# Embarrassingly Distributed and Master-Worker Paradigms on the Grid*

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

[1] Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas, Centro de Astrobiología (CSIC-INTA), Associated to NASA Astrobiology Institute, 28850 Torrejón de Ardoz, Spain.

[2] Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain.

**Abstract.** Grids constitute a promising platform to execute loosely coupled applications, which arise naturally in many scientific and engineering fields like bionformatics, computational fluid dynamics, particle physics, etc. In this paper, we describe our experiences in porting three scientific production codes to the Grid. Those codes follow typical computational models, namely: embarrassingly distributed and master-worker. In spite of their relatively simple computational structure, consisting of many "independent" tasks, their reliable and efficient execution on computational Grids involves several issues, due to both the dynamic nature of the Grid itself and the execution and programming requirements of the applications. The applications have been developed by using the DRMAA (Distributed Resource Management Application API) interface. 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. The experiments have been performed on Globus-based research testbeds that span heterogeneous resources in different institutions.

## 1 Introduction

It is becoming evident that the traditional concept of computing based on a homogeneous, and centrally managed environment is being displaced by a new model based on the exchange of information and the sharing of distributed resources by applications [1]. However, such applications often involve large amounts of data and/or computing elements and are not easily handled by today's Internet and web infrastructures. Grid technologies attempt to provide the support needed for such an infrastructure, enabling applications to use remote resources managed by widespread "virtual organizations".

The Globus project [2] has constructed an open-source toolkit to build computational grids, implementing a set of non-proprietary protocols for securely

* This research was supported by Ministerio de Ciencia y Tecnología, through the research grant TIC 2003-01321 and 2002-12422-E, and by Instituto Nacional de Técnica Aeroespacial "Esteban Terradas" (INTA) – Centro de Astrobiología.

identifying, allocating and releasing resources from the Grid. Due to its open-source nature and its increasing popularity, the Globus toolkit has become a *de facto* standard in Grid computing. Globus is a core Grid middleware that provides the following components, which can be used separately or altogether, to support Grid applications: GRAM (Globus Resource Allocation Manager), GASS (Global Access to Secondary Storage), GSI (Grid Security Infrastructure), MDS (Monitoring and Discovery Service), and GridFTP. These services allow secure and transparent access to resources across multiple administrative domains, and serve as building blocks to implement the stages of Grid scheduling [3].

Probably, one of the most challenging problems that the Grid computing community has to deal with is the fact that Grids are highly dynamic and faulty environments. *Adaptive scheduling* has been widely studied in the literature [4–6], and it has been demonstrated that periodic re-evaluation of the schedule can result in significant improvements in both performance and fault tolerance. On the other hand, *Adaptive execution* can improve application performance by adapting it to the dynamic availability, capacity and cost of Grid resources. Moreover, an application should be able to migrate to a new resource to satisfy its new requirements or preferences *(self-adaption)*.

In a previous work [7], we have presented a new Globus experimental framework that allows an easier and more efficient execution of jobs on a dynamic Grid environment in a "submit and forget" fashion. The Grid*Way* framework provides resource selection, job scheduling, reliable job execution, and automatic job migration to allow a robust and efficient execution of jobs in dynamic and heterogeneous Grid environments based on the Globus toolkit [2]. Moreover, Grid*Way* provides support for the Distributed Resource Management Application API (DRMAA)[8].

The aim of this paper is to present our experiences on using the Grid to execute three real applications belonging to the Bioinformatics, Planetary Geology and Optimization research areas. These applications follow typical loosely coupled models: embarrassingly distributed and master-worker. We also will show that DRMAA is a suitable and portable framework to express those distributed communicating paradigms. The tasks that made up the above computing models could require different complexity or instruction streams. Therefore, adaptive scheduling is again required to deal with their asynchronous temporal structure.

The main features of the Grid*Way* framework and its code porting support are described in Section 2 and 3 respectively. The synchronous and asynchronous embarrassingly distributed models in the context of a Bioinformatics and a Planetary Geology applications are analyzed in Section 4 and 5. Section 6 deals with the master-worker paradigm using a grid-oriented genetic algorithm as case of study. At last, the main conclusions and acknowledgments of this research are summarized in Section 7.

