Scientific Programming 13 (2005) 19–30
IOS Press

Development and execution of an impact cratering application on a computational Grid[1]

E. Huedo[a,*], A. Lepinette[a], R.S. Montero[b], I.M. Llorente[a,b] and L. Vázquez[a,c]
[a]*Laboratorio de Computación Avanzada, Simulación, y Aplicaciones Telemáticas, Centro de Astrobiología (CSIC-INTA), 28850 Torrejón de Ardoz, Spain*
[b]*Departamento de Arquitectura de Computadores, y Automática, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain*
[c]*Departamento de Matemática Aplicada, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain*

Abstract. Impact cratering is an important geological process of special interest in Astrobiology. Its numerical simulation comprises the execution of a high number of tasks, since the search space of input parameter values includes the projectile diameter, the water depth and the impactor velocity. Furthermore, the execution time of each task is not uniform because of the different numerical properties of each experimental configuration. Grid technology is a promising platform to execute this kind of applications, since it provides the end user with a performance much higher than that achievable on any single organization. However, the scheduling of each task on a Grid involves challenging issues due to the unpredictable and heterogeneous behavior of both the Grid and the numerical code. This paper evaluates the performance of a Grid infrastructure based on the Globus toolkit and the Grid*Way* framework, which provides the adaptive and fault tolerance functionality required to harness Grid resources, in the simulation of the impact cratering process. The experiments have been performed on a testbed composed of resources shared by five sites interconnected by RedIRIS, the Spanish Research and Education Network.

1. Introduction

Impact cratering is an important geological process of special interest in Astrobiology that affects the surface of nearly all celestial bodies such as planets and satellites. The detailed morphologies of impact craters (see [23] for a detailed description) show many variations from small craters to craters with central peaks. Furthermore, a water layer at the target influences lithology and morphology of the resultant crater. Therefore, marine-target impact cratering simulation plays an important role in studies which involve hypothetical Martian seas [22].

Our target application analyzes the threshold diameter for cratering the seafloor of a 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 [22]. In any case, this kind of study requires a huge amount of computing power, which is not usually available within a single organization.

In order to determine the range for the critical diameter of the projectile which can crater the seafloor, we will perform a high number of simulations. Each computational task solves the equations of motion for

[1]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.
*Corresponding author. Tel.: +34 91 520 64 28; Fax: +34 91 520 10 74; E-mail: huedoce@inta.es.

ISSN 1058-9244/04/$17.00

compressible media combined with the equations of state, over a subset of input parameter values, namely: projectile diameter, the water depth and the impactor velocity. Additionally, the execution time of each computational task is not uniform because of the different numerical properties of each experimental configuration.

Grid technology provides a way to access the geographically distributed resources needed for executing compute-intensive Parameter Sweep Applications (PSA), like the one described above. In spite of the relatively simple structure of a PSA, its reliable and efficient execution on computational Grids involves several issues, mainly due to the nature of the Grid itself. In particular, one of the most challenging problems that the Grid computing community has to deal with is the fact that Grids present unpredictable changing conditions, namely: high fault rate and dynamic resource availability, load and cost. Adaptive scheduling has been widely studied in the literature [1–3,26,27] and is generally accepted as the cure to the dynamism of the Grid. Moreover, the different execution times for different tasks makes critical the use of an adaptive approach. The Grid*W*ay framework [17] achieves the robust and efficient execution of PSAs by combining adaptive scheduling and execution, to reflect the dynamic Grid characteristics; and re-use of common files between tasks, to reduce the file transfer overhead. The aim of this paper is to describe and analyze the results obtained in the simulation of the impact cratering process in a Grid infrastructure based on Globus and Grid*W*ay.

In Section 2 we present the highly heterogeneous testbed used in this work. The functionality and internals of the Grid*W*ay framework are briefly described in Section 3. The target application is outlined in Section 4. We demonstrate that a Grid testbed based on Globus and Grid*W*ay provides the functionality and reliability needed to execute the simulation tasks. The performance results are described in Section 6, where some performance metrics in order to evaluate the Grid computing platform are proposed: the Grid speedup metric, which quantifies the benefits of being part of a Grid, and the resource load variability, which could be used to adjust the components of the Grid infrastructure in order to achieve higher efficiencies. Finally, some conclusions are presented in Section 7.

2. The research testbed

The management of jobs within the same department is addressed by many research and commercial systems [8]: Condor, Load Sharing Facility, Sun Grid Engine, Portable Batch System, LoadLeveler, etc. Some of these tools, such as Sun Grid Engine Enterprise Edition [16], also allow the interconnection of multiple departments within the same administrative domain. Other tools, such as Condor Flocking [9], even allow the interconnection of multiple domains, as long as they run the same distributed resource management software. However, they are unsuitable in computational Grids where resources are scattered across several administrative domains, each with its own security policies and distributed resource management systems.

The Globus toolkit [10] provides the services and libraries needed to enable secure multiple domain operation within different resource management systems and access policies. Globus is a core Grid middleware that provides the following components, which can be used separately or together, to support Grid applications: Grid Security Infrastructure (GSI), Grid Resource Allocation Manager (GRAM), Global Access to Secondary Storage (GASS), Monitoring and Discovery Service (MDS), GridFTP and Replica Management Services.

