DRMAA Implementation within the Grid *W*ay Framework.*

J. Herrera [†] R.S. Montero [†] E. Huedo [‡]
I.M. Llorente [†,‡]

Email: jherrera@fdi.ucm.es, rubensm@dacya.ucm.es, huedoce@inta.es, llorente@dacya.ucm.es

[†]Dpto. Arquitectura de Computadores y Automática
Facultad de Informática, Universidad Complutense
28040 Madrid (Spain)

[‡]Lab. Computación Avanzada, Simulación y Aplicaciones Telemáticas
Centro de Astrobiología (CSIC-INTA),
28850 Torrejón de Ardoz (Spain).

September 13, 2004

Abstract

One of the barriers that prevents the expansion and adoption of Grid technologies is the lack of a standard programming paradigm. The Distributed Resource Management Application API (DRMAA) has been proposed to support the rapid development and distribution of applications across Distributed Resource Management Systems (DRMS). In this paper we describe an implementation of the DRMAA standard on a Globus-based testbed, and show its suitability to express typical scientific applications. As a case of study, we consider the implementation of the NAS Grid Benchmarks (NGB) with DRMAA. The DRMAA routines are supported by the functionality offered by the Grid *W*ay framework, which provides the runtime mechanisms needed for transparently executing jobs on a dynamic Grid environment.

1 Introduction

In recent years a great research investment has been made in Grid computing technologies. However, deployment of applications across the Grid requires a high level of expertise and

*This research was supported by Ministerio de Ciencia y Tecnología through the research grant TIC 2003-01321 and Instituto Nacional de Técnica Aeroespacial "Esteban Terradas" (INTA).

a significant amount of effort. The most important barriers arise from the nature of the Grid itself: multiple administration domains, heterogeneity, dynamism and high fault rate. In a previous work [3] we have developed a Globus submission framework, Grid*Way*, which allows an easier and more efficient execution of jobs on a dynamic Grid environment in a "submit and forget" fashion. Grid*Way* automatically performs all the job scheduling steps [6] (resource discovery and selection, and job preparation, submission, monitoring, migration and termination), provides fault recovery mechanisms, and adapts job execution to the changing Grid conditions.

On the other hand, the Grid lacks of standard programming paradigms. The *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). In this way, DRMAA could aid scientists and engineers to express their computational problems by providing a portable direct interface to DRMS.

In this work we describe the implementation of DRMAA within the Grid*Way* framework, and investigate the suitability of the DRMAA specification to distribute typical scientific workloads across the Grid. In this sense the *NAS Grid Benchmarks* (NGB) suite [7] constitutes an excellent case-of-study, since it models distributed *communicating* applications typically executed on the Grid.

In Section 2, we describe the functionality provided by the Grid*Way* framework to overcome the barriers to efficiently exploit a Grid environment. Section 3 briefly describes the implemented DRMAA routines, and the development and execution process adopted in this work. Finally, in Section 4, we evaluate the suitability of the DRMAA implementation for executing NGB. The paper ends in Section 5 with some conclusions.

2 The Grid*Way* Framework

Although the DRMAA standard can help in exploiting the intrinsic parallelism found in some application domains, the underlying DRMS is responsible for the efficient and robust execution of each job. In particular the following aspects are considered by the Grid*Way* framework:

- Given the dynamic characteristics of the Grid, the Grid*Way* framework periodically adapts the schedule to the available resources and their characteristics [3]. Grid*Way* incorporates a *resource selector* that reflects the applications demands, in terms of requirements and preferences, and the dynamic characteristics of Grid resources, in terms of load, availability and proximity (bandwidth and latency) [4].