## 2 Main Features of the Grid *W*ay Framework

Grid *W*ay [7] is a Globus-based submission loosely-coupled framework that achieves an efficient execution of applications by combining:

- *Adaptive scheduling*: Reliable schedules can only be issued considering the dynamic characteristics of the available Grid resources [5]. In general, adaptive scheduling can consider factors such as availability, performance, load or proximity, which must be properly scaled according to the application needs and preferences. Grid *W*ay periodically gathers information from the Grid and from the running or completed jobs to adaptively schedule pending tasks according to the application demands and Grid resource status.
- *Adaptive execution*: In order to obtain a reasonable degree of both application performance and fault tolerance, a job must be able to migrate among the Grid resources adapting itself to events dynamically generated by both the Grid and the running application [9]. Grid *W*ay evaluates each *rescheduling event* to decide if a migration is feasible and worthwhile.
- *Reuse of common files*: Efficient execution of some applications profiles like parameters sweep can only be achieved by re-using shared files between tasks [10]. This is specially important not only to reduce the file transfer overhead, but also to prevent the saturation of the file server where these files are stored. Reuse of common files between tasks simultaneously submitted to the same resource is achieved by storing some files declared as *shared* in the GASS cache [11].
- *Fault tolerance*: The failures that may occur in a Grid can fall in a wide range of categories such as execution faults, network errors, hardware faults, configuration problems, etc (see for example [12]). Fault detection and recovery depends on the nature of the failure, and it may involve retrying, migrating or restarting the execution of an application.

## 3 Grid *W*ay Code Porting Support

The Distributed Resource Management Application API (DRMAA) Working Group[1], within the Global Grid Forum (GGF)[2], has developed an API specification that allows a high-level interaction with Distributed Resource Management Systems (DRMS). The DRMAA standard constitutes a homogeneous interface to different DRMS to handle job submission, monitoring and control, and retrieval of finished job status.

DRMAA allows scientists and engineers to express their computational problems in a Grid environment. The capture of the job exit code allow users to define complex jobs, where each depends on the output and exit code from the previous job. They may even involve branching, looping and spawning of subtasks, allowing the exploitation of the parallelism on the work flow of certain type of applications.

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

The target application source code does not have to be modified. However, due to the high fault rate and the dynamic rescheduling, the application should generate `restart files` in order to restart the execution from a given point. If these files are not provided, the job is restarted from the beginning. User-level checkpointing managed by the programmer must be implemented because system-level checkpointing is not currently possible among heterogeneous resources. In order to adapt the execution of a job to its dynamic demands, the application can specify its host requirements through a `requirement expression`. Also, in order to prioritize the resources that fulfill the requirements according to its runtime needs, the application must specify its hosts preferences through a `ranking expression`. The `ranking expression` uses a performance model to estimate the job turnaround time as the sum of execution and transfer time, derived from the performance and proximity of the candidate resources [13].

In this work we will analize the following loosely coupled paradigm:

– Embarrassingly distributed: Applications that can be obviously divided into a number of independent tasks. The application is asynchronous when require distinct instruction streams and so different execution times. A sample of this schema with its DRMAA implementation is showed in the figure 1.