Table 1 shows the characteristics of the machines in the research testbed, based on the Globus toolkit 2.X [10]. The testbed joins resources from five sites, all of them connected by the Spanish Research and Education Network, RedIRIS. The geographical distribution and interconnection network of sites are shown in Fig. 1. This organization results in a highly heterogeneous testbed, since it presents several resources (PCs, clusters, SMP servers), processor architectures and speeds, Resource Management Systems (RMS), network links, etc. In the following experiments, `cepheus` is used as client, and holds all the input files and receives the simulation results. In the case of clusters, we have limited to 5 the number of simultaneously used nodes, in order not to saturate these systems since they are at production level.

3. The Grid*W*ay framework

The Globus toolkit [10] supports the submission of applications to remote hosts by providing resource discovery, resource monitoring, resource allocation, and job control services. However, the user is responsible for manually performing all the submission stages in order to achieve any functionality: selection, preparation, submission, monitoring, migration and termination [24,25]. Hence, the development of applica-

Table 1
Characteristics of the machines in the research testbed

Name	Site	Nodes	Processors	Speed	Mem.	RMS
hydrus	DACYA-UCM	1	1×Intel P4	2.5 GHz	512 MB	fork
cygnus	DACYA-UCM	1	1×Intel P4	2.5 GHz	512 MB	fork
cepheus	DACYA-UCM	1	1×Intel PIII	600 MHz	256 MB	fork
aquila	DACYA-UCM	1	1×Intel PIII	700 MHz	128 MB	fork
babieca	LCASAT-CAB	30	1×Alpha Ev67	450 MHz	256 MB	PBS
platon	REDIRIS	1	2×Intel PIII	1.4 GHz	512 MB	fork
heraclito	REDIRIS	1	1×Intel Cel.	700 MHz	256 MB	fork
ramses	DSIC-UPV	12	2×Intel PIII	900 MHz	512 MB	PBS
khafre	CEPBA-UPC	1	4×Intel PIII	700 MHz	512 MB	fork

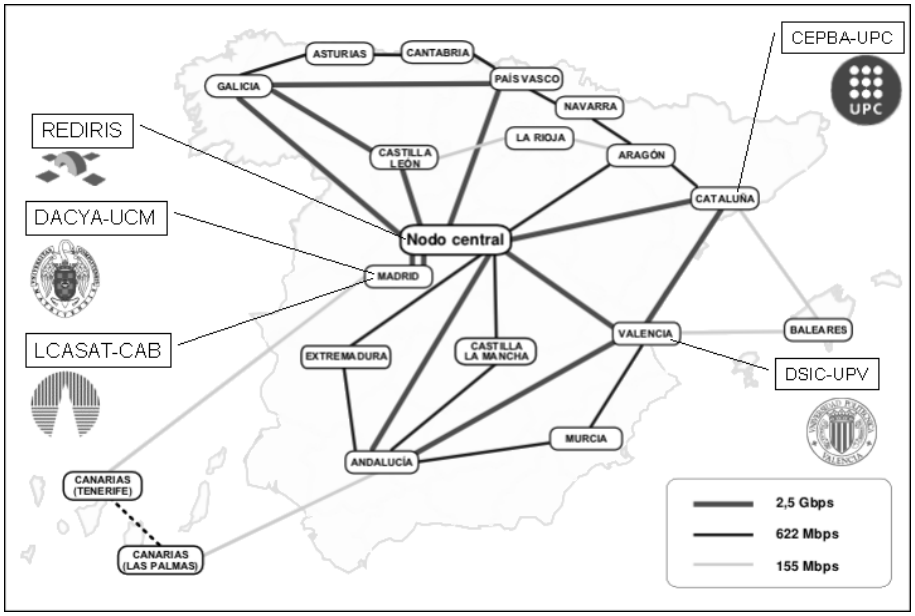

Fig. 1. Geographical distribution of the sites in Spain and interconnection network provided by RedIRIS.

tions for the Grid continues requiring a high level of expertise due to its complex nature. Moreover, Grid resources are also difficult to efficiently harness due to their heterogeneous and dynamic nature. In a previous work [17], 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 [10].

3.1. GridWay architecture

The architecture of the Grid*Way* framework is depicted in Fig. 2. The user interacts with the framework through a programming or command line interface, which forwards client requests (*submit*, *kill*, *stop*, *resume*) to the *dispatch manager*. The *dispatch manager* periodically wakes up, and tries to submit pending and rescheduled jobs to Grid resources. Once a job is allocated to a resource, a *submission manager* and a *performance monitor* are started to watch over its correct and efficient execution (see [17] for a detailed description of these components).

The framework has been designed to be modular, thus allowing extensibility and improvement of its capabilities. The following modules can be set on a per job basis:

– The *resource selector* module, which is used by the *dispatch manager* to build a prioritized list of candidate resources following the preferences and requirements provided by the user.

