the others shouldn't prove too difficult, either.

SPRUCE is easily installed and adapted, achieving the goal and answering the questions of this thesis. Generally speaking one can say that SPRUCE provides a valuable service to grid users and could be implemented in most GT5 environments.

7.2. Urgent Computing in the future

SPRUCE hasn't been actively developed in recent times, but there are still many usecases. Urgent computing is still a field in need of improvement but after a few years of stagnation there is the chance of new development in the future.

There is one use case where smartphones are used to create urgent computing client platforms [PKKB 12], meaning smartphones are used as some sort of sensory devices to collect data. If this is happening through GPS, the internal camera or other applications doesn't make much of a difference. All these can collect data to be used in urgent computing simulations.

Other interesting development may come during the ICCS (International Conference on Computer Science) in 2014, where a workshop regarding urgent computing will take place. This workshop will concern itself with the different topics around urgent computing, the methodology, simulations and various applications in real life.

Urgent Computing is being used right now, if not with SPRUCE then by using other means. While it is still in its more experimental stages right now many promising applications can be imagined and quite a few improvements can come from urgent computing.

A. Code Examples

A.1. Portal Setup and Communication

Listing A.1: getTokenInfo.php

```
<?php
# function for calling web-service and processing return value

function show_tokenInfo(){
        $tokenid = $_GET["token"];

        $options= array('trace' => 1, 'timeout' => 300);
        $ws_client = new SoapClient("http://spruce.priv.lab.nm.ifi.lmu.de
            :8080/
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣axis2/services/SpruceUserServices?
    wsdl",$options);
        $xmltokenid = '<spruce:getTokenInfo␣xmlns:spruce="http://spruce.
            priv.lab.nm.ifi.lmu.de/ws/xsd/">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<spruce:token>'.$tokenid.'</spruce:token>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</spruce:getTokenInfo>';
        $params = new SoapVar($xmltokenid, XSD_ANYXML);
        try{
                $response = $ws_client->getTokenInfo($params);
        }catch(SoapFault $e){
                print_r($e->getMessage());
        };
        $response = $ws_client->__getLastResponse();
        unset($ws_client);

        $replArray = array("spruce:", "soapenv:");
        $response = str_replace ($replArray, "", $response);
        $xmlresponse = simplexml_load_string($response);
        $xmlarray = (json_decode(json_encode((array) $xmlresponse),1));
        $body = $xmlarray['Body'];
        $spruceResp = $body['SpruceResponse'];

        $result = buildResponse("getTokenInfo",$spruceResp);
        return json_encode($result);
}
echo show_tokenInfo();

function buildResponse($method, $response){

        $resp = array();
        $resp['token'] = $response['token'];
        $resp['status'] = $response['status'];
        $resp['lifetime'] = $response['lifetime'];
        $resp['creation_date'] = $response['creation_date'];
        $resp['expiration_date'] = $response['expiration_date'];
        $resp['activation_date'] = $response['activation_date'];
        $resp['activation_ip'] = $response['activation_ip'];
        $resp['deactivation_date'] = $response['deactivation_date'];
        $resp['notify_addr'] = $response['notify_addr'];
        $resp['urgency'] = $response['max_urgency'];
        $resp['time_remaining'] = $response['time_remaining'];
        $resp['issued_to'] = $response['issued_to'];
        $resp['test'] = $response['test'];
        $resp['max_auths'] = $response['max_auths'];
        $resp['auths_left'] = $response['auths_left'];
        $resp['UserList'] = array(); #$response['UserList'];

        if (is_array($response['UserList'])){
                foreach ($response['UserList']['UserInfo'] as $l){
                        $user = array();
                        $user['real_name'] = $l['real_name'];
                        $user['identity'] = $l['identity'];
                        $user['email'] = $l['email'];
                        array_push($resp['UserList'], $user);
                }
        }
        else{
                $resp['UserList'] = $response['UserList'];
        }
        $resp['issued_by'] = $response['issued_by'];
        $resp['VO'] = array();
        $resp['VO']['abbrv'] = $response['VO']['abbrv'];
        $resp['VO']['site'] = array();
        if (isset($response['VO']['site'][0])){
                # multiple sites
        }else{
                $resp['VO']['site'][0]['resource'] = array();
                $resp['VO']['site'][0]['abbrv'] = $response['VO']['site'
                    ]['abbrv'];
                if (isset($response['VO']['site']['resource'][0])){
                        foreach ($response['VO']['site']['resource'] as
                            $res){
                                $resource = array();
                                $resource['abbrv'] = $res['abbrv'];
                                array_push($resp['VO']['site'][0]['
                                    resource'], $resource);
                        }
                }
                else{
                        $resp['VO']['site'][0]['resource']['abbrv'] =
                        $response['VO']['site']['resource']['abbrv'];
                }
        }
        return $resp;
}
?>
```