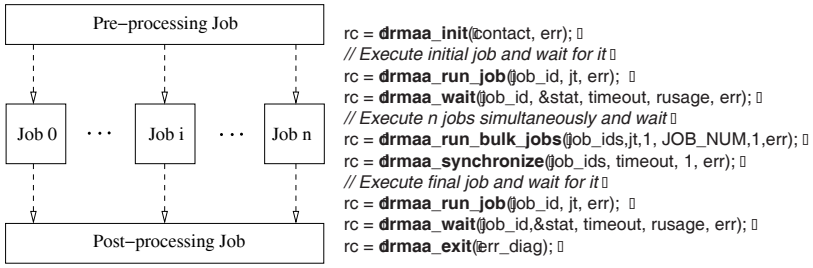


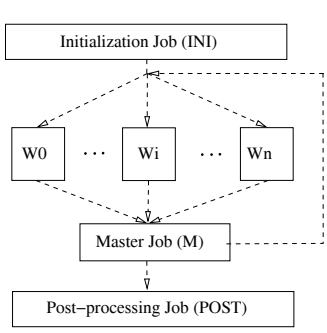

**Fig. 1.** Embarrassingly distributed paradigm and its codification using the DRMAA standard.

– Master-worker: A Master task assigns a description (input files) of the task to be performed by each Worker. Once all the Workers are completed, the Master task performs some computations in order to evaluate a stop criterion or to assign new tasks to more workers. Again, it could be synchronous or asynchronous. Figure 2 shows a example of Master-worker optimization loop and a DRMAA implementation sample.

## 4 Synchronous Embarrassingly Distributed Paradigm

### 4.1 A Protein Structure Prediction Application

Bioinformatics, which has to do with the management and analysis of huge amounts of biological data, could enormously benefit from the suitability of




```
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);
```

**Fig. 2.** Master-Worker paradigm and its codification using the DRMAA standard.

the Grid to execute high-throughput applications. In the context of this paper, we consider a Bioinformatics application aimed at predicting the structure and thermodynamic properties of a target protein from its amino acid sequences. The algorithm, tested in the 5th round of Critical Assessment of techniques for protein Structure Prediction (CASP5), aligns with gaps the target sequence with all the 6150 non-redundant structures in the Protein Data Bank (PDB), and evaluates the match between sequence and structure based on a simplified free energy function plus a gap penalty term. The lowest scoring alignment found is regarded as the prediction if it satisfies some quality requirements. In such cases, the algorithm can be used to estimate thermodynamic parameters of the target sequence, such as the folding free energy and the normalized energy gap [14].

To speed up the analysis and reduce the data needed, the PDB files are preprocessed to extract the contact matrices, which provide a reduced representation of protein structures. The algorithm is then applied twice, the first time as a fast search, in order to select the 100 best candidate structures, the second time with parameters allowing a more accurate search of the optimal alignment.We have applied the algorithm to the prediction of thermodynamic properties of families of orthologous proteins, i.e. proteins performing the same function in different organisms. If a representative structure of this set is known, the algorithm predicts it as the correct structure.

### 4.2 Results

The experiments presented in this section were conducted on a research testbed based on the Globus Toolkit described in table 1. This testbed is highly heterogeneous and it is made up of resources belonging to two different sites interconnected by a "public" non-dedicated network.

The following set of experiments shows how adaptive scheduling improves the performance and adaptive execution provides fault tolerance by restarting the execution from the beginning.
Let us consider an experiment consisting in 88 tasks, each of them applies the

**Table 1.** Research testbed for the Protein Structure Prediction Application.

| Name | Site | Architecture | Speed | Mem. | OS | DRMS |
|---|---|---|---|---|---|---|
| ursa | DACYA-UCM | 1×UltraSPARC-IIe | 500MHz | 256MB | Solaris | fork |
| draco | DACYA-UCM | 1×UltraSPARC-I | 167MHz | 128MB | Solaris | fork |
| pegasus | DACYA-UCM | 1×Pentium 4 | 2.4GHz | 1GB | Linux 2.4 | fork |
| solea | DACYA-UCM | 2×UltraSPARC-II | 296MHz | 256MB | Solaris | fork |
| babieca | LCASAT-CAP | 5×Alpha Ev67 | 450MHz | 256MB | Linux 2.2 | PBS |

structure prediction algorithm to a different sequence of the *Triosephosfate Isomerase* enzyme which is present in different organisms. The overall execution time for the Bioinformatics application, when all the machines in the testbed are available, is 7.15 hours with an average throughput of 12 jobs per hour. This experiment was reproduced in two new situations. In the first case, babieca is shut down for maintenance in the middle of the experiment during one hour. As a consequence, the framework stops scheduling jobs in this host and the average job turnaround is reduced to 10 jobs per hour. Once babieca is restarted, Grid*W*ay schedules jobs on it again and the throughput increases to nearly 12 jobs per hour.