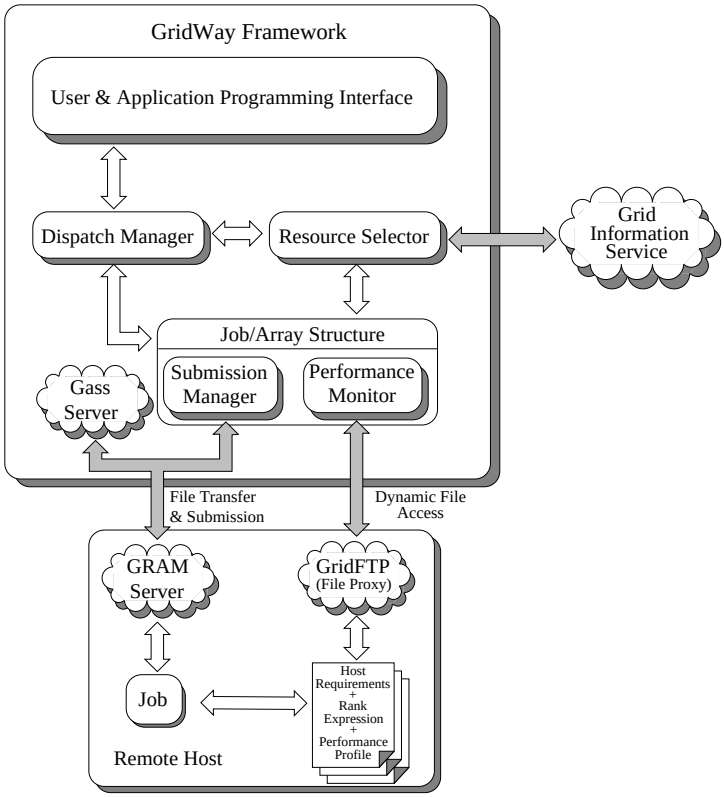

Fig. 2. The Grid*W*ay architecture.

– The *performance evaluator* module, which is used by the *performance monitor* to periodically evaluate the application performance, usually by analyzing a performance profile generated by the running application or by a monitor started along with the application.

3.2. Job execution

Job execution is performed in three steps by the following modules:

– The *prologue* module, which is responsible for creating the remote experiment directory and transferring the executable and all the files needed for remote execution, such as input or restart files corresponding to the execution architecture. These files can be specified as local files in the experiment directory or as remote files stored in a file server through a GridFTP URL. For the files declared by the user as *shared*, a reference is added to the remote GASS cache, so they can be re-used by other jobs submitted to the same resource.

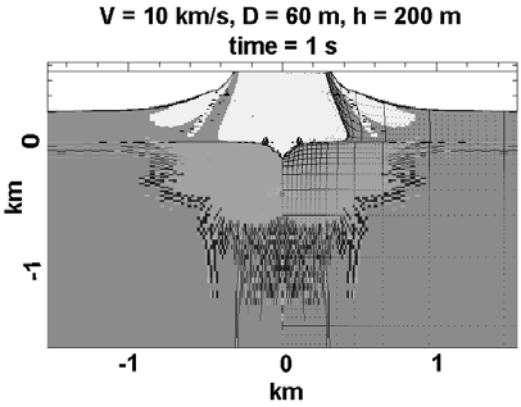

Fig. 3. Timeframes of the opening cavities at 1 second time using the 60 m impactor with 200 m water depth and a velocity of 10 Km/s for the impactor.

– The *wrapper* module, which is responsible for executing the actual job and obtaining its exit code.
– The *epilogue* module, which is responsible for transferring back output files, and cleaning up the remote experiment directory. At this point, references to *shared* files in the GASS cache are also

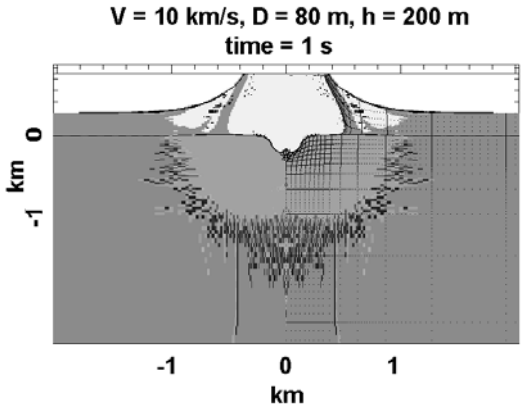

Fig. 4. Timeframes of the opening cavities at 1 second time using the 80 m impactor with 200 m water depth and a velocity of 10 Km/s for the impactor.

removed.

3.3. Grid scheduling policy for PSAs

Grid*W*ay achieves an efficient execution of PSAs by combining: *adaptive scheduling*, *adaptive execution*, and *reuse of common files* [19]. In fact, one of the main characteristics of the Grid*W*ay framework is the combination of adaptive techniques for both the scheduling and execution [18] of Grid jobs:

- *Adaptive scheduling*: Reliable schedules can only be issued considering the dynamic characteristics of the available Grid resources [2,3,6]. 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 [19].
- *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 [1,20,26]. Grid*W*ay evaluates each *rescheduling event* to decide if a migration is feasible and worthwhile [17]. Some reasons, like job cancellation or resource failure, make Grid*W*ay immediately start a migration process. Other reasons, like "better" resource discovery, make Grid*W*ay start a migration process only if the new selected resource presents a higher enough rank. In this case, the time to finalize and the migration cost are also considered [21].
- *Reuse of common files*: Efficient execution of PSAs can only be achieved by re-using shared files between tasks [6,13]. 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, which can occur in large-scale PSAs. Reuse of common files between tasks simultaneously submitted to the same resource is achieved by storing the executable file and some files declared by the user as *shared* in the GASS cache [19].

