Execution of Typical Scientific Applications on Globus-based Grids.[*]

J. Herrera[†] E. Huedo[‡] R.S. Montero[†]

I.M. Llorente[†,‡]

[†]Dpto. Arquitectura de Computadores y Automática
Facultad de Informática, Universidad Complutense
2 8040 Madrid (Spain)

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

Abstract

*One of the barriers that prevents the expansion and adoption of Grid technologies is the lack of a standard programming paradigm to port existing applications among different environments. The Distributed Resource Management Application API (DRMAA) has been proposed to aid the rapid development and distribution of these 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. As a case of study, we will consider in this paper the implementation of the NAS Grid Benchmarks with DRMAA. The DRMAA routines are supported by the functionality offered by the Grid*Way *framework, that provides the runtime mechanisms needed for transparently executing jobs on a dynamic Grid environment.*

Keywords: Grid Computing, Grid programming paradigms

1 Introduction

The deployment of existing applications across the Grid continues requiring a high level of expertise and a significant amount of effort, mainly due to the characteristics of the Grid: complexity, heterogeneity, dynamism and high fault rate. On the other hand, the lack of a standard programming paradigm for the Grid has prevented the portability of existing applications among different environments.

To deal with the characteristics of the Grid, we have developed Grid*W*ay [7]: a Globus submission framework that allows an easier and more efficient execution of jobs on dynamic Grid environments. Grid*W*ay automatically performs all the job scheduling steps [10], provides fault recovery mechanisms, and adapts job scheduling and execution to the changing Grid conditions [5].

Moreover, the *Distributed Resource Management Application API Working Group* (DRMAA-WG)[1], within the *Global Grid Forum* (GGF)[2], has recently 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, or higher level tools that use DRMAA, could aid scientists and engineers to express their computational problems by providing a portable direct interface to DRMS.

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 applications portable across DRMS adhered to the standard.

In this work, we discuss several aspects of the implementation of DRMAA within the Grid*W*ay framework, and investigate the suitability of the DRMAA specification to distribute typical scientific workloads across the Grid. In this sense the *NAS Grid Benchmarks* suite (NGB) [12] constitutes an excellent case-of-study, since it models distributed *communicating* applications typically executed on the Grid.

In Section 2, we first analyze several aspects involved in the efficient execution of distributed applications related

[*]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).

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

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

to the barriers to use the Grid, and how they are overcome by the Grid*W*ay framework. Section 3 briefly describes the DRMAA standard, and the development and execution process adopted in this work. Then, in Section 4, we illustrate how DRMAA can be used to implement several scientific applications paradigms. Finally, in Section 5, we evaluate the suitability of the DRMAA for implementing the NAS Grid Benchmarks (NGB). The paper ends in Section 6 with some conclusions.

2 The Grid*W*ay Framework

The Grid*W*ayframework [7] provides the following techniques to allow a robust an efficient execution of jobs in heterogeneous and dynamic Grids:

- Given the dynamic characteristics of the Grid, the Grid*W*ay framework periodically adapts the schedule to the available resources and their characteristics [5]. 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) [8].

- 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 [5].

- Grid*W*ay also provides the application with fault tolerance capabilities by capturing GRAM callbacks, by periodically probing the GRAM job-manager, and by inspecting the exit codes of each job.

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 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.

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

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

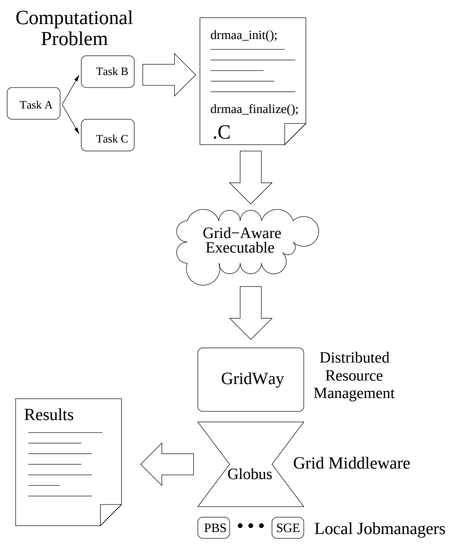


Figure 1. Development and execution cycle using the DRMAA interface

- Job template routines: `drmaa_set_attribute`, `drmaa_allocate_job_template` and `drmaa_delete_job_template`. This 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*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`. This routines are used to control (killing, resuming, suspending, etc..) and synchronize jobs, and monitor their status.

The DRMAA interface (see [9] 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.