The second experiment starts with pegasus unavailable, and it is *plugged* in to the Grid 3.5 hours after the experiment started. As could be expected, the absence of pegasus decreases the average throughput (9 jobs per hour), and increases the overall execution time to 9.8 hours. Figure 3 shows the dynamic job turnaround time during the execution of the application in the above situations.

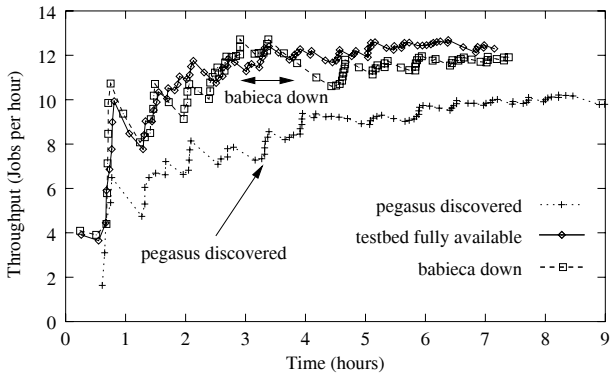



**Fig. 3.** Dynamic throughput in the execution of the application when the testbed is fully available, when pegasus is discovered and when babieca is down.

## 5 Asynchronous Embarrassingly Distributed Paradigm

### 5.1 A Mars Impact Cratering Application

Our target application analyzes the threshold diameter for cratering the seafloor of an hypothetical martian sea during the first steps of an impact. Results of this analysis can be used to develop a search criteria for future investigations, including techniques that will be used in future Mars exploration missions to detect buried geological structures using ground penetrating radar surveys, as the ones included in the ESA Mars Express and planned for NASA 2005 missions.The discovery of marine-target impact craters on Mars would also help to address the ongoing debate of whether large water bodies occupied the northern plains of Mars and help to constrain future paleoclimatic reconstructions [15]. In any case, this kind of studies requires an huge amount of computing power, which is not usually available within a single organization.

We deal in this study with vertical impacts, as they reduce to 2D problems using the radial symmetry. All simulations were conduced with spherical projectiles. The non-uniform computational mesh of the coarse simulations consists of 151 nodes in horizontal direction and 231 nodes in vertical direction and the total nodes describes half of the crater domain because of axial symmetry. The mesh size progressively increases outwards from the center with a 1.05 coefficient to have a larger spatial domain. The central cell region around the impact point where damage is greater, more extended than the crater area, is a regular mesh 80 nodes resolution in both $x$ and $y$ direction, and also describes half of the damaged zone. We use a resolution of 10 nodes to describe the radial projectile.

For a fixed water depth, we used 8 cases of projectile diameter in the range of 60 m to 1 Km, and 3 cases of impactor velocity: 10, 20 and 30 Km/s. Calculations were performed for 3 cases of water depth: 100, 200 and 400 m. Once fixed the projectile velocity and the water depth of the hypothetical ocean, we search to determine the range for the critical diameter of the projectile which can crater the seafloor [16]. Therefore, in this study we have to compute 72 cases. Its execution on a Grid environment allows to obtain the diameter range of interest within the research cycle time.

### 5.2 Results

Table 2 shows the characteristics of the machines in the research testbed, based on the Globus toolkit 2.4. The testbed joins resources from five sites, all of them connected by the Spanish Research and Education Network, RedIRIS. This organization results in a highly heterogeneous testbed, since it presents several architectures, processor speeds, DRMS and network links.

