Developing Distributed Computing Solutions Combining Grid Computing and Public Computing

M.Sc. Thesis

Christian Ulrik Søttrup   Jakob Gregor Pedersen

Department of Computer Science
University of Copenhagen

1st March 2005

Abstract

As modern research relies more and more on computers, computer cycles are becoming a scarce resource for research projects, as well as a large part of the cost. Some projects try to solve this problem by relying on computer resources donated by the public. The drawback to this approach is that the public resources are stochastic in nature. It is therefore not possible to predict when a study will finish.

Our goal is to create a resource that will supply a scientific research project with reliable compute power using computers donated by the public. To this end we will use BOINC, a newly created open platform that tries to lower the cost of computational resources for research by using donated cycles. We propose to build a bridge between this platform and Grid computing to solve the problem of reliability. Grid computing is a platform that seeks to solve the problem of scarcity. It does this by allowing researchers around the world to buy, sell, or share computational resources. We show that using a mix of these resources it should be possible to supply an inexpensive resource with guarantees on the quality of service. We also propose a design for integrating this resource into the Grid, thus making it easier to use.

Contents

List of Figures

Chapter 1

Introduction

The purpose of this thesis is to study the possibility of creating a hybrid between two methods of distributed computing, namely Grid computing and Public Resource Computing(PRC[1]). To achieve this aim we first create a PRC project that is able to supply a significant computational resource. We will then use this PRC project as a basis for creating bridges between these two types of computing. These bridges should allow the Grid to make use of the resources supplied by the public and also allow a PRC project to deliver some of the features normally found only in Grid Computing.

1.1 Motivation

In September 2004 CERN[2] celebrated its 50th anniversary. In connection with this, many different events were planned to raise the public's awareness of CERN's work in particular and natural science in general. The IT department at CERN was approached to develop a public Computing Challenge that would illustrate distributed computing, and if possible make connections to CERN's work in the area of Grid computing, specifically the Large Hadron Collider Computing Grid, or LCG. Another reason for studying the possibility of running a PRC project at CERN, is the compute power that is needed once the new particle accelerator is ready. The needed compute power is far greater than what CERN can hope to supply locally. This is the reason why CERN, which is otherwise a physics research institution, is investing in computer science, specifically Grid research. The recent appearance of BOINC[3] as a general platform for PRC provided an interesting opportunity in these respects. SETI@home[4] had already shown that not only are many people interested in donating their spare computer power in the aid of science, but also that once they have done so they are more likely to get interested in the science their machine is working on. The latest measurement of SETI@home's performance in 2000, put it at about 15 TeraFLOPS. This was faster than IBM's ASCI White, the fastest supercomputer at that time. The compute power available from a PRC project could therefore be a significant resource for CERN.
One problem with PRC projects is that the performance they provide is not reliable. You cannot be certain whether you will get any resources at all. Another problem is that the

[1] For further information on PRC, see section 2.
[2] Centre Européenne pour le Recherche Nucléaire. A nuclear research facility in Switzerland, see section 3.1
[3] BOINC is covered in section 2.2
[4] A PRC project searching for signs of extraterrestrial intelligence. Also the precursor to BOINC.

application often has to be tailored specifically to the PRC, making it time consuming to start using a PRC resource. An interesting solution to this problem would be to create a Grid PRC hybrid. It would allow ordinary Grid jobs to run under a PRC platform. It would also be able to supply a quality of service by using a mix of PRC resources and Grid resources.

1.2 Problem Description and Approach

To summarize our goals are to:

- Create a PRC project that at the same time will be a part of CERN's 50th anniversary and be proof that the concept of PRC is a feasible computational resource for CERN by supplying a significant increase in compute power for a CERN application.
- Design and implement a bridge from PRC to the Grid, to show that it is possible to satisfy some form of quality of service requirements from a PRC resource.
- Discuss and design a bridge from the Grid to a PRC platform.

We aim to tackle these problems by setting up a PRC project running SixTrack, a CERN application vital to the construction of experiments at CERN. SixTrack being an interesting choice because it does not have enough computational power at its disposal to do all the studies the physicists demand. We will use the BOINC framework to build the PRC project. We will then use the experience, and the data we gather to design and implement a bridge from BOINC to CERN's Grid implementation: LCG-2. We aim to ameliorate the stochastic nature of cycle-scavenged resources. Finally we will discuss the design of integrating BOINC into the LCG-2.

1.3 Challenges

There are several challenges to be met when solving the problems outlined above. We will discuss the main ones here.
First, there are only a handful of general platforms for doing PRC and most of these have only become available recently. We will have to choose and familiarize ourselves with a PRC platform, port an application to the PRC platform and run extensive tests on the port. Extensive testing is important, because we need to attract a large user base to show whether a PRC project is able to deliver an interesting amount of compute power.
Second, PRC and Grid computing are two very different approaches to distributed computing. Combining the two will of course require familiarizing ourselves with the concept of a Grid and with LCG-2 in particular.
To successfully build a bridge from BOINC to LCG-2 we will have to adapt the BOINC system to be able to take advantage of the Grid resources. This is because adapting LCG-2 due to its massive scale is out of the question. We will also have to decide on a QoS metric and create a way of honouring it. Lastly we have to test the bridge preferably in a production scale environment to see the results of our efforts.
Concerning the bridge from the Grid to a PRC platform we have to come up with a way of running the very generic jobs from the Grid on the non-generic resources of the PRC platform. Since Grid resources are generally trusted whereas PRC resources are not, we have to figure out how to deal with running jobs meant for trusted resources on non-trusted resources. After

solving these problems in general we have to solve them for the specific systems of LCG-2 and BOINC. We also plan to implement a partial prototype of such a LCG-2 to BOINC system to study the feasibility of the concept.

1.4 Contributions

The main contributions of this project are:

- We created a very popular PRC project, showing that the PRC concept can be a significant computational resource for CERN. It outstripped the other resources available to SixTrack by far. As a side benefit it also showed that running a PRC project can be a great tool for public relations.
- Creating a bridge from BOINC to LCG-2. Unfortunately we cannot prove that this bridge makes it possible to supply a quality of service for a PRC platform. Because the LCG-2 resources that were at our disposal, only were best effort and therefore no better than a cycle-scavenging scheme. However, initial tests show promise.
- Our discussion of the design of a Grid to BOINC bridge aims to be a helpful guide to those wishing to implement such a bridge. We have also made the first steps in integrating the BOINC server into LCG-2's job submission system.

In addition we have helped mature the BOINC platform. Both by testing the framework as well as supplying bug-fixes, documentation, and new features beyond those discussed above.

1.5 Sources and Acknowledgments

The LHC@home project became a large part of our work and our thesis. Because of the size of the project many people besides the authors contributed to it in one way or another. In this section we will introduce the people who helped us, and what role they played in the creation of the LHC@home project. When we describe different parts of the LHC@home project, in chapter 3, we will mention the appropriate person if that part was not made by us.
The project was led by Francois Grey, group leader of the IT Communications team at CERN and the otherwise retired Ben Segal, formerly the manager of EDG WP2[5]. Francois handled the contacts to upper management and Ben the day to day supervision.
The project needed to run a relevant CERN application to be of use. The application that was chosen was SixTrack, see section 3.2. The people who helped us understand and port this application were Eric McIntosh, who does error analysis and optimization on the SixTrack application, and Frank Schmidt, the creator of SixTrack.
Because of our other responsibilities we were unable to handle the daily server maintenance on our own. A server administrator was therefore needed. The first was Karl Chen, a student who had worked on BOINC at Berkeley. He helped us design and set up the needed servers for the project building on his SETI@home experience, but had to return to his studies before the project went public. The next two, replacing each other in rapid succession, were Kalle Happonen and Markku Degerholm, both of them students from Helsinki Institute of Physics.
One of the prerequisites for a successful BOINC project is doing an interesting screensaver.

[5] A workgroup working on the Data Management architecture of the European Data Grid

Since this was outside the scope of this thesis, Jasenko Zivanov, a summer student from Basel University, was hired to create this under our and Ben Segal's supervision.
We would also like to thank David Anderson, the creator of BOINC. Although not directly a member of the LHC@home team he was always very helpful in explaining the BOINC system or helping us track down bugs in the BOINC software.
The second part of our thesis, designing bridges between Grid and BOINC, was done entirely by us, but would not have been possible without the generous help of the LCG team, especially Maarten Litmaath and David Smith. They were always willing to demonstrate the workings of the LCG, whenever the documentation was insufficient or completely lacking, or discuss issues on the design of the bridges between LCG-2 and BOINC.
We would also like to thank Ben Segal and Francois Grey, our supervisors at CERN, and Jørgen Sværke Hansen, our adviser at DIKU, for guidance, comments and suggestions during the project. Thanks to Steffen Lauritzen and Tabita Holrick for proofreading.
A final thanks to Dominique Dupraz for helping us with the french authorities, apartments and any other tasks requiring the use of French.

1.6 Structure of the Report

We begin by describing the concept of PRC and introduce some of the different PRC systems through time, in chapter 2. In chapter 3 we describe the specifics of our team's work in deploying the BOINC framework and the LHC@home application at CERN. In the following chapter we introduce Grid computing particularly the LCG-2 platform. The authors' design of the bridges is treated in two separate chapters: Chapter 5 describes the bridge that allows BOINC jobs to migrate to the Grid and chapter 6 describes the other direction.

Chapter 2

Public Resource Computing

In this chapter we describe the PRC model for distributed computing. Two PRC platforms will be discussed in detail, namely BOINC and Condor. We compare their respective strengths and weaknesses and, argue why BOINC is a better PRC platform. We also give a short description of other PRC platforms including Frontier from Parabon, DCGrid from Entropia, Grid MP from United Devices, and OfficeGRID from MESH-Technologies.

2.1 The Concept

PRC describes a computational model pioneered by the distributed.net project [11]. It is also known under the names Internet Computing, Public Computing, and Meta Computing among others. We will use the name Public Resource Computing(PRC) because we feel that it best describes the model. The idea is to get anybody with an Internet connection and spare compute power to donate CPU cycles on their computer. This leads to a very heterogeneous distributed model, as the people donating their compute power range from grandmothers to cluster administrators, so can the machinery range from x86 machines on an analog modem to UNIX clusters connected with fibre optics. Both the performance and the hardware architecture can therefore differ a lot between the donating machines.

Using compute power that would otherwise go to waste is often called cycle-scavenging or cycle-harvesting and there are also quite a few systems specializing in doing just this without necessarily involving the public.

There are a number of technical differences between public resources and normal dedicated resources that cause quite a few problems to be solved by PRC platforms. First of all as mentioned above some public resources are connected via modems making them only connected at certain times, and even if an ADSL line for instance is used the actual computer may not always be on. Therefore it cannot be assumed that public resources can always be reached hence disconnected operation must be supported. Secondly public resources are usually protected by firewalls or are sitting behind NAT-enabled[1] routers with the resources using local IP addresses instead of global IP addresses. This means it cannot be assumed that public resources accept incoming connections. Most PRC platforms solve these problems by having one central queue of jobs, which gets pulled out by clients connecting to a central scheduler. Pulling jobs instead of pushing them to clients solves the problem of clients not being connected at all times and not accepting incoming connections. Because of these assumptions of

[1] Network Address Translation.

connections to the clients and the pull model, there is usually no contact between the clients and the scheduler during the computation leaving the scheduler with no idea of the computation's progress if any. It also means that scheduling decisions can only be made when a client connects to the central scheduler.
Another issue is trust. When a computation is being run on hardware within the submitter's control, the result of the computation can be trusted. When the computation is done on an unknown stranger's computer not within the submitter's control it is uncertain whether the result can be trusted or not. Trust also goes the other way around, the clients trust the project not to destroy any of their data or their hardware for that matter. They also trust the project not to go snooping around their computer for sensitive data such as passwords, credit card numbers etc. To enhance the trust of the people donating cycles many PRC platforms employ a technique called sandboxing on the clients computers. Sandboxing isolates a computation from the rest of the computer thereby preventing it from damaging anything outside the sandbox. Sandboxing is usually achieved by intercepting system calls, thus adding a layer between the application and the operating system (OS), and deciding whether a given call is safe to perform or not. It is obvious that such a feature will help increase the users' trust in a given system.
No one does anything without a reason, so the clients need a reason for donating their spare compute power. This reason could be to help out science, help cure cancer, or any other reason, but the bottom line is that projects need to motivate the clients. The most common and successful way of motivating clients is the granting of credits. Credit is a measurement of the work done by one or more clients and it can really motivate people because it brings up the competitive spirit in people. Some PRC platforms also involve payments to the clients for used compute power, but most do not.
The model lends itself very well to problems of an "embarrassingly parallel" nature, but not other types of parallel problems. This is because of the asymmetry in communication and compute power among the donated machines. For the same reason problems that require a fast response time are very ill-suited for this type of computing. On the other hand the sheer throughput capability of a 100,000 machines is astonishing. As we see in section 3.1, many of the computing tasks at CERN are ideally suited for PRC. One of the problems of PRC is that absolutely no quality of service can be guaranteed. You have no idea when results will return and you cannot be assured that your community will not suddenly start a boycott. This problem and what can be done about it will be examined in chapter 5.

2.2 BOINC

Berkeley Open Interface for Network Computing (BOINC) is an open source middleware platform for doing PRC. It is being developed by the team that made SETI@home lead by David Anderson. Their reason for doing this is that at the moment they have 3-4 times more compute power at their disposal than they can supply data for. SETI@home analyse data from a radio telescope and this telescope supplies a fixed amount of data, so once enough compute power to analyse the given amount of data has been used, excess compute power goes unused. They are therefore interested in sharing their community with other scientific projects, and to this end they have created BOINC. Like SETI@home users gain credit when their computer does work for projects, these credits are then used to display leader boards on web pages. The only use of the credits is to create a competitive spirit among the users.

No payment for use of the computers is implemented.
BOINC consists of a client which is shared among all BOINC projects, and a project specific server complex[2] and application program(s). Figure 2.1 shows an overview of the BOINC system, with which we will go into details in the next sections starting with the client. BOINC uses the above mentioned scheme of having a central work queue, depicted as the BOINC database in the figure, and of having clients connect to the scheduler. It might be a good idea to keep an eye on figure 2.1 when reading the next sections to aid in the understanding of the system.

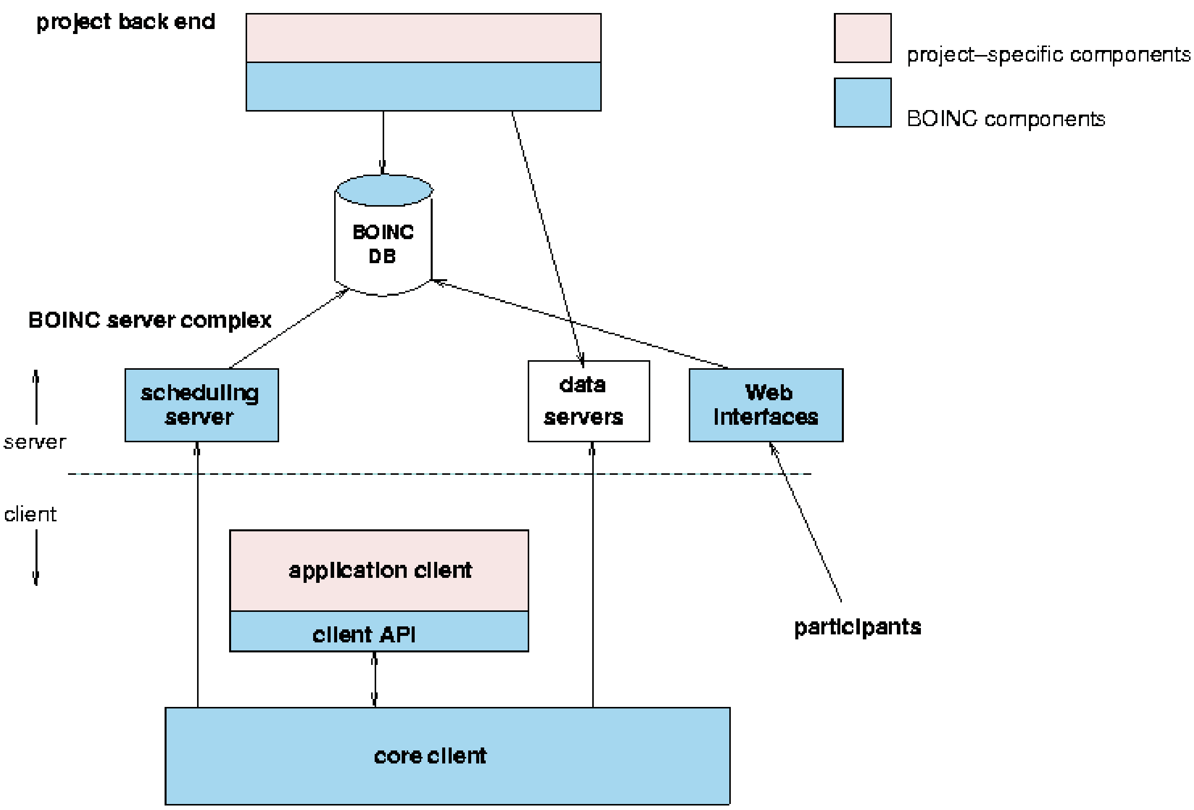


Figure 2.1: The BOINC system. From [1].

2.2.1 The Client

A user who wants to donate his computer's spare cycles to a BOINC project downloads and installs the BOINC client, also known as the core client. This client can be found at the website of every BOINC project and it is therefore usually downloaded from the first project the user signs up for. After installation of the client software, the user visits a project website and registers as a user. Shortly after having registered, the user receives an email with a unique user id. This id together with the project's URL is then entered by the user to sign his BOINC client up for the specific project. This process is called attaching the client to a

[2]The server side of the BOINC platform is called the server complex since it can be made up of several physical machines. In the following BOINC server and BOINC server complex is used interchangeably.

project and the reverse process is called detaching. The client is attachable to many projects at the same time and allows the user to specify how large a percentage of time on average should be spent on each project.
The client takes care of scheduling among jobs from different projects, possibly preempting jobs, downloading and uploading results to the different projects it is attached to.
The user has a lot of control over the client via user preferences. Some of these preferences are project specific and some are not. Among other things the user can specify whether the core client should run jobs while the user is actively using the computer or only use it after a specified period of time without user activity. As a project specific preference the user can set a minimum and a maximum amount of work the client should keep on the computer for the given project. Once the amount of work drops below the minimum amount, the client contacts the project asking for enough work to get to the maximum amount. During this session the client also reports any finished results[3] since the last contact and checks for new application versions. Finishing results is a two step process by the client, first the result is uploaded as soon as it is computed and a network connection is available, and afterwards, during the scheduling session, the result is reported i.e. the scheduler is informed that the result is computed and uploaded. This scheme means that finished results can lie for days on either the client or on the web server without the scheduler knowing about it. The local queue of jobs on the client is processed in FIFO[4] style, but since a client can be connected to many projects at once the client needs to schedule among the different projects. The clients run as many jobs at once as the number of CPUs on the host system[5]. It also reports how much CPU time was spent on each result[6]. This is normally used to calculate the amount of credit a user is awarded. This scheme has been devised to allow users without constant Internet connection to participate in BOINC projects. The client communicates with the server using normal HTTP to get through firewalls.
Besides doing all the work needed to run jobs the client is also capable of displaying application specific graphics and these graphics can be used as a screensaver. All of this is of course to attract and entertain users in the first place.
The client is available for Windows and Linux on Intel X86 architectures, for Mac OS X on PowerPC, and Solaris on Sparc architectures, but since it is open source it should not be too difficult to get it running on other platforms as well.

2.2.2 User Security

From the users' perspective security is ensured by the project hashing all programs and data, and signing the hashes with a private key. This signing should be done on a computer that is not connected to the Internet to minimize the risk of getting the key stolen. This is the only security the users have. They can be certain that the code they got was from the project they signed up with, but they have to trust that the project is not, deliberately or otherwise, delivering malicious code. Nor can they be certain that somebody had not already at the beginning spoofed the project. For example somebody could have made the SET1@home[7]

[3] The concept of results is explained in section 2.2.2.
[4] First-In, First-Out.
[5] Since hyper-threading CPU's are identified as a normal SMP system to the OS the client runs two jobs per hyper-threading CPU (Only two-way hyper-threading is available at present).
[6] Again the concept of results is explained in section 2.2.2.
[7] Notice the slight misspelling.

project to lure people who accidentally went to the wrong homepage into running a distributed mail server for spamming.
One idea to get around this problem is to make a BOINC client that can accept a list of projects to block. This way projects that are spoofing could be blocked. This of course only helps if the project where the user downloads the client from is being honest. This is currently not implemented in the core client. The client also protects users from runaway jobs by enforcing limits on the CPU-time and on the disk usage of a job. If these limits or an overall limit on the disk usage by BOINC is exceeded the job is killed and its files deleted. The job limits are set by the project when creating a job but the overall disk usage limit is set by the individual users. This scheme means users have to trust the project's limit on CPU-time but is protected against excess disk usage even without trusting the project.

2.2.3 Project Security

David Anderson gained a lot of experience with hackers during the SETI project. He found that there are generally three ways people will try to hack a PRC system. These three ways also apply to the BOINC system. We believe they are not exhaustive ways of hacking the direct functioning of BOINC, but the major ways. Some people will try to hack the web server or the other standard server components, but these are not directly related to BOINC and will therefore not be discussed here, though care should of course be taken to try and prevent these attacks as well. The three ways are:

1. Users might try to return wrong results in the hope of destroying the project goals.
2. Users might try to return very large results in the hope of overburdening the server.
3. Users might try to get more credit than what they are due.

The first problem is solved by giving out the same calculation to a number of different users. This number can be set for each job individually, but the BOINC standard is five. When the results are returned they are compared according to a comparison algorithm the project supplies. When enough results agree, a canonical result is chosen. This means that if anybody returns a wrong result it will disagree with the quorum, be discarded and the user will not be awarded any credit. The number of results that make up a quorum can also be specified for each job, but the default is three.
This redundant computing gives rise to a somewhat confusing terminology, a problem instance is called a "WorkUnit" (WU) and copies of the problem instance representing the redundant computations are called "results" even though they have not yet been computed. Whenever the administrator creates a job, he is in fact creating a WU, and from this WU a number of results to be given out to clients for computing is created. The number of results created is specified when the WU is created along with a number of different parameters regarding the results. One more reason for doing redundant computing besides the built-in safety is to cope with client errors and clients never returning the result, perhaps because the user got bored with the project and detached his client. If the initially created results all fail or timeout more result are created up until a limit set when the WU was created.
The second problem is solved by limiting the maximum size of the output files that the server will accept. This is one of the parameters given when creating a WU. When clients return results they connect to a CGI[8] program called the uploadhandler. It, as the name

[8]Common Gateway Interface.

implies, handles the upload and kills the upload if the maximum size is exceeded. The third problem does not sound so serious, after all it does not hurt the research if some user gets a higher rank[9] than he is supposed to have. The problem lies in the psychology of keeping your community willing to give you their computer time. A few users whose foremost interest is helping science will continue but other users will become disgusted with the cheaters and unfortunately also the project and will therefore stop donating time to the project. BOINC solves this problem in the same way that it solves the problem of people returning wrong results, by doing redundant computing. Users returning valid results will normally receive the average credit of the other valid results for the specific calculation.[10] So if a user tries to get too much credit he will simply end up giving more credit to everybody with a valid result for the given calculation. Thereby lessening the incentive to cheat.

As stated above we do not believe the three ways are exhaustive since it would be possible to stage a Distributed Denial of Service(DDoS) attack on the server complex by having clients constantly upload the same results. This would waste resources on the server complex and generate a lot of network traffic possibly denying behaving users of service. It is very difficult to protect against this type of attack, even if uploading of the same file repeatedly is prevented it will still require some resources to figure out that a given file has already been uploaded and turn the client away. We believe there are more ways of tampering with the system than those presented here, but at present no other methods have been used to attack the existing BOINC projects.

2.2.4 The BOINC Server

The BOINC server consists of at least one web server that handles up- and downloads and a database server that keeps track of the state of the WorkUnits and their associated results. Furthermore five different daemons periodically check the state of the database and perform any needed tasks within their area of responsibility. All these programs can be run on the same machine or they can be distributed to different servers for performance reasons. This was one of the lessons learned from SETI@home. The server runs on Linux and Solaris, but being open source other platforms is a possibility.

Below we go into details of the five daemons and a few other components that make up the BOINC server.

2.2.4.1 The Transitioner

This program is application independent. It handles the state transition of WUs and results. The transitioner checks the state of a WU in the database and updates the appropriate state fields in the database, once a WU is ready for a state transition. This is complicated by the fact that a WU does not have an overall state, but a set of substates. These substates involve the states of its associated results. For instance if they are ready for validation, and there is enough results ready for validation to grant a quorum the state of the WU is changed to "ready for validation". The transitioner generates the initial results and generates more if timeouts or errors occur. It is one of the most CPU intensive programs, it can therefore be

[9] Users are ranked according to their contribution to individual projects causing competition among them. This is a major driving force in getting users to donate their compute power.

[10] How this is actually handled is entirely up to the validator, which is project specific. For instance the minimum credit could be awarded to all users who returned a valid result.

split into many processes that each has the responsibility for a subset of WUs. These processes can then be distributed to different servers.

2.2.4.2 The Validator

The purpose of this daemon is to verify whether the returned results are valid or not. It is part of the project back end shown on figure 2.1. When run it checks the database to see if there are any new results uploaded that need validation. If there is, the validator then runs an application specific function for comparing the results. The implementer of the validator functions has to specify two functions; one that compares two results and one that compares a set of results. The first is used to decide whether to grant credit when a new result is uploaded and a canonical result has already been found. The second is used to decide on a canonical result from a set of results. The implementer might for example consider results that agree to the third decimal as good enough, but require that at least four results are within this range. Furthermore it allows the user to perform further sanity checks on the results. If we for instance were simulating gravity on a body dropping freely in the atmosphere, we could check if it ended up higher than it started. This should never happen so we would not accept the result. Whether the cause of this occurrence is errors in the client or in the simulation program, we would never want such a result to be validated. When a WU is created it is specified how many agreeing results are needed for that particular WU. This value could have been set on an application basis but since different people with different needs could potentially be submitting WUs for the same application the method chosen allows these different people to set different values. It is also possible that WUs the same person has different needs. Once there are enough agreeing results a canonical result for the WU is chosen from the set of agreeing results. The WU is hereafter flagged in the database as ready for assimilation.

2.2.4.3 The Assimilator

The assimilator regularly checks if new WUs are ready for assimilation. It is like the validator part of the project backend depicted in figure 2.1. The administrator must supply a function that determines what is to be done with the canonical results. It could for example zip up the result files and e-mail it to the person interested in it, or automatically do postprocessing on the data extracting the interesting parts and storing it on a magnetic tape storage device. Once the WU has been assimilated, the WU will be flagged as completed.

2.2.4.4 The File Deleter

The file deleter daemon simply checks for completed and assimilated WUs accumulated since its last run. If so, it cleans the web server of input and output files related to these WUs. It is therefore essential that the canonical result's output files are copied somewhere safe in the assimilation phase, if they are needed. It only deletes files not database records, so it is always possible to go through the database and find information on WUs, results etc. even after their completion (and file deletion...).

2.2.4.5 The Feeder

The feeder loads undispatched (unsent in BOINC terms) results from the database into a shared memory segment. This prefetching of results is done to improve performance of the entire BOINC system by limiting the number of queries on the database. Especially the performance of the scheduler (see below) is improved, when dispatching results to clients it only needs to access the shared memory segment, which contains many more results than needed for one client, instead of doing a query on the database. The results loaded into the memory segment were randomly chosen in the early versions of BOINC to prevent cheating as it would make it virtually impossible to have control of say 5 clients and have them connect in rapid succession to get all results associated with a specific WU. If one user got all results for a specific WU, he could easily fool the validator. Because of performance problems the new versions of BOINC load results on a first come first served basis, meaning results are loaded in the order their respective WUs were submitted. all: Command not found.

2.2.4.6 The Scheduler

The scheduler is a CGI program that is run whenever a client connects to the project and asks for work. It is shown as the scheduling server in figure 2.1. Instead of querying the database it gets work from the shared memory segment loaded by the feeder. The scheduler matches results with clients, since not all clients are identical and certain user settings also differ. For instance one user could be running a Linux version of the core client and have specified that the core client can only use 10 MB of disk space and 20 MB of RAM, while another user could be running the windows version and allowed it to use 100 MB of disk space and 100 MB of RAM, these users are clearly able to process different results. During a scheduling session the core client also reports any finished results, which have already been uploaded, since the last scheduling session. After a scheduling session the core client is left with a list of results to process and a list of URLs from which to get the needed files i.e. input files and the application files, if not already on the host machine.

