AstroGrid-D

Grid User Guide

Guide for porting User Applications to the Grid[1]

Authors:
AEI: A. Beck-Ratzka, T. Radke
AIP: S. Braune, H. Enke, I. Nickelt, S. White
ARI: R. Spurzem
MPE: A. Carlson

[1]This work is part of the AstroGrid-D project and D-Grid. The project is funded by the German Federal Ministry of Education and Research (BMBF).

Change History

Version	Date	Name	Brief summary
0.1.0	18 April 2008	Steve White	Creation, initial content (on dynamo)
0.2.0	6 June 2008	Steve White	Added section on batch jobs, formatting
0.3.0	12 June 2008	Alexander Beck-Ratzka	Added word about GAT in advanced solutions
0.4.0	17 June 2008	Art Carlson	Reorganised; added content on Clusterfinder
0.4.1	29 June 2008	Art Carlson	Finished first draft of Clusterfinder section
0.4.2	30 June 2008	Art Carlson	General editing, organisation, comments
0.5.0	11 July 2008	Alexander Beck-Ratzka	Added use case description GEO600, edited GAT
0.5.1	15 July 2008	Steve White	Spelling, capitalisation, etc
0.6.0	16 July 2008	Art Carlson	rewrote intro and architecture, general editing
0.6.1	16 July 2008	Steve White	Reorganised according to general job types
0.6.2	23 July 2008	Steve White	More text in job submission, added GridWay
0.6.3	24 July 2008	Alexander Beck-Ratzka	More text in GEO600 (overview task farming) and in GAT
0.7.0	28 July 2008	Rainer Spurzem	Edit 2.1 and 2.2, GridWay etc. / Added NBODY6++/GridWay Sect. to be continued
0.8.0	08 Aug 2008	Steve White	Re-structuring, more references, some rewording
0.8.1	12 Aug 2008	Thomas Radke	Added section on AstroGrid-D Portal
0.8.2	13 Aug 2008	Thomas Radke	Finalised GEO600 use case description
0.8.3	14 Aug 2008	Thomas Radke	Finalised AstroGrid-D Portal section
1.0	10-10-2008	Harry Enke	Fixme's and editorial work

Contents

1 Introduction

AstroGrid-D provides a hardware and software infrastructure to enable the German astronomy community to effectively utilise the potential of grid computing. Typically, an astronomer will have developed an application that produces correct results, but daily work will be restricted by the long turn-around time of running the program on a single computer. Porting the application to the grid results in faster turn-around time, in addition to the other advantages of grid computing.

This document aims to reduce the investment the astronomer must make to start benefiting from the grid. Its first part deals with some major types of astronomical applications that benefit from running on the grid, and provides examples of AstroGrid-D use cases. Its second part describes advanced methods of submitting jobs to the grid that have been used and developed by AstroGrid-D.

It is assumed that the reader has already become a grid user, that is, they have successfully

- obtained a grid certificate (see *AstroGrid-D Membership* [11])
- registered in the AstroGrid-D virtual organisation (see *AstroGrid-D VO registration* [16])
- and learned how to access grid functionality [10],

i.e. they can log into the grid.

Other documents that might be useful to consult before or while using this guide are found at *Guides for installing and using Globus Grid* [12].

1.1 Grid authentication

User authentication on the grid is different from conventional Internet authorisation methods, in that it is based on a hierarchy of trust, rather than direct relationships between users and computational resources.

The grid middleware chosen by Astrogrid-D[2] to supply this authorisation is Globus[20]. The Globus Toolkit [21] is an open source software toolkit used for building grid systems and applications provided by the Globus Alliance.

1.2 Grid hosts

A common way to work on the grid is to log in interactively as a grid user from a host (typically, a computer) that has grid client software installed, and from there to submit jobs to or transfer data with grid resources.

This document will refer to the machine from which a job is submitted as the "**submission host**", which must have grid client software installed (see [1]) but need not be a grid resource itself. From the submission host, a job is submitted to "**target resource**s". In some applications there may in addition be a distinct "**data resource**" that participates in file transfers.

[2]Within D-Grid other middlewares are also supported, but AstroGrid-D has decided to focus on the world-wide open source standard Globus.

In case of GridWay job submission, there will be another resource in the job submission process, which is the "**gateway resource**", acting as an intermediary between the submission host and the ultimate target resources.

1.3 Globus grid transactions

The fundamental transactions from the user's point of view are interactive login, file transfers and submissions of `jobs`, which are programs whose execution and completion are handled on a grid resource.

The Globus Toolkit (GTK) provides the following functions for these transactions.

- **interactive login, command execution**
  `gsissh`: an implementation of OpenSSH with grid authorisation.
- **file transfer** (to and from grid resources):
  `globus-url-copy`: uses any of various protocols including `gsisftp`, `http`, `ftp` and `file`. Between grid resources, can do parallel "streams" for high performance.
  `gsiscp`: a simple implementation of `scp` with grid authorisation.
- **job submission**:
  `globusrun-ws`[3]: combines common activities such as file stage-in and stage-out, and cleanup with remote running of the program executable, as well as providing functions such as monitoring and accounting.

Globus jobs can be monitored or cancelled using the job's EPR (*End Point Reference*).

After a successful job submission with `globusrun-ws`, an EPR file is printed to standard output. This can be redirected to a file by using the "-o" option on the submit line.

To cancel or monitor a job submitted with `globusrun-ws`, use the command again, with the job's EPR in *job*.epr, as below:

- **job status**

```
globusrun-ws -status -j job.epr
```

- **job cancellation**

```
globusrun-ws -kill -j job.epr
```

1.4 Advanced grid transactions

AstroGrid-D uses and develops several advanced mechanisms designed to streamline grid use. Each of these is meant to remove a level of complexity from the process of job submission.