The execution time for each task is different and, what is more important, unknown beforehand, since the convergence of the iterative algorithm strongly depends on input parameters. Moreover, there is an additional difference generated by the changing resource load, availability and characteristics. Therefore, adaptive scheduling is crucial for this application. Figure 4 shows the dynamic

**Table 2.** Research testbed for the Mars Impact Cratering Application.

| Name | Site | Architecture | Speed | Mem. | OS | DRMS |
|---|---|---|---|---|---|---|
| hydrus | DACYA-UCM | 1×Intel P4 | 2.5GHz | 512MB | Linux 2.4 | fork |
| cygnus | DACYA-UCM | 1×Intel P4 | 2.5GHz | 512MB | Linux 2.4 | fork |
| cepheus | DACYA-UCM | 1×Intel PIII | 600MHz | 256MB | Linux 2.4 | fork |
| aquila | DACYA-UCM | 1×Intel PIII | 700MHz | 128MB | Linux 2.4 | fork |
| babieca | LCASAT-CAB | 5×Alpha Ev67 | 450MHz | 256MB | Linux 2.2 | PBS |
| platon | REDIRIS | 2×Intel PIII | 1.4GHz | 512MB | Linux 2.4 | fork |
| heraclito | REDIRIS | 1×Intel Cel. | 700MHz | 256MB | Linux 2.4 | fork |
| ramses | DSIC-UPV | 5×Intel PIII | 900MHz | 512MB | Linux 2.4 | PBS |
| khafre | CEPBA-UPC | 4×Intel PIII | 700MHz | 512MB | Linux 2.4 | fork |

turnaround time during the execution of this experiment. Total experiment time was 4.64 hours (4 hours, 38 minutes and 33 seconds), so the achieved throughput was 3.87 minutes (3 minutes and 52 seconds) per job, or likewise, 15.5 jobs per hour.

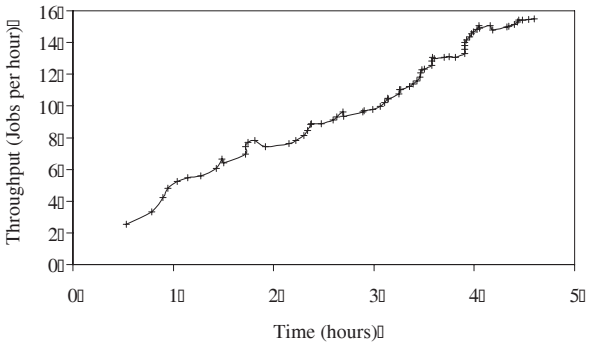



**Fig. 4.** Dynamic throughput, in terms of average turnaround time per job.

## 6 Master-Worker Paradigm

### 6.1 A Grid Oriented Genetic Algorithm

Genetics Algorithms (GA) are search algorithms inspired in natural selection and genetic mechanisms. GAs use historic information to find new search points and reach an optimal problem solution. In order to increase the speed and the

efficiency of sequential GAs, several Parallel Genetic Algorithm (PGA) alternatives have been developed. PGAs have been successfully applied in previous works, (see for example [17]), and in most cases, they succeed to reduce the time required to find acceptable solutions.

In order to develop efficient Grid-oriented genetic algorithms [18], the dynamism and heterogeneity of a Grid environment must be considered. In this way, traditional load-balancing techniques could lead to a performance slow-down, since, in general the performance of each computing element can not be guaranteed during the execution. Moreover, some failure recovery mechanisms should be included in such a faulty environment. Taking into account the above considerations we will use a fully connected multi-deme genetic algorithm. In spite of this approach represents the most intense communication pattern (all demes exchange individuals every generation), it does not imply any overhead since the population of each deme is used as checkpoint files, and therefore transferred to the client in each iteration.

The initial population is uniformity distributed among the available number of nodes, and then a sequential GA is locally executed over each subpopulation. The resultant subpopulations are transferred back to the client, and worst individuals of each subpopulation are exchanged with the best ones of the rest. Finally, a new population is generated to perform the next iteration [19]. The scheme of this algorithm is depicted in figure 5.