- The Grid*Way* framework also provides adaptive job execution to migrate running applications to more suitable resources. So improving application performance by adapting it to the dynamic availability, capacity and cost of Grid resources. Moreover,

[1] http://www.drmaa.org (2004)

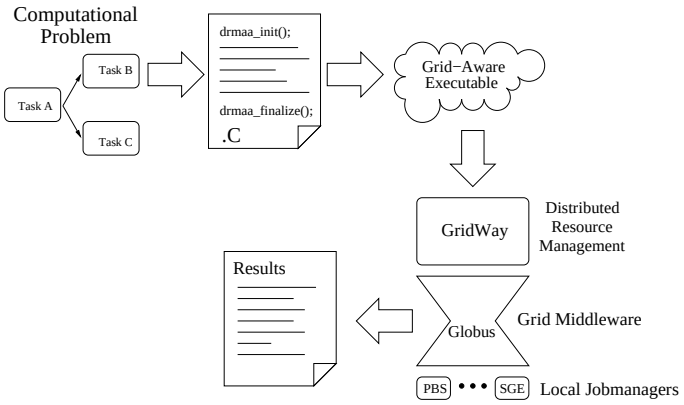


Figure 1: Development and execution cycle using the DRMAA interface

an application can migrate to a new resource to satisfy its new requirements or preferences [3].

- Grid*Way* also provides the application with fault tolerance capabilities by capturing GRAM callbacks, by periodically probing the GRAM *jobmanager*, and by inspecting the output of each job.

We expect that DRMAA will allow to explore several execution techniques when distributing applications across the Grid. For example fault tolerance could be improved by replicating job executions (redundant execution), the intrinsic parallelism presented in the workflow of several applications could be exploited, or several alternative task flow paths could be concurrently executed (speculative execution).

3 Distributed Resource Management Application API

One of the most important aspects of Grid Computing is its potential ability to execute distributed communicating jobs. The DRMAA specification constitutes a homogeneous interface to different DRMS to handle job submission, monitoring and control, and retrieval of finished job status. In this sense the DRMAA standard represents a suitable and portable framework to express this kind of distributed computations.

Although DRMAA could interface with DRMS at different levels, for example at the intranet level with SGE or Condor, in the present context we will only consider its application at global Grid level. In this way, the DRMS (Grid*Way* in our case) will interact with the local *jobmanagers* (Condor, PBS, SGE...) through the Grid middleware (Globus). This development and execution scheme with DRMAA, Grid*Way* and Globus is depicted in figure 1.

In the following list we describe the DRMAA interface routines implemented within the Grid*Way* framework:

- Initialization and finalization routines: `drmaa_init` and `drmaa_exit`.

Table 1: Characteristics of the machines in the UCM-CAB research testbed.

Name	VO	Model	Speed	OS	Memory	*jobmanager*
babieca	CAB	5×Alpha DS10	466MHz	Linux 2.2	256MB	PBS
hydrus	UCM	Intel P4	2.5GHz	Linux 2.4	512MB	fork
cygnus	UCM	Intel P4	2.5GHz	Linux 2.4	512MB	fork
cepheus	UCM	Intel PIII	600MHz	Linux 2.4	256MB	fork
aquila	UCM	Intel PIII	666MHz	Linux 2.4	128MB	fork

- 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 output streams.
- Job submission routines: `drmaa_run_job` and `drmaa_run_bulk_jobs`. Grid*Way* has native support for *bulk* jobs, defined as a group of n similar jobs with a separate job id.
- Job control and monitoring routines: `drmaa_control`, `drmaa_synchronize`, `drmaa_wait` and `drmaa_job_ps`. This routines are used to control (killing, resuming, suspending, etc..) and synchronize jobs, and monitor their status.

The DRMAA interface (see [5] for a detailed description of the C API) includes more routines in some of the above categories as well as auxiliary routines that provides textual representation of errors, not implemented in the current version. All the functions implemented in the Grid*Way* framework are thread-safe. We would like to mention that the implementation of the DRMAA routines is straightforward thanks to the functionality offered by the Grid*Way* framework.