Part II of this document details some of the following:

[3]The "ws" stands for the "web services" version[22] of Globus 4.0*x*. Earlier releases have analogous commands like `globus-job-run`, which are supported for compatability but not recommended for everyday use.

- **GridWay metascheduler**
  Jobs submitted via a GridWay metascheduler are automatically submitted to the next available target host that matches the job's specified requirements, thus freeing the user from the work of finding appropriate hosts themselves. See section 6.
- **GAT (Grid Application Toolkit)**
  a wrapper providing access to other grid middleware in addition to Globus. See section 7.
- **AstroGrid-D Portal**
  Submission of grid jobs through a web page. See section 8.
- **ADM (AstroGrid-D Data Management)**
  Allows access to data (within jobs or interactively) through a virtual file system on the grid, thus freeing the user from the concern of which grid resource actually holds a certain file. See [4].
- **Grid timeline**
  Real-time Web-based monitoring of multiple jobs. See [3].
- **AstroGrid-D WebMDS**
  Live list of AstroGrid-D grid resources. See [17].

1.5 Grid application types

There is a wide variety of arrangements of computational applications on the grid, depending on the application's use of resources involved. Part I of this document goes into detail about several major types of applications, and how a user can profit from experiences of AstroGrid-D with them.

We will discuss first submission of a simple single-process job to a single grid resource.

For bigger jobs, users will want to run multiple processes possibly on multiple grid resources, which themselves may be single computers or computer clusters. Also, it is often convenient to keep data stored in particular locations on the grid.

The multiple-process job types have been divided into three broad categories:

- cluster batch jobs

  Typically one process starts on each of a set of compute cores of a single cluster, to work on the same input data.

  Intermediate information is communicated among running processes so that they behave together as a single program to process the data. For synchronization of this communication, a single real cluster is usually required; such a job is therefore submitted to the single cluster resource on the grid.

- data-parallel jobs

  Input data is divided into parts, and each of a set of processes works independently on an assigned part of the data.

  These processes can operate on remote grid resources. Data storage and transfer issues are important here.

- task farming

  How the processes get data, and what they do with it, is the business of some external entity. Only a lot of processes need to be started.

Of course, these categories are not exhaustive, and there is a lot of overlap between the types listed.

Part I

Types of grid applications

2 Simple applications

By a simple grid job, we mean a job that runs as a system process on a single resource on the grid. It may require input data and output results to be copied, and some housekeeping to be performed with each run.

The basic Globus command used to submit grid jobs is `globusrun-ws`, which will take as input a *job description* that specifies the files to be copied to each resource, and how to start the executable program on the target resource.

One could run such simple jobs by copying the required files by hand and using `gsissh` to run the executable, but doing this is messy and limited in practice, and one loses other advantages of submission `globusrun-ws`, such as monitoring and accounting of jobs.

In practice, deployment issues come into play, such as the building binaries for specific architectures, and the presence of special libraries and services on specific systems. At this time, these issues must be dealt with case-by-case.

2.1 Basic procedures

Here we will describe a fairly general procedure to accomplish the simplest practical grid job submissions, consisting of an independent program to be run on a single grid resource.

A) Preparations for deployment

On a local host (of same architecture)

1. Set up a chosen directory and file name structure, including input files, ensure program runs properly on local resource.
2. Prepare a build system for the executable, or a statically linked binary
3. Pack input into a tar file, since `globusrun-ws` does not support directory creation at stage in

B) Resource selection

Discover what grid resources are available, and choose among them for target resources.

1. Look up the machines in the AstroGrid-D WebMDS page [17] or call `update_machinefile.sh`
2. Select suitable resources and create a "machines" file from that list

C) Submission `./submit.sh [`*`machines-file`*`]`

D) Job monitoring

1. Use `globusrun-ws -status` *epr*
2. If AstroGrid-D monitoring is installed on the resource, go to `http://is.astrogrid-d.org` for a web interface

E) Output retrieval

The data is transferred during stage-out into an specified directory (or tar file) on the submission host. By default, the directory on the remote compute resource will be deleted.

2.2 Example: Dynamo

We will use the *Dynamo* demo (see [13]) as an example of simple job submission. This demo is based on a real scientific application "dynamo" by Detlef Elstner. The dynamo application solves of the induction equation with turbulent electromotive force (alpha tensor) for modelling of turbulent dynamos in planets stars and galaxies.

The submission program of the Dynamo demo is contained in the two files

`submit.sh`	a `bash` shell script; see appendix A
`jdd.template`	a "Job Description Document" file (XML); see appendix B

In the following, we lead the reader through these two files, explaining them and how they might be altered for other applications.

2.2.1 job description file

Most of the job submission procedures require a job description. Such a description depends greatly on the specific application.

The job description document (JDD) of the "Dynamo" demo can easily be adapted for use in other simple types of grid computing applications.

The JDD file doesn't contain logic, but rather a description of which files to upload and download to the compute resources, and what to execute there.

The JDD language is a part of the Globus WS-GRAM services (see [23]). It has simple XML-like syntax. The main reference for the JDD language is the JDD Specification [24],

The example in the Dynamo demo additionally contains shell environment variables, which are replaced by `eval 'cat ....` This feature of the demo may be useful in other cases, but is not generally needed.

Note especially the section

```
<executable>
```

In the dynamo demo, an important innovative use has been made of the executable. It executes a small script (in the `<arguments>` subsection) with a shell acting as the `<executable>`. (Special characters which have special meening in XML, like e.g. input and output redirection, have to be quoted.)

You might want to change this, for example, to compile source code on the compute resource.

