Integration of GRID Superscalar and GridWay Metascheduler with the DRMAA OGF Standard*

R.M. Badia[1], D. Du[3], E. Huedo[2], A. Kokossis[3], I.M. Llorente[2], R.S. Montero[2], M. de Palol[1], R. Sirvent[1], and C. Vázquez[2]

[1] Barcelona Supercomputing Center
[2] Universidad Complutense de Madrid
[3] University of Surrey

Abstract. This paper shares the experiences with one of the BEinGRID pilots, BE14, from a technological point of view. The experiment has integrated GRID superscalar (as programming model) with GridWay (as metascheduler) through the DRMAA standard. Additionally, a portal based in GridSphere has been developed. The portal enables the management of the grid and the automatic deployment and monitoring of applications. This environment has been successfully used to speed up an application that enables new processes and products development in the Chemistry sector with considerable success.

Keywords: Grid Computing, DRMAA, GRID superscalar, GridWay Metascheduler.

1 Introduction

Business Experiments in GRID (BEinGRID), is an European Union integrated project funded by the Information Society Technologies (IST) research, part of the EUs sixth research Framework Programme (FP6). The BEinGRID consortium is composed of 75 partners who are running eighteen Business Experiments designed to implement and deploy grid solutions in industrial key sectors.

The main objective of BEinGRID project is to foster the adoption of the so-called Next Generation Grid technologies by the realization of several business experiments and the creation of a toolset repository of grid middleware upper layers. BEinGRID is undertaking a series of targeted business experiment pilots designed to implement and deploy grid solutions in a broad spectrum of European business sectors (entertainment, financial, industrial, chemistry, gaming, retail, textile, etc). Eighteen business experiments are ongoing in the initial stage of the project with a second open call that recently accepted a second bunch of experiments. Secondly, a toolset repository of grid service components and best practise will be created to support European businesses that wish to take-up the

* This research was supported by European Union, through BEinGRID project EU contract IST-2005-034702.

grid. To minimise redevelopment of components, BEinGRID will deploy innovative grid solutions using existing grid components from across the European Union and beyond.

The authors of this paper are involved in the Business Experiment 14 (BE14), "New Product and Process Development," that addresses the creation of an integrated environment that enables the automation of new products and processes in the Chemistry sector in a grid environment.

The process industry spends a lot of resources in the development of new products and processes. Nowadays these processes depend heavily on computers and are basically manual and sequential. The objective of this work is to implement a development environment that is able to automate these processes in a computational grid. Grids appear as the ideal venue to enable such an application since they offer tremendous scope to automate studies with virtual access to experts and resources and capabilities to launch integrated experiments.

The added value of the experiment is the increase of the efficiency measured by reductions in development times, systematic accumulation of industrial know-how, and effective use (and re-use) of knowledge and expertise. The application is developed on top of the integration of two powerful grid tools: GRID superscalar (GRIDSs) [1], which provides a grid-unaware programming environment and GridWay [2], a metascheduler which provides reliable execution in heterogeneous grids. The integration of both tools have been performed through the OGF standard DRMAA [3,4], which has also become the first OGF recommendation. This integration give benefits to both GRIDSs and GridWay: GRIDSs is enabled to run now with DRMAA schedulers, like GridWay, being able to rely on their features (fault tolerance, monitoring, migration...) and also with a general view of all the applications run in the grid; and GridWay benefits of a higher level programming environment, that from a sequential application is able to general a graph-dependency graph and exploit the concurrency of the application at task level.

In this work we focus on the technical aspects of the experiment: how GRIDSs and GridWay have been extended and modified to be able to work in cooperation and how an end-user application has been successfully developed on top of this environment. The paper structure is as follows: section 2 presents the Grid Applications Development solution (GridAD) developed in the framework of the BE14. Section 3 gives an overview of the enabling technologies of GridAD: GRIDSs and GridWay. Section 4 describes how GRIDSs and GridWay have been modified to enable their integration through the DRMAA OGF standard. In section 5 we present the early results obtained with the BE application. Finally, section 6 concludes the paper and presents future work.