Listing A.2: SOAP Request for token information

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
    envelope/"
                xmlns:ns1="http://SpruceUserPortal.spruce.org/types">
    <SOAP-ENV:Body>
            <spruce:getTokenInfo xmlns:spruce="http://spruce.priv.lab.nm.
                ifi.lmu.de/ws/xsd/">
            <spruce:token>
                4444-4444-4444-4444
            </spruce:token>
        </spruce:getTokenInfo>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing A.3: SOAP Reply for token information

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope
    /">
  <soapenv:Body>
    <spruce:SpruceResponse xmlns:spruce="http://spruce.priv.lab.nm.ifi.
        lmu.de/ws/xsd/">
      <spruce:token>4444-4444-4444-4444</spruce:token>
      <spruce:status>Active</spruce:status>
      <spruce:lifetime>24:00:00</spruce:lifetime>
      <spruce:creation_date>2013-11-18 14:00:00.0</spruce:creation_date>
      <spruce:expiration_date>2014-11-18 15:00:00.0</
          spruce:expiration_date>
      <spruce:activation_date>2013-11-18 15:00:00.0</
          spruce:activation_date>
      <spruce:activation_ip></spruce:activation_ip>
      <spruce:deactivation_date>2014-11-18 15:00:00.0</
          spruce:deactivation_date>
      <spruce:notify_addr>mail@spruce.priv.lab.nm.ifi.lmu.de</
          spruce:notify_addr>
      <spruce:max_urgency>red</spruce:max_urgency>
      <spruce:time_remaining>838:59:59</spruce:time_remaining>
      <spruce:issued_to></spruce:issued_to>
      <spruce:test>false</spruce:test>
      <spruce:max_auths>1000</spruce:max_auths>
      <spruce:auths_left>825</spruce:auths_left>
      <spruce:issued_by spruce:id="1">Test Admin</spruce:issued_by>
      <spruce:VO spruce:id="1">
        <spruce:abbrv>vo1</spruce:abbrv>
        <spruce:site spruce:id="1">
          <spruce:abbrv>projekt24s</spruce:abbrv>
          <spruce:resource spruce:id="1">
            <spruce:abbrv>proj24</spruce:abbrv>
          </spruce:resource>
          <spruce:resource spruce:id="2">
            <spruce:abbrv>proj15</spruce:abbrv>
          </spruce:resource>
        </spruce:site>
      </spruce:VO>
      <spruce:UserList>
        <spruce:UserInfo spruce:id="1">
          <spruce:real_name>Test User</spruce:real_name>
          <spruce:email>testuser@fakemail.com</spruce:email>
          <spruce:identity>testuser</spruce:identity>
        </spruce:UserInfo>
        <spruce:UserInfo spruce:id="4">
          <spruce:real_name>Test Name</spruce:real_name>
          <spruce:email>mail@spruce.priv.lab.nm.ifi.lmu.de</spruce:email>
          <spruce:identity>
             /O=Grid/OU=GlobusTest/OU=simpleCA-projekt24.pub.lab.nm.ifi.
                lmu.de/OU=local/CN=Test User
          </spruce:identity>
          </spruce:UserInfo>
        <spruce:UserInfo spruce:id="5">
          <spruce:real_name>12</spruce:real_name>
          <spruce:email>1</spruce:email>
          <spruce:identity>2</spruce:identity>
        </spruce:UserInfo>
      </spruce:UserList>
    </spruce:SpruceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Listing A.4: SPRUCE database dump

```
-- MySQL dump 10.11
-- Server version       5.0.41

--
-- Table structure for table `token_info`
--

DROP TABLE IF EXISTS `token_info`;
CREATE TABLE `token_info` (
  `id` mediumint(8) unsigned NOT NULL auto_increment,
  `token` varchar(19) NOT NULL default '',
  `lifetime` time NOT NULL default '00:00:00',
  `creation_date` datetime NOT NULL default '0000-00-00␣00:00:00',
  `expiration_date` datetime NOT NULL default '0000-00-00␣00:00:00',
  `activation_date` datetime NOT NULL default '0000-00-00␣00:00:00',
  `deactivation_date` datetime NOT NULL default '0000-00-00␣00:00:00',
  `activation_ip` varchar(15) default NULL,
  `issued_to` varchar(255) NOT NULL default '',
  `issued_by` mediumint(8) unsigned NOT NULL default '0',
  `max_urgency` enum('yellow','orange','red') NOT NULL default 'yellow',
  `notify_addr` text NOT NULL,
  `comment` text,
  `test` tinyint(1) NOT NULL default '0',
  `activation_host_name` varchar(50) default NULL,
  `max_auths` int(11) NOT NULL,
  `auths_left` int(11) NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Table structure for table `users`
--

DROP TABLE IF EXISTS `users`;
CREATE TABLE `users` (
  `id` mediumint(8) unsigned NOT NULL auto_increment,
  `real_name` varchar(128) NOT NULL default '',
  `email` varchar(128) NOT NULL default '',
  `identity` varchar(255) NOT NULL default '',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Table structure for table `virtual_orgs`
--

DROP TABLE IF EXISTS `virtual_orgs`;
CREATE TABLE `virtual_orgs` (
  `id` mediumint(8) unsigned NOT NULL auto_increment,
  `name` varchar(128) NOT NULL default '',
  `abbrv` varchar(16) NOT NULL default '',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

A.2. Resource Adaptions

Listing A.5: Patches for *spruce.pm*, the SPRUCE version of the LRM adapter