The section also specifies <stdout> and <stderr> streams, which may be set to files.

The first section

```
<fileStageIn><sourceUrl>
```

specifies which "Source" files to upload to the compute resource, here specified as a URI. In the demo, this is just the tar file of inputs created by the `submit.sh` script. You may want to use it for other purposes. In many simple cases, it is possible to stage in source code, which is to be compiled by the script on the compute resource.

The next section

```
<destinationUrl>
```

specifies the "Target" files (or directory trees) that constitute output to be saved. In the demo there is just one such tree; for each separate file or directory tree you will need a separate <destinationUrl>. Note also that if it is a directory tree, the URI section must end in a slash ("/").

2.2.2 multiple simple submissions

The main flow of control in the `submit.sh` script consists of a loop over a set of input files, running a process on a different compute resource for each input.

The particular input files are specific to Dynamo, but such a loop is common to many applications. You will want to change the names of the input files, at least: See the line

```
INPUTDIR=...
```

This specifies the directories that are copied for stage-in.

Also note the line starting with

```
mkdir -m 777 ...
```

The necessity to specify permissions was due to the fact that the processes run in a grid account, but the job submission may be from a non-grid user account. If you submit everything from a grid account, the setting of permissions may be unnecessary.

The files to be uploaded ("staged in") to the compute resource are all put in a temporary directory tree, which is tarred. This is an easy way to create a consistent directory structure on the compute resources.

The complicated line beginning with

```
eval 'cat ...
```

processes the input file `jdd.template`, expanding any environment variables it contains, and outputs an JDD file, which is used by Globus for job submission.

The script then does the actual submission with the call to

```
globusrun-ws
```

using the JDD file just created. It writes EPR (End Point Reference) files in the local directory

`~/.epr/`, which may be used for monitoring the state of the job.

Finally, if IDL is installed, it is used for graphical visualisation of the output. You could replace this by whatever post-processing you like.

3 Cluster applications

Many scientific applications involve algorithms that allow computation to be spread across multiple processes each running on its own CPU, with intermediate information being communicated between processes. In effect these multiple processes act on the initial data as a single large program. Computer clusters are ideal computing resources for such applications.

The grid provides researchers with many cluster resources. Each resource will typically have its own local resource management system (LRMS, sometimes called "batch system"). There are numerous different LRMSs, including `PBS`, `LSF`, `Condor`, `SGE`, `Loadleveler`. To facilitate the uniform submission of parallel jobs to resources, grid tools have been developed, that do away with much of the direct interactions with the LRMSs of different resources.

The most common means of communicating intermediate information between processes is the Message Passing Interface (MPI), which consists of libraries and special build mechanisms on the resource. This part remains rather site-specific, and research and testing is usually required to properly prepare an MPI binary.

The primary issue is that of communication of certain information with the cluster's LMRS. At least, the the desired number of parallel processes job are specified — this information usually influences a choice of cluster-dependent job queue. Also, the building of the binary for the cluster's particular libraries may require reading the user documentation for the cluster.

Note that in the case of cluster resources, a job is submitted to the cluster's head node, which is the grid resource, but the computational work of the job is performed by the cluster's compute nodes, which are typically *not* grid resources, having little or no wide-area network connectivity.

3.1 Preparation

Here we outline the typical process.

Log in to the target resource with `gsissh`, research and configure the environment (especially, determine how to build an MPI executable, if that is needed, and determine if the job must be run in a special queue of the LMRS), and optionally build the desired binary.

Do a test submission of the job directly to the cluster's LRMS.

Create a Job Description Document (JDD) file, that specifies

- path to parallel executable
- executable arguments
- execution directory

- number of parallel processes
- whether the job is to run in an MPI environment
- stage-in, stage-out (possibly with remote data servers)
- cleanup of files

A trivial example of a job description document might look like this:

```
<job>
        <executable>./mpihello</executable>
        <directory>$GLOBUS_USER_HOME/mpitest</directory>
        <stdout>$GLOBUS_USER_HOME/mpitest/grid_mpi.stdout</stdout>
        <stderr>$GLOBUS_USER_HOME/mpitest/grid_mpi.stderr</stderr>
        <count>8</count> <!--number of parallel process to start -->
        <jobType>mpi</jobType> <!-- indicates an MPI job -->
        <queue>queue-name</queue> <!--cluster-dependent -->
</job>
```

Note that the document has an XML-like syntax, and uses HTML/XML comments (`<!-- ... -->`). Also note the built-in directory path `GLOBUS_USER_HOME`.

3.2 Job submission from the grid

The command line for job submission to a cluster LRMS differs from that for single-process job submission in that the LRMS of the cluster must be specified in the "Factory Type" (`Ft`) flag of the `globusrun-ws` command:

`globusrun-ws -b -submit -Ft` *`batch`* `-F` *`rsrc-hostname`* `-f` *`jdd-file.xml`*

where *`batch`* indicates the type of LRMS on the cluster, one of `PBS`, `LSF`, `Condor`, `SGE`, `Loadleveler`.

3.3 Example: HaloFinder AHF

A practical example of submission of cluster MPI jobs over the grid is given by the AstroGrid-D use case *HaloFinder AHF*: see [13].

The implementation of this use case are complicated by several factors, including installation at grid resource sites, and some issues with Globus software. The problems are not unusual however. These issues and other details of practical grid job submission are discussed in [2].

4 Data parallel applications

Data-parallel applications achieve rapid processing by simply dividing the input data among multiple independent processors. The main issues in such applications are then how to divide up the data, how to deliver it to each of the processors, and how to retrieve the results.

In this context it is convenient to speak in terms of *workflows*, which consist of a set of jobs with a well-defined logical control flow structure.