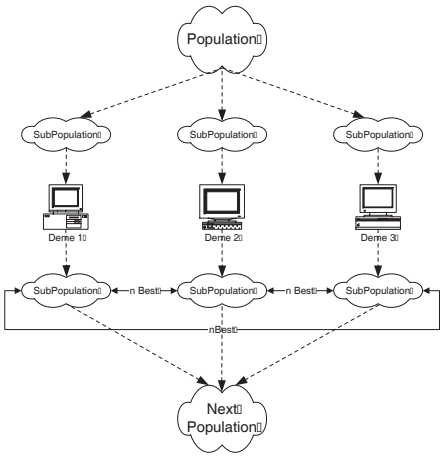



**Fig. 5.** Schema of fully-connected multi-deme genetic algorithm, with three computing nodes

The previous algorithm may incur in performance looses when the relative computing power of the nodes involved in the solution process greatly defers, since the iteration time is determined by the slowest machine. In order to prevent these situations we allow an *asynchronous* communication pattern between

demes. In this way, information exchange only occurs between a fixed number of demes, instead of synchronizing the execution of all subpopulations. The minimum number of demes that should communicate in each iteration depends strongly on the numerical characteristics of the problem. We refer to this characteristic as *dynamic connectivity*, since the demes that exchange individuals differs each iteration.

### 6.2 Results

We evaluate the functionality and efficiency of the Grid-oriented Genetic Algorithm described above in the solution of the One-Max problem [20]. The One-Max is a classical benchmark problem for genetic algorithm computations, and it tries to evolve an initial matrix of zeros in a matrix of ones. In our case we consider an initial population of 1000 individuals, each one a 20x100 zero matrix. The sequential GA executed on each node performs a fixed number of iterations (50), with a mutation and crossover probabilities of 0,1% and 60%, respectively. The exchange probability of best individuals between demes is 10%.

The following experiments were conducted on a research testbed made up of three different sites, and based on the Globus Toolkit 2.4. See table 3 for a brief description of the resources in the testbed.

**Table 3.** Research testbed for the Grid Oriented Genetic Algorithm.

| Name | Site | Architecture | Speed | Memory | OS | DRMS |
|---|---|---|---|---|---|---|
| hydrus | DACYA-UCM | 1×Intel P4 | 2.5GHz | 512MB | Linux 2.4 | fork |
| cygnus | DACYA-UCM | 1×Intel P4 | 2.5GHz | 512MB | Linux 2.4 | fork |
| aquila | DACYA-UCM | 1×Intel PIII | 700MHz | 128MB | Linux 2.4 | fork |
| babieca | LCASAT-CAB | 5×Alpha Ev67 | 450MHz | 256MB | Linux 2.2 | PBS |

Besides the need for both adaptive scheduler and execution we would like to remark the advantages of the DRMAA API to aid the rapid development and distribution across the grid of typical computational models. Figure 6 shows the execution profile of 4 generations of the GOGA, with a 5-way *dynamic connectivity*. Each subpopulation has been traced, and labelled with a different number ($P_{deme}$). As can be shown, individuals are exchanged between subpopulations $P1, P2, P3, P4, P5$ in the first generation; while in the third one the subpopulations used are $P1, P2, P4, P7, P8$. In this way the *dynamic connectivity*, introduces another degree of randomness since the demes that communicate differ each iteration and depend on the dynamism of the Grid.

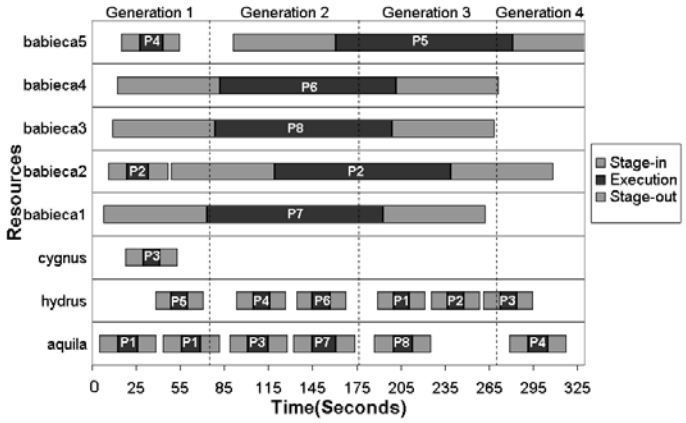



**Fig. 6.** Execution profile of four generations of the 11, each subpopulation has been labelled with $P_{deme}$

## 7 Conclusions and Acknowledgments

We have shown how an adaptive approach for job scheduling and execution is required due to both the changing conditions of the Grid resources and the asynchronous nature of some applications. The functionality, robustness and efficiency of a Grid environment consisting of Grid$W$ay and Globus have been demonstrated through the execution of typical scientific applications. We have demonstrated that DRMAA is a suitable and portable framework to express the applications studied in this work: a protein structure prediction application, a Mars impact cratering application and a Grid oriented genetic algorithm.