```
    ########################   3   ##########################
    my @hosttypes = $description->hosttypes();
    my @hostnames = $description->hostnames();
    my @hostxcount = $description->hostxcount();
    my @xcount = $description->xcount();
    my ($this_hostxcount, $this_xcount, $this);
    my $pbsnodes;
    my $real_count; # Set to value that will supersede $description->
       count()

    if(@hosttypes)
    {
            $self->log("host_types␣=␣" . scalar(@hosttypes));
            foreach my $this_hosttype (@hosttypes)
            {
                    $self->log("host_type␣=␣" . $this_hosttype);
                    $this_hostxcount = shift @hostxcount;
                    $this_xcount = shift @xcount;
                    $this = (($this_hostxcount) ? "$this_hostxcount:" : '
                       1:') .
                    $this_hosttype .
                    (($this_xcount) ? ":ppn=$this_xcount" : '');
                    $pbsnodes .= ($pbsnodes) ? '+' : '';
                    $pbsnodes .= $this;
                    $real_count += (($this_hostxcount) ? $this_hostxcount
                        : 1) * (($this_xcount) ? $this_xcount : 1);
            }
            $rc = print JOB '#PBS␣-l␣nodes=' . $pbsnodes . "\n";
            $real_count ||= 1; # initialize
    }
    elsif(@hostnames)
    {
            $self->log("host_names␣=␣" . scalar(@hostnames));
            foreach my $this_hostname (@hostnames)
            {
                    $self->log("host_name␣=␣" . $this_hostname);
                    $this_xcount = shift @xcount;
                    $this = $this_hostname .
                    (($this_xcount) ? ":ppn=$this_xcount" : '');
                    $pbsnodes .= ($pbsnodes) ? '+' : '';
                    $pbsnodes .= $this;
                    $real_count += (($this_xcount) ? $this_xcount : 1);
            }
            $rc = print JOB '#PBS␣-l␣nodes=' . $pbsnodes . "\n";
            $real_count ||= 1; # initialize
    }
    elsif(@hostxcount)
    {
            $self->log("host_xcount␣=␣" . scalar(@hostxcount));
            foreach my $this_hostxcount (@hostxcount)
            {
                    $self->log("host_xcount␣=␣" . $this_hostxcount);
                    $this_xcount = shift @xcount;
                    $this = $this_hostxcount .
                    (($this_xcount) ? ":ppn=$this_xcount" : '');
                    $pbsnodes .= ($pbsnodes) ? '+' : '';
                    $pbsnodes .= $this;
                    $real_count += $this_hostxcount * (($this_xcount) ?
                       $this_xcount : 1);
            }
            $rc = print JOB '#PBS␣-l␣nodes=' . $pbsnodes . "\n";
    }
    elsif(@xcount)
    {
            $self->log("xcount␣=␣" . scalar(@xcount));
            foreach my $this_xcount (@xcount)
            {
                    $self->log("xcount␣=␣" . $this_xcount);
                    $this = "1:ppn=$this_xcount";
                    $pbsnodes .= ($pbsnodes) ? '+' : '';
                    $pbsnodes .= $this;
                    $real_count += $this_xcount;
            }
            $rc = print JOB '#PBS␣-l␣nodes=' . $pbsnodes . "\n";
    }

