Developing Grid-Aware Applications with DRMAA on Globus-based Grids *

J. Herrera[1], E. Huedo[2], R.S. Montero[1], and I.M. Llorente[1,2]

[1] Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain.
[2] Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas, Centro de Astrobiología (CSIC-INTA), 28850 Torrejón de Ardoz, Spain.

Abstract. In spite of the great research effort made in Grid technology in the last years, application development and execution in the Grid continue requiring a high level of expertise due to its complex and dynamic nature. The Distributed Resource Management Application API (DRMAA) has been proposed to aid the rapid development and distribution of applications across the Grid. In this paper we present the first implementation of the DRMAA standard on a Globus-based testbed, and show its suitability to express typical scientific applications. The DRMAA routines have been implemented using the functionality provided by the Grid*W*ay framework.

1 Introduction

In recent years a great research investment has been made in Grid computing technologies. However, deployment of existing applications across the Grid requires a high level of expertise and a significant amount of effort. The most important barriers arise from the nature of the Grid itself:

- Complexity, in order to achieve any functionality the user is generally responsible for manually performing all the scheduling steps.
- Heterogeneity of Grid resources, that potentially belongs to multiple administration domains.
- Dynamism of the availability, cost and load of Grid resources.
- High fault rate, resource or network failures are the rule rather than the exception.

In a previous work [1] we have developed a Globus submission framework, Grid*W*ay, that allows an easier and more efficient execution of jobs on a dynamic Grid environment in a "submit and forget" fashion. Grid*W*ay automatically performs all the job scheduling steps [2] (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 [3].

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

On the other hand, the Grid lacks of a standard programming paradigm to port existing applications among different environments. Grid technologies are specified, standardized and implemented within the *Global Grid Forum* (GGF)[1]framework. GGF is structured in working and research groups focused on specific aspects of Grid Computing. In particular, the *Distributed Resource Management Application API Working Group* (DRMAA-WG)[2] 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.

There are several projects underway to implement the DRMAA specification on different DRMS, like Sun Grid Engine (SGE) or Condor. However, to the best of the authors' knowledge, DRMAA has never been implemented in a Globus-based DRMS. In this work we discuss several aspects of 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 Section 2, we describe the DRMAA standard and its implementation. Section 3 analyzes several aspects involved in the efficient execution of distributed applications, and how they are addressed by Grid*Way*. Then, in Section 4, we study the implementation of several applications using DRMAA. Finally, the paper ends in Section 5 with some conclusions.

2 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 homogenous 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 Grid level. In this way, the DRMS (Grid*Way* in our case) will interact with the local job manager (i.e PBS, SGE,...) through the Grid middleware (Globus Toolkit 2.2). This development and execution scheme with DRMAA 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`.
- 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.

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

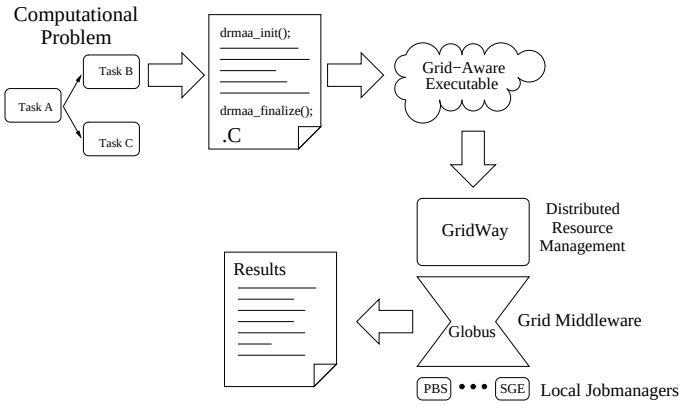


Fig. 1. Development and execution cycle using the DRMAA interface

- Job submission routines: `drmaa_run_job` and `drmaa_run_bulk_jobs`. The Grid*W*ay 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`. These routines are used to control (killing, resuming, suspending, etc..) and synchronize jobs, and monitor their status.

The DRMAA interface (see [4] 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*W*ay framework are thread-safe.

3 Efficient Execution of Grid Applications

In spite of 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*W*ay framework:

- Given the dynamic characteristics of the Grid, the Grid*W*ay framework periodically adapts the schedule to the available resources and their characteristics [3]. Grid*W*ay 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) [5].
- The Grid*W*ay 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, an application can migrate to a new resource to satisfy its new requirements or preferences [3].

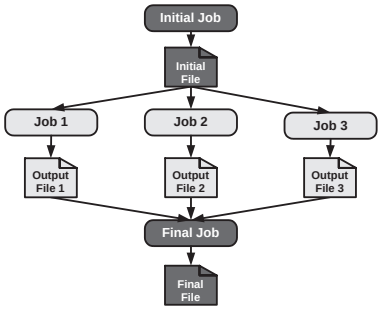
Table 1. Characteristics of the machines in the UCM research testbed.