2 Grid Applications Development Solution (GridAD)

GridAD is the result of the integration through the DRMAA OGF standard of two powerful grid tools: GRIDSs and GridWay Metascheduler. The combination of GRIDSs and GridWay (GridAD) provides a complete and powerful toolset for the development and deployment of applications in the grid. GRIDSs is specially unique for the possibility that it offers to the programmers to make the grid "invisible". On the other hand, GridWay is an efficient metascheduler, worldwide known and used. Additionally, there is no solution equivalent to the combination of both. GridAD can be used for computational grids and also on clusters, by linking to other DRMAA libraries intended for DRMS (Condor, SGE, etc). Figure 1 shows the flow that a GridAD application follows on execution. From a final's user sequential application, GRIDSs is able to find the existing task-level parallelism and schedules the tasks for execution through GridWay. GridWay then performs resource management and monitoring of the application.

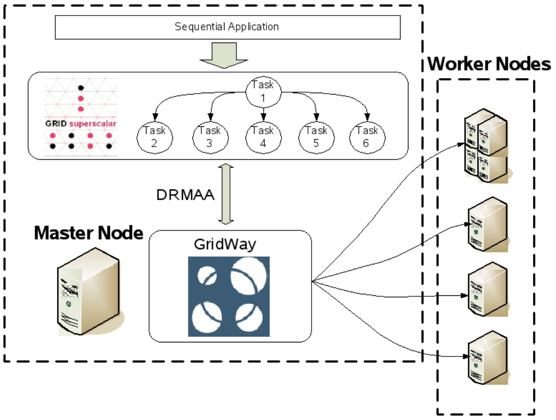

Fig. 1. Execution in GridAD environment

3 Enabling Grid Technologies

3.1 GRID Superscalar

GRID superscalar (GRIDSs) [1,5] is an innovative grid programming framework that enables non-grid experts to develop applications that can be run in a computational grid. GRIDSs provides a very user-friendly programming framework to grid environments. GRIDSs not only provides an abstract layer to program applications in the grid, but also is able to increase the performance of the applications by automatically parallelising parts of the application. Applications that can take advantage of GRIDSs are those composed of one or more coarse grain tasks that are called several times during the application execution. GRIDSs will execute the coarse grain tasks in independent grid servers and will execute sequentially those that have data dependencies. However, when no data dependencies exist between two or more tasks, GRIDSs is able to execute them concurrently.

GRIDSs is composed of:

– User Interface: GRIDSs applications are composed of three parts: main program, tasks' code, and interface of the tasks. A small set of primitives (up to seven) are offered for the main program, and two primitives for the tasks' code. The interface of the tasks is simply an interface specification that includes the direction of the tasks' parameters (input, output or input/output). Expert users can complete their applications by giving resource constraints (memory, disk, OS, ...) and performance costs models of their tasks.
– Automatic code generation: From the interface definition some code is automatically generated by gsstubgen, a tool provided with the GRIDSs distribution. This automatically generates code for stubs and skeletons that will be run on the grid servers and clients.
– Deployment Center: The deployment center is a graphical interface that performs the automatic deployment of the applications in the grid, by transfering the code files, automatic building of the binaries in the servers and configuration file generation.
– GRID superscalar Monitor: The GRID superscalar monitor (GSM) visualizes the task dependence graph (TDG) at run time, so the user can study the structure of the application and track the progress of execution.
– Run-time: The runtime is the more complex component of the system and performs: task dependency maintenance, task scheduling, file renaming (to further exploit the application concurrency), file transfer taking into account shared file systems, checkpointing, and fault tolerance. It is important to emphasize here that the runtime of GRIDSs makes the decision on which grid resource should be used to execute each task. To take this decision, several parameters are considered: location of input files, to reduce file transfers (and therefore exploiting file locality) and resource constraints specified by the user in the constraints interface. Another important feature is file renaming: this technique consists in the generation of several instances of the same file (i.e., several temporal files that are the same in the application, but in fact are different instances) to further increase the application parallelism.
  GRID superscalar run-time handles the renaming, maintaining at each moment for each renamed file the original filename and which is the last renamed filename, and this renaming is taken into account in the data dependence analysis. The run-time also keeps track of the server where each file is located. Files are transferred only on demand and if required. Together with a locality-aware scheduling policy, the number of file transfers is largely reduced.