In the case of adaptive execution, the following rescheduling events, which can lead to a job migration if it is considered feasible and worthwhile, are considered [17,18]:

- Grid-initiated rescheduling events:
  - "Better" resource discovery (opportunistic migration [21]).
  - Job cancellation or suspension.
  - Resource or network failure.
- Application-initiated rescheduling events:
  - Performance degradation.
  - Change in the application demands.

In this work, we do not take advantage of all the Grid*W*ay features for adaptive execution, since they are not supported by the application. In order to fully support adaptive execution, the application must provide a set of restart files to resume execution from a previously saved checkpoint. Moreover, the application could optionally provide a performance profile to detect performance degradations in terms of application intrinsic metrics, and it could also dynamically change its host requirements and preferences to guide its own scheduling process. We only consider adaptive execution to provide fault tolerance by restarting the execution from the beginning (see the following section).

3.4. Fault tolerance

Grid*W*ay provides the application with the fault detection capabilities needed in such a faulty environment:

- The GRAM *job manager* notifies submission failures as GRAM callbacks. This kind of failures includes connection, authentication, authorization, RSL parsing, executable or input staging, credential expiration and other failures.

```
#
# Job template for impact cratering simulation
#
# Executable parameters
EXECUTABLE_FILE="impact.$GW_ARCH"
EXECUTABLE_ARGUMENTS=""
# Experiment files
INPUT_FILES="asteroid.inp.$GW_TASK_ID asteroid.inp, \
    material.$GW_TASK_ID material.inp, \
    vibr.$GW_TASK_ID vibr.inp"
SHARED_FILES="eostables.zip"
OUTPUT_FILES="frames.tgz frames.$GW_TASK_ID.tgz"
RESTART_FILES=""
# Standard I/O
STDIN_FILE=""
STDOUT_FILE="stdout.$GW_TASK_ID"
STDERR_FILE="stderr.$GW_TASK_ID"
# Resource selection parameters
HOST_REQS="reqs.ldif"
RANK_EXPR="rank.sh"
```

Fig. 5. Job template for the impact cratering application.

– The *job manager* is probed periodically. If the *job manager* does not respond, then the GRAM *gatekeeper* is probed. If the *gatekeeper* responds, a new *job manager* is started to resume watching over the job. If the *gatekeeper* fails to respond, a resource or network failure occurred. This is the approach followed in Condor/G [11].
– The standard output of *prologue*, *wrapper* and *epilogue* is parsed in order to detect failures. In the case of the *wrapper*, this is useful to capture the job exit code, which is used to determine whether the job was successfully executed or not. If the job exit code is not set, the job was prematurely terminated, so it failed or was intentionally cancelled.

When an unrecoverable failure is detected, Grid*W*ay retries the submission of *prologue*, *wrapper* or *epilogue* a number of times specified by the user and, when no more retries are left, it performs an action chosen by the user among two possibilities: stop the job for manually resuming it later, or automatically generate a rescheduling event.

3.5. Related projects

The AppLeS project [2] has previously dealt with the concept of adaptive scheduling on Grids. AppLeS is currently focused on defining templates for characteristic applications, like APST for parameter sweep and AMWAT for master/worker applications. Also, Nimrod/G [3] dynamically optimizes the schedule to meet the user-defined deadline and budget constraints. On the other hand, the need for a nomadic migration approach for adaptive execution on Grids has been previously discussed in the context of the GrADS project [20]. The tools developed by the above projects have been successfully applied to several applications, like drug design with Nimrod/G [4], computational biology with AppLes [5], and numerical relativity with GrADS and Cactus [1].

The aim of the Grid*W*ay project is similar to that of the above projects: simplify distributed heterogeneous computing. However, it has some remarkable differences. Our framework provides a submission agent that incorporates the runtime mechanisms needed for transparently executing jobs in a Grid by combining both adaptive scheduling and execution. Our modular architecture for job adaptation to a dynamic environment presents the following advantages:

– It is not bounded to a specific class of application generated by a given programming environment, which extends its application range.

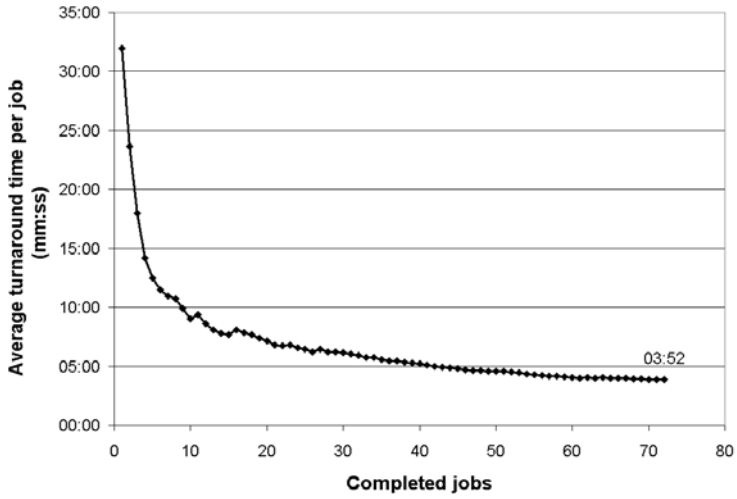

Fig. 6. Dynamic throughput, in terms of average turnaround time per job.