4.1 Example: ClusterFinder

The ClusterFinder program analyses optical and x-ray archives to identify and characterise galaxy clusters. Like many astronomical applications, the data it needs as input can be obtained by public access over the Internet, and the analysis, while covering a large fraction of the sky, only requires data from a small region at any one time. These features allow ClusterFinder to make good use of the grid. First, the calculations can be easily parallelised by assigning a different patch of sky to each target resiyrce. Second, requirements on data transfer and storage can be minimised by arranging for each resource to access the subset of data it needs directly from the archive.

ClusterFinder as a grid application has a hierarchical structure.

- The core is the Fortran 90 program, controlled by a parameter file, which points to the files on the local system to be used for input and output.
- The next level is the `makefile` that controls the workflow on a single host, including creation of the input data files by access to public databases and compilation and execution of the Fortran program.
- The next higher level consists of the mechanisms to submit a job from one host to another through the grid, including staging and monitoring. Deployment of ClusterFinder on a grid resource is also controlled at this level, with the workflow on the remote resource controlled by a separate script, rather than a `makefile`.
- The highest level is the logistics required to submit grid jobs in a coherent way to answer a scientific question. ClusterFinder uses a PostgreSQL database in this process, which involves defining parallel jobs to perform a calculation on a large region of sky, submitting these jobs to appropriate grid resources, and collecting the results in one place.

4.2 Deployment – using Subversion

In order to execute a job on a grid resource, the application must be deployed there. A straightforward way to do this is to transfer all necessary files, possibly pre-compiled and possible packed in a tar file that expands into a directory structure, just before execution. This might even appear essential, since the grid does not guarantee that the state of the account will be preserved from one session to the next.

ClusterFinder has chosen instead a method that minimises data transfer. The files needed by the application (source code, scripts, `makefile`, some data files) are maintained in a Subversion repository. Once the files exist on a resource, including directory structure, version control information and the password for the repository, any changes that have been made in the reference version can be incrementally obtained with a simple update command. This assumes, naturally, that Subversion is installed on the target resource and that the repository is reliably accessible. This is almost always the case for AstroGrid-D resources. The initial installation can also be easily carried out by submitting a Subversion `checkout` command through the grid. Normally the repository will be

password protected, so the directory containing the Subversion password (`$HOME/.subversion`) should also be copied to the target resource.

4.3 Local workflow – using `make`

Once the necessary files are available on a resource, various operations can be carried out. For ClusterFinder, these operations are much more complex than simply calling an executable. They include compilation of source code, access to public databases, and creation of directory structures and files, and each of these is composed of several sub-operations which must be performed in a certain order - or not at all, if the product already exists. To define and control these workflow operations, ClusterFinder uses a `makefile`. Make is almost always available on AstroGrid-D resources, is widely used and accepted in the world of scientific computing, and provides features comparable to other workflow systems (although arguably with a less convenient interface).

The first step in the workflow is compilation and linking of source code to create an executable. This step is well-supported by the capabilities of `make`, including provisions to automatically capture the dependencies of one program module on another. This step also creates a subdirectory to receive the compilation products.

A potential problem is establishing the required environment. Fortunately, ClusterFinder has very few dependencies. For features that are almost always present in a well-defined location (e.g., the shell `/bin/bash`, or the function `bc`), it is usually sufficient to log the occasional failures and restart the job on another resource. To avoid future failures, either the system administrator should be asked to install the feature, or that resource should be stricken from the list of potential candidates. For less common or less uniform features, more sophisticated methods are helpful. For ClusterFinder the Fortran 90 compiler is such a case. Most often `gfortran` is present, but sometimes it is `g95` or `ifort` or `pgf95` or does not exist at all. ClusterFinder deals with this diversity by adding an if-then-else block near the top of the `makefile` to define the variable `FC`:

```
ifeq ($(FC),f77)
  ifneq ($(shell which gfortran 2>/dev/null),"") #found a gfortran
    FC = gfortran
    F_PORTABILITY_FLAGS = -g -DPLANCK_GFORTRAN
  else
    ifneq ($(shell which g95 &>/dev/null),"") #found a g95
      FC = g95
      F_PORTABILITY_FLAGS = -DPLANCK_GFORTRAN
    else
      ...
```

Based on which Fortran 90 compiler can be found in the path, this block sets the `make` variable `FC` and also a flag required for one of the IO modules. This method is capable of providing powerful control of the environment, but would become unwieldy if there are many environmental dependencies. In the long term, a solution based on environment modules (which in the initial stage can be implemented even without cooperation from the system administrators) would be better.

Further steps require setting up a directory structure for the data files, the product(s) of a calculation, and some intermediate files. It turns out that this is an important design choice in grid

computing.

The problem is that, given the parallel nature of grid computing, there is no guarantee that only one job at a time will be executing on a given grid resource. In fact, there are scenarios which specifically rely on simultaneous execution. This implies a risk that input or output files from one job will be overwritten by another job.

One solution sometimes used is to create a new and totally self-sufficient subdirectory (a "sandbox") to contain all files related to one job. For ClusterFinder, this could be relatively inefficient, because compiled executables can generally be used without problems by several jobs at once, and because the input data could be used by more than one job (for example, when making a parameter scan on a small patch of sky). This suggests a multi-tier structure: a top level containing executables and templates used by all jobs, a patch level containing data and products related to patches of sky with different coordinates, and a run level containing products related to runs with different parameters on a given patch.

In ClusterFinder each job is controlled by a parameter file, which is given a unique name from which the patch number, the run number, and the catalog (RASS or SDSS) can be derived, so that one job is able to share with other jobs where possible but can be insulated from other jobs where necessary.