Name	Model	OS	Speed	Memory	Job Mgr.
babieca	5×Alpha DS10	Linux 2.2	466MHz	256MB	PBS
hydrus	Intel P4	Linux 2.4	2.5GHz	512MB	fork
cygnus	Intel P4	Linux 2.4	2.5GHz	512MB	fork
cepheus	Intel PIII	Linux 2.4	662MHz	256MB	fork
aquila	Intel PIII	Linux 2.4	568MHz	128MB	fork

We expect that DRMAA will allow to explore several common execution techniques when distributing applications across the Grid [6], for example fault tolerance could be improved by replicating job executions (redundant execution), or several alternative task flow paths could be concurrently executed (speculative execution).

4 Experiences

In this section we describe the ability of the Grid*Way* framework when executing different computational workloads distributed using DRMAA. The following examples resembles typical scientific problems whose structure is well suited to the Grid architecture. These experiments were conducted in the UCM research testbed, based on the Globus Toolkit 2.2 [7], briefly described in table 1.




```
rc = drmaa_init(contact, err);

// Execute initial job and wait for it
rc = drmaa_run_job(job_id, jt, err);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err);

// Execute n jobs simultaneously and wait for them
rc = drmaa_run_bulk_jobs(job_ids,jt,1,JOB_NUM,1,err);
rc = drmaa_synchronize(job_ids, timeout, 1, err);

// Execute final job and wait for it
rc = drmaa_run_job(job_id, jt, err);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err);

rc = drmaa_exit(err_diag);
```

Fig. 2. High-throughput scheme and its codification using the DRMAA standard.

4.1 High-Throughput Computing Application

This example represents the important class of Grid applications called *Parameter Sweep Applications* (PSA), which constitutes multiple independent runs of

the same program, but with different input parameters. This kind of computations appears in many scientific fields like Biology, Pharmacy, or Computational Fluid Dynamics. In spite of the relatively simple structure of this applications, its efficient execution on computational Grids involves challenging issues [8].

The structure of the PSA is shown in figure 2 (left-hand side). A initial job is submitted to perform some pre-processing tasks, and then several independent jobs are executed with different input parameters. Finally a post-processing job is executed.

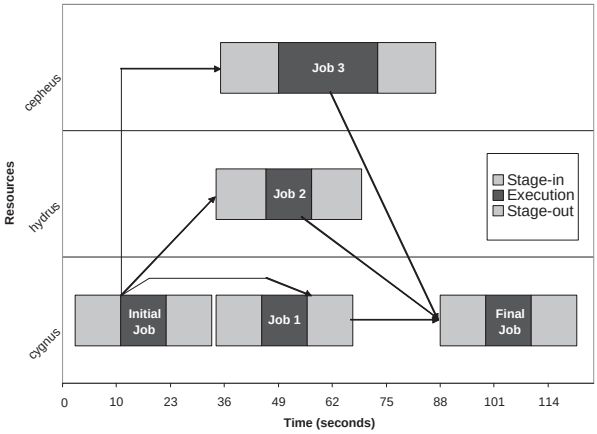


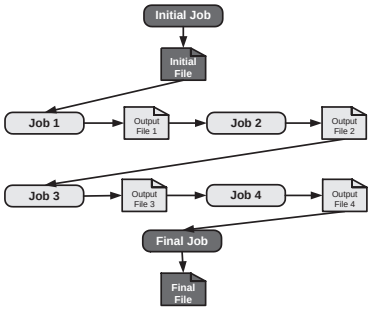
Fig. 3. Execution profile for the high-throughput application.

Figure 3 shows the execution profile of the PSA, each computational task is executed in a different Grid resource. The total turnaround time of the experiment is 121 seconds, with an average execution and file transfer times for each computational task of 15 seconds and 22 seconds, respectively. In this case the average CPU utilization during the PSA execution was 20%. In this case the overhead induced by job scheduling (i.e. querying the MDS Grid service to obtain a preliminary list of potential hosts, and to assign a rank to them [1]) is 5% of the overall execution time.

4.2 Pipelined Workflow Application

The pipelined workflow comprises the execution of a long chain of n jobs. Each job in the sequence uses the computed solution of its predecessor to initialize. Considering this dependencies each job in the chain can be scheduled by Grid*Way* once the previous job has finished. This computational scheme typically appears in long running simulations that can be broken up into a series of tasks (see figure 4).

In this case all the jobs in the pipeline sequence are submitted to the same host, cygnus, with an average turnaround time per job of 33 seconds. The total turnaround time for this application is 203 seconds, and an average resource CPU utilization of 81%.