4 The NAS Grid Benchmarks: A Case of Study

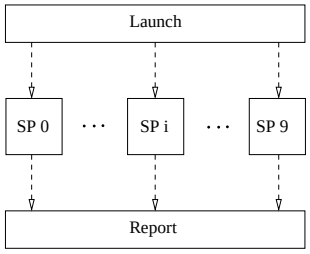
The *NAS Grid Benchmarks* (NGB) [2] have been presented as a *Data Flow Graph* (DFG) encapsulating an instance of a *NAS Parallel Benchmarks* (NPB) [1] code in each graph node, which *communicates* with other nodes by sending/receiving initialization data. NGB is focused on computational Grids, which are used mainly for running compute-intensive jobs that potentially process large data sets. Each benchmark comprises the execution of several NPB codes that symbolize scientific computation (flow solvers SP, BT and LU), post-processing (data smoother MG) and visualization (spectral analyzer FT).

Like NPB, NGB specifies several different classes (i.e. problem sizes), in terms of mesh size and number of iterations. The NGB problems considered in this section belong to the A class, as defined in [2]. The four families of problems defined in the NGB suite

model applications typically executed on the Grid and therefore constitutes an excellent case-of-study for testing the functionality of the DRMAA and the environment itself. The experiments were conducted in the UCM-CAB research testbed, based on the Globus Toolkit, briefly described in table 1.

4.1 Embarrassingly Distributed

Embarrassingly Distributed (ED) family represents the important class of Grid applications called *Parameter Sweep Applications* (PSA), which constitute multiple independent runs of the same program, but with different input parameters. In particular, each one consists in the execution of the SP flow solver [1] with a different initialization parameter for the flow field. *Parameter Sweep Applications* (PSA) like this can be directly expressed with the DRMAA interface as *bulk* jobs. The general structure of ED and its implementation with DRMAA are shown in figure 2.




```
//Initialization
jt= SP;

rc= drmaa_init(contact,err);

// Execute 9 jobs simultaneously and wait
rc= drmaa_run_bulk_jobs(job_ids,jt,1,9,1,err);
rc= drmaa_synchronize(job_ids,timeout,1,err);

rc= drmaa_exit(err_diag);
```

Figure 2: Structure and implementation of the ED benchmark using DRMAA.

Figure 3 shows an execution profile of the ED benchmark. Each suitable resource in the Grid consecutively executes three SP tasks, which results in a total turnaround time of 18.88 minutes and so a throughput of 0.5 jobs per minute. The average execution and file transfer times for each task are 4.25 minutes and 18 seconds, respectively.

4.2 Helical Chain

The *Helical Chain* (HC) family represents long chains of repeating processes, such as a set of flow computations that are executed one after the other, as is customary when breaking up long running simulations into series of tasks, or in computational pipelines. Each job in the sequence uses the computed solution of its predecessor to initialize. Considering these dependencies each job in the chain can be scheduled by Grid*Way* once the previous job has finished (see figure 4).

An execution profile of the HC benchmark is shown in figure 5. The turnaround time is 17.56 minutes, with an average resource usage of 20.21%. The MDS delay in publishing resource information results in an oscillating scheduling of the jobs. This schedule

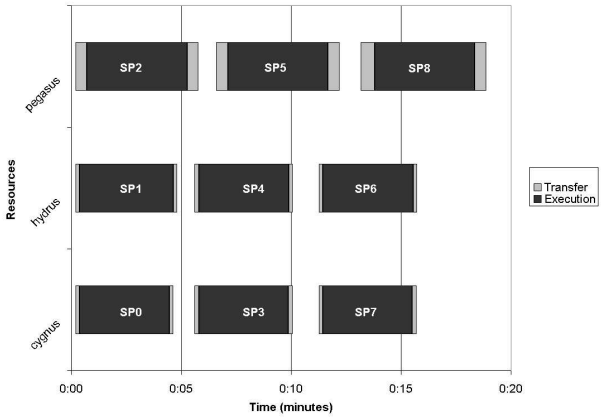
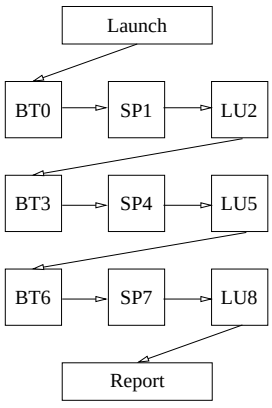