The next step is obtaining the data required for a particular patch of sky from public databases. ClusterFinder performs this step with grid technology, namely by making a call to the databases through an OGSA-DAI server [30]. OGSA-DAI access to public databases is not always available, so another application will usually have to either set up OGSA-DAI access or implement other procedures. An intermediate step in this process is creating an OGSA-DAI perform document (in XML format) by inserting coordinates parsed from the parameter file into a template document.

The final steps in the workflow involve calling the ClusterFinder executable to produce likelihood maps, but these targets may also be diverse, for example, a map may be produced from RASS data or from SDSS, the maps may be summed and/or a threshold applied, a graphical representation may be visualised, or a regression test may be called. A special target, "send-map", which does not strictly belong to the local workflow, will be discussed in the following section.

4.4 Remote submission – using gsissh

The mechanisms described for deployment and local workflow can be used for grid computations in the sense that they allow rapid, simple, and reliable use of ClusterFinder once the user has logged in to a grid resource, usually using `gsissh`. The potential of the grid is utilised more fully if the jobs can be submitted from a single grid resource in batch mode to many grid resources.

This is in principle a simple thing, since any command can be submitted with grid credentials using either `gsissh` or `globusrun-ws`. The latter command has additional functionality, such as file staging, monitoring, and the option of submitting through a broker, but often a high latency. A simple request that takes half a second with `gsissh` might take several seconds with `globusrun-ws`.

The most common function needed in addition to command submission is file staging and/or streaming. For ClusterFinder, the most important files to be transferred are the parameter file that controls the calculation, `stdout` and `stderr` and/or a log file for purposes of monitoring and post-mortem analysis, and the resultant likelihood map.

The parameter file can be easily transferred from the submission host to the target resource, and the output of the job from the target resource to a data resource using either `gsiscp` or `globus-url-copy`.

Since the submission host does not automatically know when the output is finished, but the workflow on the target resource does, the transfer of the finished product is included as the target "send-map" in the `makefile`. ClusterFinder currently uses `gsissh` for job submission, piping `stdout` and `stderr` to files on the submission host.

These files can be monitored in real time, if desired, using "`tail -f`".

4.5 Logistics - using PostgreSQL

The highest level process is creating and managing the jobs to be submitted in parallel to the grid. ClusterFinder uses a PostgreSQL [31] database to organise the relevant information. The physics parameters for the RASS and for the SDSS calculations are available as entries in database tables. The input and output files are in default locations defined by a template. A script (`create-series`) takes a defined region of sky and divides it up into a number of patches, for each patch, a run is defined and given the status "pending".

Another script (`submit-pending-runs`) examines a pre-defined list of potential grid resources to determine which ones currently are up and running (accepting `gsissh` commands). It further determines how many jobs are currently running on each of these resources and compares that to the maximum number allowed for that resource according to the list. Finally it submits any pending runs to an available resource until they all have been submitted.

Since the submission mechanism includes transfer of the results to the submission host, after definition of a series and submission of the pending runs, all the output files will be available on the submission host at known locations, so that any post-processing may proceed locally.

5 Task farming

5.1 What is task farming?

In the task farming approach, on each resource, a copy of the executable runs independently of those running on other resources, and is responsible for fetching its own data and reporting its results. That is, all that is required of the grid user is to start the executables on as many resources as desired.

5.2 The GEO600 use case — an overview

The GEO600 use case (see [13]) is such a task farming application. It uses the *Einstein@Home* application [19] for analysing the data of the GEO600 Laser Interferometer near Hannover, in order to find signals of gravitational waves.

Einstein@Home uses the *Berkley Open Infrastructure for Distributed Computing — BOINC* [18]. The data analysis consists of a heuristic search within the detector data for known gravitational

wave signal patterns. Powered by the support of its members, who donate the idle time of home and office computers to the search for gravitational waves, the project reaches a peak performance of 72TFlop/s.

Einstein@Home was identified as an ideal candidate for a grid application because of:

- multi-platform support
- well tested software base
- simple resource requirements
- build-in checkpoint and recovery methods
- fine grained adjustable run time
- linear scaling with node number

These features ensure, that the data analysis will scale with the capabilities of the developing AstroGrid-D. Starting at the very beginning of the project the application was used to farm already available grid resources for the scientific community and to drive the integration of resources into D-Grid.

Right now *Einstein@Home* runs on the Globus resources of D-Grid, using WS GRAM [23] for task submissions.

5.3 GEO600 Deployment

Before running *Einstein@Home* tasks on a grid resource, the GEO600 use case software must be deployed there. The deployment itself can be performed as a WS-GRAM job from a submission host (which has the Globus client software installed) to a number of Grid machines running the Globus WS-GRAM and GridFtp Grid services. This is triggered by a script which is pre-staged to the target resource which then installs all required software there. As prerequisites on the target resource only *SVN* (to check out the GEO600 source code) and a *Perl* interpreter (to run the deployment script) are necessary. All other required packages are installed automatically during the deployment.

5.4 Running GEO600 on the Grid

Once the *GEO600* use case has been deployed on a Grid resource *Einstein@Home* tasks can be submitted there by calling a Perl script on the submission host. This perl script invokes the WS-GRAM service in order to submit one or more Grid tasks to Globus resources. In a GEO600 resource configuration file certain task submission parameters can be set for individual Grid resources:

- the location of the deployment directory of the GEO600 software on a target machine,
- the total number of tasks to be submitted to a target resource,
- the number of tasks which should be submitted at a time,

- the walltime to be allocated for an *Einstein@Home* task

The submission script uses a local MySQL database for the management of all submitted tasks. Each task is uniquely identified by a numeric task identifier which can be used to look up its WS-GRAM end point contact file; this file is used by the submission script to check the current status of the Grid task (eg. PENDING, ACTIVE, DONE) and its exit code when it has terminated. Depending on the number of currently pending and active tasks and the parameters in the GEO600 resource configuration file, the submission script can automatically determine when to submit new tasks to a Grid resource. To establish a continuous Grid task submission scheme, it is therefore sufficient to invoke the submission script periodically, eg. as a cron job, on the submission machine.