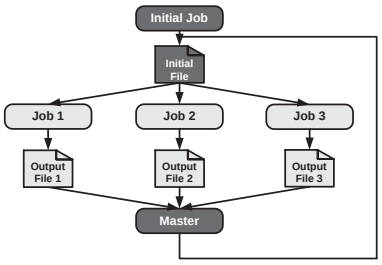



```
rc = drmaa_init(contact, err_diag);
// Execute n jobs consecutively
for (i = 0; i < JOB_NUM; i++) {
   rc = drmaa_run_job(job_id, jt, err_diag);
   rc = drmaa_wait(job_id, &stat, timeout, rusage, err_diag);
}
rc = drmaa_exit(err_diag);
```

Fig. 4. Pipelined workflow application and its codification using the DRMAA standard.

4.3 Master-Worker Optimization Loop

We now consider a generalized Master-Worker paradigm, which is adopted by many scientific applications like genetic algorithms, N-body simulations or Monte Carlo simulations among others. A Master process assigns a description (input files) of the task to be performed by each Worker. Once all the Workers are completed, the Master process performs some computations in order to evaluate a stop criterion or to assign new tasks to more workers (see figure 5).




```
rc = drmaa_init(contact, err_diag);

// Execute initial job and wait for it
rc = drmaa_run_job(job_id, jt, err_diag);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err_diag);

while (exitstatus != 0) {
// Execute n Workers concurrently and wait for them
rc = drmaa_run_bulk_jobs(job_ids, jt, 1, JOB_NUM, 1, err_diag);
rc = drmaa_synchronize(job_ids, timeout, 1, err_diag);

// Execute the Master job, wait for it and get exit code
rc = drmaa_run_job(job_id, jt, err_diag);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err_diag);
rc = drmaa_wexitstatus(&exitstatus, stat, err_diag);
}
rc = drmaa_exit(err_diag);
```

Fig. 5. Master-Worker application and its codification using the DRMAA standard.

Figure 6 shows the execution profile of three generations of the above Master-Worker applications. The average execution time per iteration is 120 seconds, with an average computational and transfer times per worker of 15.7, and 23.3 seconds respectively. In this case the total turnaround time is 260 seconds with an average CPU utilization of 22%.

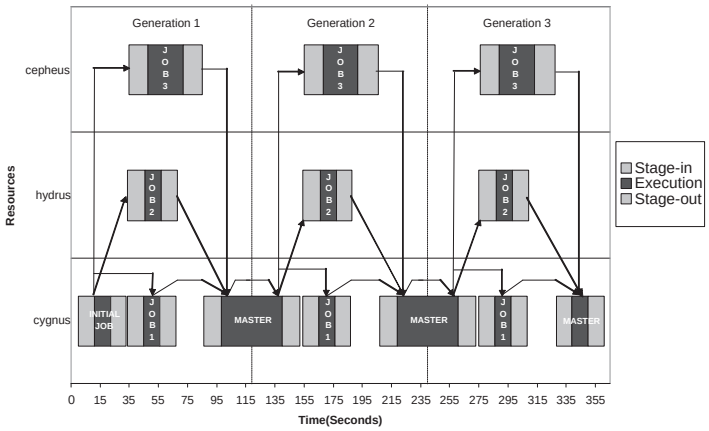


Fig. 6. Execution profile for three iterations of the Master-Worker application.

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.

It is foreseeable, as it happened with other standards like MPI or OpenMP, that DRMAA will be progressively adopted by most DRMS, making them easier and worthier to learn, thus lowering its barrier to acceptance, and making Grid application portable across DRMS adhered to the standard.

References

1. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. J. of Software – Practice and Experience (2004) (in press).
2. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report GFD-I.4, Scheduling Working Group – The Global Grid Forum (2001)
3. Huedo, E., Montero, R.S., Llorente, I.M.: Adaptive Scheduling and Execution on Computational Grids. J. of Supercomputing (2004) (in press).

4. Rajic, H., et al.: Distributed Resource Management Application API Specification 1.0. Technical report, DRMAA Working Group – The Global Grid Forum (2003)
5. Montero, R.S., Huedo, E., Llorente, I.M.: 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., Springer–Verlag (2003) 366–373
6. Badia, R.M., Labarta, J., Sirvent, R., Cela, J.M., Grima, R.: GridSuperscalar: A Programming Paradigm for Grid Applications. In: Workshop on Grid Applications and Programming Tools (GGF8). (2003)
7. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications **11** (1997) 115–128
8. Huedo, E., Montero, R.S., Llorente, I.M.: Adaptive Grid Scheduling of a High-Throughput Bioinformatics Application. In: Proc. of the 5th Intl. Conf. on Parallel Processing and Applied Mathematics (PPAM 2003). Volume 3019 of Lecture Notes in Computer Science., Springer–Verlag (2004) (in press).