2.2.4.7 The Database

A MySQL database stores all information relevant to the BOINC server complex. This includes information about registered users and their associated hosts[11], about applications and application versions, about BOINC core clients and the versions hereof and of course about WUs and their associated results. Basically the entire state of the server complex is stored in this database and queried by among others the above mentioned daemons.

2.2.5 BOINC Applications

To get any work done by BOINC, projects must implement at least one BOINC application possibly ported from an existing application. Once the application has been implemented in the form of an executable, it must be registered with the BOINC server and the administrator can start creating WUs for this application. This means that WUs are basically nothing more than a specification of input files and command line switches to an already registered application. If the project wants to run their computations on different platforms, a version of the specific application for each desired platform must be implemented and registered. WUs are not bound to a specific platform nor are they bound to a specific version of the application, the latest version for the appropriate platform is simply used.
In order for an application to be a BOINC application it must call certain functions from the

[11]One BOINC user may sign up many machines all using the same user ID.

BOINC API, which is implemented in C. BOINC applications must call a BOINC initialization function at the beginning and a BOINC finishing function at the end. Besides these two functions BOINC applications are encouraged to call a function communicating the progress in percent to the core client, so that users can see the progress on their screens. Users also expect some sort of graphics to be present so BOINC applications should implement a specific function for drawing graphics, which will be called by the core client, if the user requests that the graphics be displayed.

If the application uses input files or non temporary output files, which of course most do, the filenames must be translated before opening the files. This is done by calling a function from the BOINC API and it is done because each application is run on many different input files and produces many different output files. If each of these files from different runs had the same names, each run would require a separate directory on the server and on the client to avoid names clashing. Since the application is the same between runs it is not possible to change the files names internally in the application between runs and it would be very impractical to recompile it for every run. All this means that the application has logical file names and these files names are translated by the core client to physical file names at runtime. The physical file names are specified when the WU is created.

Checkpointing can be a big advantage, but in order to support it the application must call a BOINC API function, that tells the application if it is okay to checkpoint now. This is because the user can instruct the client to only use the hard drives on the computer at certain intervals to avoid spinning up the drives frequently (on laptops for instance). If checkpointing is allowed by the core client the application must then do all the hard work of checkpointing by itself and then call another BOINC API function to tell the core client it has finished checkpointing. This has to do with the fact that the core client records the CPU time spent on a result to calculate the credits. If a result is restarted the CPU time begins counting from the time spent at the last checkpoint instead of zero.

2.2.6 Versions

BOINC is undergoing continuous change and improvement as new versions arrive. When we first started using BOINC in the spring of 2004 it was practically in a beta stage even though this was not stated anywhere. The documentation was just beginning to be written and new chapters and sections emerged on the website almost every day. BOINC uses a major and minor version number system, major versions represent big changes that break compatibility with the former major version meaning core clients only work with servers running the same major version as themselves. Minor versions represent changes that do not break compatibility and can therefore by done quite frequently. Since updating the core client entails the users manually downloading and installing a new version, major version changes were scheduled to be once every couple of years. When we started using BOINC the latest version was 2.19 (major version 2 minor version 19). At the time of writing, the latest version is 4.19[12], meaning 2 major version upgrades in less than a year. Major version changes often require changes to applications which is particularly painful when a stable version for the many different systems it runs on, has finally been achieved. An example of a minor version change, is the fact that a BOINC application now figures out if it is running in standalone mode at runtime. Standalone mode is a test mode allowing the application to be run without running it under the core

[12] 4.19 is the latest stable version, the latest developer version is 4.69.

client, earlier versions required a flag to be set in the source code prior to compilation. This change was implemented after our implementation of the BOINC-Grid bridge and eliminates the need for the specially modified core client[13]. If the change had been available at the time we implemented the bridge, we could have made a much simpler design.

2.2.7 Summary

Via a single BOINC client users can donate CPU-cycles to many different projects at the same time. Users security is ensured by signing executables and data, however sandboxing is not used so users must trust the projects they sign up for. The system uses a pull model of job distribution and HTTP to overcome firewalls. The client is available for many platforms and since the entire system is open source, users can build their own client for whatever platform they run[14].

A job in BOINC terms is called a WorkUnit (WU). The BOINC system runs the same WU multiple times on different machines, for reliability reasons. These copies of a WU, i.e. the jobs that are actually run on the clients, are to much confusion called 'results'. The results(out data) of these 'results' are compared using a project specific algorithm to find the end result i.e. the 'canonical result'.

The server side of the BOINC system consists of a database recording the system state including jobs, a web server serving data for the clients and suppling access to the scheduler, and five daemon programs: the transitioner, the validator, the assimilator, the file deleter, and the feeder. The transitioner takes care of state transitions of the WUs and the results, the validator validates results, the assimilator does postprocessing on the canonical result once found, the file deleter does garbage collection by cleaning the server of files from finished WUs and the feeder prefetches results.

Applications have to be specially made for running BOINC by using the BOINC API. This API enables file names to be translated allowing different input files with different names to be used by the same application without changes and allowing the output files from different runs to have different names. The API also has functions for drawing graphics and for informing the core client on progress with a given job.

2.3 Condor

Condor is a very sophisticated cycle-scavenging platform, which is currently being used at CERN, so this would seem to be the obvious choice for the public computing platform part of the bridge. But the obvious choice is not always the best as we shall see, even though Condor is already integrated with the Globus Toolkit[15] for Grids. Condor is the result of the Condor research project at the University of Wisconsin-Madison. The project was started in 1988 and it builds on the results of the Remote-Unix project from the same university.

Figure 2.2 shows an overview of the Condor system and it should be consulted while reading the following description of Condor.

[13]See section 5.3.1 for information on the modified core client.
[14]Although the projects might not have applications for this platform.
[15]See section 4.2 for information about the Globus Toolkit.

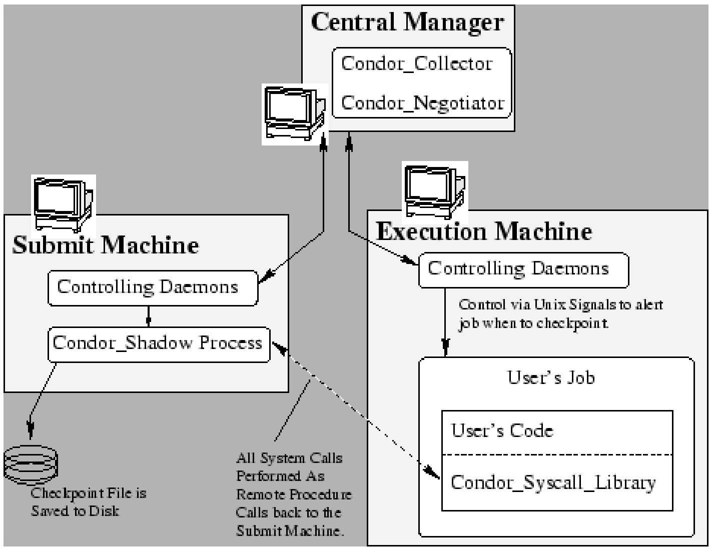


Figure 2.2: The Condor system using the standard universe. From [5].

A Condor system, called a pool, consists of a central resource manager, one or more submit machines, and one or more execute machines. The central resource manager handles the details of pairing jobs with resources and it involves knowing about both the resources and the jobs. This knowledge is obtained via ClassAds, which will be dealt with in section 2.3.2. The submit machines are machines from which is it possible to submit jobs to the Condor pool. These jobs are run on execute machines which are machines that are either dedicated to Condor jobs or normal desktop machines configured by their users to run Condor jobs when otherwise idle. The users/owners of the execute machines are called providers and the actual people submitting jobs to Condor are called users.

2.3.1 Features

Condor is very flexible and supports many features and many different computer platforms[16]. One very central concept in Condor is universes, these are runtime environments that provide different features for the Condor user. Condor supports the following universes:

- Standard
- Vanilla
- PVM
- MPI
- Globus
- Java

[16]A computer platform is defined as a combination of an operating system and a hardware architecture.

- Scheduler

The standard universe is a universe inherited from Condor's predecessor Remote Unix. It features remote system calls in which a shadow process is started on the submitting machine to process system calls, that are now transfered over the network from the execute machine. This mechanism is shown on figure 2.2 and it permits file accesses and user IDs to refer to the originating environment, and it also protects the execute host. There is potentially a serious performance impact with this mechanism, especially if a given job has many system calls. Such a remote system call is a lot slower than normal local system calls. If a submit machine has many standard universe jobs running on the Condor system, this machine has to run many shadow processes and has to service many jobs further impacting performance. The universe also supports application independent checkpointing and migration, so if an execute machine becomes unavailable a job can be checkpointed and migrated to a new execute machine to resume execution. Since Condor can be set up to only use the execution machine after a period of no user activity, an execution machine can become unavailable in the middle of executing a job. If the job can not be restarted within a reasonable time limit, it can be migrated to a new machine. The checkpointing can also be used to periodically checkpoint to the submitting machine, so that in case of the execute machine crashing, it can be restarted from the last checkpoint on a new machine. Jobs running under the standard universe requires relinking of the application with special libraries, which is not possible if the source code is unavailable. There are also a number of limitations to the applications that can be run under this universe such as only single process applications.
The Vanilla universe offers fewer limitations, but also fewer features. Jobs running under this universe cannot use checkpointing or remote system calls. The loss of remote system calls means that files used by the jobs must either be on a shared file system or files must be explicitly transferred by Condor. The loss of checkpointing means that jobs cannot be migrated and therefore might have to be restarted in case of a crash or a machine becoming unavailable due to prolonged local user activity.
The PVM and MPI universes allow for applications written for the Parallel Virtual Machine interface or the Message Passing Interface respectively to be run under Condor. To use the MPI universe the user must select the number of nodes needed and at the time of writing link with a specific implementation of MPI, called MPICH. Condor will then try and find a dedicated Beowulf cluster to run the job on and if this is not possible ordinary desktop workstations will be used instead. The PVM universe supports master-worker style jobs for PVM and uses both dedicated Beowulf clusters and normal desktop workstations. The normal workstations can of course enter and leave the virtual machine as they become available and unavailable respectively. The master application that is run on the submitting machine, never gets preempted and since it can apparently be quite tricky to write a PVM program to run on Condor resources, a special framework called MW[17] has been created to aid in the task. Because of the dynamic nature of the virtual machine, the user is required to select a minimum and a maximum number of nodes needed when submitting a PVM job.
The Globus universe allows users to submit jobs to Grids running the Globus Toolkit through Condor. This is of course very interesting and will be dealt with in the Condor-G section below.
The Java universe enables Java bytecode to be run under Condor.
The last universe, the Scheduler universe, executes jobs on the local machine. It is possible

[17]MW is a tool for making a master-worker style application. For more information visit `http://www.cs.wisc.edu/condor/mw`.

to run complicated sets of jobs which are dependent upon each other under Condor. This is done via the Directed Acyclic Graph Manager(DAGMan), which takes a directed acyclic graph(DAG) as input, where the nodes represent jobs and the edges represent dependencies. The DAGMan then works as a metascheduler for Condor and submits the jobs in the DAG in the right order (waiting for jobs to complete before submitting new jobs dependent on output from these previous jobs) to preserve dependencies.
Condor supports Linux, Solaris and Windows(NT/2000/XP), among others. It also supports different hardware architectures such as Sparc, MIPS, Alpha, Itanium and Intel X86. Not all of the mentioned platforms are fully supported by Condor, for example the Windows version does not support the standard universe.
Condor also includes heterogeneous support, which means that jobs can be submitted and allowed to run on different computer platforms, if executables for these platforms are available. The job is not bound to a specific platform before it is run, which means there will be more machines to choose from when trying to find an execute machine. This of course introduces a problem with migration, since jobs could be migrated to a different platform from the one it originally ran on. Platform independent checkpointing would require support from the application, but since one of the major advantages in Condor is the application independent checkpointing, Condor makes sure that jobs only migrate between machines with the same platform.
Finding hosts for jobs in Condor is done through a process called matchmaking, which is discussed in the following section.

2.3.2 Matchmaking

Matchmaking is a very general way of matching providers and users of a service. This matching is done via classified advertisements (ClassAds). Providers advertise their service and possibly constraints connected to this service, while users advertise their needs and wishes. These ClassAds are sent to a matchmaking service called the matchmaker, which matches providers and users based on their ClassAds. ClassAds are usually sent every 5 minutes to reflect any updates in for instance the load, but the update interval can be adjusted by the administrator. The classad has to be formatted according to a certain language, but the content can be chosen freely by users. In Condor this means, if a user has a job that needs for instance special hardware, maybe a specific OpenGL card, this user can specify a constraint in the classad, that the job can only be run on machines with this OpenGL card. Of course if no host has this card, or it has not been advertised in exactly the same way, the job will not be matched to a machine.
Figure 2.3 shows a classad describing a provider, which is pretty self explanatory except for the ranking. Providers can rank matching users (who in turn can rank matching providers) to provide for instance, a bias towards fellow research group members or friends (as in the example in figure 2.3).

```
[
Type          = "Machine";
Activity      = "Idle";
DayTime       = 36107   // current time
  // in seconds since midnight
KeyboardIdle  = 1432;   // seconds
Disk          = 323496; // kbytes
Memory        = 64;     // megabytes
State         = "Unclaimed";
LoadAvg       = 0.042969;
Mips          = 104;
Arch          = "INTEL";
OpSys         = "SOLARIS251";
KFlops        = 21893;
Name          = "leonardo.cs.wisc.edu";
ResearchGroup = {"raman", "miron",
                 "solomon", "jbasney" };
Friends       = {"tannenba", "wright" };
Untrusted     = {"rival", "riffraff" };
Rank          =
  member(other.Owner, ResearchGroup) * 10
  + member(other.Owner, Friends);
Constraint    =
  !member(other.Owner, Untrusted)
    && Rank >= 10
      ? true
    : Rank > 0
      ? LoadAvg<0.3 && KeyboardIdle>15*60
    : DayTime < 8*60*60
      || DayTime > 18*60*60;
]
```

Figure 2.3: An example ClassAd, describing a provider. From [6].

```
[
Type            = "Job";
QDate           = 886799469;
  // Submit time secs. past 1/1/1970
CompletionDate = 0;
Owner           = "raman";
Cmd             = "run_sim";
WantRemoteSyscalls = 1;
WantCheckpoint = 1;
Iwd             = "/usr/raman/sim2";
Args            = "-Q 17 3200 10";
Memory          = 31;
Rank            =
  KFlops/1E3 + other.Memory/32;
Constraint      =
  other.Type == "Machine"
    && Arch == "INTEL"
    && OpSys == "SOLARIS251"
    && Disk >= 10000
    && other.Memory >= self.Memory;
]
```

Figure 2.4: An example ClassAd, describing a job. From [6].

Figure 2.4 shows a classad describing a job, and it also shows the support for ranking. The attributes can be made up freely by the users and the providers, but they have to agree on the attribute names and types.
An important thing to note is that matchmaking matches the providers and users but it does not actually run jobs. It merely tells the provider and the user that it has found a matching counterpart. It is then up to the involved parties to handle matters further.
Compared to BOINC this is a much more flexible way of matching jobs with resources, but the flexibility comes with a loss of standards. In BOINC every client has the same attributes defined, which may not be the case in Condor. We believe that for the kind of jobs that BOINC is supposed to run, using static attributes is the best way, whereas for the kind of diversified jobs that Condor is supposed to run, matchmaking is best.

2.3.3 Scheduling

Condor takes care of scheduling by prioritizing users. Each user gets an initial priority of 0.5, which is the best priority a user can get. Each machine the user runs a job on increases his priority by 1, which also means that lower priorities are better than higher priorities. The decrease in priority can be configured by the Condor administrator, but the priority half-life is set to a day by default, meaning that if a user stops using machines, his priority will be halved every day.
Condor doesn't directly use the user priority to do scheduling. It uses the effective user priority, which is the normal user priority multiplied by some user-specific factor, allowing for preferential treatment of individual users.
The effective user priority is used to calculate the share of Condor machines each user should be allocated, which is inversely related to the ratio between user priorities. This means that a user with half the priority of another user will get twice as many resources as the other user. Condor will try to make sure that users get the shares they are entitled to, by allocating newly available machines to the user with the biggest difference between his actual share and the entitled share. The scheduler might also preempt jobs if a high priority user (meaning a user with a low effective priority) submits jobs when Condor is only running jobs from low priority users. This preempting is configurable by the administrator and care should be taken to avoid thrashing of jobs from low priority users. The default is to not preempt jobs that have been running for less than one hour. Even with Vanilla universe jobs, this scheme should not lead to starvation of jobs from low priority users, since only enough jobs will be preempted to insure high priority users get their share, meaning that the low priority user would still get a small share. Even in the case of complete preemption of a low priority user's jobs, his priority will be halved by default every day meaning that the priority will eventually be low enough to allow the jobs to run.
Besides this prioritizing among users, each user can prioritize his own jobs.

2.3.4 Condor-G

Condor-G is an extension of Condor, which makes it possible to use the normal Condor interface for submitting jobs to Grids running the Globus Toolkit. The advantages of Condor-G, compared to the "globusrun" command from the Globus Toolkit, is the familiar Condor interface, the ability to submit many jobs at once, monitoring of jobs, job completion notification and a certain amount of fault tolerance on the side of the submitting machine. To submit a job to a Grid the user needs to specify the universe as being Globus as well as the URL of a specific Globus scheduler. The scheduler can be omitted, which will cause Condor to search for a suitable Grid using ClassAds. The use of ClassAds requires the Grid sites to advertise their attributes in a classad to the resource manager. The use of Grid resources requires credentials, which of course are also needed with Condor-G. Therefore Condor-G comes with a tool to create proxy-credentials to be used by the jobs submitted to Condor-G.

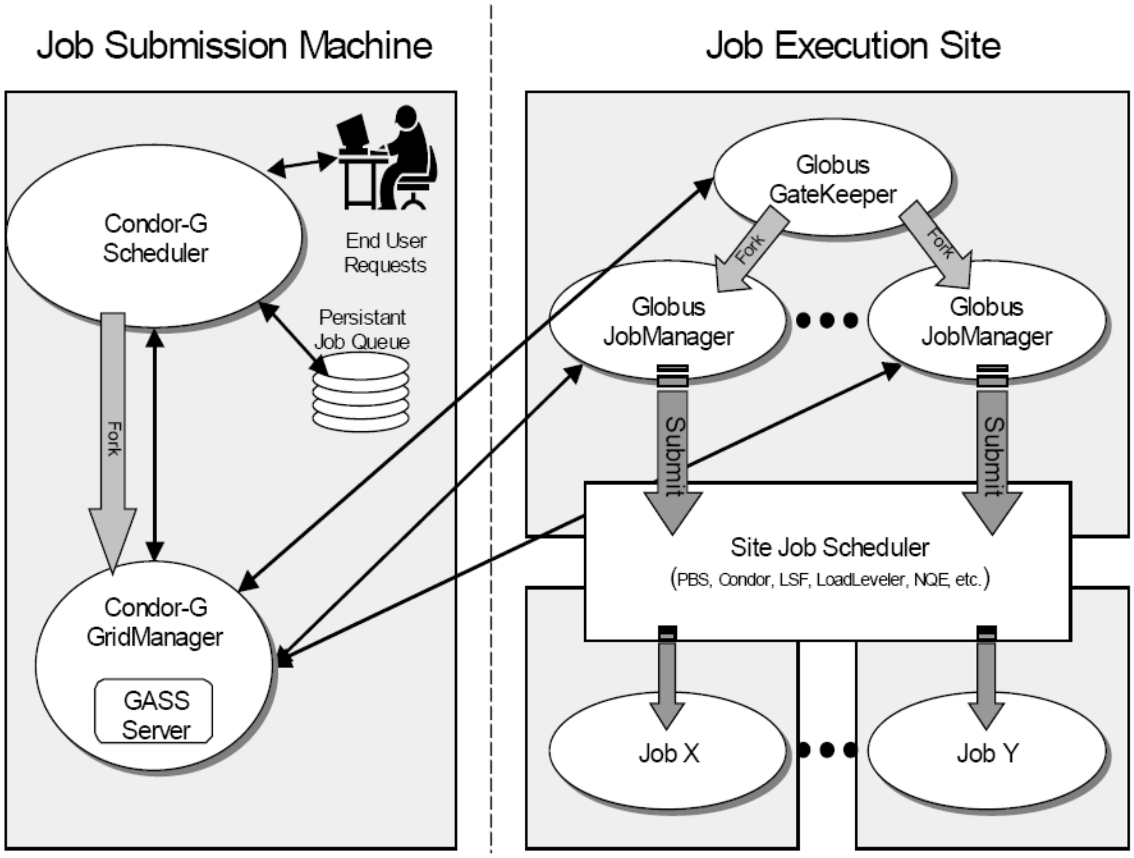


Figure 2.5: Job-execution on Globus resources via Condor-G. From [7].

Condor-G uses standard Globus protocols to execute jobs on Grids, specifically it uses Global Access to Secondary Storage(GASS) for file transfers, Globus Resource Allocation and Management(GRAM) for submitting and monitoring jobs and Globus Security Infrastructure for authentication. Figure 2.5 shows Condor-G executing jobs on a Globus-managed Grid.
Submitting jobs to the Globus universe has certain limitations. It does not support checkpointing (and migration), remote system calls and certain other features. Because of these limitations, Condor-G comes with a mechanism called Glidein, which temporarily turns a Grid machine into a normal Condor execute machine. This is done by submitting a job to the Grid machine, which in turn runs the Condor daemon like normal Condor execute machines. When the Condor daemon running on the Grid machine has been idle for a certain period of time it gracefully shuts down, turning the machine into a normal Grid resource again.

To prevent security holes, only the user that executed the Glidein (and therefore has Grid credentials) is allowed to run Condor jobs on the Grid machine. Because the jobs have to be submitted like an ordinary Condor job, he must specify in the job classad that the jobs need to be run on this particular machine. Otherwise the jobs might get dispatched to other Condor machines, leaving the Grid machine idle since only this user can run Condor jobs on it. In addition Condor-G supports something called flocking. Flocking is when jobs are run on a machine in a different Condor pool than where it was submitted. When a job cannot find resources at its home pool, it is possible to allow it to flock to another pool with available resources to run there instead. It is possible to allow jobs from different pools to flock both ways or only one way. Condor-G is used at CERN to submit jobs to LCG-2[18] mainly because it provides job monitoring via a program called the *grid monitor* and prevents flooding of jobs to the Grid. The *grid monitor* offers better performance since it is able to monitor multiple jobs as opposed to the standard Globus[19] equivalent which is only capable of monitoring a single job.

2.3.5 Condor vs. BOINC

There is a big difference between Condor and BOINC at a conceptual level; Condor is meant to run very heterogeneous jobs inside the boundaries of an organization and BOINC is meant to run very homogeneous jobs outside the boundaries of an organization. One implication of this is that Condor trusts the resources it uses, whereas BOINC does not. Because of the lack of trust, BOINC does redundant computing to try and avoid cheating and it signs executables to improve user security since it is operating on the Internet, whereas Condor does nothing of this kind although it can be configured to authenticate entities and encrypt all communication.
An important technical difference lies in the way communications are done. By default Condor uses port 9618 and 9614 to communicate. These are not "well known" ports, as specified in RFC793, and is therefore rarely open in firewalls. Condor uses the push method of distributing jobs i.e. the central resource manager connects to the execution machines. This scheme can not be used by BOINC since many of its users sit behind firewalls, and asking them to change their firewall configurations is not likely to succeed. Therefore BOINC uses only port 80[20] for communication and users pull jobs from the server complex so connections are established from the client to the server instead of the other way around.
Because of its matchmaking and multiple universes, many different job types can be run on Condor making it much more versatile than BOINC. BOINC requires specially designed server side functions tailored to the specific application, as well as linking the application with the BOINC API. This places a large obstacle in the way when it comes to running jobs under BOINC. Because porting applications to BOINC requires quite a bit of effort, this effort only pays off when the application has to be run many times, whereas in Condor basically any application with binary compatibility can be run under the Vanilla universe with very little effort. If we consider the standard universe, the migration and checkpointing allows for very long running jobs, as opposed to BOINC, which does not have direct built-in support for

[18] For more information on LCG-2 and the use of Condor-G the reader is referred to section 4.3.1.
[19] See section 4.2 for a description of Globus and its job monitor the job manager.
[20] Port 80 is open in practically any firewall, so if you want to get round a firewall this is the port of choice. The port can not be closed since it is used by the HTTP protocol, but as more and more protocols use it, it is beginning to render firewalls useless.

these features. Checkpointing can be done in BOINC, but it requires considerable work by the application specific backend.
BOINC tries harder to attract volunteers and entertain them by having graphics and progress indication of jobs as well as easy access to information about running and buffered jobs. All of this should make it easier to attract users to a BOINC based project as opposed to a Condor based project.
We have chosen to use BOINC because it supports PRC better than Condor in terms of getting reliable results, protecting users and overcoming firewalls. We feel BOINC is a true PRC platform whereas Condor is better suited for intra-organizational use. As a bonus BOINC offers access to the many users of SETI@home and without users a PRC project is useless.

2.4 Other PRC Platforms

Although we have mainly focused on BOINC and Condor in this chapter, there are many other public computing platforms available with various properties. In this section we will briefly present some of these.

2.4.1 Frontier

The company Parabon has made a PRC platform called Frontier [10]. Frontier uses sandboxing [21] to protect the clients(providers in Frontier terms.) from malicious code, but, probably because of this, they only support Java applications. The sandbox should make it impossible for the application to access the providers' files and programs, and it should also be impossible to connect to any host other than the Frontier server.
Parabon's platform is not open; instead the company is trying to make money from PRC. This means that there is only one way to reach the public with a Parabon application, and this is via Parabon's servers. These servers let you run jobs on Parabon clients connected to the Internet. Parabon supports two kinds of projects - non-profit and profit. If a project gets accepted as a non-profit project, it gets free access to their servers and of course the providers. If the project is for profit it will have to pay Parabon for the amount of CPU-time it receives from the providers, who in turn get paid by Parabon for the use of their computers.
Parabon also has a version where the project runs the servers itself, which they call the enterprise version. This version unfortunately only lets projects submit jobs to the clients in its enterprise, which makes it nothing more than an enterprise-wide job distribution system.
Since Frontier is mainly targeted at profit seeking companies, who most likely want to protect their intellectual property, Frontier uses SSL[22] for transferring data to and from the client, which makes it possible to verify that the data has not been changed en route. Furthermore the application byte-code is obfuscated on the provider's computer to prevent reverse engineering, however the amount of security provided by this obfuscation is probably very limited.
Parabon uses a different scheme from BOINC to check the validity of the returned results; they periodically send out jobs with known results to the providers to check up on them. If this test is failed the provider will not be used again.
Because Parabon is a closed platform and access to public providers is only possible via

[21] Sandboxing provides an isolated environment for applications to run in protecting the rest of the machine against malfunctioning or malicious applications in the sandbox.
[22] Secure Socket Layer.

Parabon's servers (which involves payment for used resources), we did not feel that Parabon was suited for our purpose. Besides this, the limitation that only applications written in Java can be run also made it ill-suited for our purpose.

2.4.2 DCGrid

Entropia's platform is called DCGrid. It is very similar to Parabon, but not identical. DCGrid is also a closed platform and it is very much aimed at enterprises. There is very little focus on letting the public do the computing, but instead DCGrid is aimed at being installed on the computers of a given enterprise, so that they can be used as cycle-servers by the enterprise. We do not see any problems with taking the platform to the public since the security seems to be in order. All the data on the client is encrypted and checksummed so that any changes to it can be detected. We do not have any information on how data is transferred to the client, but it should be fairly trivial to secure this too. As far as protecting the client, DCGrid uses sandboxing like Frontier, but with a different approach. Frontier only supports Java, whereas DCGrid wraps a valid win32-executable in their sandbox, effectively intercepting system calls and only allowing harmless ones to execute correctly. This means that DCGrid is totally programming language independent. On the other hand it is very operating system dependent, since they only support Windows. This limitation is not necessarily a big problem when building a PRC project, since almost all public computers run Windows [23]. We, on the other hand, have to use it to run applications intended to run on the Grid (which is running Linux), so the lack of support for this is very unfortunate. Entropia claims that DCGrid supports OGSA and Globus Grid standards, whatever this means, but it might have proven useful to us. Mainly because of the lack of support for Linux and the non-open platform, we chose not to use Entropia.

2.4.3 Grid MP

Another non-open platform is United Devices' Grid MP, which, like Frontier from Parabon, comes in different flavours. There is an enterprise version, which could also be compared to DCGrid; a pay-per-use version, where dedicated cycle-servers are used; and a global version. The global version is the one we would be interested in. It seems to work a lot like BOINC, which is not a coincidence since David Anderson from BOINC was associated with United Devices before his BOINC days: the user downloads a core client and then signs up for projects he finds interesting. On the server side things look different from BOINC. United Devices runs the servers and chooses which projects to offer to the public. There is no payment of clients like in Frontier and there seem to be only non-profit applications running.