    #####################   3   ###########################
#     if (defined $description->nodes())
#     {
#         #Generated by ExtensionsHandler.pm from resourceAllocationGroup
   elements
#         $rc = print JOB '#PBS␣-l␣nodes=', $description->nodes(), "\n";
#     }
    #####################   4   ###########################

    if($description->reservation_id() ne '')
    {
            print JOB '#PBS␣-W␣x=FLAGS:ADVRES:' .  $description->
               reservation_id()

    print JOB '#PBS␣-v␣' . join(',', @new_env) . "\n";

    #####################    4    #########################
    #####################    5    #########################

           print JOB 'touch␣' . $cmd_script_name . "\n";
        }
        else
        {
            return $self->respond_with_failure_extension(
                   "open:␣$cmd_script_name:␣$!",
                   Globus::GRAM::Error::TEMP_SCRIPT_FILE_FAILED());
        }
    #####################    5     ########################
    #####################    6     ########################
    if($description->urgency() eq 'red')
    {

            urgency($pbs_job_script_name,'r');
    }
    elsif($description->urgency() eq 'orange')
    {
            urgency($pbs_job_script_name,'o');
    }
    elsif($description->urgency() eq 'yellow')
    {
            urgency($pbs_job_script_name,'y');
    }
    #####################    6     #########################
#########################    7     #########################
sub urgency
{
        my $pbs_job_script = shift;
        my $type = shift;

        # mark the type that goes to the database call
        if($type eq 'y'){ $level = "yellow"}
        elsif($type eq 'o'){ $level = "orange"}
        elsif($type eq 'r'){ $level = "red"}
        else {print "Incorrect␣urgency␣level,␣aborting␣urgent␣job␣
           submission\n";die"\n";}

        # just tag the script with a SPRUCE comment and leave the token
           checks and policy implementations
        # to the filter code.

        open( JOB,  $pbs_job_script);
        my @lines = <JOB>;
        close (JOB);

        open( JOB, ">$pbs_job_script");
        my $num = 0;

        # add the #SPRUCE_COMMENT line as first, only after any #!bin/
           shell instructions
        if($lines[0] =~ /^#!\s*\//)
        {
                print JOB $lines[0];
                $num++;
        }

        print JOB "#SPRUCE_COMMENT␣$level\n";

        # now simply insert all other job lines

        while($lines[$num])
        {
                print JOB $lines[$num];
                $num++;
        }