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*W*ay in our case) will interact with the local job managers (Condor, PBS, SGE...) through the Grid middleware (Globus). This development and execution scheme with DRMAA, Grid*W*ay and Globus is depicted in figure 1. There are sev-

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

Name	VO	Model	Speed	OS	Memory	DRMS
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

eral 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.

The DRMAA standard can help in exploiting the intrinsic parallelism found in some application domains, as long as the underlying DRMS is responsible for the efficient and robust execution of each job. We expect that DRMAA will allow to explore several common execution techniques when distributing applications across the Grid [1]. For example fault tolerance could be improved by replicating job executions (redundant execution) [11], 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).

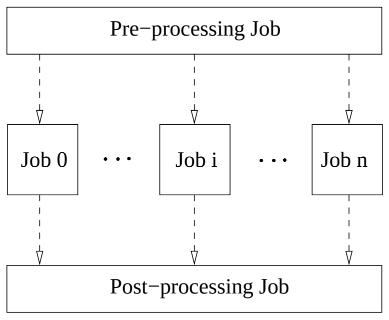
4 Experiences

We next demonstrate the ability of the Grid*W*ay 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-CAB research testbed, based on the Globus Toolkit [3], briefly described in table 1.

4.1 High-Throughput Computing Application

This example 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. 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 [7].

The general structure of a PSA and its implementation with DRMAA are shown in figure 2. An 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.



```
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
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);
```

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

In this case we consider an application that comprises the execution of 50 independent jobs. Each job calculates the determinant of an square matrix read from an input file (0.5MB). The overall execution time for the parameter sweep application is 40 minutes, with an average job turnaround time of 125 seconds. Figure 3 presents the dynamic productivity (jobs per minute) of the testbed during the execution of the PSA. Compared to the single host execution on the fastest machine in the testbed, these results represents a 35% reduction in the overall execution time.

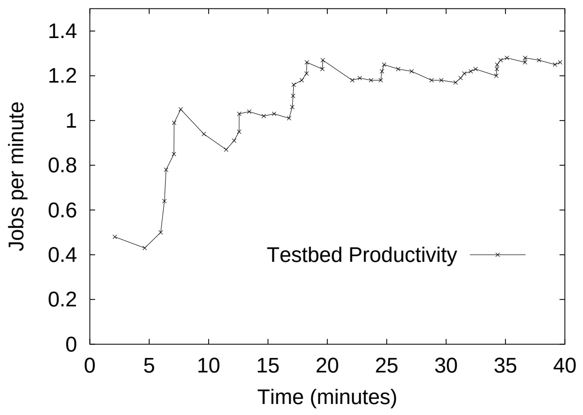
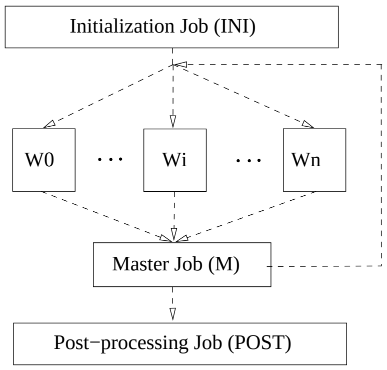


Figure 3. Testbed productivity in the execution of the high-throughput computing application.



```
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
    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, wait 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);
```

Figure 4. Master-Worker application and its codification using the DRMAA standard.

4.2 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 4).

In particular, we will consider a simple distribution scheme for a genetic algorithm. The master acts as the control process by creating worker jobs. Each worker task is initiated with an identical-sized sets of individuals, and evolves the population a fixed number of iterations. The master receives the results, evaluates the fitness function, and if convergence is not achieved it exchanges some individuals and repeats the process.

Figure 5 shows the execution profile of three generations of the above Master-Worker application. 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 360 seconds with an average CPU utilization of 22%.

5 The NAS Grid Benchmarks: A Case of Study

The *NAS Grid Benchmarks* [4] have been presented as a data flow graph (DFG) encapsulating an instance of a *NAS Parallel Benchmarks* (NPB) [2] code in each graph node, which *communicates* with other nodes by sending/receiving initialization data. The NGB suite models 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.

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 or problem sizes, in terms of mesh size and number of iterations. The four families defined in the NGB are:

- *Embarrassingly Distributed* (ED) models High Throughput Computing applications, whose structure and implementation with DRMAA has been discussed in section 4.1.