For large-scale taks farming applications it is likely that errors occur during the submission or execution of Grid tasks, eg. because a Grid resource or service is temporarily down, or the application itself returns an error on a given machine. In order to keep a continuous GEO600 Grid task submission scheme going, another script should be invoked periodically to inspect the exit code of failed GEO600 tasks (ie. their exit code is non-zero), and to clean up the task status in the database in the case of recoverable errors (which are defined in an error state configuration file). Jobs with unknown error states are left untouched and eventually need human interaction to be resolved.

The GEO600 use case runs now in production mode, and it consumes around **100 000** CPU hours a day on D-Grid resources. This is an amount of computing time which cannot be achieved easily on single cluster.

Part II

Advanced job submission

6 The GridWay Metascheduler

For many applications, it is undesirable to deal directly with specific grid resources (and their individual hardware profiles, and when they may be available). It is then better to submit a job to run on whatever adequate hardware is available, when it is available. This is the purpose of a resource broker/scheduler.

A grid *resource broker* is a mechanism for submitting a job to the grid, that automatically chooses the resources on which the job will run, based on the user's hardware requirements. A grid *scheduler* is a mechanism for automatically deciding when submitted jobs will run, based on resource availability.

6.1 GridWay job submission

The GridWay meta-scheduler is a flexible instrument for the implementation of grid scheduling and brokerage of jobs. It is part of the Globus open source project [21], developed by the "Distributed Systems Architecture Group" of the Univ. Compl. de Madrid in Spain.

Its provides an intermediate resource, the GridWay server (henceforth gwserver), which functions as a gateway between the submission host and the target resource.

A gwserver provides a command line function for submission of GridWay jobs (`gwsubmit`). This function is similar to `globusrun-ws`, but provides to GridWay's scheduling and brokering facilities. GridWay accesses the data from all available target resources, e.g. through the MDS system, which is available to the user via the `gwhost` command.

Submission of jobs through `gwsubmit` requires the specification of certain job characteristics by the user (type of job, single or parallel, special hardware requested or not).

The job is automatically scheduled to run on the next available free resource matching the request. The matchmaking parameters and the scheduling algorithm and priorities can in principle be freely configured through the GridWay server.

6.2 Example NBODY6++/phiGRAPE

We have encountered many practical difficulties to support this concept in a general way in our heterogeneous AstroGrid-D environment. Synchronising user requests with information available from resources, observing certain boundary conditions regarding data transfer has not fully been achieved in the AstroGrid-D yet. Rather, we have restricted ourselves to one application, the NBODY6++/phiGRAPE use case (see [13], [15], [29]), and demonstrate with it some of the elementary functionalities.

NBODY6++ is a member of a family of high accuracy direct N-body integrators originating from

Sverre Aarseth (see [32]) used for simulations of dense star clusters, galactic nuclei with massive black holes, and problems of planetary system formation. It is a special version of Nbody6 optimised for massively parallel computers. phiGRAPE is a variant of the code optimised for special purpose GRAPE or graphical processing unit (GPU) hardware.

7 The Grid Application Toolkit (GAT)

Although Globus has been chosen as the standard grid middleware in AstroGrid-D, but it is beneficial to have a uniform interface for submission of jobs to the grid independently of grid middleware (see [26]).

For example a user may want to access resources running other middleware like UNICORE, or vice-vers, a user may have software they have been running on other middleware, and want to run it on systems using Globus as well.

The GAT (Grid Application Toolkit, see [25]) provides grid functionality to grids built with a variety of grid middlewares.

7.1 GAT adaptors

The GAT offers adaptors for

- the Portable Batch System (PBS),
- the Sun Grid Engine (SGE),
- Globus, as well GRAM as WS GRAM,
- Unicore[4] (version 6),
- gLite,
- Local adaptors.

Due to the separate treatment of pre- and post-staging by GAT, it is also possible to use the SGE and PBS adaptors to submit jobs to an SGE or PBS based LRMS from a host which has no PBS or SGE installed. The only need is at least an `ssh` connection to the host with PBS or SGE.

The Local adaptors are very helpful, because they enable the development of a program against the GAT-API without having access to a grid. The calls remain the same, independent of the GAT adaptor which is invoked for doing the work. So one can first implement the program, and then switch it to the grid. This can reduce the time of development remarkably.

[4]The Unicore adaptor has been developed recently by Alexander Beck–Ratzka at the AEI, and is right now not delivered with the JavaGAT. In case of interest please contact `mailto:alexander.beck-ratzka@aei.mpg.de`

7.2 JavaGAT Command-Line-Interface

This is possible using the Command-Line-Interface (CLI) of JavaGAT. This CLI is a development of the AstroGrid-D project, and it is available in the JavaGAT-Release in the sub-directory `astrogrid-packages/GATJobRun`.

The entry point to the GAT-CLI is the script `gat-job-run`, which is located in the sub-directory `scripts` of `astrogrid-packages/GATJobRun`. It enables a user to submit a job with pre-staging and post-staging quite simply, e. g. by the call:

```
gat-job-run -stdin Input -stdout Output \
-prestage file1, file2 ...  -poststage file1, file2, ...  \
-RB.jobmanager SGE gridhost.grid.org <executable> <arguments>
```

The statement above submits an executable with arguments to *gridhost.grid.org*. Some stdin, stdout, and pre- and post-staging files are defined. The argument `RB.jobmanager` defines SGE to be used as the jobmanager on the target resource. An RSL dataset needed for this job is created. A detailed description can be found in [6].