What Grid MP applications look like is a bit of a mystery, because of the non openness of the platform. They do claim, however, that most applications can be run "as is" whatever this entails, but they also hint that specially designed applications offer better performance. Security is very much a high priority in Grid MP. Clients are authenticated when communicating with the server using certificates, so that misbehaving clients can be rooted out. Data transfers are encrypted as well as data stored on clients, and executables require signature validation. Furthermore, sandboxing is used, but how this is accomplished is uncertain.

[23] 91.6% of the machines donating CPU-time to the LCH@home project were Windows machines. See section B.2 for more information on the machines donating compute power to LCH@home.

Grid MP also supports MPI[24] to facilitate fine-grained parallel jobs, but this does not seem usable for the global version since fine-grained parallel jobs usually require tightly coupled machines with fast, robust, and always available connections between them.
United Devices market their platform as a Grid computing platform, however according to the definition of a Grid, which we define in section 4.1, Grid MP is not a Grid.
BOINC offers the same features as Grid MP but Grid MP is not open source. Therefore BOINC is a much better choice, when we possibly have to modify our chosen PRC platform to build our bridges.

2.4.4 OfficeGRID

The contribution from MESH-Technologies is called OfficeGRID and it is unfortunately also a closed platform. It is targeted at enterprises and the harvesting of spare cycles within those. OfficeGRID applications must use a special OfficeGRID API to take full advantage of the system in terms of parallelization. Fortunately this API is available for both C/C++ and Fortran. Presumably standard applications can be run as well, but without the parallelization of specially designed OfficeGRID applications. The platform supports Windows and Linux as the operating system on the execution machines.
Specially designed applications for OfficeGRID use a distributed shared memory model called TMem, which should make it easier to parallelize applications than with message passing systems like MPI. TMem uses tuple spaces, supports dynamic networks, meaning nodes that enter and leave the network during execution, and is resilient to network errors.
Because it is a closed platform without much support for public computing in terms of trusting the results and protecting the clients, we have chosen not to use OfficeGRID although the distributed shared memory model looks very promising.

2.5 Summary

In this chapter we discussed the Public Resource Computing(PRC) model, as well as a range of specific implementations of it. We studied Condor and BOINC in detail. The greatest advantage of Condor was that a program can be run under it without modification. BOINC on the other hand is more demanding, but also has many advantages when doing really public projects, such as a pull model for job distribution so that clients behind firewalls can contribute. As well as many security features lacking in Condor. The analysis of the different systems showed us that the best choice for the LHC@home project would be BOINC. This project is described in the next chapter.

[24] Message Passing Interface.

Chapter 3

LHC@home

This chapter describes the work we did setting up a PRC project, namely LHC@home, at CERN. After we give an introduction to CERN, we describe the application we chose to be the target for our PRC project, before delving into the particulars of setting up BOINC at CERN. Finally we provide some advice for other people thinking about starting a PRC project based on our experiences. It will of course focus on BOINC, but it will also have general advice applicable to any PRC system.

3.1 An Introduction to CERN

CERN, an acronym for Centre Européenne pour la Recherche Nucléaire, is as the name implies a European research institution specializing in nuclear and particle physics. It was founded in 1954 and is situated on the border between France and Switzerland just outside Geneva. The main focus for the moment is the construction of the LHC, the Large Hadron Collider. The LHC is a machine where particles are accelerated to near light speed and then they are brought to collide inside detectors. The many different signals from the detectors are then used to reconstruct the event. These events are then studied for clues to the nature of the laws of physics. The rate of data production for these detectors is estimated to be around 10 petabyte a year.

3.1.1 Computing Needs at CERN

Computers are used for numerous heavy computational tasks at CERN, not only for analyzing the data from the LHC. The accuracy needed in building the LHC also requires simulations of many different physical systems affecting the calibration of the LHC, a funny, but true story involves a mysterious error in the energy calibration of LEP[1] that was later found to be caused by the TGV[2] departing from the Geneva train station. The collisions also have to be simulated because only a small fraction of them will be of interest to the physicists and it is simply not feasible to do a full analysis of every collision. It is predicted that there will be around 800 million collisions per second and each will produce on the order of 50000 particles. This means that every second about $40 * 10^{12}$ particles are produced that need to be analyzed, which is about 10000 times faster than the rate a modern processor can add two

[1]Large Electron and Positron collider. The predecessor of LHC

[2]Train Grand Vitesse. French high speed trains.

floating point numbers. The analysis of each particle is of course much more involved than just adding two floating point numbers. Physicists therefore simulate interesting interactions to establish a kind of fingerprint for it. These are then used to filter the collisions.
One interesting common feature of all these different simulations is that they are independent for each collision and therefore easily parallelized. Another feature is that, because most of the studies are statistical of nature, a single result is not of interest only the aggregate of all the results are. The most interesting benchmark for the scientist is throughput and not response time. Only in the case where a single computer does not have the resources to process a simulation, or where parallelization leads to super linear speed ups are regular clusters or supercomputers an advantage. For these reasons Distributed Computing is ideal for most of the computations done at CERN. This is also reflected by the IT centre at CERN, where they have migrated away from the use of supercomputers (IBM, CRAY) to a large batch farm of about 2000 PCs, expected to have increased to about 5000 by the time LHC is ready in 2007.

3.2 SixTrack

The LHC simulation that was chosen to be the basis for LHC@home was SixTrack. SixTrack has been developed over the last 20 years by Frank Schmidt, and is used by CERN and other synchrotrons[3] around the world to do beam studies. Like much of CERN's older software it is written in Fortran. On the initiative of Eric McIntosh it is also used as part of the SPEC[4] floating point benchmark. So CERN, when buying new computers, never have to estimate how well a new processor will run SixTrack, but can look it up directly. Another added benefit is that manufacturers will optimize their hardware and compilers to get the best benchmark score, so modern CPUs and compilers are directly optimized for SixTrack performance.
SixTrack simulates a number of particles as they are accelerated around the LHC ring. It calculates in small time steps the movement of the particles and the effect of the bending magnets, focusing magnets and so on. The purpose is to determine whether the particles stay in the ring or are lost. As the different parts of the accelerator ring begin to arrive at CERN, they are studied and measured and the data added to the simulation. This way it is possible to correct the placement of the different parts and to discover parts that are unusable before the LHC is put into operation. It is imperative that the stability of the ring is ascertained before the LHC goes into operation. If the beam dumps[5], equipment worth many million CHF may be destroyed.
SixTrack is ideal for loosely coupled distributed computing, because many different starting points and velocities of the particles have to be tried. Each new set of starting conditions then gives rise to a completely independent calculation. The amount of data transfered per result is different for each calculation, but is on the order of 300 KB. A typical calculation takes 30 minutes on an average modern computer. This gives about 2 minutes of transfer time for people using dial-up connections. That is a compute to transmit ratio of 15:1 for the slowest connections. This seems very reasonable, but there is one catch; Sometimes the particles will be lost very soon after starting the simulation. The simulation will then stop, maybe after only one minute, because it serves no purpose to continue the calculations. Because the particles in SixTrack are on the border of chaotic motion, there is no way to predict which starting

[3] A special type of particle accelerator.
[4] Standard Performance Evaluation Corporation.
[5] The particles do not stay inside the ring.

conditions will lead to such a result. However, even though a few calculations with a low compute to transfer time ratio might be very annoying for users on dial-up, it would not pose a problem internally at CERN, nor would it for volunteers who have ADSL or similar connections. We also foresee that this problem will disappear in the future, as more and more people migrate to some form of broadband connection.
SixTrack was chosen as a pilot project for PRC, because of its high importance compared to the funding it receives. The funding was probably the reason why the people behind SixTrack were enthusiastic about running it on public resources, when other better funded groups were more sceptical. The enthusiasm of the SixTrack people along with the support offered by them was also a deciding factor when we chose SixTrack.

3.3 CPSS

Andreas Wagner and Eric McIntosh at CERN have already been working on making a screensaver version of SixTrack that could run on CERN desktop PCs, in hope of gaining more compute power for SixTrack. They have pursued this project in their spare time, and have come up with a system they call the CERN Physics ScreenSaver (CPSS). They had never thought of going beyond the CERN desktops though, and CPSS was consequently not designed with any kind of security in mind. The architecture is somewhat reminiscent of BOINC. The server consists of a database storing data about the clients, and a repository of available and completed jobs on the server. This server can be split up across multiple machines for scalability, though this has not been tested in practice. The server does not automatically support sending multiples of the same job for later comparison or validating the results. This has to be done separately if it is desired.
The client contacts the server through the HTTP protocol just like the BOINC client. It is though not nearly as sophisticated. It does not sign binaries, so it is vulnerable to a man in the middle attack. It does not allow the user to specify usage limits or to run applications in the background. It runs as a screensaver and in case of user interaction, it immediately kills the running application and any progress is lost.
The work on CPSS meant that the SixTrack Fortran code was already ported to run under Windows, and had been rigorously tested to insure that the results did not differ from the Unix version.

3.4 CPSS vs BOINC

The CPSS system has some advantages over BOINC. The most important is that the application can be run without modification, thus making it easier to port legacy applications to the platform. Another advantage is that the client software is automatically updated. That this does not happen in BOINC can cause some problems. Under BOINC, the interaction between the application and the client software might cause the application to fail. This is then fixed in the next version of the client, but you cannot be certain that all your users have updated their BOINC clients. BOINC tries to solve this by tagging WUs with a minimum client version, so WUs will only be distributed to a client fulfilling this requirement. This only partially solves the problem because then you are relying on the user to actually update their client to the new version which they might or might not do, thus lowering the performance until everyone has upgraded.

BOINC lacks tools for deleting WUs. So if a mistake is made in generating the studies, the physicist has to contact the administrator who has to remove the WUs manually from the database.
A final advantage is that the system was already undergoing testing when we started our project, so the servers were deployed and the client program distributed at CERN. Using CPSS for LHC@home might therefore have seemed like the logical choice.
CPSS does, however, have some serious drawbacks when it comes to taking the project to the arena of public computing. There is, as we saw in section 3.3, no security. CPSS was primarily designed with SixTrack in mind and not as a general purpose tool. The screensaver part for example is tightly bound to the output files of SixTrack. It uses data from them to display progress and other data. This could though be generalized. If the machine is used while CPSS is running, it will kill the application and let the user take over, whereas BOINC merely suspends the application. Because it was not designed for the public, CPSS contains no credit system or forums, which according to David Anderson, is a crucial part to keep the users' interest. This would not be a problem at CERN, because the screensaver could be installed on all machines automatically by the IT department, but it could prove to be one when taking the project to the public. On this basis it was therefore decided that CPSS would continue to run and be distributed internally, but BOINC would be used for the public LHC@home project. This arrangement would then allow us to compare the performance of the two platforms.

3.5 Deploying BOINC at CERN

Deploying BOINC at CERN consisted of three main tasks. First of all, to set up a server complex to handle the BOINC project. Second, to port SixTrack to the BOINC platform and third, to create an interesting screensaver for SixTrack that would please the users.

3.5.1 The Test Project

Our very first concern was to gain familiarity with the BOINC platform. To this end we set up a test project on an ordinary desktop machine. This server we would use to develop LHC@home on, until we had the project ready for public release at which point we would move the project to real server machines dedicated to this project. After we had moved the project we would continue to use the test project for testing new versions of SixTrack before releasing them to the public as well as develop the bridges described in chapters 5 & 6. For the test project we developed a small wrapper program that would take care of doing the necessary BOINC calls as well as packing and unpacking data and running the main SixTrack program. This would eliminate the need for adding the BOINC calls to SixTrack. The test project showed that this wrapper approach, while very useful and easily adapted to other BOINC projects, had some shortfalls. We will cover this in more detail in section 3.5.5.

3.5.2 Server Requirements

Once we had gained experience with our test server, our first job was to work out the server requirement. We made this estimation together with Karl Chen, who had extensive experience in this area from his work with the Seti@home project. We based estimate on the average case. Peak needs were much higher and would demand a more rigorous statistical analysis

than what we have room for here, and the worst-case scenario is that all the clients will try to access the server at the same time.
The first number to estimate is the number of BOINC clients that are going to participate. This number is of course very difficult to estimate because it is based on how popular LHC@home becomes. In the first two weeks after SETI@home switched to BOINC, the number of clients returning results rose to 40,000. Based on the experience of SETI@home we have concluded that the number of effective full-time clients will be on the order of 10,000. These 10,000 full-time clients are probably going to be made up of several part time clients, for instance 40,000 clients donating time to on LHC@home 6 hours a day. 1000 clients is probably too low, since we could get this number just by installing BOINC at CERN, a thing that is likely to happen, either because people at CERN are very interested in this or as an order from the management so that the computers that are idle can be used for something sensible. 100,000 seems too optimistic since SETI@home has about 50,000 clients and these are not even full-time[6].
Next we need to estimate the amount of data each client is going to retrieve from and send to the server. The first time a client tries to download a WU, it will also download the SixTrack program from the server. This will only happen once, until a new version of the SixTrack program is put into production. SixTrack will, once it is given the "go ahead" for this project, hopefully be very stable and only be upgraded very infrequently during the lifetime of LHC@home. The bandwidth needed for transferring the application can therefore be ignored. The WU holds a description of the entire ring and the starting 4-vector of all the particles, because the ring description can and sometimes is changed from WU to WU, the entire description must be sent every time. This data is on the order of 2,5 MB uncompressed and about 300 KB compressed. The latest versions of BOINC, which we did not have when we started the project, allows for something they have termed sticky files. This means that we can mark an input file as sticky. It will then not be deleted along with the other input files when the WU finishes. New jobs needing this file will then preferably be sent to clients already in possession of this file. This should reduce the needed upstream bandwidth. No projects are using this feature in production yet, and it is therefore very hard to predict how well this feature is going to work.
The result data is on the order of 12 MB uncompressed and 4 MB compressed, however this can vary greatly depending on the options specified by the physicists[7]. Up to this point they had done studies with at most 100,000 turns. Because this is only about 2 seconds of real time they were very interested in increasing that number, but had not been able to because of lack of compute power.[8] The result contains the final position of the particles together with a lot of numerical data from the entire run, in some cases the latter is needed for analysis, but not often. We have convinced the maintainers of SixTrack and the physicists that in case the bandwidth for transferring results or disk space becomes a bottleneck, they would only need the file that contains the summarized result data. If the summarized data calls for it, they can always rerun a WU locally to get the entire data. In the case of the reduced output only

[6]We are referring to the BOINC version of SETI@home, the classical version has on the order of 500,000 clients.

[7]The physicists can specify how many turns should be simulated and how often the particle position should be written to disk.

[8]CPSS had increased the amount of computer power they had at their disposal, but because it simply kills the application such a long running job would only have a small chance of finishing. A 24 hour job would only have a chance of finishing during the weekend wasting the compute power during workdays.

about 50 KB of data would need to be uploaded per WU.
At the time we started this analysis, an average WU would take about one hour to compute on a 2 GHz P4 machine. This number is also highly influenced by the options the physicist specify. During our work with the application we had to change Fortran compilers many times, sometimes just versions and sometimes also the compiler manufacturer, and each time the execution time for a WU became smaller. The version we finally used for the public had almost halved the time an average WU takes.

3.5.2.1 Estimated Network Requirements

We need to estimate the bandwidth partly to determine the server requirements and partly because of the network setup at CERN. If our bandwidth requirements are large, we have to advise the network administrators at CERN and we have to request that packets destined for our server get high priority in routers and firewalls. The estimated network requirements are calculated as the size of a WU multiplied by the number of WorkUnits an average client can process per time unit and the number of clients. For example the estimated compressed upstream bandwidth is:
$300KB/WU * 2WU/(user * hour) * 10000users = 1.7MB/s$

bandwidth in MB/s	uncompressed	compressed
upstream	13.9	1.7
downstream	66.7	22.2
downstream(reduced output)	0.3	0.2

Table 3.1: Estimated bandwidth requirements for SixTrack under BOINC based on 10,000 users.

From these estimates we see that it is going to be necessary to use the reduced output, because 22 MB/s are at the limit of what harddrives and Internet connections (it is a sustained transfer rate larger than 100 MBit/s) can deliver. Because of the massive reduction in bandwidth requirements when compressing files versus the wasted CPU-time, on the client and at submission, compression was chosen.

3.5.2.2 Estimated Disk Requirements

The bandwidth requirements for the disks are of course the same as for the network plus some local traffic, for example if the tables in the database become to big to keep in memory. The estimation on the disk usage is based on a result redundancy of 3 per WU, and a verification redundancy of 2 agreeing results.[9] A result redundancy of three means that three results will be able to share one set of input files. The result files will need to be kept on the server until they have been validated and then they can be transfered to their final destination. Likewise the input files have to be kept on the server until the WU has finished. The standard preferences for the BOINC client, will make it request enough results to keep it busy for 3 days. This is about 130 WUs on average.
Thus the input files will need:

[9]The actual values are set by the physicists when submitting a job, however these are the expected values.

$300KB/WU * 0.33WUs/result * 130results/User * 10000users = 386GB$
The output files will need:
$4MB/result*1(unassimilated)result/WU*0.33WUs/result*130results/user*10000users = 1.7TB$
The reduced output files will need:
$50KB/result*1(unassimilated)result/WU*0.33WUs/result*130results/user*10000users = 21GB$
These numbers are based on an immediate removal of the files when a WU has been assimilated and that we submit exactly enough jobs to sustain the queues. If we submit jobs too fast, the disk will run out of space and if we do it too slow, we will not gain the full performance of the system. Because jobs will normally be submitted in batches, the disk space needs will be higher than the steady state above.

3.5.2.3 Proposed Configuration

From the bandwidth estimates above it was clear that we would need to request high priority Internet access for our servers. It was also clear that we would need some type of RAID[10] system for our disks, because no single disk could deliver the sustained bandwidth requirements. We made the following proposal for a LHC@home server complex:

Web Server(WS):

will run	will store	hardware characteristics
Web server (Apache)	input files	disk: 0.5 to 2.0 TB hardware RAID
BOINC scheduler	output files	RAM: 1.0 to 2.0 GB
BOINC validator		CPUs: 1-2, 1-4 GHz
BOINC Assimilator		network: 1 GBit/s + 100MBit/s dedicated to DBS

Database Server(DBS):

will run	will store	hardware characteristics
database server (MySQL)	database files	disk: 50+ GB, RAID
BOINC work generator		RAM: 2.0+ GB
BOINC Transitioner		CPUs: 1-4, fast (lots of cache) e.g. Xeons
BOINC file deleter		Network: 100MBit/s dedicated link to DBS

3.5.3 Server Setup

With the proposal above we approached Jan van Eldik of CERN's IT-FIO[11] department. He found a discarded disk server that we could have. It had the following hardware characteristics:

Disk: 1.2 TB hardware RAID 5, 700 GB formatted capacity
RAM: 1.0 GB
CPU: 2 P3 Xeon 1.2 GHz
Network: 1000 MBit/s

[10] Redundant Array of Independent Disks
[11] Fabrics infrastructure and operations.

While this machine was adequate for running the Web server, it was unfortunately smaller than what we had hoped for for the database server, that now would have to be run on the same machine. It was therefore agreed that if we got into performance problems, we could get one more machine of similar setup within a reasonable time frame.
Now the BOINC server system is made in such a way that it is easy to distribute the server functions across many servers should the need arise. It is though very important that it is realized from the start that you may later have to add more servers to the project. Because we anticipated having to add a new machine, we made sure that all accesses went through DNS[12] aliases, so we would just have to move a service to another machine and then change the definition of this alias to point to the new machine.

3.5.3.1 Job Creation

Jobs for LHC@home are created by Eric McIntosh based on studies requested by the physicists involved with building the LHC. Mr. McIntosh has a set of bash shell scripts to extract data from special accelerator databases and generate the input files for the SixTrack application. The generation actually involves running a small simulation on the data for every new seed and it is therefore quite CPU-intensive. The shell scripts were already prepared to be used with different execution systems like LSF and the CERN developed cycle-scavenger CPSS. So to ease integration with BOINC, we wrote a set of bash shell scripts to be used by Eric McIntosh's scripts instead of the ones for LSF or CPSS, when he wanted to generate jobs for LHC@home. The first version of these scripts used Secure CoPy(SCP) to copy the input files to the BOINC server and then used Secure SHell(SSH) to log in to the BOINC server and run the BOINC supplied program to create WU. To avoid having to type username and password for every connection to the BOINC server, a public key system supported by SSH and SCP was used to authorize Eric McIntosh. This first version was found to be too slow due to the overhead of establishing secure connections and a new version was written.
The new version functions in two stages, a dedicated AFS[13] directory, which the submitting machine and the BOINC server both have access to, is used as a buffer. The script on the submitting machine copies the input files to the AFS directory and creates a description of the job. The jobs are picked up by a script on the BOINC server run at regular intervals by cron[14]. The script copies the input files to the server's local storage and runs the BOINC supplied WU creator program. There is a small issue concerning simultaneous access to the AFS directory since the two scripts run asynchronously. Left unattended it would be possible for the server side script to pickup a job not fully transfered to the AFS directory. The submit side script deals with this issue, by transferring the input files first and then creating the description file using a unique temporary filename that does not show up when the server side script lists the directory looking for description files. Since the server side script does not find the description file of a job being created by the submit side script, it does not look for the input files and does not create the WU. Once the description file has been created, the temporary file name is changed and the job is ready to be picked up. This system ensures robustness without the need for locking. It is even possible to use more than one machine to submit jobs, since every submit side script uses a unique name for the temporary file name. We have used more than one machine to submit jobs without conflicts. The new version has

[12]Domain Name Service.
[13]The Andrew File System. A network file system.
[14]Daemon to execute scheduled commands.

not presented any performance problems.

3.5.3.2 Validation

The first validator we used required complete agreement between the result files. It compared the different values in the result files and made sure that they were pairwise equal. We found that this validator discarded a relatively large amount of results. The reason was very small differences, 1 bit, in the result from different platforms. Results calculated under Win2000 would differ from results from other OSes (e.g. XP and Linux) and results computed on an AMD chip would differ from those computed on an Intel chip. This peculiarity had never been observed by CPSS, but the explanation is that most computers at CERN are installed by the IT division that uses the same operating system and chip manufacturer for reasons of economy. Because there is no way to know which platform is doing the computation correctly, except for doing the calculations by hand, the physicist decided that such a strict accordance was not necessary until the reason for the discrepancies could be found. A new validator was therefore written by Markku Degerholm that allows for a relative difference of a millionth i.e. if $ABS(a-b) > 1e-6*a$ or $ABS(a-b) > 1e-6*b$ the results are discarded. The old validator is planned to go into use again once the problem has been resolved.

3.5.3.3 Assimilation

The assimilator on LHC@home is very simple. Once a WU is finished it merely copies the output files of the canonical result to a given directory. This directory is currently an AFS directory where Eric McIntosh can access the results, do postprocessing on them or move them to permanent storage for later analysis.

3.5.3.4 Database

When we created the database we used the MySQL standard MyISAM type of tables. It quickly became apparent that this was a big performance bottleneck. We therefore migrated the database to use the InnoDB tables. Although these tables are bigger than MyISAM and therefore will require more disk space and may be slower because more data has to be transfered for each query they have one big advantage. InnoDB allows row level locking instead of table level. We experienced that the server might run nicely under heavy load and suddenly the performance would degrade heavily for a short period. We found that this problem was caused by queries waiting on locks. After we switched to InnoDB this problem became much less pronounced.

We also found that the database performance degraded significantly during the project. The culprit was the tables containing results and WUs. They had become so large that they could no longer fit in memory and therefore gave rise to a lot of disk traffic. Because none of the BOINC daemons need access to assimilated WUs or results of assimilated WUs, we created two new tables where we once a day moved the assimilated WUs and the results belonging to it. The feature is now a standard part of BOINC.

3.5.4 Creating a Secure Code Signing Method for LHC@home.

As we saw in section 2.2, trust is very important for users of BOINC. One of the ways BOINC insures the safety of the user, is to use code signing when distributing the executable to the

clients. The clients can then be sure that the code does in fact come from CERN and nobody else has intercepted the connection and sent their own virus program instead of the official SixTrack program. This procedure is though hinged on the fact that LHC@home has a secure method for signing executables. We will use the following method.
First of all, the machine that signs the executable must have access to the private key. This key must never be discovered and copied by a malignant entity. The only way to secure this key properly is therefore to not allow the key signing machine to be accessible from the Internet. Secondly the machine should also be physically difficult to tamper with. Yet we have to have a method of copying the executable to the machine and to copy the signature off of the machine again.
Our solution to this problem is to generate a bootable CD image of Linux with the private key on it. This CD should then be locked away in a safe when not needed. When a new signature is needed, almost any X86 computer could be turned into the code signing machine, without any fear of anybody having installed Trojans or viruses on the system since the booted system would be completely isolated from the normal system. You can even leave the network attached to the machine as long as we do not load any network drivers on the image CD operating system. The executable could be transferred to the system using a CD or a USB memory stick, and the signature could be copied out using the USB stick again.
We believe this is the safest and most convenient way of keeping the code signing secure. The most unsafe part of this procedure is when the CD-image is made. We have to be absolutely sure that the key is generated and moved to the image in a secure way. We also have to be sure that any trace of the private key is removed from any other place than the final CD.

3.5.5 Porting SixTrack to BOINC

SixTrack is a simulation that does calculations on the particles in the region where their behaviour becomes chaotic. It is therefore very important that the results of running the program with the same input on two different machines agree, when the final statistics are compiled. For that reason it is very important that the results are as similar as possible across platforms. The only compiler that was found to provide this behaviour was the Lahey Fortran compiler. The other tested compilers all had some small differences in their floating point calculations. This difference only appears in about 1000 out of the 60 million returned floating point numbers in the result. However these very small errors are significant enough to cause errors in the final analysis according to Frank Schmidt, the creator of SixTrack.
The BOINC API only exists in a version for C/C++. We then had a choice of porting SixTrack to C/C++ or the BOINC API to Fortran. Because of SixTrack's high level of numerical dependency, serious testing over the years and its size, it was out of the question to port it to C/C++. One way to get around this barrier is to run SixTrack through a wrapper program, like we had done during the test project. The wrapper would then communicate with BOINC. The test project had though shown some problems with using this method. When SixTrack was run through the wrapper it was for example not possible to have the graphics access data structures from the SixTrack application. It was also possible to kill the BOINC client in such a way that only the wrapper was killed and not SixTrack leading to unpredictable results when BOINC was restarted among many other small problems. Although the wrapper still worked fine from a purely computational point of view, we did not feel that it was something we could let the public run. We therefore had to make a Fortran API for BOINC. The API went through many phases, because each Fortran compiler we used had different ways of

linking with C code. The final version, which we will describe here, should work with any modern Fortran compiler because it creates object files that follow the calling conventions for Fortran. We have also made a version for the Fujitsu-Compaq compiler a link to a description of it can be found in appendix D.

3.5.5.1 The Fortran BOINC API

The API is a small library that you include in your BOINC build if you want to call BOINC functions from Fortran. It makes sure the relevant symbols are available for linking as well as doing the necessary translations of the function calls.
In Fortran a function definition and a function call have underscores pre- and appended in the object code. We have to make sure that this symbol is available during linking. If we from Fortran call the function `Call BOINC_init(0)` the compiler will create a symbol `_BOINC_init_` that it expects to find during linking. C compilers automatically create the prepended underscore, so all we have to do is create new aliases of the BOINC functions with an underscore appended. Because of overloading, symbols in C++ object files have their names mangled to include the calling conventions, the types of their input and other information about the symbol. Even though the C++ standard includes a description of how to implement this, some compilers chose not to honour this[15]. We therefore must also make sure to declare the functions in the module as `extern "C"` so we are certain that the symbol is created with the C conventions.
The function names will now be accessible from Fortran. The remaining problem is that Fortran and C have different calling conventions. Everything in Fortran is call-by-reference, whereas some types are call-by-value in C, for example integers. So the BOINC Fortran API functions have to accept pointers as arguments and then dereference the pointers before calling the C API. The final difference is the way strings are handled[16]. In C a string is a null terminated character array but in Fortran it is a structure containing the character array and an accompanying length. When a string is used in a function call in Fortran it is passed as a pointer to the start of the character array(like in C) and the length is passed as an integer appended to the list of arguments of the function. If for example you wanted to call the function `test` which takes two inputs: a string and an integer, its definition in C could be:

```
Void test_(char *ingredient, int amount)
```

In Fortran you would call it with:

```
Call test("pepper",3000)
```

The call that the Fortran compiler generates would be similar to the C code:

```
char[6] p = 'p','e','p','p','e','r';
Test(p,3000,6);
```

We therefore have to create a translation function from this to the ordinary null terminated strings of C. We have worked together with Karl Chen from the BOINC project and our BOINC-Fortran API is now a standard part of BOINC. Links to the code for the API can be found in appendix D. Figure 3.1 illustrates the function of the BOINC-Fortran API.