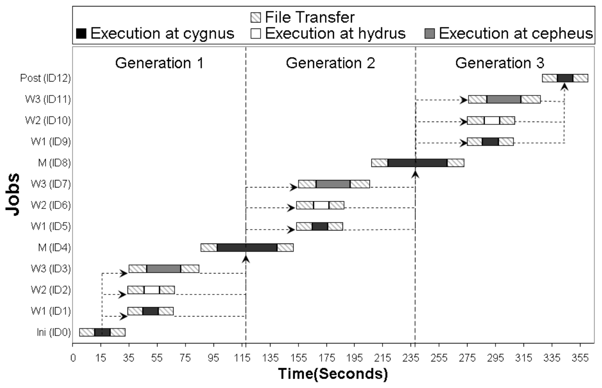


Figure 5. Execution profile for three iterations of the Master-Worker application.

- *Helical Chain* (HC) 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.

- *Visualization Pipe* (VP) represents chains of compound processes, like those encountered when visualizing flow solutions as the simulation progresses.

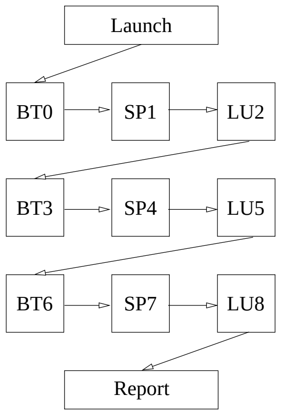
- *Mixed Bag* (MB) again involves the sequence of flow computation, post-processing, and visualization, but now the emphasis is on introducing asymmetry.

Grid benchmarks should provide a methodology to assess the functionality, performance and quality of service provided by a Grid environment. In this work we will concentrate in testing the functionality of our testbed made up of: local schedulers (fork and PBS), middleware (Globus toolkit), and high level tools (Grid*W*ayand DRMAA). In the NGB reports presented below, for the shake of completeness, we also include some performance metrics like job turnaround time, resource usage, and data transfers and execution times. Moreover the Globus overhead, as well as the Grid*W*ay overhead (scheduling time), are included in all measurements.

5.1 Helical Chain

The HC benchmark consists in a sequence of jobs that model long running simulations that can be divided in different tasks. Each job in the sequence uses the computed solution of its predecessor to initialize. Considering this dependences each job in the chain can be scheduled by Grid*W*ay once the previous job has finished (see figure 6).

Results of the HC benchmark (class A) for this scheduling strategy are shown in figure 7. The turnaround time is



```
// 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 6. Structure and implementation of the HC benchmark using DRMAA.

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 clearly reduces the performance obtained compared to the optimal turnaround time[3] of 6.18 minutes.

Nevertheless, this kind of applications can be submitted through the Grid*W*ay framework as a whole. The output files of each task in the chain are handled by the framework as checkpoint files. In this way the application can take advantage of the self-adapting capabilities provided by Grid*W*ay:

- The application can progressively change its resource requirements depending on the task of the chain to be executed. So, the application does not have to impose the most restricted set of requirements at the beginning, since it limits the chance for the application to begin execution [6].

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

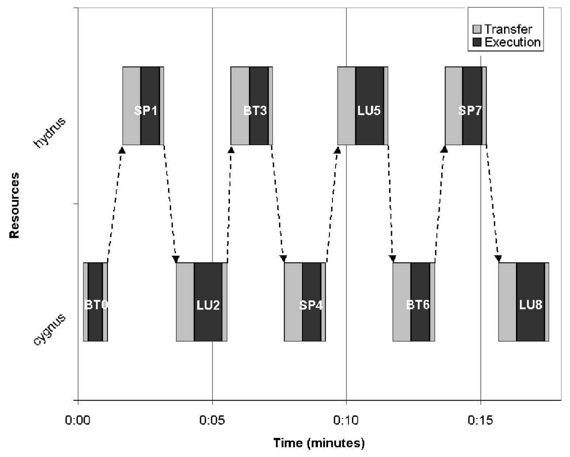


Figure 7. Results with the HC.A benchmark.

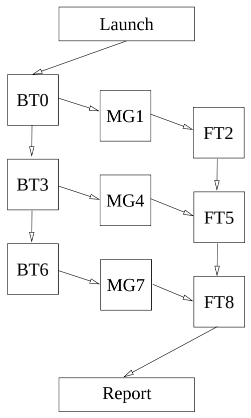
- The application can generate a performance profile to provide monitoring information in terms of application metrics (for example time to perform each task of the chain). This performance profile can be used to guide the job scheduling. Thus, the application could migrate to other host when some resources (disk space, free CPU...) are exhausted [6].
- The application can be migrated when a *better* resource is found in the Grid. In this case the time to finalize, and file transfer costs must be considered to evaluate if the migration is worthwhile [8].

When the HC benchmark is submitted as a whole job, the average resource usage increases to 91%, since the nine tasks of the same chain are scheduled to the same host (cygnus). In this case, the turnaround time is 7 minutes and the average execution time is reduced to 6.4 minutes. This supposes a decrement in the job turnaround time of 60% compared to the first scheduling strategy and an increment of only 11% compared to the optimal case.

5.2 Visualization Pipe and Mixed Bag

Although this kind of benchmarks could be serialized and adaptively executed like the previous one, they are more suitable to be implemented as a workflow application to exploit the parallelism they exhibit.

Since GridWay does not directly support workflow execution, we have developed a *workflow engine* taking advantage of the DRMAA programming interface, see figure 8. This algorithm follows a greedy approach, although different policies could be used to prioritize the jobs submitted to at each moment, for example, submit first the job with a more restricted set of requirements, with more computational work or with more jobs depending on it.