GRIDSs is distributed as Open Source under Apache v2 license [5].

3.2 GridWay Metascheduler

GridWay [6] provides the end-user with a working environment and functionality similar to those found on local DRM systems, such as SGE, LSF or PBS. The end-user is able to submit, monitor and control his jobs by means of DRM-like commands (gwsubmit, gwwait, gwkill...) or standard programming interfaces.

– Efficient, reliable and unattended execution of jobs: GridWay automatically performs all the job scheduling steps, provides fault recovery mechanisms, and adapts job scheduling and execution to the changing grid conditions
– Broad application scope: GridWay is not bounded to a specific class of application generated by a given programming environment and does not require application deployment on remote hosts, which extends its application range and allows reusing of existing software. GridWay allows Submission of single, array or complex jobs consisting of task dependencies, which may require file transferring and/or database access.
– DRM-like command line interface: The GridWay command line interface is similar to that found on Unix and resource management systems such as PBS or SGE. It allows users to submit, kill, migrate, monitor and synchronize jobs, that could be described using the OGF standard JSDL.
– DRMAA application programming interface: GridWay provides full support for OGF standard DRMAA to develop distributed applications (C, JAVA, Perl, Python and Ruby bindings).

Moreover, GridWay modular architecture (see Figure 2) offers easy deployment, adaptability and extension capabilities, as well as support for site autonomy and dynamic environments. GridWay and the Globus Toolkit support the deployment of enterprise grids, that enable diverse resource sharing to improve internal collaboration and achieve a better return from their information technology investment; partner grids, allowing access to a higher computing performance to satisfy peak demands and also provide support to face collaborative projects; and outsourced grids, managed by dedicated service providers, that supply resources on demand over the Internet.

Since the release of GridWay 4.0, intended for Globus Toolkit 4 components, in January 2005, it is distributed under Apache v2 license [6]. GridWay is a Globus project [7] and, starting with GridWay 5.2.2, it is included in Globus Toolkit [8].

4 Integration

In this section, we discuss about the integration of the technologies explained above by means of DRMAA. The OGF *Distributed Resource Management Application API Working Group* (DRMAA-WG)[1] has developed an API specification for job submission, monitoring and control that provides a high level interface with *Distributed Resource Management Systems* (DRMS) [3]. In this way, DRMAA could aid scientists and engineers to express their computational problems by providing a portable direct interface to DRMS. DRMAA has been the first recommendation proposed by the OGF.

The functional description of the system was devised as follows: GRID superscalar runtime generates tasks that will be submitted to the GridWay metascheduler taking into account data dependencies between the tasks. The GridWay

[1] http://www.drmaa.org

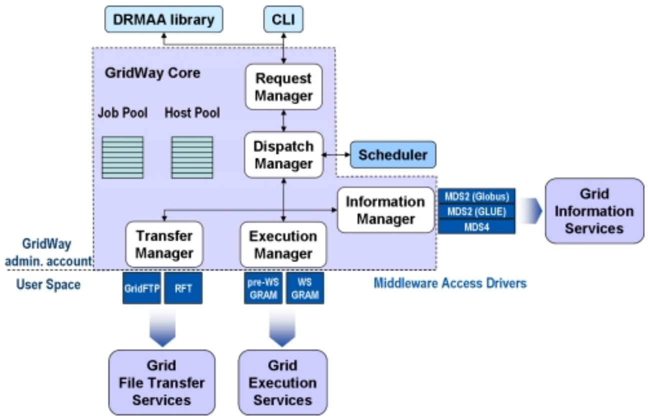

Fig. 2. Architecture of the GridWay Metascheduler

metascheduler receives the tasks submitted by GRID superscalar runtime, following the DRMAA standard, and run them in a remote resource through Globus Toolkit 4. GRIDSs polls GridWay for notifications on job state changes. The application makes several requests to GRIDSs runtime, which derives in the creation of a task in the task-graph. This task, whenever does not have data dependencies with other tasks, is submitted to GridWay for execution in a remote resource.