[15] For example Microsoft Visual C++.

[16] There are more differences between C and Fortran, for example the way multi-dimensional arrays are handled but these differences are not important in relation to the BOINC API.

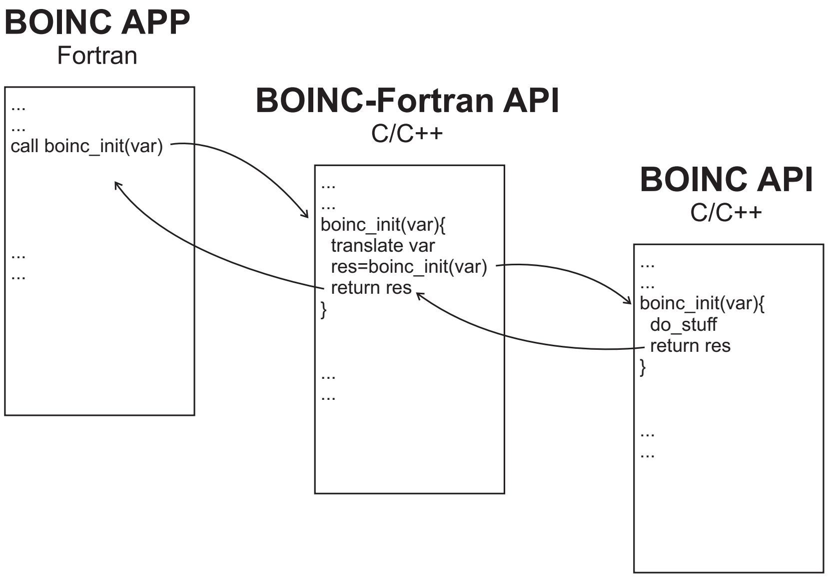


Figure 3.1: Using the BOINC-Fortran API to call BOINC API functions from within a Fortran application.

3.5.6 The Screensaver

The BOINC framework allows an application to run graphics, and these can be used as a screensaver if the user so desires. It is though not necessary for a project to supply any graphics, in that case BOINC has a standard screensaver that will be displayed. Together with our supervisors at CERN it was decided early on that we would need a screensaver. Partly to attract users and partly to satisfy our objective of getting users interested in the science behind the computations. Many ideas for how this screensaver could look were discussed, but it was decided that we would follow the lead of SETI@home and do a visualization of the particles flying around the LHC along with displaying different user statistics[17]. Acknowledging our lack of artistic abilities, we therefore created and tested a basic framework for doing OpenGL graphics and transferring data from SixTrack to the graphics routines, but left the work of finishing the look of the screensaver and coding the graphics to someone else.
To this end Jasenko Zivanov was hired and worked with us during the summer. A screen shot of the finished result can be seen in appendix B.5. The green smudges represents the protons as they are accelerated around the ring. The movement of the protons were supposed to be linked to the SixTrack simulation and a 3D model of the LHC. Jasenko worked hard with Eric McIntosh to solve this problem, but it proved to be difficult to show the data in a fluent and pleasing way. The simulation can complete a large part of a turn around the LHC for every frame drawn to the screen, the movement of the particles on screen will therefore appear very jittery. Unfortunately Jasenko had to return to his studies in Basel before he could finish his work on this problem. The graphics shown are instead based on a simple simulation of gravitationally attracted particles running simultaneously with the SixTrack simulation.
The users were very satisfied with the graphics and thought it was the most visually pleasing of the different BOINC projects (see the forums at [13] for user comments). Though the

[17] Like credits and percentage of current work done.

screensaver actually had no physical relation to the simulation, it still sparked many discussions on the properties of the LHC. The screensaver was therefore a success in both our goals.

3.6 Results from LHC@home

Setting up a PRC project is a lot of work and it is not the only demanding task, keeping a PRC project running also takes a lot of work, but it is well worth the expenditures. During the two months the LHC@home project ran, from Sept 1 until Nov, we amassed work equal to 61.6 years of CPU time[18]. To supply this we used two of CERN's discarded server machines both of them with slower processors than the average of the donated machines, to finish a job that would otherwise have required us to buy more than 360 dedicated machines. In figure B.1 we can see that there is potential for even larger gains. We only slowly increased the amount of users we allowed to join the project to see how the servers behaved. This was a decision made by our supervisors at CERN to avoid bad publicity. If we had allowed the maximum users from the beginning we would have been able to process much more data. Another problem we faced was that the physicists could not supply enough data to keep the queues populated. The process of creating new jobs was not automated enough and demanded more user interaction than there was available manpower from their side. This was not a problem, because they got all the studies done they wanted, but simply an indication that the LHC@home system is able to deliver much more CPU time per real time than what we got from it.

3.6.1 Donated Platforms

One aspect of creating a PRC project is deciding on a platform for the application. The application will often already exist and be designed for whatever platform you have locally. Since this, at CERN, would almost always be some type of UNIX system, the application would be written and compiled for this. SixTrack for example was written in Fortran to be run on the Linux batch clusters at CERN. It is therefore interesting to have an idea of what can be expected of the prevalence of the different platforms donated to a project. From the beginning we wanted to have a Windows version of SixTrack because we believed it to be the most widespread OS. Although Linux users have a reputation for being more interested in science and more knowledgeable about computers, we did not expect this to have a large influence on the ratio of Linux users to Windows users. From the table in B.2 we see that the machines offered for use in LHC@home are comprised of 91.6% Windows machines and 7.8% linux. It is therefore necessary for the application to run under Windows to really utilize the full potential, just like we expected. These numbers may be skewed a bit because we only offered a Linux and a Windows version. There may be more machines of other types, but they did not bother to join because they would not be able to do any work. We do not believe that the numbers are significantly off, mainly because they seem follow the general relative prevalence of the different operating systems on desktop machines.

We will therefore recommend, to anyone setting up a PRC project, to create applications that support Windows, Linux and MAC in that order of importance. Although the group of Macintosh users may seem very small, only 0.5% , compared to the other operating systems,

[18]This is the pure number adjusted for the fact that we calculate results at least twice because we do not trust the results from BOINC.

we will still recommend supporting a version for them for three reasons. Firstly, we believe the amount of MAC users may be bigger than our numbers may suggest. We stated on the LHC@home webpage that we would not support MAC, yet 0.5% of the machines that attached where MAC. Secondly, the MAC community seams very eager to join. This means that the community will offer to help develop and test the application. The Advanced Computation[19] group of Apple Computers even offered us their help in porting SixTrack to the MAC OS X platform. Porting the application to MAC therefore seems to be relatively easy. Finally, having a tightly knit community can lighten the load of running the project. They will very often help each other solve problems and post these solutions to the projects forums. We observed this behaviour in our group of alpha testers at LHC@home, they were very helpful in answering bug reports and explaining our decisions to do things one way or the other to the other users, thus lightening the administrative burden on us.

3.6.2 Keeping the Users Happy

Another manpower-intensive aspect is keeping the contributers happy, so that they keep contributing. This includes adding a pretty screensaver that the users like. We say add because very few applications will have graphics included when they were destined for execution on a cluster. It is also important to create a community that the users enjoy, this will keep them interested in the project. BOINC's way of doing this includes discussion forums and user rankings. These have to be monitored daily because the users will get annoyed if their requests for help are not reacted upon. Many annoyed users do not just leave, but they try to get others to leave as well. As an example we decided on rules for the Forums, among these were one stating that all communication in the "Problems" forum had to be in English. This would make it much faster for us to read and reply and the answers would be usable by most of the other users as well. This would keep us from having to answer the same questions in many different languages. One user got so angry because of this rule that he sent us emails threatening us and he encouraged people to leave the project through the forums. It took quite some time explaining to him why the rules where a good idea and not an attack on him or his particular language of choice. Handling problems like this is both difficult and time consuming, but a very important aspect of running a PRC project. People might leave because they sympathize with the complaining user, or even if they do not they might leave if they feel the problem is handled incorrectly.

3.6.3 Failure Rate - the Dreaded Tail

One of the biggest drawbacks of PRC is the relatively high probability of a job failing in some way. In BOINC this problem is lessened by running the same job on many machines, three being the default. The best case for a failure is if the job fails and that the failure is reported to the server. The server will then immediately create a new copy of the job and run it on another machine. Thus only delaying the final result with the time the job took to fail. The worst case is if there is a communication error or the job fails, but this is not communicated to the server. In this case the server will keep waiting for the answer to come back until a deadline is reached, normally two weeks after the job is distributed. Although distributing more copies of the job than needed lowers the rate of failure (for the final result not individual jobs), it does not eliminate the problem. If two of the three jobs fail we are in

[19]The group in charge of scientific computing issues.

the same situation as if we had only distributed the two jobs needed for a quorum.
To analyze how much of a problem this is in practice we chose a representative study that was run on LHC@home. The data on the "v64lhc100proeigth" study can be found in appendix B.4. It consist of 160090[20] results of which 4% fail and 9% time out. A time out seems to be equally likely across our supported platforms, but the Linux client fails more than twice as often as the windows client, 10% vs 4%. This probably means that we have an undiscovered bug in our Linux version of SixTrack. Under the assumption that these time-out errors are independent, these failure rates mean that there is a 0.8% probability that a WU will have to wait for one time-out period before finishing. This might not seem like a high probability, but this study consist of 35362 WUs. Therefore 264 WUs will have to wait at least two weeks for a result.

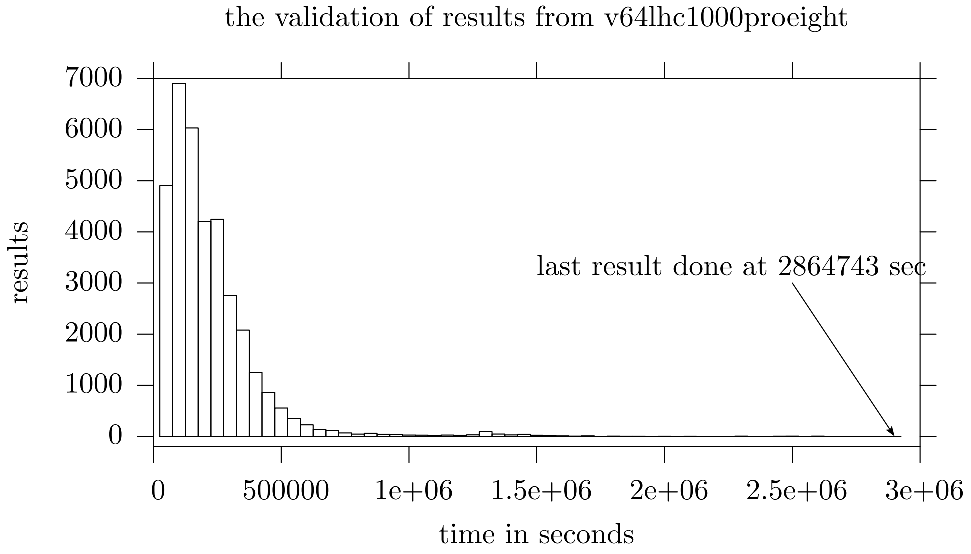


Figure 3.2: The time to finish a study.

Graph 3.2 shows the distribution of the time it took to finish the WorkUnits of this particular study. It displays the amount of WUs that finished within a given time interval. For example 6,903 WU finished within 50,000 and 100,000 seconds after they were issued. The actual value used is not the finishing time for the WU because this is not stored anywhere in the database. We instead use the time at which the result that was chosen as the canonical result was returned to the server. These two values are not necessarily equal, because the transitioner and validator are only run once in a while. So the real time value would be a little bit larger. The processing time we see in the graph is more akin to minimum time, but that is fine for what we want to show; That the majority of WUs are finished within a reasonable amount of time, but there is a tail of WUs that can take a very long time to finish. We see that while almost 90% of the jobs are finished after 11 days, the final WU is not finished until after 33 days. This is very annoying for the physicist, because he needs all the results before he can do postprocessing. He can see that 90% of his jobs are done and expect his study to finish

[20]Only results associated with a WU for which a canonical id has been found are considered. When a WU reaches ten failed results it is discarded because it is very likely that the job crashes with that input.

soon. He would normally use a dedicated resource and therefore expects that the last jobs should finish within a tenth of the time already spent and not two times as long. The bridge from BOINC to the Grid should be able to ameliorate this problem. It is based on the work of Chris Kenyon and it will be covered in greater detail in chapter 5.

3.6.4 Benefits for SixTrack

The increased compute power available to the SixTrack group allowed them not only to finish their studies in record time, it also allowed them to run studies they would never have considered before. Many of the studies were run with a tenfold increase in the number of turns, which helps study the long term stability of the beam. They were also able to study the effect of the angles of the particles' initial velocity on the simulation. Because of time concerns, only a few standard angles are normally used in a study. It was assumed that if these angles were chosen representatively enough, the end result would be the same as with many angles. With LHC@home they were able to test this hypothesis in depth.
Running SixTrack under BOINC meant that it was run on a much more diverse set of machines than was the case at CERN. In the first instance this led to many discrepancies in the results, but these will be studied to understand what effects are caused by the physics and what are caused by the numerical inaccuracies in software and hardware.
LHC@home delivered more than 600,000 finished jobs to SixTrack during our public run. For a comparison CPSS has delivered 200,000 finished jobs during its lifetime, which is more than a year. This means that LHC@home has delivered more than three times the amount of results in a quarter of the time as CPSS has. Furthermore the complexity of the jobs run on LHC@home has been much higher.

3.7 The Future of LHC@home

Since the middle of November 2004 the LHC@home project has been shut down for maintenance. It is the plan that it will be ready to run again sometime during February 2005. During this time the physicist have gone over the results and devised new studies to confirm and extend the series run in 2004 as well as including new and more refined physics models in the simulation. Eric McIntosh has used this time to fix some minor bugs that were found in the checkpointing code of SixTrack. He has also been able to track down the small numerical differences that sometimes occurred and which lead us to change the validator, see section 3.5.3 on the validator. The problem was that a few standard math library functions(for example `exp` the exponential function) returned results that were a bit different on different platforms. Eric McIntosh has worked together with the people in the Arénaire project at the École normale supérieure de Lyon. They have developed a new math library. They hope it has solved the problem and are at the moment running tests on the LHC@home project.
At the end of the project, we experienced a lot of performance issues with the database server, so the LHC@home hardware will also be upgraded during this period. The newest types of server machines are not put directly into production use at CERN, but are tested for a period of time to measure the stability. Luckily, the LHC@home project has been chosen to be a part of this test, so we can exchange our deprecated servers with some of the newest and fastest, and we hope this will increase the amount of users the system is able to handle.
The positive result of the project which was initially an experiment led by a small sub group of the IT division at CERN has impressed many people at CERN. So the decision has been

taken to make BOINC a more permanent facility at CERN. BOINC will probably be implemented as one of the services the IT division offers. This means that an experiment can approach the IT division and have their simulations ported and run through the IT divisions BOINC servers.

3.8 Summary

In this chapter we described how we set up the LHC@home project at CERN. We chose the CERN application SixTrack to be the application for LHC@home. We made a call translation module that allows Fortran programs to use BOINC, because SixTrack is written in Fortran. This module is now a standard part of BOINC. The project was a success, far outstripping the other resources available to SixTrack. It also showed the shortcomings of a PRC system. The unreliability of a PRC meant that although the throughput is very large, the finishing time of a particular study can be very long. For a representative SixTrack study we saw that the last 10% of the jobs took three times as long to finish as the first 90%. The project is now being turned into a permanent facility at CERN, who hope that it will also be a significant resource of computational power for other CERN applications. We address these two concerns later in the thesis. In chapter 5 we will supply a method for solving the problem of very long running jobs using the Grid. Then we design a general way of submitting jobs to BOINC through the Grid in chapter 6. This will remove the need for porting applications to BOINC, and is therefore a possible computational resource for any CERN application. However, we first need to introduce the Grid concept and the CERN specific implementation. This is done in the next chapter.

Chapter 4

The Grid

This chapter first deals with the concept of a Grid, defines what constitutes a Grid and explains about 'the Grid'. The concept of a virtual organization is discussed as well as the current status of Grid technologies.
An important building block in most of the Grids today, called the Globus Toolkit is discussed after the introduction of the Grid concept. The discussion is mostly focused on version 2 of the Toolkit because the LHC Computing Grid version 2(LCG-2), which we will use to constitute the Grid-side of the bridges to and from PRC, makes use of this version. A short discussion of the newest version of the toolkit, version 3, is also included for the sake of completeness.
After discussion of the toolkit we introduce the LCG-2 and describe a selected set of the systems forming the infrastructure of the Grid. In this section we focus on tools from the Globus toolkit used in the LCG-2 and specifically on deviations from recommended usage of these tools.
This chapter together with chapter 2 should provide basic understanding of Grid Computing and PRC in general as well of LCG version 2(LCG-2) and BOINC. This knowledge should render it easier to understand chapters 5 and 6 dealing with the two bridges.

4.1 The Concept

Grids, computational grids that is, take their inspiration from power grids. Power grids enable quick and easy access to power. The users do not have to consider where the power comes from and how it actually gets to the outlet. Architects for computational grids imagine the same properties for compute power instead of electrical power. They imagine vast quantities of computers connected via a large network, possibly the Internet, to form a worldwide computational grid. Users wanting compute power then 'plug in' to the Grid and use the resources as if they had a supercomputer of their own right next to them. Compute power is not the only resource that the Grid is supposed to offer; Storage of all kinds of data including programs is also envisioned. The Grid can be thought of as a very large virtual supercomputer. All the details of finding available compute power, finding the needed data, transferring data and applications to the actual computers doing the computations, is to be handled by the software running on the machines forming the Grid, all of this totally transparent to the user.
The observant reader noticed the word 'imagine' in the sentences above and this word means that the Grids of today are not quite as developed and commonplace as power grids.
The word 'Grid' is a very hyped word and sometimes it is easy to suspect that it is being used

to attract funding or to sell a product, which in reality has little to do with computational grids. This leads to many different definitions of what a Grid is and greatly increases the difficulty of getting to the bottom of the concept. From the many definitions of a Grid we have chosen one, which we think is the best. It is a three point checklist by Ian Foster [17]: A Grid is a system that:

1. Coordinates resources that are not subject to centralized control.
2. Uses standard, open, general-purpose protocols and interfaces.
3. Delivers nontrivial qualities of service.

If we use this definition on BOINC we see that it is clearly not a Grid since it does not use standard general-purpose protocols and does not deliver nontrivial qualities of service, although it coordinates resources not subject to centralized control. Likewise if we use it on Grid MP from United Devices and OfficeGRID from MESH Technologies we see that they too fail the second point and possibly also the third.

Now we have a definition of a Grid, but what about 'the Grid'? The Grid is envisioned to become the standard among Grids like the Internet is the standard among internets, and like the Internet this will probably be achieved by standardizing the protocols the different Grids use, thereby opening up for the possibility of connecting them together and creating one big Grid, an inter-Grid.

The challenges when building a Grid are numerous. Many of them arising from the problem of sharing resources across organizational boundaries. Users from a given organization must be able to safely and without effort run programs and store data on computers within a different organizational domain. This involves making a standard system of job submission and, because of the different organizations with their different hardware and software, an execution system capable of running jobs on many different subsystems. It also involves implementing security without centralized control, since the different organizations are independently managed, otherwise it would not be a Grid according to the definition above. Knowing about the existence and status of available resources across the entire Grid, again across organizational boundaries, in order to use these resources and to use them effectively is needed as well. These challenges all have to be met while attempting to make the use of the Grid as easy and as transparent to the users as possible.

An often used term in connection with Grids is a Virtual Organization(VO), which is a collection of individuals, institutions and/or organizations all sharing a common goal. To reach this common goal the members of a given VO share their resources in the form of compute power, data, applications etc. VOs are also a way of organizing users of a given Grid to lighten the administrative burden. As an example of a VO the people, who come from all over the world and many different institutions, working on the ATLAS experiment are grouped together to form a VO on LHC Computing Grid. Likewise people from the other LHC experiments are grouped together to form VOs on the LHC Computing Grid.

Most Grids today are testbeds and are used by computer scientists to implement and test new ideas within the field of grid computing. There are a few operational Grids today and they are mostly used to connect research institutions (physics research for instance) and their machines. They are used by the scientists of the collaborating institutions and not by the common public as opposed to the Internet.

The software used to 'glue' the many computers of a Grid together into a virtual supercomputer, is often referred to as middleware since it sits between the applications and the operating system. Most middleware today is based on the Globus Toolkit, which is the topic of the next section.

4.2 The Globus Toolkit

The Globus Toolkit(GT) is a set of open source tools, in the form of services and software libraries, that can be used to create a computational grid. It is not a complete Grid infrastructure, but instead components identified by the Globus organization as basic building blocks of value to practically any Grid. The tools offer functionality such as security, data transfer, data management, resource management and resource discovery. Most of the tools are generic and can be connected to components specific to a given Grid by writing a small piece of glue-software to interface with the API of the given tool. As stated earlier the toolkit is practically the standard when it comes to building a Grid infrastructure. It is being used by, among others, the Grid Physics Network(GriPhuN), Particle Physics Grid, the Network for Earthquake Engineering and Simulation(NEES), the Earth Systems Grid, NorduGrid and most importantly in LCG-2, which is the Grid we will use when building the bridges. LCG-2 uses version 2 of the toolkit, which is why the following discussion of the tools in the toolkit is mostly targeted at version 2, although a newer version of the toolkit is available. Since it is beyond the scope of this thesis to discuss the entire toolkit in detail only the most important and interesting details and tools are discussed.

4.2.1 Grid Security Infrastructure

The Grid Security Infrastructure(GSI) offers security measures needed in basically any Grid. One of these measures is authentication of the different elements of the Grid including the users of course. Since entering a password every time communication is initiated is very tedious, GSI supports 'single sign-on'. This means that a user enters his password once and then has access to potentially all Grid elements. 'Single sign-on' is greatly complicated by the fact that Grids consist of many separate organizational units making a centralized security system impractical. Instead GSI uses public key cryptography for authentication and certificates[1] for establishing trusts. GSI takes it one step further and introduces proxy-certificates, a certificate representing another certificate. Proxy-certificates can be used to delegate control, for instance they can be used to allow a computation to request other Grid resources by issuing a proxy-certificate from the user supplied certificate and to use this proxy to request the extra resources, hence allowing the computation to act on the user's behalf. To limit the security risks of proxy-certificates, these usually have a very limited lifetime. Proxy-certificates can also be issued from proxy-certificates since there is always a link to the original certificate and thereby a link to the certificate authority[2]. Because of the limited lifetime, proxy-certificates do not need to be encrypted allowing for them to be used without entering a password. This can be used to offer the 'single sign-on' since users can create a proxy-certificate from their original certificate (entering their password to decrypt the certificate) and then use the proxy-certificate to authenticate with the Grid resources.
Once authenticated a shared key can be used to encrypt the communication between Grid

[1]For more information about certificates and the way GSI uses them we refer to http://www-unix.globus.org/toolkit/docs/3.2/gsi/key/index.html.

[2]A certificate authority is an entity issuing certificates thereby for the certificate holders identity. The certificate authority has to be trusted by the authenticating entity in order for it to trust the certificate holder.

elements and if the overhead of encryption is not justified, communication integrity[3] is offered instead.

4.2.2 GridFTP

GridFTP is a data transfer protocol intended to be used in Grid environments. GridFTP is a secure, reliable, high performance data transfer protocol designed for wide-area networks with a high bandwidth. It is as the name hints based on the well-known FTP protocol, which has been extended to offer features specifically needed by Grids. GSI is used to secure both the control- and the data channel of the FTP communication and parallel data transfers has been added. Parallel data transfer involves splitting a given file into chunks and transferring the chunks simultaneously; This opens up for a potential performance increase because the chunks can be transferred from different servers that stores the same file. Even if only one copy of the data is available, parallel data transfer still has the potential to offer better performance on wide area networks because the individual data streams can be routed individually thereby using multiple WAN links instead of a single link.
GridFTP also supports third party data transfer allowing for data transfer between to servers without going through a middleman, but controlled by a middleman. This third party data transfer is of course also secured by GSI.
Since entire files are not always needed by Grid applications support for a partial transfer of a file is also part of GridFTP.
Reliable data transfer is also needed in Grid environments, therefore GridFTP includes features to restart failed transfers.

4.2.3 Replica Management

Grids are not only used to give access to compute power, but also to store data as mentioned previously. Often multiple copies of the same data, called replicas, are stored on the Grid for the sake of performance[4], robustness and scalability. Replica management manages the data and can select the best suited replica(s) in a given situation. It provides the ability for users to register files, enabling other Grid users to find them, create and delete replicas and to find files and replicas as well as information about the resource storing them.
Replica management uses another Globus component called the Replica Catalog to actually store the meta data. The Replica Catalog stores mappings between logical filenames and the physical file names which are often stored as URLs, which can be used to access the files. This mapping scheme allows the physical filename to change, i.e allowing the file to be moved, and it allows for better performance because applications are not bound to a specific instance of data, but the best replica can be chosen at runtime. The Replica Catalog also stores information about the resource actually storing the replica and this is the information used by the Replica Management to select the best suited replica(s). The Replica Catalog groups files into logical collections, probably to improve scalability. The Replica Management system is not distributed and therefore has scalability issues.
Replica management uses GridFTP to transfer data.

[3] Data cannot be changed in transit without being detected.
[4] Access to local copies is often faster than access to non-local copies.

4.2.4 Grid Resource Allocation and Management

To be able to run jobs on a Grid, mechanisms to start and monitor jobs on remote machines are needed. Grid Resource Allocation and Management(GRAM) offers such mechanisms in a secure way by using GSI. Via GRAM it is possible to submit, cancel and check the status of remote jobs. A GRAM client, which is not a part of GRAM, but uses the GRAM client API[5], is used to interact with the remote machines. To submit a job to a remote machine the client needs a specification of the job, which among other things names the remote machine on which to run. This means that GRAM does not include any scheduling logic, it only runs jobs on already identified machines.

The remote machines run the server component of GRAM called the Gatekeeper, which the clients connect to. Once connected, GSI is used for authentication and once authenticated and authorized the Grid user, who submitted the job, is mapped to a local user, which runs a GRAM job manager managing the given job. The job manager interacts with the local job scheduling system, which could be any system such as for instance Condor[6], to run the job. It is very important to note that the job manager does not directly start the job on the remote machine, it offers a general API which local schedulers must implement in order for GRAM to use them. The job manager runs while the job is active and can be queried by the client in regards to any changes in the job status, such as active (executing), suspended, done, or failed.

GRAM also takes care of staging[7] files using Global Access to Secondary Storage(GASS). GASS, which is also a part of the Globus Toolkit, is designed to enable easy access to remote files possibly stored on the submitting machine. It works similarly to the remote system calls under the standard universe of Condor, but not identically. As part of an `open` call the entire file is transfered in and as part of a `close` call the file is transfered out.

To sum up GRAM offers mechanisms for managing jobs on already identified remote machines that have a job submission system implementing a specific API from GRAM.

4.2.5 Monitoring and Discovery Service

The Monitoring and Discovery Service(MDS) is a system for publishing and querying the status of resources and their configuration. Together with GRAM they can be used to create a scheduler for a Grid.

The design idea of having a standard service in the middle able to connect to different subsystems and a Grid-specific super-system, as was used in GRAM, is also used in MDS. MDS consists of three major components; the Grid Index Information Service(GIIS), the Grid Resource Information Service(GRIS) and Information Providers (IPs). If we start from the bottom the IPs are interfaces, which receive information about the resource from resource-specific monitoring systems. Some IPs are readily available, others will have to be written for the specific resource. The information about the resource is passed on by the IP to the GRIS using a standard API. Usually both the IP and the GRIS run locally on the resources, but the GIIS does not. It collects the information from several GRIS enabled resources to allow searching through the information to find a suitable resource. A GIIS can connect to another GIIS introducing several levels of GIISes, for instance a Grid could have one GIIS per site and

[5] The client is most likely some sort of scheduler specifically developed for a given Grid such as LCG-2 or NorduGrid.

[6] Or BOINC for that matter. See chapter 6 for more information on this subject.

[7] Transferring files to and from the remote machine.

one Grid level GIIS. This along with caching of information throughout the system makes it quite scalable.
MDS is a LDAP[8]-based service and data is organized in schemas. It supports two schemas, the MDS core schema, which is a schema containing basic information, and the Grid Laboratory Uniform Environment(GLUE) schema, which is a joint effort between some Grid projects to define information needed to represent Grid resources. The information offered by MDS about resources could be load status, CPU, disk, memory and network information as well as OS information depending on the information provider.
GSI can be used to secure the system and GRAM can be configured to publish information about queues and job status on the local resources to MDS.

