ARTICLE IN PRESS

Future Generation Computer Systems xx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Monitoring and steering Grid applications with GRID superscalar

S. Reyes [a], C. Muñoz-Caro [a], A. Niño [a,*], R. Sirvent [b], R.M. Badia [b]

[a] QCyCAR. E. S. Informática, Universidad de Castilla-La Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain

[b] Barcelona Supercomputing Center, Building Nexus II, Jordi Girona, 29. E-08034 Barcelona, Spain

ARTICLE INFO

Article history:
Received 18 April 2009
Received in revised form
30 November 2009
Accepted 7 December 2009
Available online xxxx

Keywords:
Grid computing
GRID superscalar
Monitoring Grid applications
Steering Grid applications

ABSTRACT

We present the design and implementation of a general task monitoring and steering system for Grid applications (GSTAT). The system is integrated in the GRID superscalar (GRIDSs) programming framework. Information at the application, Grid node, and individual task levels are supplied upon request. Using the steering capabilities, individual tasks or the whole application can be cancelled. The corresponding jobs can be restarted using fault tolerance and checkpointing capabilities based on GRIDSs. In addition, the computational resources assigned to an application can be modified. GSTAT is tested using high throughput and high performance computing cases on an Internet-based Grid of computers.

© 2009 Published by Elsevier B.V.

UNCORRECTED PROOF

1. Introduction

A computational Grid is a large-scale distributed system that Q1 supports integration of different computational resources from different administrative domains [1]. With the advent of Grid and related technologies, scientists and engineers are building more and more complex applications to manage and process large data sets, or to implement system models on distributed resources. Some applications are translated in large amounts of individual tasks needed to obtain the answer to the problem being considered. This is the case of parameter sweep problems, which arise naturally in several scientific and engineering fields. In these problems, we have a system defined by a set of independent parameters. The behavior of the system can be determined by varying these parameters. This produces a set of tasks that can be independently computed. Depending on the size of the parameter space, the number of tasks can be very large. It has been shown that the Grid approach is very efficient in dealing with these kinds of problems [2,3].

The above mentioned scientific or engineering application scenarios require means for composing and executing complex workflows. The workflow concept is concerned with the automation of procedures where, to achieve an overall goal, data are passed between participants according to a defined set of rules. In this context, the Grid programming environment GRID superscalar (GRIDSs) is able to implement and handle dynamic workflows [4]. In these kinds of applications, users must be supported in finding and keeping track of the available resources, and on the state of the processes running on the Grid. To address these objectives, it is necessary to collect systematically information on the present, and sometimes on the past, status of resources or processes. This is nothing but a process of monitoring. In addition, access to application level steering is a desirable feature. According to [5], only AutoPilot [6] and Mercury [7] incorporate some kind of application oriented steering at the Grid level.

In this work, we present an approach to the monitoring and steering of Grid applications consisting in the incorporation of these capabilities in the general Grid programming environment GRIDSs. In this form, any application developed or migrated to the Grid with GRIDSs can be supervised and handled by the user. In the present approach we consider application and task oriented steering capabilities. Thus, the user can modify the resources associated with the application, terminate and restart the whole application or even cancel (and reschedule) individual tasks. In addition, in contrast with the existing tools, we also incorporate an off-line monitoring working mode. This approach is especially useful when the application is composed by a large number of independent tasks and the user needs to control the time evolution and behavior of the process. This is the case in parameter sweep problems.

As test cases we use two examples. The first one belongs to the high throughput computing realm. This case corresponds to an application for the automatic generation of molecular potential energy hypersurfaces in Internet-based Grids of computers [2]. The second example belongs to the high performance realm. It consists in a concurrent implementation of block matrix multiplication.

The rest of this paper is organized as follows. Section 2 presents an overview of monitoring in Grid systems and of the GRIDSs

* Corresponding author. Tel.: +34 926 295300x6474; fax: +34 926 295354.
E-mail address: alfonso.nino@uclm.es (A. Niño).

0167-739X/$ – see front matter © 2009 Published by Elsevier B.V.
doi:10.1016/j.future.2009.12.002

programming framework. Section 3 presents an overview of the design and implementation issues of the proposed monitoring and steering system. Section 4 focus in the text interface proposed in the present version of the system. Section 5 shows the examples. Finally, Section 6 presents the conclusions.