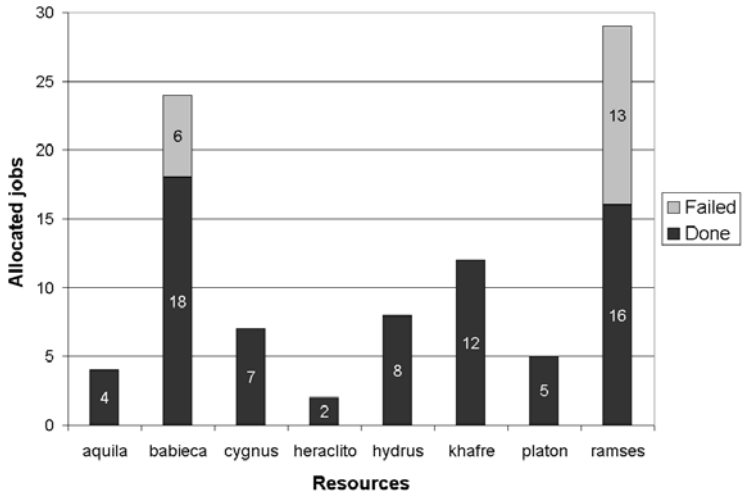

Fig. 7. Schedule performed by Grid*Way*, in terms of jobs allocated to each resource.

– It does not require new services, apart from Globus basic services, which considerably simplify its deployment.
– It does not necessarily require code changes, which allows reusing of existing software.
– It is extensible, which allows its communication with the Grid services available in a given testbed.

We would like to mention that the experimental framework does not require new system software to be installed in the Grid resources. The framework is currently functional on any Grid testbed based on Globus. We believe that this is an important advantage because of socio-political issues; cooperation between different research centers, administrators and users is always difficult.

4. Impact cratering simulations

The impact process can be described as a transfer of energy process. The initial kinetic energy of the projectile does work on the target to create a hole –the crater– as well as heating the material of both projectile and target. We focus our attention in high-velocity impacts which can be separated into several stages dominated

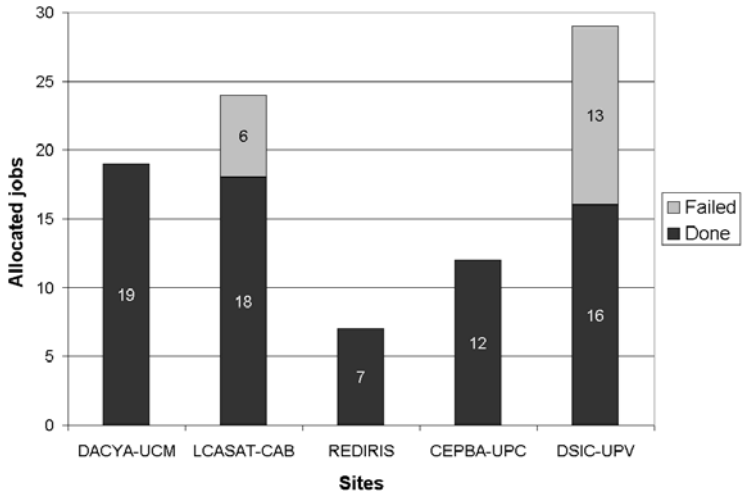

Fig. 8. Schedule performed by Grid*Way*, in terms of jobs allocated to each site.

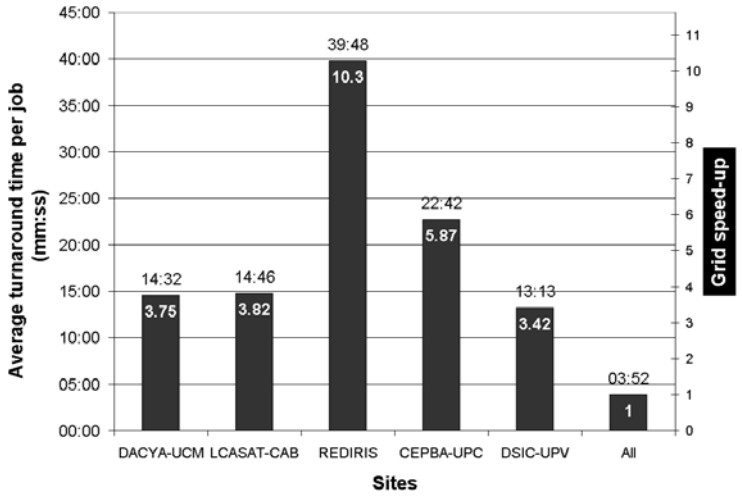

Fig. 9. Throughput, in terms of average turnaround time per job (left-hand axis and values on top of columns) and distributed speed-up (right-hand axis and values inside columns), for each site and for the whole testbed (rightmost column, labelled as "All").

by a specific set of major physical and mechanical processes.