Figure 3: Execution profile of the ED benchmark class A.




```
// Initialization
jobs[0].jt= BT; jobs[1].jt= SP; jobs[2].jt= LU;
jobs[3].jt= BT; jobs[4].jt= SP; jobs[5].jt= LU;
jobs[6].jt= BT; jobs[7].jt= SP; jobs[8].jt= LU;

drmaa_init(contact, err);

// Submit all jobs consecutively
for (i= 0; i<9; i++) {
  drmaa_run_job(job_id, jobs[i].jt,err);
  drmaa_wait(job_id, &stat, timeout,rusage, err);
}

drmaa_exit(err_diag);
```

Figure 4: Structure and implementation of the HC benchmark using DRMAA.

clearly reduces the performance obtained compared to the optimal turnaround time[2] of 6.18 minutes.

4.3 Visualization Pipe and Mixed Bag

Visualization Pipe (VP) represents chains of compound processes, like those encountered when visualizing flow solutions as the simulation progresses. *Mixed Bag* (MB) again involves the sequence of flow computation, post-processing, and visualization, but now the emphasis is on introducing asymmetry. These benchmarks are combinations of the ED (fully parallel) and HC (fully sequential) benchmarks described above. They exhibit some parallelism that should be exploited, but it is limited by the dependencies between jobs. In the case of VP, the parallelism is even more limited due to the low pipe width (only 3, for all classes) and the long times to fill and drain the pipe (with class A, it only executes

[2]Belonging to a serial execution on the fastest machine.

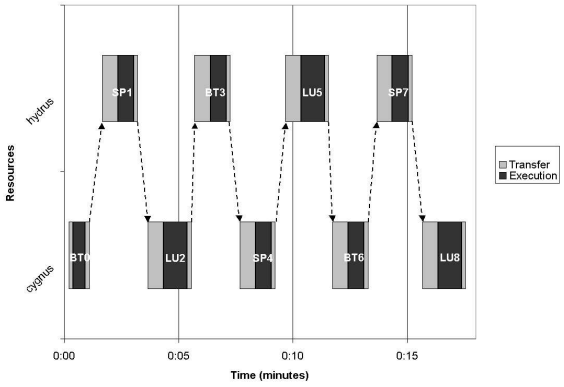


Figure 5: Execution profile of the HC benchmark class A.

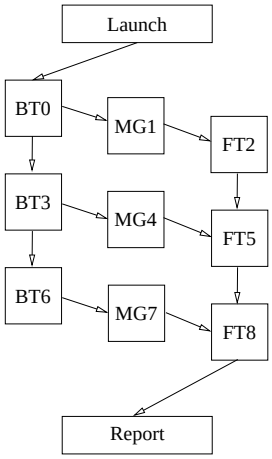
once with full parallelism).

Since neither Grid*W*ay nor DRMAA directly support workflow execution, we have developed a simple *workflow engine* taking advantage of the DRMAA programming interface, see figure 6. This *workflow engine* submits jobs as the jobs they depend on are completing.

```
drmaa_init(contact, err);

// Loop until all jobs are finished
while (there_are_jobs_left(jobs)) {
  // Submit jobs with dependencies solved
  for (i= 0; i<num_jobs; i++)
    if (is_job_ready(jobs,i))
      drmaa_run_job(jobs[i].id,jobs[i].jt,err);

  // Wait any submitted job to finish
  job_id= "DRMAA_JOB_IDS_SESSION_ANY";
  drmaa_wait(job_id,&stat,timeout,rusage,err);
  set_job_done(jobs,job_id);
}

drmaa_exit(err_diag);
```

Figure 6: Implementation of the *workflow engine*.