        close(JOB);
}
##################   7   ###########################
```

B. Installation Guide

B.1. Requirements

- The Grid
  - A Globus Toolkit 5 Grid Environment (with polling, not SEG packages)
  - Root access to create a new queue and to install the jobmanager and submitfilter
  - The underlying LRMs should be TORQUE/Moab, TORQUE/Maui, LoadLeveler, LSF, PBS Pro or Cobalt for BG/L, since these are supported by the former version of SPRUCE and should be easily adaptable with this guide.
  - An account with the name ”globus” should exist.
- The SPRUCE server
  - An Apache Webserver with a Tomcat Servlet Container and an Axis2 Web Service Stack on top
  - A MySQL database

B.2. Portal Setup

Filling the Database

- DB (creatable by using the database dump provided in the enclosed CD as *sprucedb.sql*)
  `create database <db name>;` (databasename and the root password need to be changed in the Web service files)
  `use db;`
  `source sprucedb.sql;`
- Search for the function *private Connection getDBConnection* in the WS files in the **Webservices** directory on the CD and replace the ”user” and ”password” for the correct database authentications. If the database name isn't **db**, this needs to be changed as well in the connectionUrl.

Deploying the Web services

Then, the Web services need to be made available by dumping them into Axis2 on the Web server. This is done by compiling all the Java-files and then creating a .aar file for the Web services. In our case we only need the submitfilter and the userportal.
`cd <dir>`, where dir is Webservices/submitfilter or Webservices/userportal
`jar cf <name>.aar *`

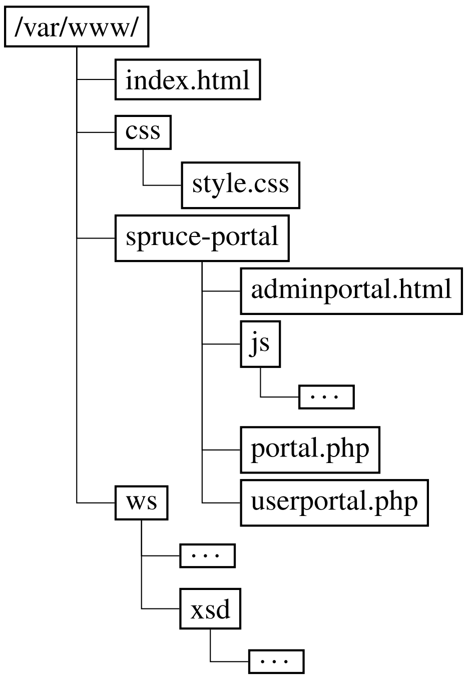
```
 mv <name>.aar /var/lib/tomcat6/webapps/axis2/WEB-INF/services
```
And then add the names of the .aar files in
`/var/lib/tomcat6/webapps/axis2/WEB-INF/services/services.list`.
The location of these files and directories can change with different Tomcat versions.

After restarting Apache and Tomcat, the Web services can be found at
`<websiteURL>:8080/axis2/services/listServices`.

If the database and the Web services are not on the same server, two functions in the submitfilter need to be slighlty altered. They have been changed to automatically assume localhost instead of querying for the database IP address. The two functions are called *success* and *fail*. The former solution threw an unknown axis fault and had to be replaced.

Web Frontend

The code for the portals can be found in the **Portal/** directory on the CD. They need to placed in `/var/www/` like this:



This provides a functional Web portal for SPRUCE with the ability to communicate with the Web services and the database. Now that the portal section is up and running, the installation on the resources has to be considered.

B.3. SPRUCE on Resources

SPRUCE installation

For the installation of SPRUCE one can simply follow the guides on
`http://spruce.teragrid.org/software.php`, but instead of using the provided install script and the provided *spruce.pm*, the files in **Spruce/** on the CD need to be used. These files are

only applicable with TORQUE/Maui, the other schedulers need to be examined and adapted accordingly.
To create and execute the installation files, simply enter the `src/resource_provider/` directory and execute `make`. Then enter the `build/resource_provier/` directory and execute `./install-spruce`.

This can be done by examining the directory structure and either symlinking to obtain the wanted structure or to rewrite the install script.

LRM Configuration

If TORQUE/Maui is installed, the *maui.cfg* can look like the one in **LRM/**. The important parts of configuring TORQUE are creating the second queue:

```
qmgr -c "create queue spruce queue_type=execution"
qmgr -c "set queue spruce started=true"
qmgr -c "set queue spruce enabled=true"
qmgr -c "set queue spruce resources_default.nodes=1:ia64-compute"
qmgr -c "set queue spruce resources_default.walltime=00:15:00"
```

Adding the available nodes to:
`/root/torque-4.2.5/tpackages/server/var/spool/torque/server_priv/nodes`
or like this:
`qmgr -c "create node <name>"`

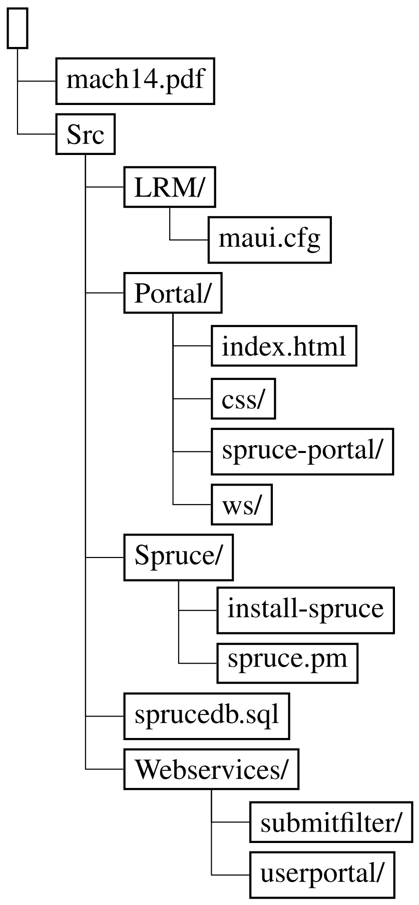
TORQUE requires *pbs_mom* on every compute node and *pbs_serv* on the head node. Maui is only require on the head node.

Since polling is used to query the job status a */etc/grid-services/* entry needs to be added. The file */etc/grid-services/jobmanager-spruce-poll* must contain this line:
`stderr_log,local_cred - /usr/sbin/globus-job-manager globus-job-manager -conf /etc/globus/globus-gram-job-manager.conf -type spruce`
Copy this file to */etc/grid-services/available/*, restart the Globus Gatekeeper and then execute
`globus-gatekeeper-admin -e jobmanager-spruce-poll -n jobmanager-spruce`.
If polling is no longer desired the SEG-module can be downloaded for Globus. Then a SPRUCE SEG-module would need to be written.
With these installations and changes, SPRUCE should now be running and is accessible by calling Globus commands like this:
`globusrun -r <hostname>/jobmanager-spruce -o -f <filename>.rsl` where *<filename>.rsl* contains (among others) an urgency parameter:

```
&
(rsl_substitution = (HOMEDIR "<homedir>"))
(executable = $(HOMEDIR)/<textexecutable>)
(jobType = single)
(urgency = red)
(name = <jobname>)
(savejobdescription = yes)
```

Now jobmanager-spruce should prioritize this job higher and start it in the next possible free slot.

The attached CD has the following structure:



Bibliography

[Adaptive Computing] *TORQUE Documentation*, http://docs.adaptivecomputing.com/-torque/help.htm . Accessed: 01 December 2013.

[BNTB 06] BECKMAN, PETE, SUMAN NADELLA, NICK TREBON and IVAN BESCHASTNIKH: *SPRUCE: A System for Supporting Urgent High-Performance Computing*. In *IFIP WoCo9 Conference Proceedings, Arizona*, 2006.

[BWE$^+$ 00] BUTLER, RANDY, VON WELCH, DOUGLAS ENGERT, IAN FOSTER, STEVEN TUECKE, JOHN VOLMER and CARL KESSELMAN: *A National-Scale Authentication Infrastructure*. Computer, 33(12):60–66, December 2000, http://dx.doi.org/10.1109/2.889094 .

[DCC$^+$ 04] DROEGEMEIER, KELVIN K., V. CH, RICHARD CLARK, DENNIS GANNON, SARA GRAVES, MOHAN RAMAMURTHY, ROBERT WILHELMSON, KEITH BREWSTER, BEN DOMENICO, THERESA LEYTON, DANIEL WEBER, ANNE WILSON, MING XUE and SEPIDEH YALDA: *6.1 Linked Environments for Atmospheric Discovery (LEAD): A Cyberinfrastructure for mesoscale meteorology research and education*. In *Proc. 20th Conf. Interactive Information Processing Systems for Meteorology, Oceanograpgy, and Hydrology, Am. Meteorological Soc.*, 2004.

[FFM 07] FELLER, MARTIN, IAN FOSTER and STUART MARTIN: *GT4 GRAM: A functionality and performance study*, 2007.

[FKT 01] FOSTER, IAN, CARL KESSELMAN and STEVEN TUECKE: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. Int. J. High Perform. Comput. Appl., 15(3):200–222, August 2001, http://dx.doi.org/10.1177/109434200101500302 .

[Fost 02] FOSTER, IAN: *What is the Grid? A Three Point Checklist*. June 2002, http://www-fp.mcs.anl.gov/˜foster/Articles/WhatIsTheGrid.pdf .

[GENIUS] *Grid Enabled Neurosurgical Imaging Using Simulation*, http://wiki.realitygrid.org/wiki/GENIUS . Accessed: 10 January 2014.

[Globus Alliance] *About Globus Toolkit*, http://www.globus.org/toolkit/about.html . Accessed: 05 December 2013.

[Globus Alliance Documentation] *GRAM5 Key Concepts*, http://www.globus.org/toolkit/docs/5.2/-5.2.4/gram5/key . Accessed: 01 December 2013.

[Globus Homepage] *GT5: C Common Libraries*, http://toolkit.globus.org/toolkit/-docs/5.0/5.0.0/common/ccommonlib . Accessed: 11 December 2013.

[GM1] *UNICORE Homepage*, http://www.unicore.org/unicore/ . Accessed: 22 November 2013.

[GM2] *gLite Homepage*, http://glite.web.cern.ch/glite/ . Accessed: 25 November 2013.

[GT5] *Globus Toolkit 5*, toolkit.globus.org/toolkit/docs/5.2/5.2.4 . Accessed: 10 November 2013.

[HTCondor] *HTCondor Homepage*, http://research.cs.wisc.edu/htcondor/ . Accessed: 05 December 2013.

[LRZ] *Leibniz-Rechenzentrum Homepage*, http://www.lrz.de/services/-compute/grid_en/grid-middleware_en/globus_guide_en/ . Accessed: 15 January 2014.

[LSF] *LSF Homepage*, http://www-03.ibm.com/systems/-technicalcomputing/platformcomputing/products/lsf/ . Accessed: 05 December 2013.

[Maui] *Ten Reasons to Switch from Maui Cluster Scheduler to Moab HPC Suite*, http://www.adaptivecomputing.com/wp-content/uploads/-collateral/TenReasonsToSwitchFromMauiToMoab2012-01-05.pdf . Accessed: 15 January 2014.

[PBS] *PBS Homepage*, http://www.pbsworks.com/ . Accessed: 01 December 2013.

[PKKB 12] PALMER, NICHOLAS, ROELOF KEMP, THILO KIELMANN and HENRI E. BAL: *The Case for Smartphones as an Urgent Computing Client Platform.* In ALI, HESHAM H., YONG SHI, DEEPAK KHAZANCHI, MICHAEL LEES, G. DICK VAN ALBADA, JACK DONGARRA and PETER M. A. SLOOT (editors): *ICCS*, volume 9 of *Procedia Computer Science*, pages 1667–1676. Elsevier, 2012, http://dblp.uni-trier.de/db/journals/procedia/procedia9.html#PalmerKKB12 .

[RN] *Globus Toolkit Release Notes*, http://toolkit.globus.org/toolkit/-releasenotes/ . Accessed: 10 November 2013.

[SCOOP] ALLEN, G., P. BOGDEN, T. KOSAR, A. KULSHRESTHA, G. NAMALA, S. TUMMALA and E. SEIDEL: *Cyberinfrastructure for Coastal Hazard Prediction.* CTWatch Quarterly, 4(1), March 2008, http://www.ctwatch.org/quarterly/articles/2008/03/-cyberinfrastructure-for-coastal-hazard-prediction/ .

[SPRUCE] *SPRUCE Homepage*, http://www.spruce.teragrid.org/ . Accessed: 05 December 2013.

[Teragrid] *TeraGrid Archives*, http://www.xsede.org/tg-archives . Accessed: 20 December 2013.

[Treb 11] TREBON, NICHOLAS: *Enabling urgent computing within the existing distributed computing infrastructure*. PhD thesis, Chicago, IL, USA, 2011. AAI3472964.

[WS GRAM] *WS GRAM Release Notes*, toolkit.globus.org/toolkit/docs/4.0/-execution/wsgram/WS_GRAM_Release_Notes.html . Accessed: 10 December 2013.