The main stages are contact and shock compression, transient cavity growth by crater material ejection, and finally, transient cavity modification. Impact cratering begins with a sufficient compression of target and projectile materials. The energy released by deceleration of the projectile results in the formation of shock waves and its propagation away from the impact point. The projectile's initial kinetic energy redistributes into kinetic and internal energy of all colliding material. The internal energy heats both the projectile and target and, for strong enough shock waves, this may result in melting and vaporization of material near the impact zone.

To describe the impact process we solve equations of motion for compressible media using a hydrocode. The standard set of equations of motion expresses 3 basic law: mass, momentum, and energy conservation. It must be combined with the equations of state (EOS), a system of relationships which allow us to describe the thermodynamic state for materials of interest. In its basic form, an EOS should define what is the pressure in the material at a given density and temperature. In

an extended form, an EOS should define also the phase state of the material (melting, vapor, dissociation, ionization process) as well as all useful derivatives of basic parameters and transport properties (sound speed, heat capacity, heat conductivity, etc.).

Numerical simulations use the Eulerian mode of SALE-B, a 2D hydrocode modified by Boris Ivanov based on SALES-2 [12]. The original hydrocode, Simplified Arbitrary Lagrangian-Eulerian (SALE), permits to study the fluid-dynamics of 2D viscous fluid flows at all speeds, from the incompressible limit to highly supersonic, with an implicit treatment of the pressure equation, and a mesh rezoning/remapping philosophy [7]. The PDE solved are the Navier-Stokes equations. The fluid pressure is determined from an EOS and supplemented with an artificial viscous pressure for the computation of shock waves. SALES-2 can also model elastic and plastic deformation and tensile failure.

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 [15]. 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

Figures 3 and 4 show the timeframes of the opening cavities at 1 second time using the 60 and the 80 m impactor, respectively, with 200 m water depth and a velocity of 10 Km/s for the impactor. The shape difference between the 60 m case and the 80 m case illustrates the water effect. Due to the water layer, in that case, the impactor diameter has to be larger than 80 m to crater the seafloor.

5. Grid*W*ay programming model

The Grid*W*ay application programming and command line interface allow scientists and engineers to express their computational problems in a Grid environment. The capture of the job exit code allows 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 [14].

Figure 5 shows a fragment of the job template used in the following experiments. Files are specified as "source destination" pairs separated by commas, where the destination file can be omitted if it has the same name as the source file.

The experiment files consist of the executable ($\sim$0.5 MB), some parameter files (12 KB) for each task, and a table with values for the EOS equations (1.3 MB, when compressed) shared by all the tasks. The final name of the executable is obtained by resolving the variable **$GW_ARCH** at runtime for the selected host. Similarly, the final names of the parameter files are obtained by resolving the variable **$GW_TASK_ID** at runtime for the current task. Once the job finishes, the standard output (0.5 MB) and the files with the figures of the simulation timeframes in PNG format (0.5 MB) are transferred back to the client.

The experiments have been performed with a resource selection script that queries MDS for potential execution hosts, attending the following criteria:

– Host requirements are specified as a LDAP filter, which is used by the *resource selector* to query MDS and so obtain a preliminary list of potential hosts. In the experiments below, we impose a minimum main memory of 100 MB, enough to accommodate each task:

(Mds-Memory-Ram-Total-sizeMB>=100)

The *resource selector* also performs an user authorization filter (via a GRAM ping request) on those hosts.

– A rank is assigned to each potential host following the preferences specified by the user in a ranking expression, which is a script that receives the monitoring data of the resources and outputs the rank value. Since our target application is a computing-intensive simulation, the ranking expression benefits those hosts with less workload and so better performance. The following expression was considered:

Table 2
Mean, standard deviation and coefficient of variance (CV) for the transfer and execution times on each resource

		Transfer time			Execution time		
Name	Site	Mean	Dev.	CV	Mean	Dev.	CV
hydrus	DACYA-UCM	0:00:32	0:00:21	67%	0:28:51	0:27:43	96%
cygnus	DACYA-UCM	0:00:42	0:00:29	69%	0:37:17	0:20:56	56%
aquila	DACYA-UCM	0:00:33	0:00:08	25%	0:50:20	0:24:54	49%
babieca	LCASAT-CAB	0:05:32	0:02:42	49%	0:47:06	0:50:24	107%
platon	REDIRIS	0:00:29	0:00:07	22%	1:48:00	1:39:09	92%
heraclito	REDIRIS	0:00:36	0:00:11	31%	1:10:04	0:53:49	77%
ramses	DSIC-UPV	0:00:26	0:00:47	177%	0:38:31	0:54:24	141%
khafre	CEPBA-UPC	0:01:06	0:00:01	2%	1:22:26	1:19:15	96%

Table 3
Mean, standard deviation and coefficient of variance (CV) for the wall times on each resource

		Wall time		
Name	Site	Mean	Dev.	CV
hydrus	DACYA-UCM	0:29:22	0:27:51	95%
cygnus	DACYA-UCM	0:37:59	0:21:06	56%
aquila	DACYA-UCM	0:50:52	0:25:03	49%
babieca	LCASAT-CAB	0:52:38	0:49:20	95%
platon	REDIRIS	1:48:29	1:39:13	91%
heraclito	REDIRIS	1:10:40	0:54:01	76%
ramses	DSIC-UPV	0:38:57	0:54:33	140%
khafre	CEPBA-UPC	1:23:32	1:19:15	95%

$$rank = \begin{cases} FLOPS \text{ if } CPU_{15} \geqslant 1; \\ FLOPS\cdot \text{ if } CPU_{15} < 1, \\ CPU_{15} \end{cases} \quad (1)$$

where $FLOPS$ is the peak performance achievable by the host CPU, and CPU_{15} is the average load in the last 15 minutes.