4.1 DRMAA Implementation in GridWay Metascheduler

In the following list we describe the DRMAA interface routines implemented in GridWay [4]:

- Initialization and finalization routines: `drmaa_init` and `drmaa_exit`.
- Job template routines: `drmaa_set_attribute`, `drmaa_allocate_job_template` and `drmaa_delete_job_template`. These routines enable the manipulation of job definition entities (job templates) to set parameters such as the executable, its arguments or the standard input/output streams.
- Job submission routines: `drmaa_run_job` and `drmaa_run_bulk_jobs`. GridWay has native support for *bulk* jobs, defined as a group of n similar jobs with a separate job id.
- Job monitoring and control routines: `drmaa_control`, `drmaa_wait`, `drmaa_ps`, `drmaa_synchronize`... These routines are used for holding, releasing, suspending, resuming and killing jobs, to monitor job status (see Figure 3), to wait for the completion of a job and check its exit status, or to synchronize jobs.
- Auxiliary routines: These routines are needed to obtain a textual representation of errors and other DRMAA implementation-specific information.

GridWay provides both C and Java bindings for DRMAA, as a dynamic library (`libdrmaa.so`), and as a JAR package (`drmaa.jar`), respectively. It also provides binding for scripting languages like Python, Perl and Ruby by using SWIG. Thanks to the use of DRMAA, only slight changes (mainly related to file manipulation) were needed in GridWay.

4.2 DRMAA Usage in GRIDSs

We faced here problems with the job submission with DRMAA and with the direct transfer of files, both of them related to the implementation of the file transfer and resource selection policy in GRIDSs. As explained in section 3.1 originally, GRIDSs decides where to submit a job taking into account the task resource requirements and its own information about file location. With the integration with GridWay, we faced two options: either to delegate GridWay the decision where to execute the tasks, but this would have meant either losing the file-locality exploitation policy or a large reimplementation of GridWay; or to allow GRIDSs to take the resource selection decisions. Current implementation is based on the second option, using GridWay in order to get information regarding the available machines.

Another problem detected is that GRIDSs is event-driven, while DRMAA only provides polling and blocking synchronization routines. This is due to the fact that GRIDSs originally worked with Globus directly, which gives notifications about the change of the jobs status. DRMAA blocking synchronization routines are just about one job status change, and that is when the job status changes to finished. This is not enough for GRIDSs, since it needs to know other changes as well, like from Queued to Running (as seen in Figure 3), to take better advantage of file locality, since then it knows that data needed for one job and that may be useful for another is already present in a particular worker node. To overcome this problem GRIDSs changed the way it waits for events, implementing a proactive polling to GridWay using DRMAA polling routines that enables it to be aware of GridWay jobs state changes.

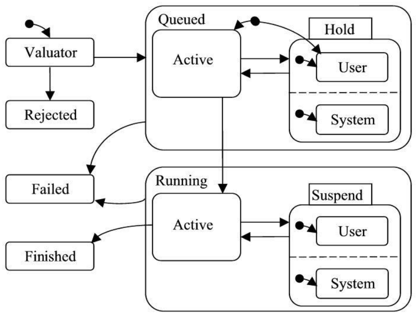

Fig. 3. DRMAA job state transition diagram [3]

4.3 Portal Development

We've implemented a Web Portal for providing the end-users a graphical, easy to use interface for some of the main functionalities of GridAD, these are: File management, uploading and downloading of files, let it be source code files or data files; and binary application deployments to worker nodes. The second functionality requires the portal the ability to deal with source code and being able to send the code to all the machines in the grid as well as compiling that code in the machines and checking the results.

The portal is implemented using GridSphere 2.2.9 [9], which is a portlet based open-source portal framework which supports an interface for working with the Globus Toolkit version 4. The deployment functionality of the portal is based on the GRID superscalar Deployment Center [1], a java based tool, which deploys the code and compiles it in all worker nodes on an execution. This application has been migrated to portlets, which run within the GridSphere framework and we've used the GridSphere grid framework (Gridportlets) to implement the usage of grid services from the portal. With this new grid portlet and MyProxy [10], the user can easily upload the source code of his/her application into the portal, from there select the machines that will run the application, splitting them between workers and a master node and then deploy the code, which will be compiled in a transparent way in each machine, and leave the system ready to be run.