8 The AstroGrid-D User Portal

An *AstroGrid-D User Portal* is available as a standard Web service under the URL

`http://cashmere.aip.de:8080`

As a common point of contact for German astrophysicists and their collaborators, this portal integrates many of the scientific applications and grid services developed in the AstroGrid-D project. The user interface provides a common look-and-feel environment to the scientist independent from their location, just like other portals on the Internet. It uses standard Web technology to present and search for information and data so that any Web browser of choice is sufficient to invoke the offered services—no other software needs to be installed on the client side.

8.1 Accessing Services through the Portal

Access to individual applications and grid services integrated in the AstroGrid-D portal is either publicly available (e.g. for an overview of computational resources registered in AstroGrid-D and D-Grid, or to see the job statistics for the GEO600 use case) or restricted to authorised AstroGrid-D portal users only.

In the latter case, an AstroGrid-D scientist must register for a portal user account first which then needs to be approved (or rejected) by a portal administrator. After a successful registration, they can authenticate and log in to the AstroGrid-D portal with the user name and password supplied in the account creation request. Currently there are the following applications available for authorised AstroGrid-D portal

users:

- ClusterFinder (see section 4.1 and [14])

- Dynamo (see section 2.2 and [9])
- PhiGRAPE (see [29])

When logged into the portal, the scientist can navigate between dedicated Web pages for each of these use cases. Each Web page is, naturally, designed specifically for the use case it belongs to.

As a working example, the portal pages for the ClusterFinder use case include

- a page giving a list of all ClusterFinder simulations previously performed by the user, along with a link to download the resulting data file
- a page with an HTML form where the user can specify certain parameters for a new ClusterFinder simulation, along with a button to submit this simulation as a grid job
- a user preferences page where default parameters can be set, such as the list of grid resources to be used for ClusterFinder job submissions, as well as the machine and directory where resulting simulation data files are to be stored

8.2 Integrating new Services in the Portal

Access to each of the above mentioned applications and grid services is provided through portlets. Portlets are Java software modules embedded into the portal container (e.g. an Apache Tomcat) through well defined Web interfaces. While the container takes care of general things such as the layout of multiple portlet windows, each portlet can import its own Web pages with application-specific contents. In order to integrate new applications or grid services in the AstroGrid-D portal, a corresponding Java portlet needs to be implemented.

The AstroGrid-D portal is entirely based on GridSphere[28], an open source Web development framework to build portals following the JSR 168 Web standard. In addition to basic Web portal functionality such as user authorisation, group/role management, access control support, and data persistence, the GridSphere framework also provides a generic portlet API for portal developers to implement their own application-specific portlets.

Documentation on how to install the AstroGrid-D portal is given in the AstroGrid-D Deliverable D7.1[5]. A guideline for AstroGrid-D developers how to design and implement their own GridSphere portlet is given in the deliverable document D7.5[7]; detailed information on the metadata management in existing AstroGrid-D application portlets is available in the deliverable document D7.6[8]. It might also be helpful to take a look at the source code of existing AstroGrid-D portlets which are available via anonymous SVN access (with user name `anonymous` and the password left blank):

```
svn checkout svn://svn.gac-grid.org/software/astrogridportlets
```

For more information on GridSphere, users should browse the GridSphere project Web pages [28]. Questions can be sent to the GridSphere developers mailing list `mailto:gridsphere-dev@gridsphere.org`.

Part III

Appendices

A Dynamo demo file `submit.sh`

```
#!/bin/bash

PROJECT=dynamo
EXECUTABLE=dynamo.x

cd ‘dirname $0‘
HOST=‘hostname --fqdn‘
MACHINES=(‘egrep ’^[^#]’ machines‘) #static machine file
NUM_INPUTS=‘ls -d1 ../input/input[0-9]* | wc -l‘

mkdir -p $HOME/.epr
rm -f visualisation.data
I=0
while [ $I -lt $NUM_INPUTS ]
do
    true & EPID=$!
    MACHINE=${MACHINES[$(($I % ${#MACHINES[*]}))]}
    UPLOAD=${PROJECT}_upload_${MACHINE}_${EPID}
    RESULT=${PROJECT}_results_${MACHINE}_${EPID}

    mkdir -m 777 -p $UPLOAD $RESULT
    INPUTDIR=‘cd ../input/input$I; pwd‘
    ln $INPUTDIR/* $INPUTDIR/../$EXECUTABLE $UPLOAD
    tar cf $UPLOAD.tar $UPLOAD

    eval ’cat <<-HERE
            ’$(< jdd.template)’
        HERE’ > $UPLOAD/$PROJECT.jdd

    globusrun-ws -submit -f $UPLOAD/$PROJECT.jdd -b -J -S -F $MACHINE \
        -o $HOME/.epr/${PROJECT}_${MACHINE}_${EPID}.epr
    if [ $? -eq 0 ]
    then
        echo $MACHINE, $HOME/.epr/${PROJECT}_${MACHINE}_${EPID}.epr, \
            $PWD/$RESULT, $UPLOAD, $INPUTDIR >> visualisation.data
        I=$(($I + 1))
    else
        rm -rf $UPLOAD $RESULT
        unset MACHINES[$(($I % ${#MACHINES[*]}))]
        MACHINES=(${MACHINES[*]})
    fi
done

test -x /usr/local/bin/idl && idl -vm=dynamo_vis.sav &
```