```
// Initialization
jobs[0].jt = bt; jobs[0].dep = "";\\
jobs[1].jt = mg; jobs[1].dep = "0";\\
jobs[2].jt = ft; jobs[2].dep = "1";\\
jobs[3].jt = bt; jobs[3].dep = "0";\\
jobs[4].jt = mg; jobs[4].dep = "3";\\
jobs[5].jt = ft; jobs[5].dep = "2 4";\\
jobs[6].jt = bt; jobs[6].dep = "3";\\
jobs[7].jt = mg; jobs[7].dep = "6";\\
jobs[8].jt = ft; jobs[8].dep = "5 7";\\
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 8. Structure and implementation of the *workflow engine* for the VP benchmark.

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 once with full parallelism).

Figure 9 shows the results for the VP.A 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%. Execution and transfer times are 22.93 and 8.1 minutes, respectively.

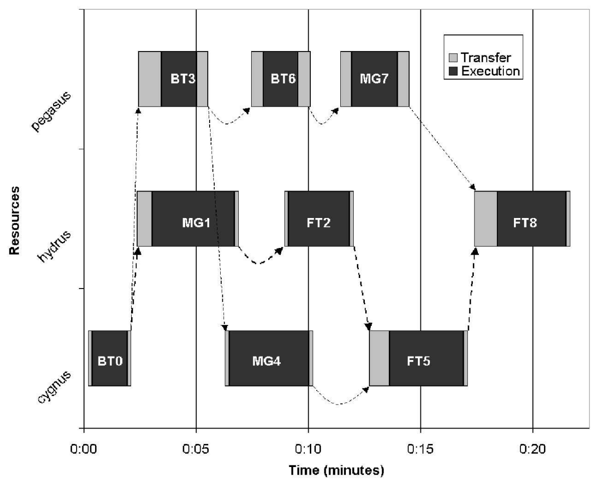


Figure 9. Results with the VP.A benchmark.

6 Conclusions

DRMAA can clearly aid the rapid development and distribution across the Grid of typical scientific applications. In this work, we have presented an implementation of DRMAA on top of the Grid*W*ay framework and Globus. The functionality, robustness and efficiency of this environment have been demonstrated through the execution of typical scientific applications, in particular the DRMAA implementation of the NGB suite.

The use of standard interfaces allows the comparison between different Grid implementations, since neither NGB nor DRMAA are tied to any specific Grid middleware. This kind of comparisons will be the object of future work.

We believe that DRMAA will become a standard for Grid application development. This would help users, making Grid applications portable across DRMS adhered to the standard, and DRMS vendors, making DRMS easier and worthier to learn.

References

[1] R. M. Badia, J. Labarta, R. Sirvent, J. M. Cela, and R. Grima. GridSuperscalar: A Programming Paradigm for Grid Applications. In *Workshop on Grid Applications and Programming Tools (GGF8)*, pages 26–37, 2003. to be published in J. of Grid Computing.

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

[3] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl. J. of Supercomputer Applications*, 11(2):115–128, 1997.

[4] 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.

[5] 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.

[6] E. Huedo, R. S. Montero, and I. M. Llorente. Adaptive Scheduling and Execution on Computational Grids. *J. of Supercomputing*, 2004. (in press).

[7] E. Huedo, R. S. Montero, and I. M. Llorente. Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications. In *Proc. of the 12th Intl. Conf. on Parallel, Distributed and Network based Processing (PDP 2004)*. IEEE Computer Society, February 2004.

[8] 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.

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

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

[11] P. Townend and J. Xu. Fault Tolerance within a Grid Environment. In S. Cox, editor, *Proc. of UK e-Science All Hands Meeting, Nottingham (UK)*, September 2003.

[12] 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.