5 Experiences

In the framework of the BE14 experiment, an application is presented that integrates the computation stages of a high-throughput environment for product and process development. The experiment allows the integration of models for optimization and simulation providing a flexible environment on top of GridAD [11,12]. As a result of using the grid, the performance of the experiment is dramatically increased. The high-throughput environment involves generic stages common to a variety of industrial problems, product synthesis applications, process design, materials design, and high-throughput experimentation in specialties, pharmaceuticals, and high-value chemicals. Such problems involve multiple runs, each using availablephysico-chemical and economic data, to target and screen options for products and processes. The combined use of computer and experiments is seen as the future environment for the development of novel products and processes.

In the specific experiment a process and catalyst development problems are modelled mathematically and solved with a combination of stochastic algorithms, deterministic algorithms, and graph-based methods. Among others, the application inputs contain (see Figure 4) superstructure models, kinetic data, configuration seeds and solver controls. The stochastic search takes the form of a Tabu search with parallel steps for intensification and diversification. In each step of the Tabu search, m different initial solutions goes through s slots, and in each slot 4 tasks are executed. Therefore, the number of tasks in one slot is $m \times s \times 4$. The system then updates the solutions with the best results, and the

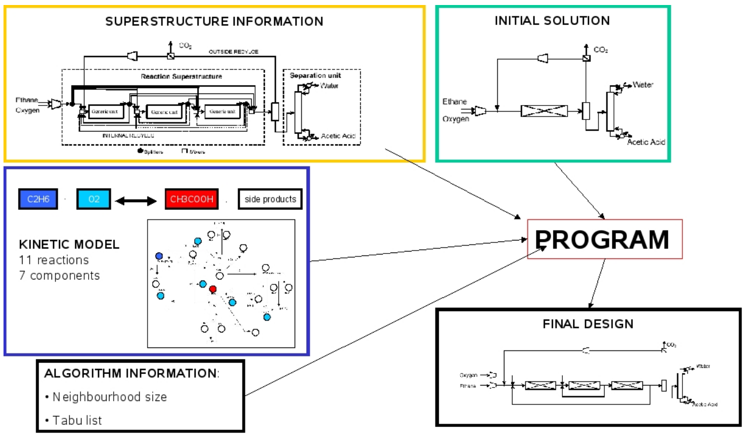

Fig. 4. Application environment

process is repeated i iterations and h times for a neighborhood factor. The total number of simulations is calculated as $m \times s \times 3 \times i \times h$ since one of the 4 initial tasks is very small.

The original sequential code was written in Fortran. Since GRIDSs does not currently support Fortran, the main program of the application, that follows an optimization TABU search scheme, was re-written in C. The computational intensive parts of the application were kept in Fortran and linked to the main program through C wrappers. This new code is still sequential and to adapt it to GRIDSs, a few calls to the GRIDSs API are added in the main program (2 or 3 calls) and the GRIDSs IDL file is written, which includes the interfaces of the computational tasks. The application code is then deployed in the grid using the portal and afterwards run in the grid. At runtime, the GRIDSs runtime exploits the concurrency between the tasks of the application and GridWay manage and monitors the task execution.

Table 1 presents a summary of statistical information for three different processes: a Van de Vusse kinetic scheme, a catalyst design experiment for acetic acid production, and a biotechnology (biocatalytic) process with excessive requirements for computing. The number of tasks and simulations required for each experiment are summarized in table 1. Real-life applications would require multiples of such simulations (typically by 3-5 orders of magnitude).

Main barriers for the adoption of the grid technology have been the differences in the supporting environment: whereas the language bindings offered by GRIDSs are mainly C/C++ (and Java) and GRIDSs can only run on Linux/ UNIX based platforms (due to GT4), the user application is in Fortran (as many similar industrial models) and the users' environment is based on Windows OS. The language barrier was overcome by the use of wrappers from C to Fortran. The implementation of the portal subsequently allows now to link Windows OS to a grid based on Linux/UNIX machines.