4.2.6 Globus Toolkit 3

Version 3 of the Globus Toolkit (GT) introduces web services to the toolkit. Explaining web services is beyond the scope of this thesis.[9] Web services are brought into the toolkit in an effort to use standard mechanisms to publish and discover Grid resources instead of the proprietary mechanisms of GT version 2 in order to improve generality and flexibility.
The Open Grid Services Architecture(OGSA) introduces the notion of a subclass of web services called *grid services*, and defines a set of interfaces and conventions which these must follow. Since a grid service is a web service, its interfaces are defined using the Web Services Description Language(WSDL), its presence is published and discovered using the Universal Description, Discovery & Integration(UDDI) system and data are structured using the eXtensible Markup Language(XML), all of them standard web service technologies.
The Globus Toolkit version 3(GT 3) provides tools to create grid services and even offers some, many of them replacing the GT 2 tools. The Reliable File Transfer(RTF) service provides just that by using GridFTP. The Replica Location Service(RLS) replaces the Replica Management of Globus Toolkit version 2 and provides better scalability by distributing the service. Web service versions of GRAM and MDS are also part of the toolkit.
The Community Authorization Service(CAS) is also part of GT 3 and has been created to reduce the administrative burden of assigning privileges to Grid users. Instead of having every resource provider specify individual privileges for every valid Grid user, users are grouped into communities and these communities are then assigned privileges to the given resource. Each community is then in charge of assigning privileges to the individual users of the community and issuing GSI certificates to its members, who use the certificates to gain access to CAS managed resources.

4.3 LHC Computing Grid

Version 2 of the LHC Computing Grid(LCG-2) is the Grid we will be using when creating bridges between PRC and Grid computing, and for that reason we will present an overview of this particular Grid in this section. The LCG-2 is a worldwide computational Grid targeted at providing compute power and storage space for the LHC. The LCG-2 is probably the largest

[8]Lightweight Directory Access Protocol(LDAP) is a set of protocols for accessing information in directories. It is based on a subset of X.500, also a set of protocols for directory access, but it is simpler and supports TCP/IP. The actual data could be stored in a standard relational database since the protocol only specifies how to access information.

[9]Interested readers are referred to 'http://www.w3.org/2002/ws/' for information on Web Services.

computational Grid in the world when measured by the number of sites. In the LCG-2's case it spans about 115 sites with a total of about 11.000 machines on average and this number is expected to rise to about 100.000 by 2007 when the LHC goes in to operation. The biggest task of the Grid will be to process and store the incredibly large amounts of data coming from the LHC experiments once the collider is operational.
LCG-2 takes its software components from many places and like many other Grids it is based on the Globus Toolkit, specifically version 2 of the toolkit. It is composed of a number of systems: the Workload Managements System(WMS), the Data Management System(DMS), the Information System(IS), the Authorisation and Authentication System and the Accounting System. The infrastructure of LCG-2 is large and complex and it is beyond the scope of this thesis to describe the entire system in detail. We will therefore only describe the three most interesting systems, the WMS, the DMS and the IS, below.

4.3.1 The Workload Management System

The WMS system is shown in figure 4.1. To submit jobs to LCG-2 users log in to a machine, which has the WMS User Interface(UI) installed. Once jobs have been submitted, the UI machine connects to the Resource Broker(RB). The RB takes care of scheduling, submitting jobs to remote machines, transferring files, and logging.
The RB uses GSI to authenticate users and once authenticated it copies the input sandbox, which is a collection of files stated by the user as present on the UI and needed for the job, to its local storage. It uses the matchmaker, the IS and the Replica Location System(from the DMS described in section 4.3.2) to find the best suited resource for a given job taking many points into account such as user specified constraints, queue lengths and replica localities. The matchmaker works according to the same principles as the matchmaking mechanism of Condor and it also uses ClassAds. Condor-G is used to submit the jobs to the Computing Element(CE) that is found to be best suited. Along with the job, a monitoring job called the *grid monitor* mentioned in section 2.3.4, is also submitted.
The CEs run the gatekeeper, which accepts incoming jobs from Condor-G and starts a GRAM job manager. The job manager submits the job to a site specific batch system like Condor, LSF[10], or OpenPBS[11]. In LCG-2 the job manager is only used to submit and cancel jobs, not query their status. This task is carried out by the *grid monitor* submitted along side the jobs as mentioned above. The reason for using the *grid monitor*, and not the job manager to monitor jobs, is performance. The job managers use a lot of resources on the CE since the gatekeeper starts a job manager for every job and these job managers run until the jobs finish. The *grid monitor* on the other hand can monitor all the jobs from the same user on a CE and the job managers can be instructed to exit as soon as jobs have been submitted to the local batch system, resulting in a much smaller load on the CE. The actual machines running jobs are called Worker Nodes(WNs) in LCG-2.

[10] For information on LSF the reader is encouraged to visit http://www.platform.com/products/LSF/.
[11] For information on OpenPBS the reader is encouraged to visit http://www.openpbs.org/about.html.

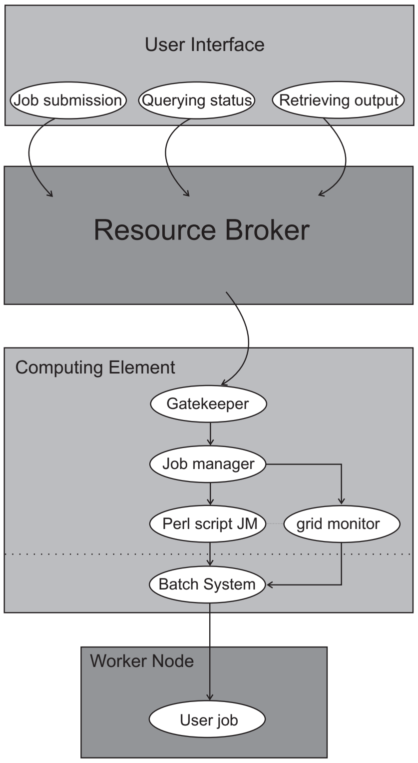


Figure 4.1: The Workload Management System of LCG-2. Modified from [29].

4.3.2 The Data Management System

The DMS consists of the Replica Location System(RLS) and Storage Elements(SEs). The RLS is queried to locate data and retrieve metadata as well as information about the SE actually storing the data.

Storage elements are computers with access to large amounts of data storage, which in some cases are large disk arrays and in others multilevel storage systems with tape robots at the lowest level. These storage elements run a GridFTP server, which makes the storage available to Grid users once they have the physical file name(PFN) of the data in question. The PFN is discovered by querying the RLS, which contains mappings from logical file names(LFNs) to PFNs. The RLS system used in LCG-2 is not the one from GT3, in fact nothing from GT3 is used in LCG-2. The GT3 RLS was developed in collaboration with EDG[12], but their paths split and two versions of RLS were implemented. LCG-2 uses the EDG version of the RLS, which is also designed to run as a distributed service. This is however not the case in LCG-2, where the system is based on a set of central servers, one for each VO on the Grid. As mentioned above the RLS is used by the matchmaker in the WMS to find the best suited CE for a given job.

[12] The European DataGrid project(EDG) is a Grid project commissioned by the European Union to develop Grid technologies.

4.3.3 The Information System

The IS of LCG-2 is based on the MDS system from GT2 and it uses the already mentioned GLUE schema to organize the information. The MDS system has however been modified to deal with issues of scalability and robustness. Like the standard MDS the IS uses information providers to provide information to a GRIS, and the GRIS relays this information to a site level GIIS, but there is no regional or Grid level GIIS in the IS. The overhead of the regional GIIS-system was lowering performance and the Grid level GIIS was unstable when collecting information from many sites and being queried by many users and RBs at the same time. These shortcomings mean that there is one GIIS per site in LCG-2 and none of them are connected to other GIISes. To serve as the Grid-level information service instead of the GIIS, LCG-2 has implemented the Berkeley Database Information Index(BDII). The BDII consists of two LDAP-servers, where one of them contains a read-only database and the other a write-only database. The system works by doing queries from users and RBs on the read-only database, while the write-only database is being updated with information from the GIISes. Once updated the two servers change roles and user queries are being done on the recently updated and former write-only database. This improves scalability and robustness.

4.4 Summary

The concept of a computational Grid in its purest form, is the idea of access to vast quantities of compute power and data storage as easy as todays access to electrical power. This idea has not been turned into reality yet, but less pure versions do exist. The Grids of today connect many organizations and offer a high quantity of compute power and storage. The infrastructure of Grids have to solve the problems of job submission, security and discovery of the existence of resources and their status all on resources not subject to centralized control. To aid in the task of building a Grid, the Globus organization has made a set of tools available. These tools include a security system, a safe data transfer protocol, a system for managing data, a system for submitting jobs and an information system keeping track of resources and their status. The toolkit is used as a basis for the Large Hadron Collider Grid version 2(LCG-2), but because of performance issues, some of the tools are not used exactly the way the Globus organization envisioned. Changes to the usage have been made to the job submission and monitoring system by using parts from Condor-G, which is described in section 2.3.4, and to the information system by using a custom made top level LDAP service.

Chapter 5

The BOINC-Grid Bridge

In this chapter we deal with connecting PRC and Grid Computing. We build a bridge from a PRC platform to a Grid allowing PRC jobs to run on Grid resources. We use BOINC as the PRC platform and LCG-2 as the Grid. The idea behind the bridge is to improve the overall usefulness of a PRC platform. The inspiration for the bridge, the work by Kenyon and Cheliotis, is discussed.

The extended BOINC system is introduced and design decisions are discussed. Afterwards the implementation is described and the results from it presented. We also discuss possible benefits for both BOINC and LCG-2 in having such a bridge. Towards the end of the chapter we discuss limitations inherent in the implementation and we present ideas that would improve it. Lastly we discuss features not present in LCG-2 that would make it more suitable for the purpose of bridging with a PRC platform.

5.1 Creating a PRC System With Hard Guarantees

We got the idea of building a bridge from BOINC to Grid from the work by Kenyon and Cheliotis[32]. They describe how it is possible to guarantee, what they call, a hard stochastic quality of service (HSQ) using a mix of cycle-scavenging and dedicated resources. HSQ is a set of statistical properties of QoSs that are guaranteed with certainty. An example of a HSQ could be that we guarantee that the user has an 80% chance of getting 5-6 CPU days and a 20% chance of 4-5 CPU days during the next 3 hours. A single user may be unlucky and only get the lowest amount of service, but the distribution is guaranteed. The HSQs that the method supports is of course more complicated than the simple example above. It can also be used to guarantee a distribution on the individual length of CPU slots. A CPU slot is an uninterrupted session on a machine. This is a very important ability, because if you have 3 jobs lasting 20 minutes and you are guaranteed 1 CPU hour, you would be satisfied with one slot of an hour or 3 slots of 20 minutes. However, while 60 slots of one minute would also honour the guarantee of 1 CPU hour, it would not allow any of your jobs to finish.

They argue that the concept of HSQ has merit also in a commercial setting, using the example of Chicago's Mercantile Exchange's Random Length Lumber. This lumber is sold with a guaranteed distribution of the length, but the actual length of the individual pieces of lumber is not guaranteed. They therefore believe that it is possible to sell computer resources based on the same principles.

The method is based on using dedicated resources to help shape the actual distribution of the

unreliable cycle-scavenging resources. Their architecture consists of an HSQ Controller that is able to submit and monitor jobs on both a cycle-harvested resource(CSR) and a dedicated resource. A dedicated resource in their terminology is a reliable resource under the sole control of the HSQ Controller. This definition is unnecessarily strict. Their method will still work if the dedicated resource is able to deliver a guaranteed QoS and the possibility to reserve this QoS or a guaranteed minimum QoS available at any point in the future. The HSQ Controller accepts or rejects incoming contracts[1] based on whether it can be honoured with the available resources. The idea is to use cycle-harvested resources until the point is reached where the contract has exactly enough time left to be able to finish on the dedicated resource. The HSQ Controller continuously monitors the available resources on the dedicated resource and the status of a contract, updating it whenever one of its jobs finishes on the CSR. Once a contract has no spare time left, i.e. it can exactly be delivered at the deadline by the dedicated resource, the HSQ Controller transfers the execution of the contract's jobs to the dedicated resource. This is why we demand that if a dedicated machine is not under the sole control of the HSQ Controller, we have to have a reservable or minimal QoS and not just a QoS. Otherwise it is possible that the HSQ Controller checking if there is spare time left given the QoS of the dedicated resource, decides that there is and keeps the jobs on the CSR, but immediately after the QoS could deteriorate to a level where the contract could no longer be honoured.
They show that the method described is a viable way of providing HSQ for a CSR, with only a small[2] requirement for dedicated resources.

5.2 Bridging From a PRC Platform To a Grid

Our project does not have any of the economical aspects of the article, but our plan is to use simplified ideas from this article to help resolve the problem of the unreliability of PRC resources. We saw an example of what this unreliability means for projects like LHC@home in section 3.6.3, where a small portion of the jobs took an extraordinary long time to finish. The idea is that we use the Grid as our dedicated resource to deliver a HSQ from a PRC resource. Unfortunately a Grid is not a dedicated resource, but according to Ian Foster's definition of a Grid it should deliver a non trivial QoS, see section 4.1. The QoS concept it delivers might still not satisfy the requirements we set for it in section 5.1. However, we are not trying to create a marketable PRC resource, but simply a more reliable one. We will therefore be satisfied with a simple QoS guarantee, and therefore only be able to deliver HSQs after best effort. Likewise we will not reject contracts we cannot honour, but satisfy them to the best of our ability.
We replace the dedicated resource in the architecture described above with Grid resources to enhance a PRC system. The part we have to make is the HSQ controller or as we refer to it, the bridge. There are very few general issues in bridging from a PRC platform to a Grid, it is mostly a matter of overcoming technical obstacles specific to a given PRC platform and a given Grid. Generally the kind of jobs that can be run on a PRC platform are much more limited than the kind of jobs that can be run on a Grid. Therefore PRC jobs can usually be run on a Grid without too many problems. This is not the case the other way around as we discuss in the next chapter. One inherent problem is the fact that PRC systems usually run

[1] A contract is a set of jobs with an associated HSQ.
[2] 1% - 10% based on the size of the contracts

Windows applications while Grid resources usually run some form of Unix.
When it comes to security there are also very few problems. In general, the public resources normally used by a PRC platform cannot be trusted and therefore certain measures have to be taken. Being able to run on Grid resources, which can be trusted, these measures are no longer necessary, but not directly harmful. However, the security of the Grid still has to be dealt with. This probably involves mapping a set of certificates for the Grid to the set of jobs to be submitted.
To supply QoS a suitable QoS metric must be chosen. The metric will be used by the bridge to decide which PRC jobs are candidates for Grid execution and used to negotiate the appropriate QoS from the Grid.
In the following sections a specific bridge from the BOINC PRC platform to the LCG-2 platform is described.

5.3 The BOINC To LCG-2 Bridge

To supply QoS from public resources, according to our modified version of the architecture by Kenyon and Cheliotis, BOINC needs a way to submit jobs to a Grid. In our case the Grid is LCG-2, so the system needs a way to run jobs on LCG-2. According to the architecture it should of course still be possible to run jobs on BOINC clients, since the entire idea is to use as many public resources as possible and as few dedicated resources as possible. Normally BOINC clients connect to the BOINC server and pull jobs from the job queue however this approach will not work when trying to run jobs on LCG-2. There is no support for the pulling of jobs on the LCG-2, only pushing is supported. Therefore the bridge has to actively push jobs to LCG-2, which is a major break from the normal operation of the BOINC system. How to actually run a BOINC job on a LCG-2 WN is the topic of the next section.
Choosing the suitable QoS metric is the topic of section 5.3.4.

5.3.1 Running BOINC Applications on LCG-2

BOINC applications are specifically built for BOINC. They have calls to BOINC functions which interfaces with the core client to execute certain tasks such as file name translation. Because of this they cannot be run unmodified without the core client[3]. To be able to run BOINC WUs on the Grid, a program is needed to interface with the application and basically do the job the core client normally does. The standard core client cannot be used though because it connects to one or more BOINC servers and requests work etc. The object is just to run a single BOINC WU as if it was just another Grid job, and return the result via the normal Grid channels. For efficiency reasons one could argue that running more than one WU per Grid job is a better solution, since the overhead of network connections, job generation and scheduling would be distributed among the WUs in a single Grid job. However batching WUs in this manner greatly increases the execution time of a Grid job. The matter will be discussed further in section 5.3.4.
Using the normal BOINC mechanisms to return results would require major changes to the server software, since results from Grid must be handled differently from normal results. Having the bridge handle the difference the normal server software does not have to be

[3]Though this was the case at the time the bridge was implemented using version 4.03 of BOINC it is no longer the case with newer versions of BOINC.

changed. The matter is further discussed in section 5.3.2. Using the BOINC mechanisms would also require allowing the client to communicate with the BOINC server to return jobs, but not to retrieve new ones and it would require the WNs having outside network access, which most do not. Because of these issues the Grid mechanisms of returning results will be used.
A specially made program to impersonate the core client could be built to interface with the BOINC applications, but since the standard core client is almost usable, we chose a different approach. Instead of starting from scratch, we chose to make minor modifications to the standard core client to make it act in a way that best suits our needs. These needs are to run a single job already shipped with the core client to the Grid machine, not contact any servers or upload any results and exit when this single job finishes or crashes (whichever comes first. . .).
The core client is structured as a set of finite state machines each in charge of a specific task, such as doing the computations, transferring files, or talking to the scheduler. These state machines are also in charge of changing the state of a result[4]. The state machines are repeatedly called in a loop in the core client, where they check for actions to be taken within their area of responsibility. To disable all communication the state machines in charge of communication were disabled from this loop. Since the communication state machines were no longer running, additional changes were needed to simulate the normal state-change of a result, that is from the state of no longer computing to uploaded and finally reported. Without these changes the result would never get deleted from the state file[5] of the client and it would not exit, thinking there is still work to be done.
Normally when a result is finished the garbage collector kicks in and deletes the result since it has been uploaded to the server. In our case this is not desirable, because the result has not been transfered off the machine by LCG-2 yet, so the actual deleting of files in the garbage collector had to be disabled as well.
Finally the core client has a commandline-switch that instructs it to exit when there is no work to be done, but unfortunately this only happens when the scheduling server has been contacted and the client has made sure that there really is no more work to be done. Since the communication state machines are not running, the client is never satisfied that there is no more work to be done. Minor changes to the exit code were required to fix this problem.
The core client now runs results in the state file, never communicates at all and exits without deleting results when it runs out of results to process.

5.3.2 The Bridge Daemon

The first thing to consider is how to get the (right) jobs from the BOINC database and run them on the Grid. The BOINC server complex is highly modularized making it possible to split the different daemon of the complex between several machines for scalability [6], it seemed obvious to use this idea when constructing the bridge. It was therefore chosen to create a separate daemon responsible for doing the bridging. This way the bridge daemon can

[4] Each WorkUnit gets turned into multiple results because of redundancy, which means a result is actually a job. For more information about the BOINC system consult section 2.2.

[5] The core client maintains a file representing the state of this particular client including downloaded files, pending results and so on.

[6] Even run some of the daemons in duplicate.

be run on a separate machine and avoid putting more load on the existing BOINC servers[7] except for the database server which it needs access to. One of the many other possibilities would have been to incorporate it into the transitioner, but for the sake of scalability and modularity this solution was not chosen. It also makes it easier to enable/disable the bridge - either add it to the list of daemons to be run in the BOINC server configuration file or not.
For each WorkUnit the BOINC server creates multiple results to represent the actual job because of the redundant computing used by BOINC. This is important when it comes to deciding what to do when a WorkUnit is submitted to the Grid. Creating a new result to represent the WU executing on the Grid has several advantages; For one it is keeping in line with the BOINC-way of doing things. It also makes it much easier to assimilate the result and to validate incoming results from BOINC clients against the Grid result, since all of this is already done by the server complex requiring no changes at all. For these reasons it is chosen to create a new result for jobs executing on the Grid at the cost of database space and performance.
When results return from normal BOINC clients, they are validated against other results from the same WorkUnit, but WorkUnits being run on the Grid are only run once since it is a trusted resource. Because it is a trusted resource, there is also no need to validate the results from it. They are therefore immediately made the canonical result for the corresponding WorkUnit, if such a result has not already been found. Making a result the canonical result without validation can be achieved by manipulating the WorkUnit's record in the database.
It is possible that a WU finishes through normal means while the WU is running on the Grid. Such a WU should be cancelled to efficiently use Grid resources would. This is however not done in the current implementation. By cancelling the jobs, the ability to validate the canonical result against the result from the Grid is lost. This is not a big loss though since the only possible action if the canonical result is found to be invalid is to take away the credit from the involved users. The damage has already been done since the canonical result has been made the final result because the assimilation has already been run.

5.3.3 The Extended BOINC System

The extended BOINC system, which includes the bridge to LCG-2 is shown in figure 5.1. It illustrates the two ways the system has for executing the jobs in the queue, the first and regular way involve normal clients connecting to the scheduler asking for work and once done with the work connecting again to report the results. The second and new way is a very different approach, instead of the clients pulling jobs from the server by connecting to it, the server instead connects to the resource broker of the Grid and pushes jobs to the Grid, in essence working more like a client than a server. Also instead of the clients reporting the results automatically, the server now has to actively monitor the running jobs by polling the logging and bookkeeping service of the LCG-2 for the status of the jobs and actively get the results once the jobs finish. This is a major break from the normal operation of the BOINC server complex.
The bridge consists of three major parts; the BOINC server complex, a modified BOINC core client and the bridge daemon. The server complex is standard and was covered in section 2.2.4, the last two parts will be covered in this chapter.

[7] BOINC servers seems to be suffering from scalability problems.

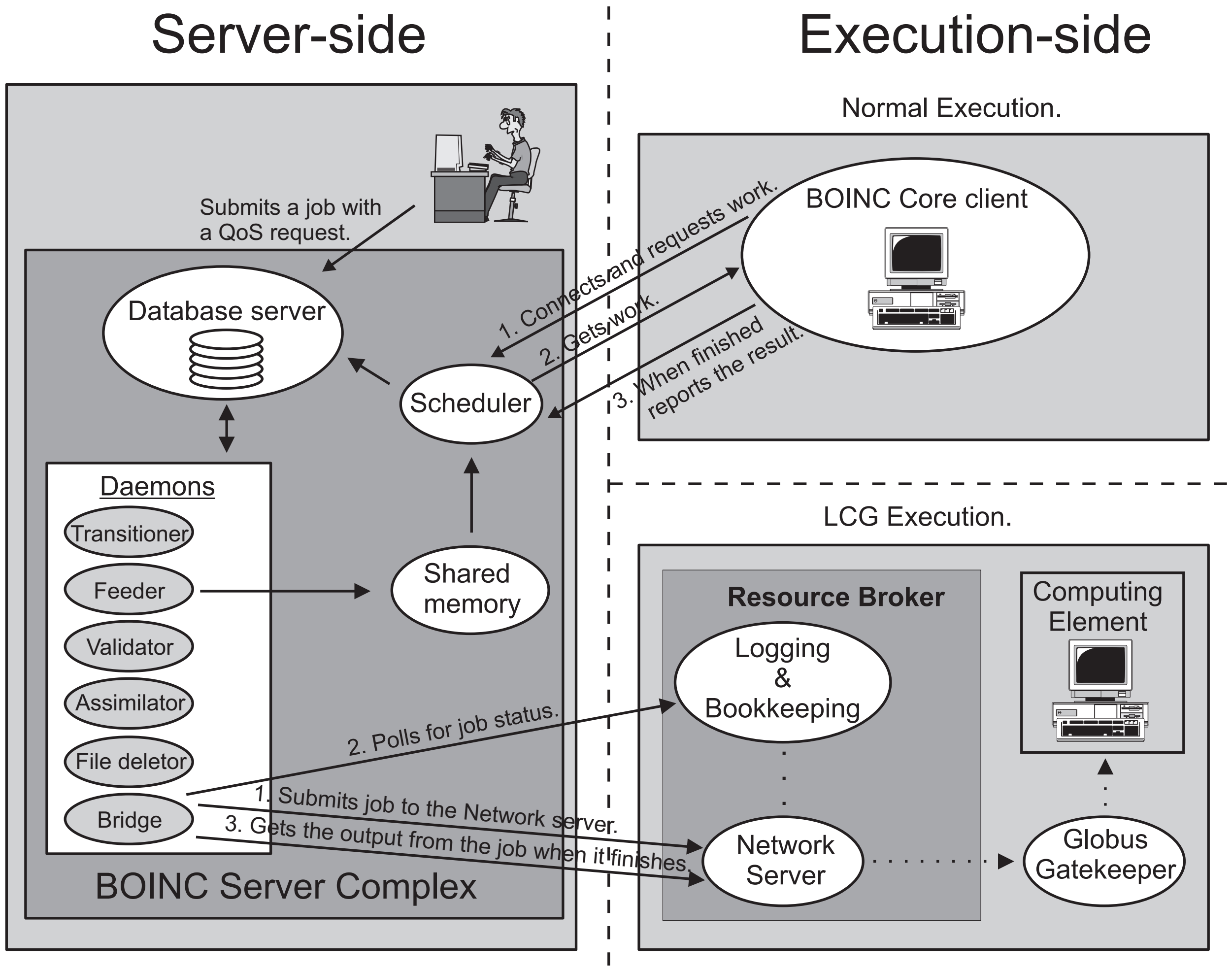


Figure 5.1: Job-execution on the modified BOINC platform, which includes the possibility of submitting jobs to Grid resources.

5.3.4 QoS

As stated in section 5.2 we have to decide on a QoS metric. This is to be used to decide which WUs should go to Grid on to negotiate the appropriate QoS from the Grid. However our Grid, LCG-2, does not supply QoS in the common sense of the word. This fact could present somewhat of a problem regarding the third point on Ian Fosters checklist as to what constitutes a Grid. However according to Mr. Foster himself QoS contains more dimensions than the traditional sense, including security, reliability and performance, justifying calling LCG-2 a Grid. The LCG-2 submits jobs to the job-queue which appears to have the shortest waiting time, therefore best effort. Most Grids of today do not supply QoS. There is work going on in this field to provide QoS from Grid resources, but until this is a reality it is best effort instead, which translates into best effort for the bridge as well. Readers interested in QoS on Grid resources should consult [21] for more information. Though it does not support QoS it is still a more predictable resource than BOINC clients and certainly a more valuable resource and should therefore not be wasted. This still means jobs should only go to the Grid

when absolutely necessary, although this is impossible to achieve in practice.
Traditional QoS metrics include response-time and average response-time for interactive systems, delay, variation in the delay (called delay jitter) and average throughput for networks, but none of these are relevant for PRC jobs. What matters for these jobs is when the result is ready, so this is the metric of choice. This sort of scheduling is known as deadline-scheduling. With LHC@home the jobs were almost always grouped into studies, where each study looked into a specific area of the accelerator's behavior[8], hence the scientists were not particularly interested in when a single job finished, but instead when a study finished. Whether the metric operates on a group of jobs or a single job the same effect can be achieved by setting the group size to 1 in the former case or by setting the finish time of a bunch of jobs to the same time in the latter case, therefore, for reasons of simplicity, it is chosen to operate on a single job.
In order for the bridge to try and finish the job in time, it needs a way of figuring out when to submit a given job to the Grid; Since our design is based on the work by Kenyon and Cheliotis we chose to use the method they propose. However since our resources are not dedicated we have to account for the fact that our jobs get queued. Therefore it is chosen that a job is submitted to the Grid when the time left for the job to complete in is less than two times the estimated time to run it on an average Grid machine. There are many problems with this way of deciding when to submit a job. First of all, why is two times the right number? If the Grid is heavily loaded the number might be too small to finish the job in time and if it is not heavily loaded it might be too high causing possibly unneeded jobs to be submitted[9]. This brings up the issue from section 5.3.1 about batching WUs together to form a single Grid job. Since this would result in a much bigger job than single WU jobs it would need to be submitted much earlier than single WU jobs, since they can and will be run on many different worker nodes in parallel resulting in a faster turnaround. There is however one way of tricking the LCG-2 to run a multiple WU job on several WNs. Since about 10% of the LCG-2 sites offer and publish MPI-support a multiple WU job could be submitted as a MPI job causing the wanted effect. There are downsides to this approach, MPI jobs are not started until all the needed CPUs are available, possibly and very likely causing a large portion of these CPUs to sit idle while waiting for the last CPUs to become available. First of all, this wastes Grid resources, but it also means a greater turnaround time for all, but the last of the jobs in a batch[10]. The turnaround time for all the WUs is further worsened by the fact that 90% of the Grid resources will not be usable because of the lack of MPI support. A greater turnaround time means a higher probability of wasting Grid resources, because it necessitates earlier submission to Grid. This introduces the probability of a WU finishing normally because of the extended timeframe needed by the earlier submission, thereby making the Grid job a complete waste. Batching WUs together is a good idea, since it reduces scheduling overhead, but more importantly offers the possibility of only transferring the files common among a collection of WUs once, thereby significantly reducing overall submission time. It does however complicate matters quite a bit and given LCG-2's limited support for such jobs, we chose the simpler approach of one WU per Grid job.
Another problem is that longer running jobs get more time to use in queues than short run-

[8] For instance given a fixed set of amplitudes and a fixed set of seeds what happens when the angles are varied?