2. Background

2.1. Related work

Monitoring is classically defined as the collection, interpretation and display of information concerning the status of the hardware or software system of interest [8]. For instance, to produce parallel applications that perform well on today's parallel architectures, different tools for collecting and analyzing data do exist. As representative tools of this kind we have Tau [9], Paradyn [10], or MATE [11], for a review see [12].

In a Grid, its information system (Grid Information System or GIS) is in charge of monitoring the status of the resources forming the Grid. However, monitoring is also of key interest for job scheduling, data replication, accounting, data analysis, or for the optimization of applications [13]. Monitoring the status of an application in a Grid environment can be carried out in principle from the information provided by the Grid resource management system. However, in this form, we only can get information about individual tasks. This is not a practical approach when our application handles tens or hundreds of thousands tasks.

The monitoring of tasks in distributed systems involves dynamically collecting information about the concurrent individual tasks making up the processes. In this context, different works have been carried out from different standpoints. Thus, Garcia-Molina et al. [14] describe a methodology for debugging distributed systems from a bottom-up perspective. This work involves the use of trace files obtained in a monitoring tasks. Snodgrass [15] considers monitoring as an information processing activity, proposing a relational model for organizing the information generated in distributed systems. In turn, Harrison [16] proposes an approach for minimizing the impact of monitoring on the tasks being monitored. This approach is based in the use of program state analysis and dynamic traces of behavior. A detailed review of monitoring distributed systems, in the pre-Grid era, is that of Joyce et al. [8].

With respect to Grid systems monitoring, it is necessary to refer to the specifications of the Open Grid Forum (OGF) [17]. Here, a set of fifteen scenarios where Grid monitoring is needed for obtaining and using performance data [18] are identified. In particular, scenario 6 corresponds to job progress monitoring. Also, the OGF has proposed a putative standard for the Grid Monitoring Architecture (GMA) [19]. The GMA model describes the major components of a Grid monitoring system and their essential interactions. In this model, any networked resource (processors, storage media, network links or applications) is defined as an entity. Data associated to a resource are timestamped. The result is defined as an event. The proposed architecture is based in three types of components. First, the Producer that is an entity providing events. Second, the Consumer that is any process that receives events. Finally, the Directory service or registry that allows producers to publish the event types that they produce. The Directory allows consumers to find the events they are interested in. A detailed description of task monitoring in Grid systems can be found in [5,20,13]. Also, the task execution supervision is an important functionality for the user. When developing monitoring tools for Grid system it is interesting to conform to the OGF recommendations about Grid Monitoring Architecture.

Several tools are available for monitoring computational Grid resources. For instance, the Monitoring and Discovery Service (MDS) [21], formerly known as the Metacomputing Directory Service, constitutes the information infrastructure of the Globus Toolkit [22]. Another tool is Ganglia [23] that is a scalable distributed monitoring system for high performance computing systems such as clusters and Grids. Also, we have the Network Weather Service (NWS) [24]. This is a portable and non-intrusive performance monitoring and forecasting distributed system. Another example is the Resource Monitoring for the Grid (GridRM) [25]. GridRM is based on a three-tier architecture and it accesses information of resources from other services such as MDS or Simple Network Management Protocol (SNMP). This tool uses a relational database for storing information. Finally, GridICE [26] is a distributed monitoring tool designed for Grid systems. Additional information on Grid monitoring services can be found in Reference [13].

In contrast with system monitoring tools, few monitoring and performance analysis tools for Grid applications have been introduced. For instance, the Network Application Logger Toolkit (NetLogger) [27] is used for performance analysis of complex systems such as client–server and/or multi-threaded applications. OCM-G [28] is an infrastructure for Grid application monitoring which uses OMIS [29] as a communication interface. ProActive [30] is a Java library for parallel, distributed, and concurrent computing. It also features mobility and security in a uniform framework. ProActive applications can be monitored transparently by an external application: the IC2D (Interactive Control and Debugging for Distribution) [31]. AutoPilot [6] is a distributed monitoring and tuning system. It is used in the GrADS [32] system to monitor performance contracts via application level AutoPilot sensors. The Mercury [7] Monitor is designed to satisfy requirements of Grid performance monitoring. It supports monitoring of Grid entities such as resources and applications. Finally, the job monitoring system AMon [33] provides the user with sufficient information on the jobs and the resource usage. The monitoring data are pre-analyzed to give the user hints on possible problems. The data are presented in a graphical way, allowing interactivity.