6. Computational results and performance evaluation

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 and the testbed resources are heterogeneous. Moreover, there is an additional difference generated by the changing resource load and availability. Therefore, adaptive scheduling is crucial for this application. Figure 6 shows the dynamic throughput, in terms of average turnaround time per job (i.e. the elapsed time divided by the number of completed jobs), as the experiment evolves. 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, 15.51 jobs per hour.

Figures 7 and 8 show the schedule performed by Grid*Way*, in terms of number of jobs allocated to each resource and site, respectively. Most of the allocated jobs were successfully executed, but others failed and were dynamically rescheduled. Given these results, we can calculate the fault rate for each resource or site. The two failing resources (sites) show a fault rate of 25% and 45%, respectively, which result in an overall fault rate of 21%. These failures are mainly due to a known Globus problem (bug id 950) related to the NFS file systems and the PBS resource manager used in the clusters, which causes the job manager not to be able of getting the standard output and error of the job. This problem is mitigated, but not avoided, on `babieca`, where a patch related to this bug was applied.

Figure 9 shows the achieved throughput, also in terms of average turnaround time per job, by each site and by the whole testbed for the above schedule. In the right axis, the distributed or Grid speed-up, i.e. the performance gain obtained by each site, is also shown. We introduced Grid speed-up as a valuable metric for resource users and managers on each site in order to realize the benefits of being part of a Grid. Performance metrics like this can help to curb their selfishness sharing resources on the Grid [25]. It is defined as follows:

$$S_{\text{site}} = \frac{T_{\text{site}}}{T_{\text{Grid}}}, \quad (2)$$

where T_{Grid} is the Grid turnaround time, i.e. the waiting time from the application execution request until all

the tasks are completed and all the results are available when all the resources in the testbed are used, and T_{site} is the site turnaround time, i.e. the turnaround time when only the resources of a given site are used. This metric should be obtained or estimated in a distributed way for each site.

Table 2 shows the mean, standard deviation and coefficient of variance (CV) for the transfer and execution times on each resource. Table 3 shows the same statistics for the wall times on each resource. These values are nearly the same as for the execution time, since that is the dominating term.

In the case of the transfer time, we can see differences due to the heterogeneity of network links and resource configurations. For example, `khafre` runs the 2.2 version of the Globus toolkit, which has a polling period for the GRAM *job manager* of 30 seconds (whereas the 2.4 version polling period is 10 seconds). That produces a mean transfer time of one minute (two polling periods) with a very low standard deviation. The results also report a very high standard deviation in `babieca`, which has a nearly flat probability density function, that revealed several problems in the GRAM *job manager*. It is interesting to note that the best mean transfer time corresponds to `platon`, although it is located in a different site from the client. This is due to the file reuse policy implemented by Grid*W*ay, as `platon` is a SMP node with two processors that executes two simultaneous tasks sharing common files.

In the case of the execution time, there are two sources of variance: the dynamism and heterogeneity of the Grid resources and the different time needed by each task to converge. Processor speeds have the greater impact on the mean, while the use of RMS like PBS in some clusters or the existence of SMP nodes make the standard deviation to be greater.

7. Conclusions

The Globus toolkit provides a way to access the distributed resources needed for executing the compute and data intensive applications required in several research and engineering fields. However, the user is responsible for manually performing all the submission steps in order to achieve any functionality, and the adaptive execution of applications is not supported. The Grid*W*ay framework provides the runtime mechanisms needed for submitting applications and dynamically adapting their execution.

The suitability of a Grid environment based on the Grid*W*ay framework and the Globus toolkit has been demonstrated for the execution of a high throughput computing application that simulates impact cratering. The application comprises the execution of a high a number of tasks that exhibit different execution times due to both the heterogeneity and dynamism of the Grid resources and the convergence properties of the algorithm. Such computing platform will help to develop a search criteria for future investigations and exploration missions to Mars. Moreover, if they are successful in their hunt, they would also help to address the ongoing debate of whether large water bodies existed on Mars and, therefore, they would help to constrain future paleoclimatic reconstructions.

The Grid speed-up has been introduced as a valuable metric for resource users and managers in order to realize the benefits of sharing resources over the Grid. On the other hand, the study of the execution and transfer time provides a measure of the variability in the resource load and could be monitored when adjusting the components of a Grid in order to improve its performance.

8. Acknowledgements

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 Research and Education Network (RedIRIS). All of them are part of the Spanish Thematic Network on Grid Middleware.

We would like to also thank Jens Ormö and Jesús Martínez-Frías, in the Planetary Geology Laboratory at CAB, for their help in understanding the impact cratering simulations.