B Dynamo demo file `jdd.template`

```
<?xml version="1.0"?>

<job>
  <executable>/bin/sh</executable>
  <directory>/\${GLOBUS_USER_HOME}/</directory>
  <argument>-c</argument>
  <argument>
    tar xf ${UPLOAD}.tar;
    cd ${UPLOAD};
    ./${EXECUTABLE}
  </argument>
  <stdout>/dev/null</stdout>
  <stderr>/dev/null</stderr>
  <maxWallTime>100</maxWallTime>
  <maxMemory>2</maxMemory>
  <fileStageIn>
    <transfer>
      <sourceUrl>gsiftp://${HOST}${PWD}/${UPLOAD}.tar</sourceUrl>
      <destinationUrl>file:///\${GLOBUS_USER_HOME}/${UPLOAD}.tar</destinationUrl>
    </transfer>
  </fileStageIn>
  <fileStageOut>
    <transfer>
      <sourceUrl>file:///\${GLOBUS_USER_HOME}/${UPLOAD}/</sourceUrl>
      <destinationUrl>gsiftp://${HOST}${PWD}/${RESULT}/</destinationUrl>
    </transfer>
  </fileStageOut>
  <fileCleanUp>
    <deletion>
      <file>file:///\${GLOBUS_USER_HOME}/${UPLOAD}.tar</file>
    </deletion>
    <deletion>
      <file>file:///\${GLOBUS_USER_HOME}/${UPLOAD}/</file>
    </deletion>
  </fileCleanUp>
</job>
```

References

[1] AstroGrid-D: *How to access grid resources from non-grid machines*
http://www.gac-grid.org/project-products/grid-support/grid-from-outside.html

[2] AstroGrid-D Deliverable 1.4, *Running MPI Jobs on Grid Resources*
http://www.gac-grid.org/project-documents/deliverables/wp1/Run_MPI_Jobs_on_Grid-D_1_4.pdf

[3] AstroGrid-D Deliverable 2.6, *Advanced Prototype Implementation of Metadata Information Providers*
http://www.gac-grid.org/project-documents/deliverables/wp2/D2_6_Information_Providers.pdf

[4] AstroGrid-D Deliverable 3.3, *Distributed File Management*
http://www.gac-grid.org/project-documents/deliverables/wp3/D3_3_test_of_file_management.pdf

[5] AstroGrid-D Deliverable 7.1, *Installation of the GridSphere Portal Framework on a GACG Portal Server*
http://www.gac-grid.org/project-documents/deliverables/wp7/M2.pdf

[6] AstroGrid-D Deliverable 7.4, *GAT Software components and documentation*
http://www.gac-grid.org/project-documents/deliverables/wp7/7-4.pdf

[7] AstroGrid-D Deliverable 7.5, *Simulation-specific Portal Components and Documentation*
http://www.gac-grid.org/project-documents/deliverables/wp7/7-5.pdf

[8] AstroGrid-D Deliverable 7.6, *Portal User Interfaces for Metadata Management*
http://www.gac-grid.org/project-documents/deliverables/wp7/7-5.pdf

[9] AstroGrid-D Demo: Dynamo
http://www.gac-grid.org/project-products/Applications/DynamoIntro.html

[10] AstroGrid-D Hands-on-Globus workshop *Basic knowledge to use the grid*
http://www.gac-grid.org/project-overview/events-meetings/workshops/HandsOnUserWorkshop2007/

[11] AstroGrid-D: *Membership Guide*
http://www.gac-grid.org/project-products/grid-support/globus-user-guide.html

[12] AstroGrid-D: *Guides for installing and using Globus Grid*
http://www.gac-grid.org/project-products/grid-support

[13] AstroGrid-D Usecases,
http://www.gac-grid.org/project-documents/UseCases.html

[14] AstroGrid-D Application: ClusterFinder
http://www.gac-grid.org/project-products/Applications/ClusterFinder.html

[15] AstroGrid-D Application: Nbody6++
http://www.gac-grid.org/project-products/Applications/nbody6.html

[16] AstroGrid-D VO Registration
https://vomrs.gac-grid.org:8888/vomrs_astrogrid-d/vomrs

[17] AstroGrid-D WebMDS
http://www.gac-grid.org/project-products/grid-status/astrgrid-overview.html

[18] BOINC Project
http://boinc.berkeley.edu/

[19] Einstein@Home
http://www.einsteinathome.org/

[20] Globus Alliance
http://www.globus.org/

[21] Globus Toolkit Manuals
http://www.globus.org/toolkit/docs/4.0

[22] Globus, *Web Services in GTK*
http://www.globus.org/toolkit/docs/4.0/common/key/index.html#id2501285

[23] Globus, *WS-GRAM User's Guide*
http://www.globus.org/toolkit/docs/4.0/execution/wsgram/user-index.html

[24] Globus, *WS-GRAM Job Description Document (JDD) schema*
http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html

[25] Grid Application Toolkit
http://www.gac-grid.de/project-products/Software/GAT.html

[26] Rob V. van Nieuwpoort and Thilo Kielmann and Henri E. Bal, User-Friendly and Reliable Grid Computing Based on Imperfect Middleware
Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07)
http://sc07.supercomputing.org/schedule/pdf/pap232.pdf

[27] GridWay Metascheduler
http://www.gridway.org/

[28] GridSphere Project
http://www.gridsphere.org/

[29] phiGRAPE
http://wiki.cs.rit.edu/bin/view/GRAPEcluster/phiGRAPE

[30] OGSA-DAI Project
http://www.ogsadai.org.uk/

[31] PostgreSQL
http://www.postgresql.org/

[32] sverre.com
http://www.sverre.com/