[9] The whole point is to rely as much as possible on public resources and only use the Grid as a last option, which means waiting as long as possible before submitting the job, but not too long. . .

[10] In the case of SixTrack this would not matter since the entire study is not finished until the last WU is.

ning jobs, since jobs get submitted when it is estimated that the time left is less than twice the execution time. The extra 'execution time' is of course there because jobs usually do not get to run as soon as they are submitted to the Grid, but spend some time queuing, but there is no particular reason why a long running job should spend more time in a queue than a short-running job.
The next problem comes from the estimates of the performance of an average WN and the job's requirements, which is exactly what it is, estimates. For the concept to be sound, the bridge needs an estimate of the required number of floating point operations needed to complete a given job[11] and it needs an estimate of the performance in FLOPS on an average WN. The machines we have seen on LCG-2 have been 1 GHz Pentium IIIs and according to SETI Spy[14] the peak performance of such a machine is about 166 million FLOPS. We estimate this peak performance to correspond to an average of 100 million FLOPS. Therefore the default estimate of the performance of an average WN is set to a 100 million FLOPS. Since this is only the default value, server administrators can consult [14] to estimate the average performance of their machines.
The estimates are difficult to get right and there are probably many other problems with the algorithm chosen, but since all of this is best effort and it is beyond the scope of this thesis to develop elaborate QoS algorithms, it will do.
Instead of having the administrator of the bridge estimate the floating point performance of an average Grid CE, it would be possible to query the Grid. The sites in LCG-2 report their number of CPUs and the SpecInt performance of the CPUs or a site specific reference machine if the site is composed of different machines[12]. This SpecInt value can be used to estimate the floating point performance since there is a strong correlation between the two in normal PCs, which is what the LCG-2 is mostly made up of. Once the floating point performance of the WNs from different sites has been estimated, it could either be used to make the average floating point performance estimate or it could be used together with other improvements to create a much more accurate scheduling of jobs. A more accurate scheduling could involve either knowing the queue lengths in time for the sites, which is not possible with the current LCG-2, or running shadow jobs on different sites as placeholders, to be replaced by actual jobs if needed. The placeholder scheme is a way of reserving slots in queues on the Grid. When a placeholder reaches the front of a queue, an appropriate job could be transferred to it. If the placeholder's slot is not needed it could simply die allowing the next job in line to use the resource. The placeholder scheme would require a way of figuring out that a placeholder has reached the front of a queue and it would require a way of replacing a placeholder with an actual job. A drawback to this approach is that while cancelling an unneeded placeholder does not burden the WN it would still waste resources on the RB and on the CEs. Nevertheless it offers a much more accurate way of figuring out when to submit jobs to the Grid.
Below is the formula used to decide if a given WorkUnit is to be submitted or not:

$$2 \cdot \left(\frac{estimated\_floating\_point\_operations\_for\_WorkUnit}{estimated\_FLOPS\_on\_an\_average\_Grid\_machine}\right) \geq deadline - present\_time.$$

Since BOINC keeps all of its jobs in one big FIFO queue, the problem of deadline scheduling could also have been solved by monitoring the job completion rate. The rate could be calcu-

[11] Each WorkUnit in BOINC already has an estimate of the required floating point operations to complete it, since this is used when clients connect and ask for say two days of work given their FLOPS performance, so this is of course the same estimate used by the bridge.

[12] If the machines are too different the LCG-2 recommends splitting the site into more homogeneous sites.

lated on the server by monitoring the job completion. It could also be calculated by having each client give an estimate of its expected throughput based on its history and have the server compile the statistics. The rate could then be used to estimate whether or not a given QoS-enabled WU would finish in time given its position in the FIFO queue and its deadline. This way WUs with only a very slight chance of finishing could be submitted to the Grid long before their deadline ensuring that the QoS is upheld. If the estimate is right, there will be no waste of Grid resources. However we stick with the simple solution of Kenyon and Cheliotis though slightly modified.

5.3.5 Queues

Since the BOINC system with the addition of the bridge becomes quite complicated, a description of the many queues in the system is warranted. Figure 5.2 shows the queues and their connections with each other and serves as a graphical backup to the following description. Once a WorkUnit is created the results[13] are put in the single job queue of the BOINC

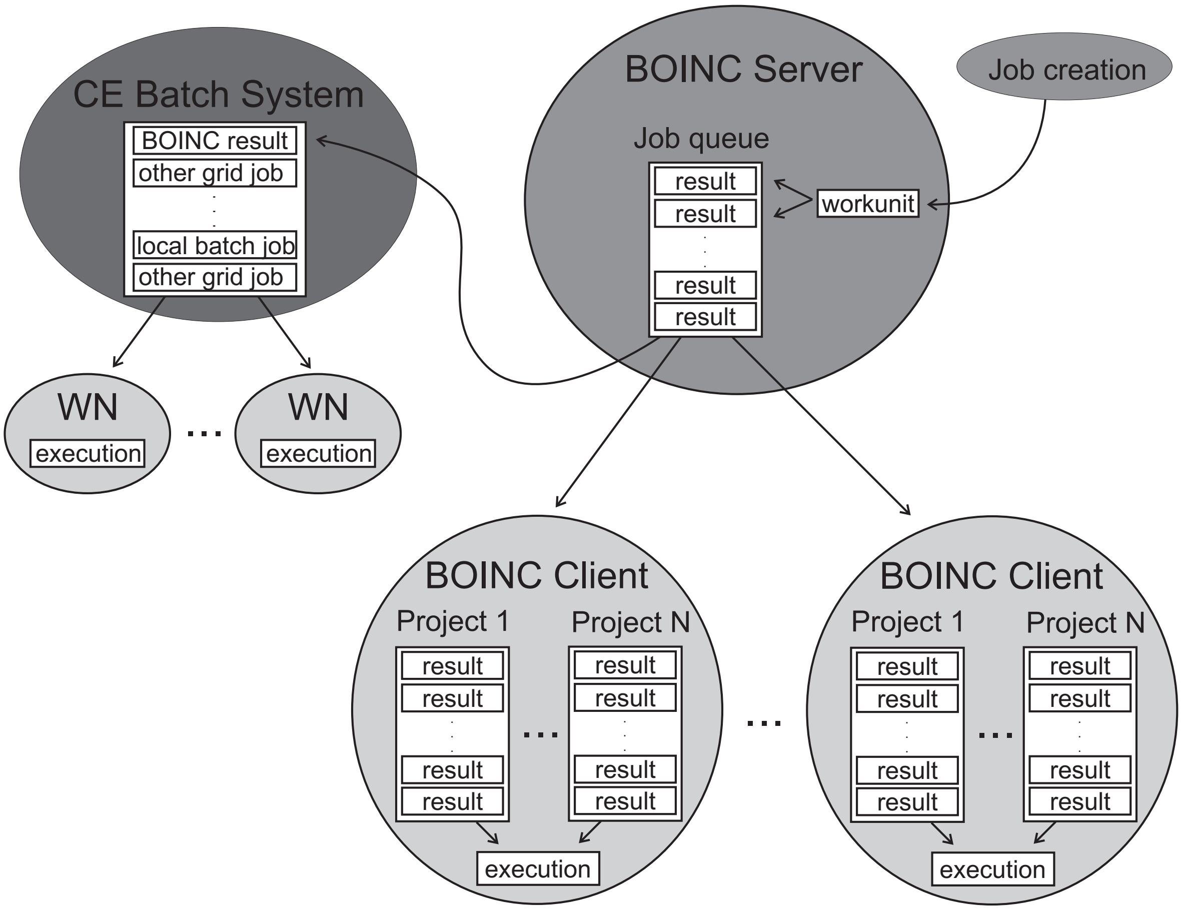


Figure 5.2: Job-queues in the extended BOINC system, which now includes the bridge to LCG-2.

[13]Redundant computations.

server. During normal operation these results are then picked up by the clients, who pick up a buffer of jobs during one scheduling session to limit network usage. This means a picked up result is put in a local queue on the client and awaits execution. Once executed the result is uploaded and its journey through the queues ended. If the normal operation fails to satisfy QoS-enabled WUs the bridge kicks in according to the section above. This means a new result is created and immediately submitted to the Grid. Existing results from the same WU are still somewhere in the normal BOINC queues, i.e. in the server's queue or in some local client(s) queue(s). Since BOINC resources are vast and inexpensive, this scheme maximizes the chances of the WU finishing in time without any real disadvantages except for stealing compute power from other QoS-enabled WUs possibly forcing them to be submitted to the Grid as well. Once submitted to the Grid, the RB of LCG-2 finds a suitable CE for the job and it is queued on the CE-local batch system. After execution the result is retrieved by the bridge and the WU is done. The normal results from the WU, if not completed, are still in the normal BOINC queues and should probably be deleted. However this is not done in the current implementation. There are quite a few issues with deleting WUs and they are all dealt with in section 6.5.2.2 concerning the delete tool for the Grid-BOINC bridge. The tool could actually be used by this bridge to delete the results.
The queue on the BOINC server is FIFO by default but later versions of BOINC supports prioritizing among WUs[14]. Each WU is assigned a priority value and the scheduler hands out the WUs based on this value. We propose setting this value to reflect the ratio between the estimated number of required floating point operations of the WU and the time until its deadline when creating a QoS-enabled WU. This value should always be higher than on non QoS-enabled WorkUnits thereby ensuring that the BOINC system tries to finish the QoS-enabled WUs before normal WUs. We further propose updating this as the WU gets closer to the deadline thereby enhancing its chances of finishing in time.
The local queues on the clients are also FIFO, but since the client has a queue for each project it is attached to, and the user can assign shares to the different projects the FIFO-property is only assured on a project basis. The batch systems of the CEs have their own queues and can therefore use whatever scheduling scheme they like, such as Condor's user-priority based scheduling from section 2.3.3.

5.3.6 Database Changes

The standard database of the BOINC server does of course not support bridging with the Grid, so a few modification/additions are needed. The first thing that is needed is a way to record the QoS agreement for each WorkUnit, in our case a Unix-time indicating when the result of the WorkUnit is needed. Since the QoS agreement is connected to one and only one WorkUnit, the most intuitive solution to this need is to add an extra field to the WorkUnit table. This is the solution chosen and the field has been named "finish_before".[15] A new table containing workunitids and their corresponding QoSs could also have been used to satisfy the need, but the former solution is both simpler and better performing, since it doesn't require the joining of two tables to find WorkUnits in danger of not honouring their QoS agreement.

[14] Unfortunately this feature was not available when we implemented the bridge.

[15] Since not every WorkUnit may require bridge support it is possible to disable this by setting "finish_before" to zero, which is exactly what happens if no "finish_before" time is specified when a WorkUnit is created. This also means that users wanting their WorkUnit to be submitted to Grid immediately should set "finish_before" to at least 1.

The next thing that is needed is a place to record the job-id given by the Grid to submitted jobs. This id is the only way to identify a job on the Grid and is needed when polling for the status of a job and when retrieving the result of a job. Since it is very inefficient to poll the status of a job continuously, a status-time, indicating when next to poll for the status of a job is almost a necessity. This will be set to an initial estimate and then incremented when a polled job has not yet finished.[16]. In theory at least, a job could go into an infinite loop or something else causing it to never finish or never get detected as finished. In this case, it is not optimal for the bridge to keep polling for the status of the job. A time-limit is needed on each job. When the limit is exceeded, the job is considered failed and the database updated to reflect this. Since a new result will be created for each job submitted to the Grid, the result-id needs to be connected to a job-id and the job-id needs to be connected to the status-time and the time limit. These fields could be added to the result table achieving the needed effect. Since it is very likely that only a few of the results will actually be run on the Grid, it would be inefficient use of the database and underlying disk-space to add these fields to all results. For this reason a new table called "grid_job" is created:

jobid	resultid	workunitid	status_time	report_deadline

The observant reader will notice a field called workunitid not mentioned above. This is a redundant field recording the id of the WorkUnit connected to the job. It is redundant because the result also records the id of the connected WorkUnit, so the id could be discovered by joining the result table and the grid_job table on the resultid, but for performance reasons it is included in the grid_job table as well. As an example, when a job finishes it needs to be recorded to both the result table and the WorkUnit table and since the workunitid is already known, there is no need to do a join on the workunitid between the result table and the WorkUnit table [17] hereby saving a query every time a job finishes at the expense of a slightly bigger database.
This concludes the changes to the database.

5.3.7 User Defined Settings

In order for the bridge to submit jobs to the Grid, it needs a few pieces of information from the administrator. Besides these necessary pieces of information there are a few settings which can be tuned by the administrator for better performance etc. The bridge needs access to this information, the usual ways of getting these configuration setting include a configuration file, environment variables etc. The rest of the server daemons get their configuration settings from a shared configuration file, it therefore seemed obvious that the bridge should operate in the same way. This solution means that the administrator only needs to look one place for all his configuration needs. The downside is of course the size of the single configuration file which could make it very difficult to cope with. Since the bridge only has 8 configurable settings and only three of them must be set, we chose to use the existing BOINC configuration file. The 8 settings are as follows:

- Virtual organization name - Has to be set (Unless 'default' is valid on the particular

[16]Callbacks would have been more efficient but also much more complicated for Grid architects to create.
[17]The two largest tables in the database.

Grid).

- NS (Network Server) server name - Has to be set.
- LB (Logging and Bookkeeping Service) server name - Has to be set.
- NS server port number - Defaults to 7772.
- LB server port number - Defaults to 9000.
- An estimate of the number of floating point operations per second an average Grid machine can do - Defaults to 100.000.000.[18]
- The interval in seconds between each execution of the main loop in the bridge[19] - Defaults to 30. Can be used to tweak performance.
- The number of times a failed job is retried by the Grid - Defaults to 2.[20]

5.3.8 Implementation

The bridge daemon is written in C/C++, with C being the language in which the majority of it is written as is BOINC. That also explains the rather sad looking class-diagram that is figure 5.3, but the few classes that do exist deserves to be mentioned. The `GRID_APP_VERSION` and the `GRID_USER` are both container-classes. The former contains information relevant to a specific version of a registered BOINC application, this information being the version-number, information about program files and the name of the application. All of this is needed to make sure that the latest version of the right application and all of its program files are transfered to the WN, where execution will take place, as well as a correct wrapper script. The role of the wrapper script will be discussed later in this section.
In BOINC every result is tied to a specific user, a specific host and a specific team, so when a new result for Grid execution is created, it needs to be tied to a user, a host and a team as well as every other result. For this reason and to make it easy to see which results are being executed or were executed on the Grid, a BOINC user, a host and a team to represent the Grid is being used. This user information is contained `GRID_USER`-class.

[18] This is a very important number since it controls when the bridge considers a WorkUnit of being in danger of breaking its QoS agreement.
[19] In the main loop the bridge checks for finished WorkUnit, submits new jobs etc. When not in the main loop it sleeps.
[20] Retrying, is the RB resending a job because of a failure in the Grid system. Not to be confused with resubmission to the RB, which is not supported by the bridge.

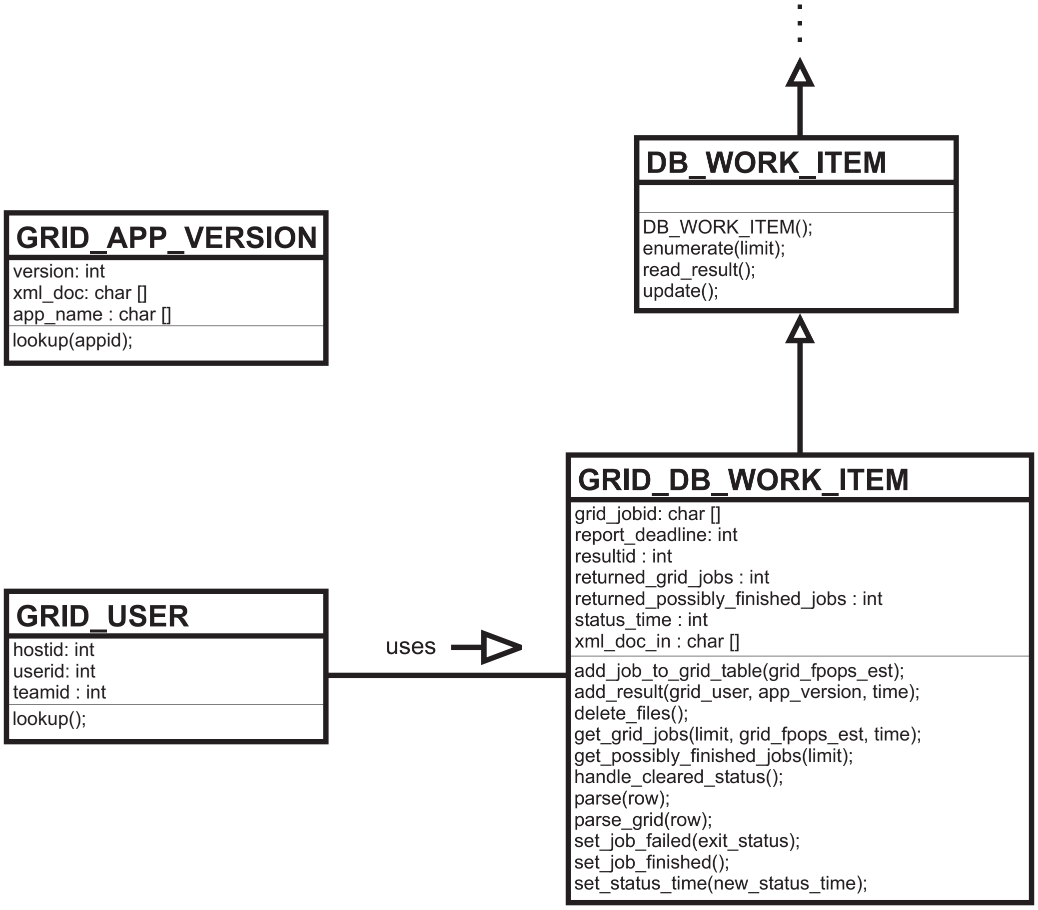


Figure 5.3: The class diagram for the bridge.

The `GRID_DB_WORK_ITEM`-class takes care of all the database operations needed by the bridge. The class represents a BOINC job on the Grid meaning it contains both the resultid representing a normal BOINC job, and the jobid representing a normal Grid job. It contains a method for getting the WorkUnits from the database that are in danger of not honouring their QoS agreement and a method for creating new results to be executed on the Grid. It also contains a method for getting jobs requiring a status check and a method for setting all the necessary fields indicating a job finish and many more. The `GRID_DB_WORK_ITEM`-class inherits from the `DB_WORK_ITEM`-class, which is part of the standard BOINC implementation. The `DB_WORK_ITEM`-class inherits from other standard BOINC classes.
To be able to submit jobs, query the status of the jobs and retrieve the output from Grid resources a proxy-certificate is needed, so naturally the bridge needs this as well. To create a proxy-certificate for the LCG-2 the user types `grid-proxy-init`, which asks for the user's password before creating the proxy certificate. A proxy certificate is only valid for a certain period of time set by the user when creating it, thereby limiting security hazards should it fall into the wrong hands. Automating the creation of proxy certificates or changing the procedure somehow, could easily open up to large security problems, so no changes to this procedure have been made. There is however a service called MyProxy, which holds a long-lived proxy and uses this to issue short-lived proxies or even renew them, if a job runs for

longer than the short-lived proxy's lifetime. This could be used to issue short-lived proxies for the BOINC jobs and would only require the administrator to renew the long-lived proxy very rarely. The downside is of course if someone breaks into the server and steals the long-lived proxy or tricks the server to issue unlicensed short-lived proxies. For reasons of safety and simplicity it was chosen not to use the MyProxy server, this means that the administrator of the BOINC server is required to continuously create a new proxy certificate before the old one expires, otherwise the bridge fails to interact with the Grid. This setup will most likely lead to a lot of administrators forgetting to renew the proxy certificate, so a simple solution is proposed, though not implemented. A small program that monitors the time left before the proxy certificate expires could be implemented. It should send off a reminder, perhaps an email to the administrator, when a user-defined minimum amount of time before expiration is left. This way no new security issues have been introduced, the administrator can choose how big a risk he wants to take when setting the expiration of the proxy certificate and how much time in advance he wants to be reminded before expiration.

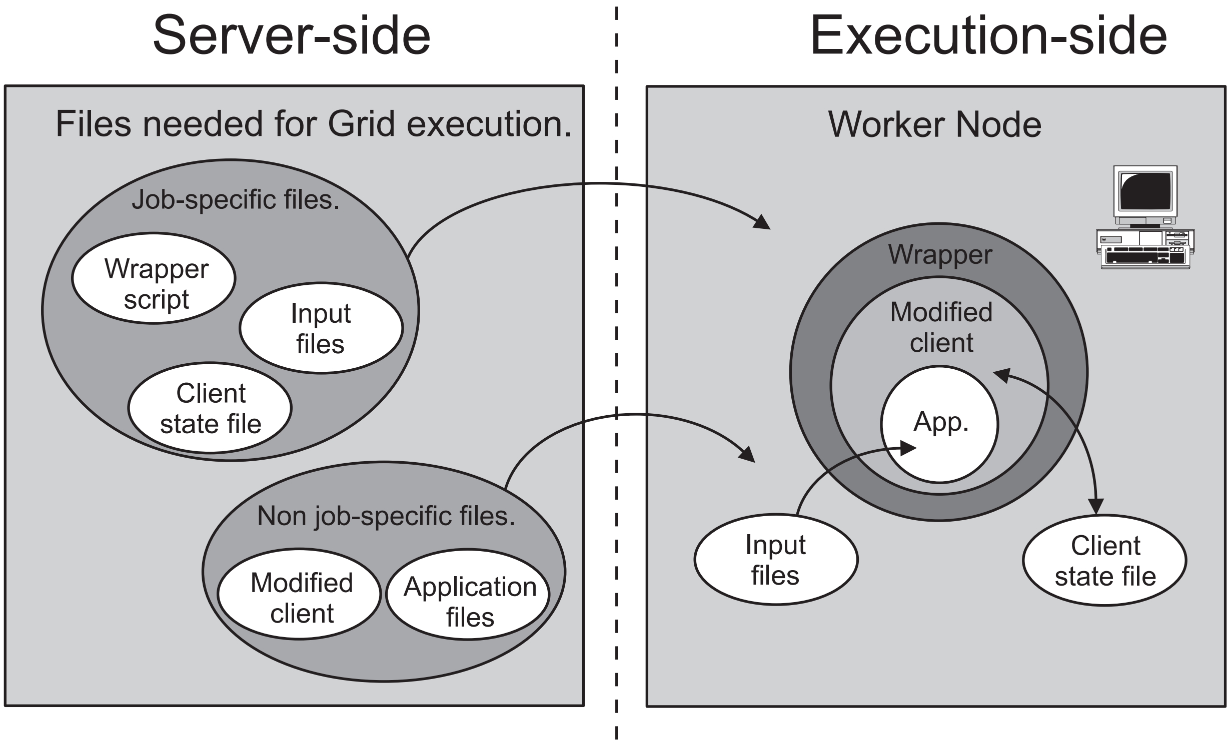


Figure 5.4: Executing a BOINC job on a Grid WN.

Figure 5.4 illustrates a BOINC job being executed on Grid resources. As mentioned earlier, a modified client is needed to run the job on the Grid, also needed are: a wrapper script, the application with all its associated files, the input files specific to this job, and a custom generated client_state file. All of this is transfered to the CE as the input sandbox. The input sandbox is the set of files that needs to be transfered from the submitting machine, in this case the BOINC server, to the computing element in order for a job to run. The wrapper script is first started on the WN, and it takes care of setting execute permissions on files needing this, setting up the correct client directory structure and executing the modified BOINC client. When the client runs, it reads the custom generated state-file, from which it learns that it has all the needed files to run a specific result, so the application is started. Once the application finishes the modified client exits as well because no more work is available. Finally the wrapper script cleans up and exits as well, and the result can be downloaded by the bridge daemon. The heart of the bridge is the main loop, which is run at user defined intervals.[21] Every time round the loop a series of steps is performed by the bridge, these steps are listed below.

- Query the database for possibly finished Grid jobs by selecting jobs with a status time of less than or equal to the present time.
  - Query the Grid for the status of each of these possibly finished jobs.
  - Get the output from the finished jobs and update the database for all of the queried jobs to reflect the finishing of the jobs or to increment the status time of the non-finished jobs.
- Query the database for WorkUnits about to break their QoS agreement.
  - Create a new result destined for Grid execution for each of the found WorkUnits.
  - Create all of the needed files to run the job on Grid resources. This includes the client state file, the wrapper script and a JDL-file describing the job to the resource broker. The last file is not strictly necessary, the same result could have been achieved by passing the description directly to the LCG-2 API. By making a file instead resubmission[22] of a failed Grid job is easier.
  - Submit the jobs to the Grid via the C++ API.
  - As a final step create an entry for each submitted job in the grid_job table of the database so a record of the submissions are kept and the status of the jobs can be queried during the first step of the main loop.

There are a few problems with stopping the daemon in the middle of a loop, namely database corruption issues. If it for instance is stopped just before the last step, there is no record of the submitted job since the database entry to the grid_job is never made. This causes the bridge to loose all track of that job i.e. not querying the status of it. It also causes it to be returned again, when the bridge queries the database for WorkUnits in danger of not honouring their QoS agreement. The really bad part comes when the bridge tries to create a new result for this WorkUnit, causing the database to fail because of clashes with unique fields in the result table. This is avoided by using a 'replace' statement instead of an 'insert' statement, causing the database to simply overwrite the existing result entry. Since the database no longer fails, it is just a matter of submitting the job again, which granted is not ideal, but then again not catastrophic. There are other problems with stopping the bridge in the middle of the loop though, one of them being stopping it after the output from a job has been retrieved, but before the database is updated to reflect the finishing of the job. When the bridge is restarted it queries the Grid again for the status of the particular job and is told that it is 'cleared', meaning output has been retrieved and the job is no longer active. During normal operation of the bridge the status of a job should never be 'cleared' so clearly it is an odd situation.

[21] Se section 5.3.7 for more information on user defined settings.

[22] Resubmission is not supported by current version of the bridge though.

It is of course is relatively easy dealt with by checking whether the output is actually on the BOINC server, and if so updating the database to reflect the finishing of the job. However it would be best if the bridge is not stopped at critical places in the loop, therefore signal handlers have been installed to catch certain signals and defer stopping the bridge to a safe spot. This is not bulletproof though as some signals cannot be caught, causing the bridge to quit immediately, however it should be robust enough to handle this.
If a canonical result is found from normal clients while the same WorkUnit is being processed on the Grid the result from the Grid is simply thrown away when the job finishes.

5.3.9 Testing

By the time we were ready to test the bridge, the LHC@home project was unfortunately no longer running. We therefore only had rather limited resources available for testing the bridge. Nevertheless we managed to stage a small test involving a single BOINC client and the test project mentioned in section 3.5.1. We submitted 30 WUs with a deadline of 2 days and we demanded 2 matching results for a quorum to be found. We left the default setting of generating 5 initial results untouched. Each job takes about 40 minutes to run on the client and is run at least twice, sometimes more because of the disconnected operation of BOINC clients. This leaves the single BOINC client just short of finishing the 30 WUs within 2 days.

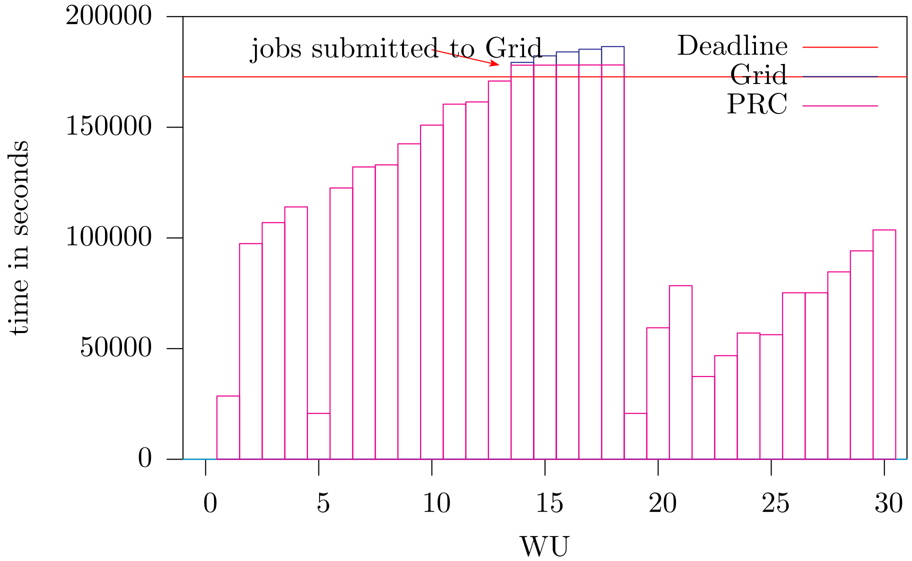


Figure 5.5: A small test of the BOINC-Grid bridge.