References

[1] G. Allen, D. Angulo, I. Foster et al., The Cactus Worm: Experiments with Dynamic Resource Selection and Allocation in a Grid Environment, *Intl. J. High-Performance Computing Applications* **15**(4) (2001), 345–358.

[2] F. Berman, R. Wolski, H. Casanova et al., Adaptive Computing on the Grid Using AppLeS, *IEEE Trans. Parallel and Distributed Systems* **14**(4) (2003), 369–382.

[3] R. Buyya, D. Abramson and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computation Grid*, In Proc. 4th Intl. Conf. High Performance Computing in Asia-Pacific Region (HPC Asia), 2000, pp. 283–289, IEEE CS.

[4] R. Buyya, K. Branson, J. Giddy and D. Abramson, The Virtual Laboratory: Enabling On Demand Drug Design with the WorlWide Grid, *Concurrency and Computation: Practice and Experience* **15**(1) (2003).

[5] H. Casanova, T. Bartol, F. Berman et al., The Virtual Instrument: Support for Grid-enabled Scientific Simulations, *Intl. J. High Performance Computing Applications* (Spring, 2004).

[6] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*, In Proc. 9th Heterogeneous Computing Workshop, IEEE CS, 2000, pp. 349–363.

[7] G.S. Collins, H.J. Melosh and B.A. Ivanov, Modeling Damage and Deformation in Impact Simulations, *Meteoritics and Planetary Science* **39**(2) (2004), 217–231.

[8] T. El-Ghazawi, K. Gaj, N. Alexandinis and B. Schott, *Conceptual Comparative Study of Job Management Systems*, Technical report, George Mason University, February 2001. Available at http://ece.gmu.edu/lucite/reports/.

[9] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers and J. Pruyne, A Worldwide Flock of Condors: Load Sharing among Workstation Clusters, *Future Generation Computer Systems* **12**(1) (1996), 53–65.

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

[11] J. Frey, T. Tannenbaum, M. Livny, I. Foster and S. Tuecke, Condor/G: A Computation Management Agent for Multi-Institutional Grids, *Cluster Computing* **5**(3) (2002), 237–246.

[12] S.C. Gareth and H.J. Melosh, *SALES 2: A Multi-Material Extension to SALE Hydrocode with Improved Equation of State and Constitutive Model*, Available at http://www.lpl.arizona.edu/gareth/publications/sales 2, 2002.

[13] A. Giersch, Y. Robert and F. Vivien, *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, pp. 364–371.

[14] J. Herrera, E. Huedo, R.S. Montero and I.M. Llorente, *Execution of Typical Scientific Applications on Globus-based Grids*, in Proc. 3rd Intl. Symp. Parallel and Distributed Computing, IEEE CS, 2004, pp. 177–183.

[15] K.R. Housen, R.M. Schmidt and K.A. Holsapple, Crater Ejecta Scaling Laws: Fundamental Forms Based on Dimensional Analysis, *Journal of Geophysical Research* **88**(B3) (1983), 2485–2499.

[16] How Sun Grid Engine, Enterprise Edition 5.3 Works. Technical report, Sun Microsystems, 2002. Available at http://www.sun.com/software/gridware/sgeee53/wp-sgeee.

[17] E. Huedo, R.S. Montero and I.M. Llorente, A Framework for Adaptive Execution on Grids, *Intl. J. Software – Practice and Experience* (*SPE*) **34**(7) (2004), 631–651.

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

[19] E. Huedo, R.S. Montero and I.M. Llorente, *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, pp. 28–33.

[20] G. Lanfermann, G. Allen, T. Radke and E. Seidel, *Nomadic Migration: A New Tool for Dynamic Grid Computing*, In Proc. 10th Symp. High Performance Distributed Computing (HPDC10), IEEE CS, 2001, pp. 429–430.

[21] R.S. Montero, E. Huedo and I.M. Llorente, *Grid Resource Selection for Opportunistic Job Migration*, In Proc. 9th Intl. Conf. Parallel and Distributed Computing (Euro-Par 2003), (vol. 2790) of LNCS, August 2003, pp. 366–373.

[22] J. Örmo, J.M. Dohm, J.C. Ferris, A. Lepinette and A. Fairten, Marine-Target Craters on Mars? An Assessment Study, *Meteoritics & Planetary Science* **39**(2) (2004), 333–346.

[23] J. Örmo and M. Lindstr, om. When a Cosmic Impact Strikes the Seabed, *Geological Magazine* **137** (2000), 67–80.

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

[25] J.M. Schopf and B. Nitzberg, Grids: The Top Ten Questions, *Scientific Programming, special issue on Grid Computing* **10**(2) (August 2002), 103–111.

[26] S. Vadhiyar and J. Dongarra, *A Performance Oriented Migration Framework for the Grid*, In Proc. 3rd Intl. Symp. Cluster Computing and the Grid (CCGrid 2003), IEEE CS, 2003, pp. 130–137.

[27] R. Wolski, G. Shao and F. Berman, *Modeling the Cost of Redistribution in Schedulling*, In Proc. 8th SIAM Conf. Parallel Processing for Scientific Applications, 1997.