Table 1. Summary of application cases configuration

Application case	Van de Vusse	Acetic Acid	Biocatalytic
Number of initial solutions (m)	10	6	6
Number of slots (s)	50	20	50
Number of tasks (N)	2,000	480	1,200
Number of iterations per slot (i)	5	5	35
Neighborhood size (h)	7	20	20
Total number of tasks (n)	52,500	36,000	157,500
Sequential execution time	5 hours 22 mins	90 hours	55 hours
Grid execution time	40 mins 45 secs	15 hous 6 mins	7 hours
Speed up	8	6	7.9

GridAD environment has been installed and deployed in a grid composed by 3-site machines (UCM in Madrid, BSC in Barcelona and UniS in Surrey) with up to 20 servers. Preliminary execution results are also shown in Table 1. For these runs, 5 machines were used, with 8 workers. The speedup regarding the initial execution time is given as a reference of the speedup observed by the final user, although this speedup can be missleading: the reference sequential execution time was obtained using a different environment (the original workstation used by the final user) with different Operating System and performance characteristics. In any case, this results have been considered very promising for the final users. The results explain that a computer-based, high-throughput experimentation is now possible and viable. The deployment of a larger network of computers and better automation could further offer much smaller times to handle the actual volume of experiments in real-life problems. Moreover, the authors currently research asynchronous versions of the optimization search to reap additional benefits in the intelligence of the search and a better parallelization of the computing.

6 Conclusions and Future Work

GridAD benefits from its components. From GRIDSs takes the ability to be able to write gridified applications in a really easy way, by just defining which functions we want to execute remotely on an IDL file. GRIDSs then automatically generated the necessary code and in run-time it uses techniques to assure an optimal use of data locality. From GridWay it takes the fault tolerance mechanisms, the ability to dynamically deploy the application, the ability to interoperate with different middlewares (EGEE, TeraGrid, Open Science Grid ,etc) and the resource provisioning among others. Also, there are benefits for their separated components. We can see this in the use of DRMAA for the components communication, how it benefits GRIDSs since now it can be plugged to traditional DRMS and run the application on a local cluster or it can be plugged to GridWay and run it on a grid infrastructure.

The paper presented the application of a high-throughput prototype that would enable the synthesis and design of novel products and processes. The use of grids facilitated the combined deployment of optimization and simulation searches, using problems and data from real-life cases. Grid-enabled computing produces realistic times to complete the experiments and reports a rather promising message for the future of similar applications. Work in progress includes the gridification of the algorithmic stages, the automation of the underlying workflows, and the parallel visualization of the synthesis search, all beyond the scope of our initial effort.

One should note that there exist hundreds of thousands of models in reaction, separation, catalysis, and energy integration. On the basis of the evidence shown in the paper, one could envisage future utility services that could be offered through the grid. Apparently, future models are left with a task to upgrade their communication capabilities, possibly through ontologies and semantics, so that to fully exploit the available computing power and enable their abilities to integrate in similar high-throughput experiments.

References

1. Badia, R.M., Labarta, J., Sirvent, R., Pérez, J.M., Cela, J.M., Grima, R.: Programming grid applications with grid superscalar. Journal of Grid Computing 1(2), 151–170 (2003)
2. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. Software - Practice and Experience 34(7), 631–651 (2004)
3. Rajic, H., Brobst, R., Chan, W., Ferstl, F., Gardiner, J., Robarts, J.P., Haas, A., Nitzberg, B., Tollefsrud, J.: Distributed Resource Management Application API Specification 1.0. Technical report, DRMAA Working Group – The Global Grid Forum (2003)
4. Herrera, J., Huedo, E., Montero, R.S., Llorente, I.M.: GridWay DRMAA 1.0 Implementation – Experience Report. Document GFD.E-104, DRMAA Working Group – Open Grid Forum (2007)
5. GRID superscalar website, `http://www.bsc.es/grid/gridsuperscalar`
6. website, G.: GridWay Metascheduler website, `http://www.gridway.org`
7. GridWay dev.globus website, `http://dev.globus.org/wiki/GridWay`
8. Globus Toolkit website, `http://www.globus.org`
9. Gridsphere website, `http://www.gridsphere.org`
10. MyProxy website, `http://grid.ncsa.uiuc.edu/myproxy`
11. Antonopoulos, N., Linke, P., Kokossis, A.: A prototype GRID framework for the Chemical Process Industries. Chemical Engineering Communications 192(10-12), 1258–1271 (2005)
12. A., K.: Modelling power as a utility. White Paper for the future of simulation, optimization, and engineering computing (2005)