Figure 5.5 shows the finishing times of the jobs, whether they were submitted to Grid and their deadline. Rather embarrassingly the five WUs submitted to Grid did not make the deadline. Even though the five Grid jobs did not make the deadline, they did finish before the single BOINC client could finish either of them. The explanation as to why the jobs did not finish in time is quite embarrassing. If figure 5.5 is studied closer one discovers that the jobs were submitted to Grid after the deadline had passed. This is because the proxy certificate had expired causing the bridge to fail in submitting the jobs. The mistake was discovered two hours after the bridge first tried to submit the jobs. If the two hours are deducted from the five

WUs they would not only have been submitted before the deadline, but also quite possibly made the deadline[23]. As stated in section 5.3.8 a small program to monitor the expiration of the proxy certificate would probably be a good idea. ;-)
Although this test was successful except for the proxy certificate glitch other tests might fail since the bridge unfortunately only operates under the best effort scheme. A full scale test on a project like LHC@home would have been beneficial.

5.3.10 Security

The security from the point of view of the BOINC platform is unchanged if not improved, since some of its jobs are now running on trusted resources. Because of the bridging with the Grid, the BOINC platform is no longer the only system to consider when evaluating the security. Since the server complex now holds a (hopefully) valid proxy certificate for Grid resources, there is a chance of this falling into the hands of an intruder resulting in possibly malicious use of Grid resources. Extra care must therefore be taken to try and avoid break-ins to the server complex, although this already is a high priority to protect the many regular BOINC users. Even if the proxy certificate is kept safe, in for instance the MyProxy server, intruders could still misuse Grid resources. This is because while the long-lived proxy is safe, short-lived proxies still get issued by the MyProxy server and these proxies could be used to run malicious code by a possible intruder.
Then there is the fact that the BOINC server automatically submits jobs to the Grid, which also opens for security issues. It means that anyone with permission to submit jobs to the BOINC server automatically get permission to submit jobs to the Grid whether they have certificates for the Grid or not. This problem could be fixed by having everyone wanting to use the bridge for deadline assistance submit a proxy certificate of their own to be used to submit their jobs to the Grid. This however means that the BOINC server now has to manage many proxy certificates, and opens up for even bigger security issues. A partial fix to this problem could be to submit links to MyProxy servers instead of the actual proxy certificates, but these MyProxy servers can still be misused as described above. Besides this, the BOINC server has no way of telling who submitted a job, so information about the proxy certificates has to follow each WU.
So basically keep an even tighter security on the server complex when bridging with the Grid.

5.3.11 Limitations and Improvements

There are quite a few things that would improve the bridge in one way or another, but did not make it into the current implementation. These things, together with some other limitations in our approach, are the subject of this section.
It would probably be an improvement to change the scheduler of the BOINC server so that it does not send out results from WorkUnits already submitted to the Grid, since it would seem some what of a waste, since the WorkUnit is already being executed on a trusted resource. On the other hand, since it is being executed on the Grid, it is presumably pressed for time, running it on more computers should increase the chances of it finishing in time[24] and public resources are considered vast and inexpensive. We however still feel it would be a waste and that the public resources could be put to better use by running other WorkUnits.

[23] unless the queue length changed dramatically within those two hours
[24] Albeit only by a little since the turnaround time on BOINC clients is quite big.

The current implementation does not record the CPU time used by jobs on the Grid. Improving this by recording certain metrics, CPU time among others, on the WN, would make it very easy to implement payment for used Grid resources.
One major limitation is the size of the input sandbox on the LCG-2. This limit is set in the configuration of the resource brokers of the LCG-2, therefore out of reach for many BOINC server administrators and if the limit is hit, the jobs fail to be submitted. The effect of this limit is that all the files needed to run the BOINC job, i.e. the modified client, the application files, the input files, the client state file and the wrapper script must fit within the limit. To remedy this major limitation, static files should be stored on the Grid itself in particular the modified client and the application files, which are also usually the largest components of the BOINC job. Storing static files on the Grid has not been implemented in the current version of the bridge, so in order to make it a little less painful to live with, the modified client is stored in a zipped format on the BOINC server and unzipped once transfered to the CE. It is also recommended to zip the application files and the input files as well, however it is the responsibility of the application developer to ensure the unzipping of the files.[25]
When the bridge interacts with the Grid i.e. submitting a job, querying the status of a job or getting the output of a job, it is on a single job basis. The problem with this is the overhead of connecting to the Grid for each job, for instance when querying the status of 20 jobs a connection to the Logging and Bookkeeping service is made for the first job and when the answer comes back, a connection for the second job is made and so on. A performance improvement would be to do these interactions with the Grid in bulk, for instance querying the LB service for the status of the 20 jobs with a single connection. Fortunately the C++ API for the LCG-2 offers job submission, status querying and output retrieval methods that work on groups of jobs, so there should be no problem in implementing this improvement.
Another limitation is that we only allow WUs which have an associated application that supports Linux to run on the Grid. This is because none of the machines on LCG-2 are running Windows. It would therefore make no sense to run an application built for Windows on the Grid, as it would immediately fail. One way around this limitation is to use Wine, a windows emulator on Linux, to run the application. It is available on some LCG-2 nodes and the Windows version of the BOINC client is known to function under Wine. It is though not certain that every application made for BOINC and Windows will run without problems.

5.4 Possible Benefits For BOINC And LCG-2 From the Bridge

Being able to run regular BOINC WUs on LCG-2 resources could possibly make some jobs suited for BOINC, which were not suited previously. For example jobs which do not have a strict deadline, but where the result is wanted sooner than say 2 weeks. Being able to support such jobs makes the BOINC system more usable. The bridge could also help with the problem of some WUs finishing very slowly on normal BOINC clients. The problem was experienced on LHC@home and is described in section 3.6.3. Getting rid of the dreaded tail would definitely be of value to the physicists using SixTrack on BOINC.
Having the bridge could also benefit BOINC by making the applications more versatile. Now BOINC applications do not have to be run under the BOINC client but can be run on LCG-2. If for some reason a user wanted to run a BOINC application on LCG-2 he would not have to

[25]Even if a BOINC application is never intended to run on the Grid it is still recommended to zip the application and the input files, since many users treasure their Internet bandwidth.

remove the BOINC calls from the application or recompile it with standalone mode[26], learn how to use LCG-2 and create a job description for LCG-2, but instead just create a normal BOINC WU with a very optimistic deadline.
In an indirect way the ability of BOINC to use Grid resources could actually reduce the load on the Grid resources. It sounds contradictory, but a more reliable BOINC system is now able to handle jobs that would otherwise have gone to the Grid. A portion of these jobs end up on the Grid anyway, but some would now be handled by BOINC clients. Thus reducing the load on LCG-2. However the price of using the bridge still has to be paid, i.e applications have to be ported to BOINC and a slower turnaround time has to be expected.
If we take the bridge from LCG-2 to BOINC, introduced in the next chapter, into consideration the bridge from BOINC to LCG-2 could further benefit LCG-2 by improving the usefulness of the LCG-2 to BOINC bridge. With no bridge from BOINC to LCG-2 the other bridge would probably only be used for low priority jobs with no deadline at all. However with the BOINC to LCG-2 bridge the other bridge can safely[27] be used, since if the BOINC clients are not fast enough, the job will just go back to LCG-2. These two bridges working together could thereby reduce the load on LCG-2 further, however the fact that Grid resources are trusted and public resources are not, should not be forgotten. This fact makes the cooperation between the two bridges a bit more complicated, but if a Grid user wants to trust public resources and has a bit of extra patience running jobs via the LCG-2 to BOINC bridge makes more sense because of the BOINC to LCG-2 bridge.

5.5 Features Missing In LCG-2

The LCG-2, although a nice platform, is not perfect in respect to bridging with a PRC platform. Features that would improve on this are discussed in this section.
The most important shortcoming is the lack of QoS from LCG-2. A minimum QoS or a reservable QoS from the LCG-2 would have made it possible to supply a QoS from the combined resources of BOINC and LCG-2 as described in section 5.1. The limit on the size of the input sandbox limits the jobs that can be run without storing files on the Grid. It would be beneficial to the bridge to have this limit removed since this would allow the non-static files such as the WU input files to be unlimited in size. An unlimited input sandbox size does not make it less of a good idea to store static files on the Grid as proposed in section 5.3.11 since this reduces the size of a job, thereby reducing network transfers and speeding up job execution. It only makes it easier to submit WUs of any size, since this would otherwise need to be taken care of by storing the non-static files on the Grid. An unlimited input sandbox would on the other hand not be beneficial to LCG-2 since it could increase the demands on the system. The problem could possibly be solved by using quotas instead of a static limit for all jobs. This would allow jobs with a large input sandbox to run if the given user's quota allows for it. It would also avoid increasing the average demand on LCG-2 since in order for the quota to allow the running of jobs with a large input sandbox, either small input jobs or no jobs at all would have to be run in a given period.
To improve the efficiency of the bridge callbacks regarding the job status would have been beneficial. Polling introduces an unnecessary overhead since a lot of times there is no sta-

[26] Which mentioned previously is no longer needed with BOINC anyway.

[27] Not safely as in the public resources can be trusted but safely as in getting the jobs back within a reasonable amount of time.

tus change between polls. Being able to for instance register a program to be called when a job changes status, could remove the overhead. The program would get the job id along with the status as input parameters and then on the basis of this information, decide what to do next. This could for instance be to retrieve the output and update the BOINC database.

5.6 Summary

In this chapter we showed how Kenyon and Cheliotis proposed to turn public resources into a resource able to deliver a certain measure of quality of service using a small set of dedicated resources. We then showed how this design can be modified to supply a measure of reliability using a Grid as the dedicated resource. This method can reduce the overall cost of such an extended PRC system, because we only have to pay for the reliable resources when we actually need them. On the other hand the method is less reliable, because Grids in practice are less reliable than dedicated resources. We implemented the method by creating a bridge from BOINC to LCG-2. This bridge allows BOINC WUs to be submitted to LCG-2. It also keeps track of WUs with deadlines and decides if and when they should be submitted to LCG-2. Unfortunately the LCG-2 does not supply a QoS, so instead of being able to guarantee a certain QoS, the system still relies on best effort. However, initial tests shows that relying on the combined BOINC and Grid resources provides better reliability. The new system allows job submitters to specify a deadline on jobs which the system will try and uphold, while attempting to use as many public resources as possible. Finally we described limitations of the bridge and suggested possible improvements.

Chapter 6

The GRID-BOINC Bridge.

In this chapter we describe the difficulties in creating a bridge that allows a Grid job to be run on a PRC platform. We then describe the security issues one encounters in doing so. At last we turn our discussion to the specifics of a Grid-BOINC bridge, and describe the modifications we have made to BOINC to make it better suited to run Grid jobs.

6.1 Grid Jobs on a PRC System

Finding a defining characteristic that is common for all Grid jobs is very difficult. This is a result of the nature of the Grid. The idea is that the Grid should be a ubiquitous computing resource able to satisfy any computing need that users may have. This utopian ideal is of course not implemented by any of the Grids that are being developed today. A general characteristic of the Grids today, is that they run some form of Unix, most of them Linux. All this means that Grid jobs are very general, whereas the type of jobs that can be run under the different PRC platforms are more limited. How limited of course differs from one PRC platform to another. On one end of the spectrum we have a PRC platform like Condor which allows any job to run as long as it is compiled for the platform of the client. Condor is though so lacking in features and security that it is arguably not usable for truly public PRC projects. On the other end of the spectrum we have PRC platforms like Frontier that require that the programs you want to run must be written in Java, on the other hand this probably means that Frontier should be easy to port to any platform that supports Java, but we cannot be sure because Parabon has not divulged very much information on Frontier.

6.2 Security

As we have seen in the Grid and PRC chapters the way these systems handle security is very different. The Grid's security model is based on complete trust, at least in its ideal form[1]. Just like you would not expect that it makes a difference for your toaster whether your electricity is produced in a nuclear power plant in France or a windmill in Denmark, you would not expect the output of your program to be different depending on where it was executed on the Grid. This is diametrically opposite to how security works on a PRC platform. The end execution machines cannot be trusted on a PRC platform, whether because of hardware errors

[1] As a computational metaphor for the electricity grid

or because of malignant tampering by users is not important. The important issue is that a person submitting a job to the Grid would expect that the result, being executed on the Grid, can be trusted. The bridge, to conform to Grid semantics, would therefore have to have a way of turning the untrusted PRC machines into trusted resources. Luckily many of the PRC platforms (see chapter 2) have this feature built in. Once Grids allow or require payments, it would also make sense to allow distinction between resources. So a user could decide to use a less secure, but also less expensive PRC resource.
There is though one type of security a PRC platform can never supply. You can never trust your code to stay secret. Some PRC platforms, like Parabon, see 2.4.1, tries to solve this problem by encrypting data and obfuscating the code, but it would have to be decrypted when it is run. A determined hacker would therefore be able to gain access to the source and input data in RAM if nowhere else. There is, at present, no way around this limitation and a Grid to PRC bridge cannot be used for applications that need to stay secret. A group of hardware and software manufacturers have created a workgroup called The Trusted Computing Group. One of their aims is to solve the problem outlined above, see [36] and [37] for more information.
Most PRC platforms provide mechanisms to insure that a result can be trusted as being correct[2]. We believe that using these mechanisms, for example BOINC's multiple execution strategy, will allow you to have the same amount of trust in the result found by a PRC platform and the one returned by the Grid. An interesting area for further research, would be to study this in more detail, for example by running a program, where the outputs are known in advance, both on the Grid and on the different PRC platforms and compare the error rates.
If the Grid to PRC bridge is to be a success, the public donating their computer power has to trust the Grid. If people are to donate their compute power, they have to trust that what they are running will not harm their computer. Some PRC platforms use sandboxing to supply this security whereas others, like BOINC, have no such security at all and the users have to trust the application completely. Another problem is trusting the validity of what the user is donating his compute power for. Often users give their compute power away, because they are interested in the subject and expect the results of the research to benefit everyone. A company might use this bridge to do research on chemical weapons and if the users found out they would probably not donate their machine. It can be very hard to prove that only "good" research is using the bridge, if any entity is allowed to execute code through the bridge. One way to solve these two trust issues, would be to extend the Grid authentication mechanism to the PRC client. Each user could then decide for himself which certificates to trust and through those only execute code from entities he trusts. This is not a very feasible solution for our bridge. It would increase the needed bandwidth and it would require the clients to handle the granting and revocation of rights on their machine. We fear that this would be more complicated than what the average user is able to handle, and that it is also much more time consuming for the users than they are willing to bear. Another solution is therefore to let the bridge act as a proxy trust server. This means that all the PRC system's clients would implicitly trust whomever the bridge trusts. This off course means that whomever is running the bridge has to be very careful when deciding which Grid users to trust with access to the bridge.

[2] Correct meaning that the result is the same as if run on a trusted machine, not that the program actually does what you expect it to.

6.3 The Ideal Bridge

An ideal bridge should be completely indiscernible to the user. It would automatically be a resource candidate if it was suited to run a specific job. The idea of an ideal bridge will probably not be possible to implement. Practically all of the PRC platforms require specific information on a job, that the job has been compiled specifically for this PRC platform or that the job conforms to some standard. This means that the user has to be aware of the bridge to conform to whatever demands the specific PRC platform has to a job.
Since the ideal bridge is not possible, we have to decide what constitutes a bridge from Grid to a PRC system. We will use the following definition:
A bridge must be able to run jobs submitted through the Grid interface on the PRC system and it has to be more general than just a new submission system through Grid. This definition means that while it might be necessary to specify more metadata about the job, the application should not be recompiled to fit the PRC system. If it was recompiled it might no longer be runnable on the Grid and therefore the Grid would simply become a new submission system for the PRC system. We will use this definition to discuss a design for the bridge.

6.4 A BOINC bridge

Here we will discuss the issues specific to creating a bridge from the LCG-2 Grid to BOINC.

6.4.1 Running Generic Jobs on BOINC

The jobs that run under BOINC and under Grid are very different. A Grid job can be any binary compatible executable, whereas a BOINC job has to be compiled with the BOINC API, so the application can tell the core client when it has started, get the translated name of the files it needs and so on. This means that we could require that all Grid jobs that try to run using the GRID-BOINC bridge should conform to the BOINC standards.
Another possibility is to create a wrapper program that takes care of communicating with the core client and translating the files before running the program. It would also have to handle the different suspend and kill signals that the core client generates. The problem with this approach is that only the job itself is capable of supplying the correct information to the core client. An application run under a BOINC client is encouraged every once in a while to tell the client how many percent of the current job is done. This is then displayed to the user. A universal wrapper program will have no way of calculating this percentage correctly.
When we started this project BOINC was in no way suited for these kind of jobs. But because ClimatePrediction.net(CP.net) needed something like a wrapper, the newer version of BOINC has incorporated many features that will allow such a wrapper program to function. It will still lack some of the features such as the percentage display discussed above. The reason the CP.net wrapper works correctly, is because it knows how the different programs that runs under it work. Their wrapper can extract the needed information from the output/checkpoint files written. This is not possible for a general wrapper though.
Another important aspect of keeping your users happy is a nice screensaver. Most of the programs that run on the Grid do not display any graphics. It would not make sense when the program is very likely being executed on a machine that is not able display it. BOINC also supports that the graphics can be a completely separate program. This feature was implemented on the behalf of CP.net, a climate modelling project also using BOINC. The

bridge could therefore supply a standard graphics binary that would run if the Grid program did not supply graphics. It is not even necessary for the bridge to supply such a graphics program, because the BOINC client has a built-in screensaver that is run if the application does not supply graphics. We will therefore not design such a graphics program for the bridge. But because the users seem to put such a great weight on a nice screensaver, we will suggest that it is done by anyone who wants to deploy the bridge.

6.4.2 Security

From section 2.2.2 we know that the security in BOINC is based on trusting that the project only gives safe jobs to the client. There are two ways to deal with this as we saw in section 6.2. Using the first way, we could add a list of trusted certificates to the preferences (see section 2.2.1) a user sets. Thus the user could allow only executables from VOs he trusts to be run. As we stated before, we believe that this method will be too unpopular with some users and too difficult to administrate for other users. We will therefore use the second method, which is simply to resign a job with our own key when we receive a job from an entity we trust. This requires that the private key is present on the BOINC server, which is of course a security risk.

Another problem with BOINC's security measures is the limits on computations. These measures are in place more to protect from bugs and mistakes than from actual conscious attempts of cracking. When a job is sent to a client, it has to specify a maximum number of floating point operations it is going to perform, as well as how much disk space the output is going to take up. The BOINC client will then from the number of operations, with the help of a benchmark, estimate a time limit for the job. If this time limit is reached, the job is killed and fails. The same applies for the disk space; A user can specify that he only wants to let BOINC use a 100 MB and if this limit is reached the job is killed. The BOINC scheduler will only send a job to a client if the job says it needs less space than what is available on the client. These numbers are not easy to estimate for the bridge. It could simply use the largest possible value for the time limit, thus insuring that the job will not be killed, but also removing all the protection from run away processes and infinite loops that this feature was meant to give. This method would also present a problem if the BOINC-Grid bridge from chapter 5 is in use. That bridge would have a hard time doing deadline scheduling when the estimate for the runtime of the job is far from the real value.

The disk limit is a much harder problem, if we simply maximize the disk usage estimate to make sure that a job is allowed to finish, it will probably not be distributed to any client because their maximum disk usage limit is much smaller. We could also set the estimate to the preferences standard disk usage limit, since the average user will probably not change this setting. This then gives us the problem that jobs, which have output files larger than this standard, will fail because they reach the limit. The only reasonable way to handle this problem is to require that jobs run through the bridge, have to specify disk usage in the job description. This, however, is a departure from our design philosophy, that the user should not need to know anything, specific to BOINC, to use the bridge. It will demand that the submitter estimates how much disk space each of his jobs are going to use, which is not normally done at LCG-2. Validation is also an obstacle for such a bridge. We need to use BOINC's multiple execution strategy to supply confidence in the correctness of the result. To compare the results we need a job specific validator, but this is of course not supplied by a Grid job. We propose that a bridge could use the validator, we used at first in the LHC@home project. It insures that the results are identical. This would probably discard a lot of results that would have been good enough, but the validator has no way of knowing that.

6.5 The Prototype

We had not planned to implement a GRID-BOINC bridge, only study the possibility of such a bridge. Through our difficulties with implementing the LHC@home project, specifically the discrepancies between the documentation and the actual software, we became convinced that we had to try to write some software for the bridge to get an actual idea of the issues involved. We have not made a full implementation of the bridge. But we will describe some of the specific challenges in such an implementation.

6.5.1 Architecture of a Grid-BOINC Bridge

In chapter 4 we saw how the LCG-2 was built out of many different elements. We saw that the GRAM protocol is used to submit jobs to the local job queue of a CE and cancel them. The system behind this machine can be many different types. In LCG-2 some sites run clusters with the LSF clustering software, and others run with the public domain PBS systems. There is even a backend for the cycle-scavenging Condor system that we described in section 2.3. For the Grid to work many different batch systems and other backend architectures would have to be supported or it should at least be relatively easy to add support for an unknown architecture. It would be difficult to get new organizations to join a Grid if they had to change their entire computing infrastructure to one supported by the Grid. The GRAM protocol therefore consists of a generic part that communicates with the Grid and a JobManager interface that has to be implemented specifically for the backend architecture. Many of the most common architectures already have implementations of this API. We will describe the API in the next section(6.5.2). Based on this our approach for adding the BOINC server as a Grid resource, would therefore be to make a new JobManager that supports BOINC as the backend system. Once this new JobManager was installed on the BOINC server, we would only need to publish the presence of the BOINC server to the Information System to have a functioning Grid to BOINC bridge.

6.5.2 The GRAM Job Manager API

LCG-2 uses the GRAM Job Manager from the Globus toolkit to handle job submissions. This job manager supports many different backend systems through its scheduler interface, described in [24]. We have to write a new Perl class that implements the Globus::GRAM::JobManager interface. Specifically we have to inherit the JobManager and implement at least the following functions:

- submit(): A function that takes a Globus::GRAM::JobDescription and queues the job described therein on the backend system.
- poll(): This function should return the status of the job in the JobDescription.
- cancel(): Removes a job from the queue or stops it if it has begun execution.

There are many more functions that can be tailored specifically to the backend, but these are the bare necessities to make a working job manager. In the first case we already have a library on the BOINC servers that contain a program, namely create_work, that will submit a job to the BOINC servers. However more work is needed to decide on the options to this program and moving files to the BOINC server, this will be covered in section 6.5.3 below. The two last ones would have been fairly straightforward if BOINC had any notion of polling or canceling jobs. Unfortunately that is not the case. Because we believe these functions would be interesting in other contexts than just a Grid to BOINC bridge, we have decided to separate these functions from the bridge and make them standalone tools. They are described in the next subsections.

6.5.2.1 The Poll Tool

This tool queries the BOINC server and deducts the state of a WorkUnit and then returns the corresponding GRAM::jobstate value. There is unfortunately not a one-to-one correspondence between the states in the GRAM protocol and the states of a BOINC WorkUnit. The GRAM states were based on the fact that a job would run on a reliable cluster where it is easy to figure out the state of a job. In the BOINC system we do not know when or if a job will actually be returned. Furthermore there is no documentation of what the different GRAM::jobstate values denote, except what their names imply. We have therefore defined the meaning of the different values in a BOINC context here:

GRAM::jobstate	BOINC state
DONE	the WorkUnit has been assimilated and a canonical result found
FAILED	the WorkUnit has failed in some way. The WU's error mask is not null.
UNSUBMITTED	there are no results[3] associated with the WU
ACTIVE	the WU's results have state IN_PROGRESS, they have been dispatched to clients
PENDING	results present, none have state IN_PROGRESS, the results are queued for dispatch

The tool takes the name of the WU as input and then decides which state the WU is in by querying the BOINC database. It looks in the standard BOINC project configuration file to figure out the database settings. The poll tool has been implemented and tested. An internal test was used, WUs were artificially created to belong in one of the states above and the poll could correctly recognize each state as well as fail gracefully if the WU was not found. It is now a part of the standard BOINC server distribution.

6.5.2.2 The Delete Tool

One of the prerequisites for creating a new GRAM JobManager implementation is that it is possible to delete a job. This feature is not included in BOINC. We have therefore had to write our own utility program that allows you to cancel a WU on the BOINC server. This has shown itself to be really useful for the LHC@home project where the physicists will launch a study and then suddenly discover that their input parameters were wrong. Using the standard BOINC implementation, they would have to ask the server administrators to manually remove the WU and results from the database. This process can now be automated. The delete utility is complicated because of the distributed nature of the WU. The state of a WU can be divided into 4 distinct cases.

In the simplest case, the WU has not been sent to any BOINC clients. In this case, we can simply delete the WU and its related results from the BOINC database and delete the WU input files from the download directory.
In the second case, results have been dispatched to clients, but no results have come back. The problem is that when we have dispatched results to clients, there is no way to contact clients from the server. We will therefore have to wait until the clients contact the server to upload the results. One way to solve the problem is simply to delete the WU and results from the database. The client will then get an error message when trying to upload results and everything is fine. This works fine if all the clients are located in-house, but presents a big problem in public projects like LHC@home, because users will not understand why this happened.
It can also happen that some clients have returned results, but not enough good results have been returned to grant a quorum. In this case, as the one above, we have outstanding results and we use the same approach as above to solve this problem.
Common for both of the cases above is the question of credits. Do we grant credits? We have no way of verifying the results as we do not have a quorum. Now an honest client has spent time on calculating the WU even if we do discard them, which speaks in favour of granting credits. The reason not to give credit is to avoid cheating. Once the WorkUnit has been deleted there is no way to validate incoming results. We have chosen the second approach.
The final case is when all results have been returned and the WU validated. This case is also simple, set the WU to have been cancelled, but let the clients keep the credit.
The delete tool has also been implemented, it takes the name of a WU as input. The WU's error mask is set to the symbolic value JOB_CANCELLED and all unsent results for this WorkUnit are flagged as not needed. Results that have been allocated to a client are left to finish normally. This tool has to be used with care, because it has write access to the database. It is very important that this tool is only runnable by trusted server administrators. The delete tool is executable by the BOINC administrator and can delete any WorkUnit. It is only runnable by users outside the BOINC servers through the JobManager from section 6.5.2. The JobManager makes sure that the user's certificate is matched to the job. It is therefore not possible for a user to delete another user's jobs.

6.5.3 Job Submission

Submitting a job is the most difficult of the three operations needed for the GRAM job submission and it will highlight one of the shortcomings of the BOINC system. We believe that a BOINC bridge should allow for two ways to run a job. If the application is a standard Grid application, the bridge should be able to run the application through a wrapper program. It is then the responsibility of the wrapper to do all the calls to the BOINC client. It should also be possible to run an application that has been specifically compiled for BOINC. A program that is compiled with the BOINC API will discover at run time whether a core client is available, if not it will then run as an ordinary standalone program.[4] Such a program would then just as easily run on a Grid machine, but allow for more user friendliness than the wrapper approach when running under BOINC. This approach is therefore not just a new submission system for BOINC.
The problem with the second approach is that BOINC treats applications and data very

[4]This is only true for the newest versions of BOINC.

differently. The BOINC system is designed to run one or a few applications per project. The idea is that the applications are downloaded to all BOINC clients and the WorkUnits are then run by the appropriate application. A physicist at CERN would need a very different approach, because every analysis with for example the ATLAS software[5] is a new executable. The physicist fills in a kind of schema and then compiles the ATLAS software to get an executable that will do the needed analysis. Because a WorkUnit is not bound to a version of an application, it will always choose to run the newest version. We therefore have to create a new application in the BOINC framework for each Grid job. This can be done by appending some unique character sequence to the name of the executable and making a database table that relates the new name with the Grid job. Then the renamed executable is registered as a BOINC application and the data is added as a WorkUnit for this application. This is necessary because two different applications might have the same name. This will put a load on the server because it is now storing all the applications run through the bridge, even though they will only be used once. The BOINC server should be modified so that once all the WUs belonging to a Grid application have finished, the application is deleted to save space. It should also be possible to specify that a certain application will be used by many jobs, and therefore should be kept on the BOINC server. A job submission should therefore contain a flag that tells whether or not the job wants to use an application kept on the server. If the flag is not present, the job will be treated as a standard job.
The wrapper approach does not have this problem because all incoming applications to the GRID-BOINC bridge will be treated as input data to the wrapper. Another interesting feature with this approach, is that with an appropriate wrapper it would allow us to run the applications in a sandboxed environment.
A submission system should transfer the needed data and applications to the BOINC server complex. It is not good enough to leave them where they are, because the BOINC clients will have to fetch the data and they will not have access to machines internally on the Grid. The submission system should do the necessary file translations, create a script for the wrapper if necessary, add the application to the database if needed and add the WorkUnit afterwards. The two latter functions are already available on the BOINC server.
A Grid-BOINC bridge implementation should at least support our wrapper approach or a similar method that is able to run jobs not compiled for BOINC. We also recommend that the bridge allows jobs specifically compiled with BOINC to run without the wrapper, so the job can fully utilize the BOINC framework.