We would like to thank all the research centers that generously contribute resources to the experimental testbed. They are the European Center for Parallelism of Barcelona (CEPBA) in the Technical University of Catalonia (UPC), the Department of Computer Architecture and Automatics (DACyA) in the Complutense University of Madrid (UCM), the Department of Information Systems and Computation (DSIC) in the Polytechnic University of Valencia (UPV), the Laboratory of Advanced Computing, Simulation and Telematic Applications (LCASAT) in the Center for Astrobiology (CAB), and the Spanish National Research and Education Network (RedIRIS). All of them are part of the Spanish Thematic Network on Grid Middleware.

## References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufman (1999)
2. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl. J. Supercomputer Applications **11** (1997) 115–128

3. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report WD8.5, The Global Grid Forum (2001) Scheduling Working Group.
4. Buyya, R., D.Abramson, Giddy, J.: A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems (2002) Elsevier Science.
5. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In: Proceedings of the 9th Heterogeneous Computing workshop (HCW2000). (2000) Cancun, Mexico.
6. Allen, G., et al.: The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. International Journal of High-Performance Computing Applications **15** (2001)
7. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. Intl. J. Software – Practice and Experience (SPE) **34** (2004) 631–651
8. Herrera, J., Huedo, E., Montero, R.S., Llorente, I.M.: Execution of Typical Scientific Application On Globus-based Grids. IEEE Computer Society (2004)
9. Vadhiyar, S., Dongarra, J.: A Performance Oriented Migration Framework for the Grid. In: Proceedings of the 3rd IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGrid). (2003)
10. Giersch, A., Robert, Y., Vivien, F.: Scheduling Tasks Sharing Files on Heterogeneous Master-Slave Platforms. In: Proc. 12th Euromicro Conf. Parallel, Distributed and Network-based Processing (PDP 2004), IEEE CS (2004) 364–371
11. Huedo, E., Montero, R.S., Llorente, I.M.: Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications. In: Proc. 12th Euromicro Conf. Parallel, Distributed and Network-based Processing (PDP 2004), IEEE CS (2004) 28–33
12. Medeiros, R., Cirne, W., Brasileiro, F., Sauv, J.: Faults in Grids: Why Are They so Bad and What Can Be Done about It? In: Proc. of the 4th Intl. Workshop on Grid Computing (Grid 2003). (2003)
13. Huedo, E., Montero, R.S., Llorente, I.M.: Experiences on Grid Resource Selection Considering Resource Proximity. In: Proc. of 1st European Across Grids Conf. Lecture Notes on Computer Science (2003)
14. van Ham, R., et al.: Reductive Genome Evolution in *buchnera aphidicola*. Proc. Natl. Acad. Sci. USA **100** (2003) 581–586
15. Ormö, J., Dohm, J.M., Ferris, J.C., Lepinette, A., Fairén, A.: Marine-Target Craters on Mars? An Assessment Study. Meteoritics & Planetary Science **39** (2004) 333–346
16. Housen, K.R., Schmidt, R.M., Holsapple, K.A.: Crater Ejecta Scaling Laws: Fundamental Forms Based on Dimensional Analysis. Journal of Geophysical Research **88** (1983) 2485–2499
17. Kang, L., Chen, Y.: Parallel Evolutionary Algorithms and Applications. (1999)
18. Imade, H., Morishita, R., Ono, I., Ono, N., Okamoto, M.: A Grid-oriented Genetic Algorithm Framework for Bioinformatics. New Generation Computing **22** (2004) 177–186
19. Cant-Paz, E.: A Survey of Parallel Genetic Algorthms (1999)
20. Schaffer, J., Eshelman, L.: On Crossover as an Evolutionary Viable Strategy. In Belew, R., Booker, L., eds.: Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann (1991) 61–68