Figure 7 shows the structure and *workflow engine* initialization of the VP benchmark. Figure 8 shows an execution profile of the VP benchmark. Dashed lines represent dependencies between jobs and thicker lines represent the critical path. In this case, the turnaround time is 21.68 minutes, with an average resource usage of 35.25%. Total execution and transfer times are 22.93 and 8.1 minutes, respectively. Total execution time is slightly greater than the turnaround time, which shows that the parallelism is very limited due to the low pipe width, the long times to fill and drain the pipe, and the Grid overhead.




```
// Initialization
jobs[0].jt= BT; jobs[1].jt= MG; jobs[2].jt= FT;
jobs[3].jt= BT; jobs[4].jt= MG; jobs[5].jt= FT;
jobs[6].jt= BT; jobs[7].jt= MG; jobs[8].jt= FT;

jobs[0].dep= "";  jobs[1].dep= "0"; jobs[2].dep= "1";
jobs[3].dep= "0"; jobs[4].dep= "3"; jobs[5].dep= "2 4";
jobs[6].dep= "3"; jobs[7].dep= "6"; jobs[8].dep= "5 7";
```

Figure 7: Structure and *workflow engine* initialization of the VP benchmark.

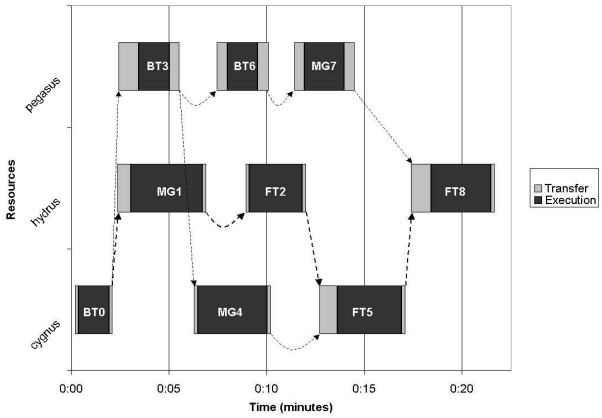


Figure 8: Execution profile of the VP benchmark class A.

5 Conclusions

We have shown how DRMAA can aid the rapid development and distribution across the Grid of typical scientific applications, and we have demonstrated the robustness and efficiency of its implementation on top of the Grid*W*ay framework and Globus. The functionality of this environment has been demonstrated through its ability to execute the NGB suite using DRMAA. In this sense, the use of standard interfaces allows the comparison between different Grid implementations, since neither NGB nor DRMAA are tied to any specific Grid middleware.

References

[1] D. H. Bailey, E. Barszcz, and J. T. Barton. The NAS Parallel Benchmarks. *Intl. J. of Supercomputer Applications*, 5(3):63–73, 1991.

[2] M. A. Frumkin and R. F. Van der Wijngaart. NAS Grid Benchmarks: A Tool for Grid Space Exploration. *J. of Cluster Computing*, 5(3):247–255, 2002.

[3] E. Huedo, R. S. Montero, and I. M. Llorente. A Framework for Adaptive Execution on Grids. *Intl. J. of Software – Practice and Experience (SPE)*, 34:634–651, 2004.

[4] R. S. Montero, E. Huedo, and I. M. Llorente. Grid Resource Selection for Opportunistic Job Migration. In *Proc. of the 9th Intl. Conf. on Parallel and Distributed Computing (Euro-Par 2003)*, volume 2790 of *Lecture Notes in Computer Science*, pages 366–373. Springer–Verlag, August 2003.

[5] H. Rajic et al. Distributed Resource Management Application API Specification 1.0. Technical report, DRMAA Working Group – The Global Grid Forum, 2003.

[6] J. M. Schopf. Ten Actions when Superscheduling. Technical Report GFD-I.4, Scheduling Working Group – The Global Grid Forum, 2001.

[7] R. F. Van der Wijngaart and M. A. Frumkin. NAS Grid Benchmarks Version 1.0. Technical Report NAS-02-005, NASA Advanced Supercomputing (NAS), NASA Ames Research Center, Moffett Field, CA, 2002.