6.6 Future Areas of Interest

As we have seen above, the public nature and the structure of BOINC puts limitations on the usefulness of the GRID-BOINC bridge. The pull model of resource allocation in BOINC is though very interesting because it diverges strongly from the push model that is otherwise used by all the other back-end systems running under GRAM. It would be very interesting to further study the possibility of directly integrating the pull model into the Grid infrastructure. Instead of a Grid user directly asking each resource, if it can accommodate his request, he could put it in a global job queue and the clusters could then take the jobs out of this queue. We touched briefly upon sandboxing in section 6.5.3. This is another interesting subject of future development. At the moment the security of a client machine is completely based upon

[5] ATLAS is the main simulation software for the largest experiment at CERN, also called ATLAS.

the user trusting the project he has signed on to. If anybody can run any application through the GRID-BOINC bridge, this trust is being misused. One remedy is to make sure that any program from the bridge is run in a sandboxed environment. This could be done through the wrapper or changing the BOINC client itself to always run applications sandboxed.

6.7 Related Work

A group at the University at Maryland is developing the Lattice project, see [33], that aims to create a unified Grid architecture which includes both push and pull models for job distribution as well as the BOINC platform as a computational resource. They have a general submission system that will choose a suitable resource either the Grid, batch clusters or BOINC for a given job. Since their "bridge" does not allow general programs to run on BOINC, but only predetermined set of applications, their approach does not meet our criteria for a Grid-BOINC bridge, see section 6.3. However, they have worked on running general programs under BOINC without recompiling it with the BOINC API. Their method consists of intercepting system calls and then passing the execution to their own functions where the required BOINC functionality is done. One example could be the way BOINC handles files: Each file is given a unique physical filename and the program has to use a BOINC function to get this filename from the logical filename present in the program. The way the Lattice project handles this is by installing a shared library that will intercept all `fopen` calls. They then do the needed BOINC calls before running the real `fopen` call. How they do this is described in [34]. This method would work as a wrapper as well as sandboxing, because the functions called could be checked for permissions by the wrapper. Unfortunately they have not been able to make it work for a general binary yet.

An interesting new project starting at the Danish Grid Center aims to port coLinux[35] to BOINC. CoLinux has implemented the Linux kernel as a Windows driver. Because drivers can run in kernel mode, or Ring 0 in x86 terminology, coLinux allows a full Linux distribution to run on a Windows machine achieving almost the full performance potential.
If this project succeeds it would, coupled with our prototype work on modifying the BOINC servers to use the GRAM protocol, allow a real Grid-BOINC bridge. First of all no modification to the binaries would be necessary because coLinux would handle all the BOINC calls. Furthermore many of the security issues would be solved because the applications would now be effectively sandboxed inside coLinux. Even better the Grid Linux applications would now be able to utilize the much more numerous Windows machines as computational resources.

6.8 Summary

A bridge between PRC and Grid Computing is an interesting concept, it would increase the computing power of a Grid such as LCG-2 markedly. For a comparison LCG-2 consists of approximately 10,000 CPUs, whereas LHC@home could deliver the equivalent of 360 dedicated CPUs and it was only so small because our server capacity had limited the amount of users to 6000. We believe that it is possible to extend the size of the user base to at least the 500,000 users that SETI@home has. We therefore believe that it is possible to increase the effective computational power of a Grid, such as LCG-2, markedly by utilizing a bridge to a

PRC project. Unfortunately, we do not believe that it will be a good idea to use BOINC for creating such a bridge. As the preceding chapter has shown, it would be very time consuming to create a bridge without modifying BOINC in such a way that it becomes so general that its advantages over other PRC platforms are removed or specializing the Grid submission procedure for the bridge in such a way that you might as well submit directly to the BOINC server complex. We believe that it would probably be better and easier to turn Condor into a real PRC platform instead of creating a bridge to BOINC. Using coLinux together with BOINC also seems like a very promising line of enquiry, but since this project is only in the planning stage it is too early to tell whether it will be successful.

Chapter 7

Conclusion

In this thesis we set out to create a PRC project as the basis for creating a system that could supply a better quality of service than ordinary PRC systems. We envisioned that this could be done by mixing PRC and Grid resources.

We created a popular PRC project at CERN, which provided a considerable computational resource. During the two months the project ran, 61.6 years of effective CPU-time was donated by the public. This CPU-time was used to run a large amount of studies with the SixTrack application to ensure beam stability for the LHC. As the backbone of this project, we used the newly developed BOINC platform for public computing. We also made improvements to the BOINC system such as a FORTRAN API which is now a standard part of BOINC.

After gathering experience with the BOINC system, we went on to extend it in an effort to supply QoS from public resources. To supply QoS from public resources, our scheme requires a way to reserve a certain level of QoS from Grid resources. The Grid available to us, the LCG-2, does not offer such a level of QoS, in fact it does not support QoS at all. So by combining BOINC and LCG-2 we were not able to supply QoS from the BOINC resources. We did however make a more reliable system as initial tests show.

We also examined the possibility of joining the public resources to the pool of Grid resources. We found that doing this in a transparent way is quite difficult. One of the reasons for this is the limitations set by many PRC platforms for the applications which they run. These limitations are much stricter than the ones normally found for Grid jobs. After dealing with the general problems, we made a design of a LCG-2 to BOINC bridge. We added functionality to the BOINC server so it could support LCG-2's job submission method. We believe that the concept of a bridge from Grid to a PRC system is a good and realizable idea. However, our design showed that BOINC is not ideally suited for such a bridge. The bridge would either require a large knowledge of the BOINC system, simply making the bridge a second submission system for BOINC, or it would require that some of the BOINC client's security features were removed. The first method would not live up to our criteria for a bridge. The second would not likely gain widespread acceptance in the BOINC community.

7.1 Future Work

The following areas are interesting and merit further research:

- *The BOINC-LCG-2 bridge in a production environment*

  We were first ready to test our bridge from BOINC to LCG-2 when LHC@home had shutdown for maintenance. It would therefore be interesting to test the bridge in a production environment once LHC@home is running again. It would also be interesting to let the bridge take advantage of the improvements that have been added to BOINC during this period. For example, BOINC now supports prioritizing jobs. Letting the bridge take advantage of this would improve the scheduling of jobs.

- *QoS from Grid*

  Delivering QoS from Grid resources would greatly benefit the current user base as well as increasing the applicability of the Grid concept. Applications with real-time constraints would probably never be suited for the Grid due to the scheduling and data transfer overheads. It would also allow a PRC to Grid Computing bridge to offer HSQs from public resources, as mentioned in section 5.1. We therefore believe this is an area worthy of further research.

- *A new PRC platform*

  PRC platforms today are either aimed at intra institutional use, or they are made specifically for one type of job. A new type of PRC platform that supports very general jobs (like Condor), but is also usable in a truly public setting, would be an intriguing prospect. Such a PRC platform would also alleviate our concerns for a Grid to BOINC bridge.

Bibliography

[1] The Berkeley Open Infrastructure for Network Computing(BOINC) homepage
http://boinc.berkeley.edu/

[2] Remote Unix - Turning Idle Workstations Into Cycle Servers
Michael J. Litzkow - University of Wisconsin.

[3] Condor - A Distributed Job Scheduler
Todd Tannenbaum, Derek Wright, Karen Miller & Miron Livny - The MIT Press - 2002.

[4] The Condor Project homepage
http://www.cs.wisc.edu/condor/

[5] Condor Version 6.6.1 Manual
Condor Team, University of Wisconsin-Madison, February 12, 2004.

[6] Matchmaking: Distributed Resource Management for High Throughput Computing
Rajesh Raman, Miron Livny & Marvin Solomon - University of Wisconsin.

[7] Condor-G: A Computation Management Agent for Multi-Institutional Grids
James Frey, Todd Tannenbaum, Miron Livny, Ian Foster & Steven Tuecke.

[8] United Devices' Homepage
http://www.ud.com/

[9] Entropia's Homepage
http://www.entropia.com/

[10] Parabon's Homepage
http://www.parabon.com

[11] Distributed.Net's Homepage
http://distributed.net/

[12] MESH-Technologies' Homepage
http://www.meshtechnologies.com/

[13] LHC@home's Homepage
http://lhcathome.cern.ch/

[14] SETI@home Processing Efficiency - SETI Spy
http://www.cox-internet.com/setispy/efficiency.htm

[15] Hunting for Wasted Computing Power - New Software for Computing Networks Puts Idle PC's to Work.
Scott Fields - University of Wisconson-Madison - 1993.

[16] Towards 100,000 CPU Cycle-Scavenging by Genetic Algorithms
Al Globus - NASA Ames Research Center - September 2001.

[17] What is the Grid? A Three Point Checklist
Ian Foster - Argonne National Laboratory & University of Chicago - July 20, 2002.

[18] The Anatomy of the Grid - Enabling Scalable Virtual Organizations.
Ian Foster, Carl Kesselman & Steven Tuecke.

[19] Grid Café
http://www.gridcafe.org/

[20] The Grid: Blueprint for a New Computing Infrastructure
Ian Foster & Carl Kesselman - Morgan Kaufmann - 1999.

[21] Analysis and Provision of QoS for Distributed Grid Applications
Rashid J. Al-Ali et al.
http://www.wesc.ac.uk/resources/publications/pdf/final-jogc.pdf

[22] The Globus Alliance Homepage
http://www.globus.org/

[23] EDG VMS API Documentation
http://www.to.infn.it/grid/workload_management/apiDoc/edg-wms-api-index.html

[24] WS GRAM Developer's Guide
http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/developer/scheduler.html

[25] GridFTP - Universal Data Transfer for the Grid
The Globus Project - September 5, 2000.

[26] Getting started with the Globus Replica Catalog
http://www-fp.globus.org/datagrid/deliverables/replicaGettingStarted.pdf

[27] A Replica Management Service for High-Performance Data Grids
The Globus Data Management Group - January 29, 2001.

[28] Globus Toolkit 2.2: MDS Technology Brief
January 30, 2003.

[29] LCG-2 Middleware Overview
Simone Campana, Maarten Litmaath, Andrea Sciabà - October 1 2004.

[30] LCG-2 User Guide Manual Series
Antonio Delgado Peris et al. - September 7 2004.

[31] LCG-2 User Interface Manual Installation and Configuration
Antonio Retico, Alessandro Usai and Guillermo Diez-Andino - August 20 2004.

[32] Creating services with hard guarantees from Cycle-Harvesting systems
Chris Kenyon and Giorgos Cheliotis - IBM Research Zurich - October 30 2002.

[33] The Lattice project's Homepage
http://lattice.umiacs.umd.edu/

[34] Intercepting Arbitrary Functions on Windows, Linux and Macintosh OS X Platforms
Daniel S. Myers and Adam L. Basinet - UMIACS-TR-2004-28

[35] The coLinux Homepage
http://www.colinux.org/

[36] The TCPA FAQ
http://www.cl.cam.ac.uk/ rja14/tcpa-faq.html

[37] TCG's Homepage
https://www.trustedcomputinggroup.org/home

Appendix A

Definitions

List of definitions for different terms and acronyms used in this thesis.

ADSL	Asymmetric Digital Subscriber Line
AFS	The Andrew File System
API	Application Programming Interface
API	Application Programming Interface
Application	Represents a collection of related computation.
Application version	A specific instance of the application. Consists of programs and possibly data needed to process a result on a specific platform.
BDII	Berkeley Database Information Index
BOINC	Berkeley Open Infrastructure for Network Computing.
CAS	Community Authorization Service
CE	Computing Element
CERN	Centre Européene pour le Recherche Nucléaire
CGI	Common Gateway Interface
CGI	Common Gateway Interface
CHF	Swiss francs, the swiss unit of currency, worth about 0.7 of a Euro.
Condor	A cycle-scavenging platform developed at the University of Wisconsin-Madison.
CPSS	Cern Physics ScreenSaver
CPU	Central Processing Unit
CPU	Central Processing Unit
DCGrid	PRC platform from Entropia
DDoS	Distributed Denial of Service
DIKU	Datalogisk Institut Københavns Universitet(Department of Computer Science, University of Copenhagen, Denmark)
DMS	Data Management System
DNS	Domain Name Service
EDG	European Data Grid
EDG	European Data Grid
FIFO	First-In, First-Out
Frontier	PRC platform from Parabon

FTP	File Transfer Protocol
GASS	Global Access to Secondary Storage
GIIS	Grid Index Information Service
GLUE	Grid Laboratory Uniform Environment
GRAM	Globus Resource Allocation and Management
Grid MP	PRC Platform from United Devices
GRIS	Grid Resource Information Service
GSI	Globus Security Infrastructure
GT	The Globus Toolkit
HSQ	Hard Stochastic QoS
HTTP	HyperText Transfer Protocol. Mainly used on the world wide web to transfer web pages
IP	Information Provider
IS	Information System
LCG-2	Large hadron collider Computing Grid version 2
LCG	Large hadron collider Computing Grid
LDAP	Lightweight Directory Access Protocol
LEP	Large Electron and Positron collider
LFN	Logical File Name
LHC	Large Hadron Collider
Master URL	Address where the master page resides which is where the clients get the URL of the scheduling servers.
MDS	Monitoring and Discovery Service
MPI	Message Passing Interface
NAT	Network Address Translation
OfficeGRID	PRC from MESH-Technologies
OGSA	Open Grid Services Architecture
OS	Operating System
PFN	Physical File Name
Platform	A compilation target.
PRC	Public Resource Computing. The concept of using donated cycles from the public to do computations.
Project	A group of distributed applications, run by a single organization.
PVM	Parallel Virtual Machine
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
RB	Resource Broker
Remote Unix	The Predecessor to Condor
Result	An instance of a computation. Generated by BOINC from a WorkUnit.
RLS	Replica Location Service
RTF	Reliable File Transfer

SCP	Secure CoPy
SE	Storage Element
SETI@home	The largest PRC project in the world. Does analysis on radio signals.
SPEC	Standard Performance Evaluation Corporation
SSH	Secure SHell
SSL	Secure Socket Layer
TeraFLOPS	10^{12} FLOPS
TGV	Train Grand Vitesse
UDDI	Universal Descrition, Discovery & Integration
UI	User Interface
URL	Uniform Resource Locator
VO	Virtual Organization
WMS	Workload Management System
WN	Worker Node
WorkUnit(WU)	Describes a computation to be performed
WSDL	Web Services Description Language
XML	eXtensible Markup Language

Appendix B

LHC@home

In this appendix we will present some statistics from the LHC@home server complex and other relevant information on the LHC@home project.

B.1 Results and CPU-time

Total CPU time	212.0 years
Total Results	2524177
Total Successful results	1968084
Total CPU time(without redundant results)	61.6 years
Total CPU time(w/o redundant) contributed by Linux	3.4 years
Total WUs	615444
Total successful WUs	613099

B.2 Number of Hosts Divided by Operating System

The table below displays the number of hosts than runs a given operating system. Only hosts that have received credits are taken into consideration. So hosts that have joined but never done any calculations are not used. Neither are hosts that have only returned wrong results.

OS	nb. of hosts	total nb. of hosts
Microsoft Windows 2000	2946	
Microsoft Windows 2003	409	
Microsoft Windows 95	17	
Microsoft Windows 98	653	
Microsoft Windows Longhorn	1	
Microsoft Windows Millennium	174	
Microsoft Windows NT	175	
Microsoft Windows XP	10030	
Windows Total		14231
Linux	1210	
Linux Total		1210
OpenBSD	2	
FreeBSD	2	
Darwin	82	
Unix total		86
SunOS	8	
Other Total		8
Total		15535

B.3 Number of Users and Hosts Over Time

Graph B.1 that shows the amount of users and hosts the LHC@home project got. The reason for the steps-like appearance is that we had a limit on the amount of users. We would then make sure that the server could handle the load before we allowed more users to register.

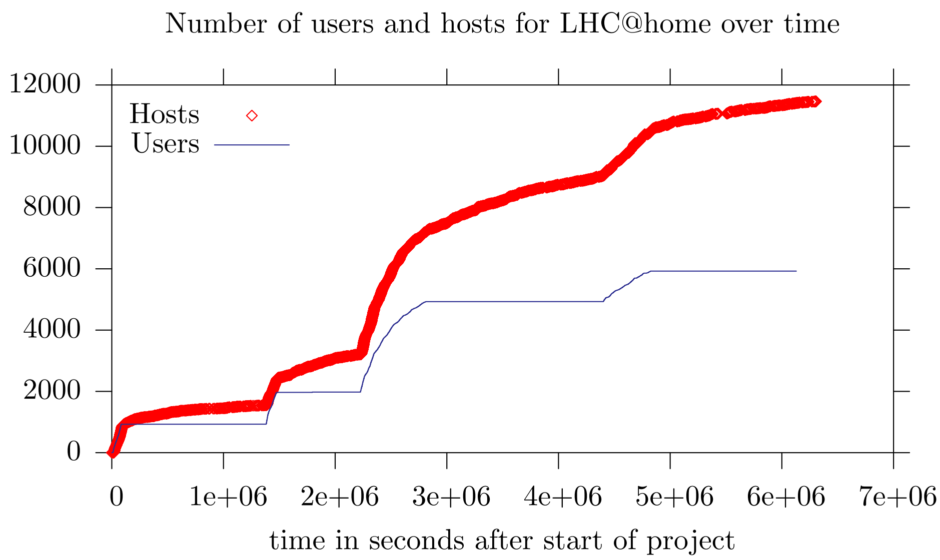


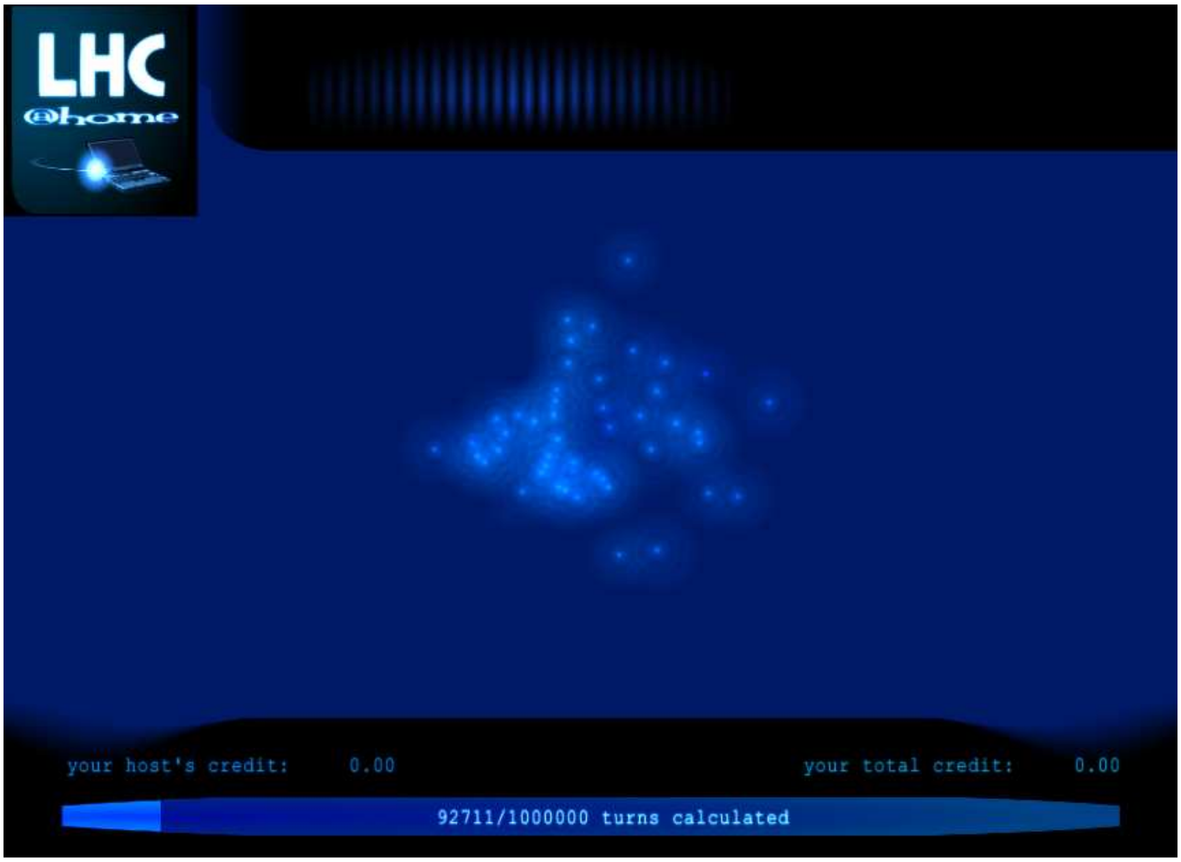
Figure B.1: The number of users and hosts donating to LHC@home

B.4 Time to Finish a Study

The table below shows the groupings of the results for the study v64lhc1000proeight.

results (Windows)	149568
results (Linux)	10522
total results	160090
failed (Windows)	5766
failed (Linux)	1075
failed total	6841
no reply (Windows)	13447
no reply (Linux)	1094
no reply total	14541

B.5 SixTrack Screenshot



Appendix C

BOINC-Grid Bridge Documentation

C.1 Installation Guide

For brave souls, who wants to attempt to install the BOINC-Grid bridge we include this installation guide. The bridge has only been tested with version 4.03 of the BOINC server complex so unexpected and unfortunate results may occur when trying to use it with other versions of the server. The installation is a lengthy process and goes as follows:

1. Install the LCG-2 User Interface system on the BOINC server, where the bridge is planned to run:
   - Install all rpms from: http://grid-deployment.web.cern.ch/grid-deployment/download/UI-rpm-LCG-2_2_0.html except for the lcfg-rpms(their names contain 'lcfg', ie. *lcfg*.rpm).
   - Install the rpm 'vdt_compile_globus_core-VDT1.1.14-4.i386.rpm' containing a libtool compatible with LCG-2. The package can be found at /afs/cern.ch/project/gd/RpmDir/vdt/globus/ but probably also other places for people without access to CERN's AFS filesystem.
   - Configure the User Interface on the BOINC server according to chapter 4 and 7 of [31] found at http://grid- deployment.web.cern.ch/grid-deployment/gis/release-docs/LCG-2_2_0/UI/UI.pdf.
   - Obtain and install a certificate for LCG-2 as describe in [30] found at https://edms.cern.ch/file/454439//LCG-2-UserGuide.pdf.
   - Test the UI by running a very simple grid job, possibly a job returning the host name of the WN. If it does not work contact people from the LCG-2.
2. Configure the server environment:
   - If not already present install GCC 3.2.3.
   - Run `export LD_LIBRARY_PATH=/!GCC 3.2.3 install-dir!/lib:$LD_LIBRARY_PATH` replacing `!GCC 3.2.2 install-dir` with the local directory containing GCC 3.2.3.

- Run `export PATH=/!GCC 3.3.2 install-dir!/bin:$PATH` also replacing `!GCC 3.2.3 install-dir!` with the appropriate directory.
- Run `export X509_USER_PROXY=/!proxy-dir!/!proxy-file!` replacing `!proxy-dir!` with the directory, where you prefer to keep the LCG-2 proxy-certificate and replacing `!proxy-file!` with the filename of the file, where you prefer to store the proxy-certificate.
- Possibly add the last three commands to the startup script of the user supposed to run the bridge, i.e. probably the .bashrc script.

3. Install and make the bridge:

   - Download the bridge from:
     http://www.fatbat.dk/thesis/bridge.tar.gz.
   - Make backup copies of the following files, possibly just renaming them:
     - !boinc-dir!/client/client_state.C
     - !boinc-dir!/client/Makefile
     - !boinc-dir!/db/boinc_db.C
     - !boinc-dir!/db/boinc_db.h
     - !boinc-dir!/sched/sched_config.h
     - !boinc-dir!/sched/sched_config.C
     - !boinc-dir!/tools/create_work.C

     , where !boinc-dir! denotes the directory containing the BOINC source code.
     This is very IMPORTANT since the bridge contains its own version of these files, which overwrites the standard BOINC files.
   - Copy the bridge.tar.gz file to the !boinc-dir! or move it there.
   - Run `tar -xzf bridge.tar.gz` from !boinc-dir! to extract the source code of the bridge.
   - Create a directory called 'grid' in the project directory, i.e the directory containing the config.xml for your project.
   - Create another directory called 'bin' in the 'grid' directory you just created.
   - Either copy the modified client 'boinc_client.gz' supplied with the bridge to the previously created 'bin' directory or run make in the !boinc-dir!/client/ directory to make your own. If you choose the last option, which is not recommended, you have to gzip the newly made client, copy or move it to the 'bin' directory and you must name it 'boinc_client.gz'.
   - Adapt the file 'Makebridge' in the !boinc-dir!/sched/ directory to fit your environment including changing the line 'cp bridge /lhcathome/projects/lhcathome/bin/' to point to the 'bin' directory of your project, i.e. the directory containing the projects transitioner, validator, assimilator etc. This directory it not to be confused with the previously mentioned 'bin' directory under the 'grid' directory.
   - Run the script 'buildbridge' or run `make -f Makebridge` both from the !boinc-dir!/sched/ directory to build the bridge and copy it to the project's 'bin' directory.
   - Run `make` from the 'tools' directory under !boinc-dir! to make the bridge-enabled version of 'create_work'.

- Copy or move the newly made 'create_work' to your project directory replacing the non-bridge-enabled version. For safety reasons you could make a backup of the old 'create_work'.

4. Extend the database to support the bridge:

   - Run `mysql !project-database! < !boinc-dir!/sched/bridge_schema.sql` replacing `!project-database!` with the name of the database holding your projects data and `!boinc-dir!` with the name of the directory containing the BOINC source code.

5. Enable and configure the bridge:

   - Add `<daemon> <cmd>bridge -d !logging level!</cmd> </daemon>` to the deamons section in the config.xml file of the BOINC project replacing `!logging level!` with either 1,2 or 3 for logging of critical events only, normal logging, or debug logging respectively.
   - Configure the bridge according to 5.3.7 by setting the following attributes under the config section of the config.xml file:

```
<VO>
    !VO_name!
</VO>
<grid_fpops>
    !estimated FLOPS on the average grid machine!
</grid_fpops>
<NS_hostname>
    !network server host name!
</NS_hostname>
<NS_port>
    !network server port number! - only needs to be set if default
    (7772) is not right.
</NS_port>
<LB_hostname>
    !Logging and Bookkeeping service server name!
    </LB_hostname>
<LB_port>
    !LB server port number! - only needs to be set if default (9000)
    is not right.
</LB_port>
<bridge_loop_interval>
    !loop interval!
</bridge_loop_interval>
<grid_retries>
    !number of retries on the grid!
```

```
</grid_retries>
```

If all goes well the bridge should be installed by now. If all does not go well send an email to 'jakob@stonefire.dk' and hopefully we will be able to fix it.

C.2 User Manual

Using the bridge is quite simple, however for it to work a valid proxy-certificate for LCG-2 must be available. This is created by running the command `grid-proxy-init -valid H:M` replacing `H:M` with the hours and minutes of wanted validity respectively. Since the proxy-certificate expires it must be periodically renewed for the bridge to function.
To create QoS-enabled WorkUnits a switch must be added to the commandline when calling the bridge-enabled 'create_work' or if a project-specific work-generator program is used it must follow the implementation of the bridge-enabled 'create_work'. The switch is `-finish_before !deadline!` where `!deadline!` is the desired deadline of the WU in UNIX-time.
The server administrator is encouraged to check the log file located in the log directory (!projectdir!/log_!servername!/bridge.log) periodically and as a tool for troubleshooting.

Appendix D

Sourcecode

All sourcecode for the thesis, along with explanations, can be found at:

`http://www.fatbat.dk/thesis/`