Another approach to the monitoring of applications relies on workflow mapping engines. Reporting the progress of a workflow is a complex task and few tools are available for that. One of this is Pegasus [34]. Pegasus performs a mapping from an abstract workflow to the set of available Grid resources. This tool generates an executable workflow that is monitored with the shell primitive pegasus status. Another option is Triana [35]. Triana is a visual workflow oriented data analysis environment integrated with Grids via the GridLab GAT (Grid Application Toolkit) interface [36]. In turn, a generic architecture for monitoring and steering legacy applications in Grid environments (gridMonSteer or GMS) has been proposed and implemented [37]. The GMS architecture consists of two parts: an application wrapper that runs in the same execution environment as the application, and an application controller that generally runs locally to the user. Using GMS with Triana is possible. Another tool is Askalon [38]. The main objective of Askalon is to simplify the development and optimization of, mostly, Grid workflow applications. The monitoring and performance analysis component provides information of computational resources, network and applications.

2.2. GRID superscalar (GRIDSs)

GRID superscalar is a Grid programming environment that allows the parallelization of sequential applications in computational Grids [39]. Although GRIDSs applications are programmed sequentially, they follow the master–worker paradigm when executed in the Grid, with a local master host running the main program and several worker hosts running one or more tasks concurrently. The underlying runtime library is able to automatically parallelize the application. The GRIDSs runtime library implements the following features: data dependence analysis, task

Please cite this article in press as: S. Reyes, et al., Monitoring and steering Grid applications with GRID superscalar, Future Generation Computer Systems (2009), doi:10.1016/j.future.2009.12.002

scheduling, file renaming, shared disk management and locality policy, checkpointing and fault tolerance. The main objective is to reduce the complexity of programming applications in the Grid. In most cases the execution of these applications implies the execution of multiple tasks in the Grid.

GRIDSs is a programming model that eases the development of Grid applications to the point that writing such an application can be as simple as programming a sequential program to be run on a single processor, whereas the hardware resources remain totally transparent to the programmer. The GRID superscalar programming interface is based on high-level programming languages, such as C, C++ or Java. From the code, a directed acyclic graph, with the data dependencies between the tasks that form the application, is generated and executed in the Grid infrastructure.

An application written with GRIDSs follows the master–worker parallel programming paradigm. The main part of the application is the master, that will automatically generate tasks for the runtime when the programmed algorithm is executed. The functions that must be executed in the Grid will be executed by creating a worker in a remote machine that will call to the corresponding function with the correct parameters. We can consider GRIDSs applications master–worker applications, although this structure is hidden to the application programmer.

To develop an application with GRIDSs, the application developer provides a sequential program, composed of a main program with calls to application tasks, together with an Interface Description Language (IDL) file. The programming model takes sequential programming as a reference. This allows to build a very small an simple API to be used in the applications.

The API is composed of six calls, which can be used in the master, and two in the worker. Despite the simplicity of the API, several advanced features can be specified, such as the speculative execution of tasks and the specification of constraints and cost for tasks. In the IDL file the user must specify the functions that must be run in the Grid, together with their parameters. This IDL file is not only used to know the type and direction of the parameters in the functions, but also to build a master–worker application.

GRID superscalar defines the behavior of the application to execute. Thus, for each task candidate to be run in the Grid, the GRID superscalar runtime inserts a node in a task graph. Then, the GRID superscalar runtime system seeks for data dependencies between the different tasks of the graph. These data dependencies are defined by the input/output of the tasks, which are files. If a task does not have any dependence with previous tasks which have not been finished or which are still running (i.e., the task is not waiting for any data that has not been already generated), it can be submitted for execution to the Grid. Renaming is applied in the task graph in order to eliminate some data dependencies between tasks and, therefore, increase the possibilities of executing tasks in parallel. This technique is a unique feature only available in GRID superscalar, and it allows to solve concurrency problems when two tasks want to write in the same files, or when a task must overwrite some files that other tasks must read.

GRIDSs provides a simple broker that selects between the set of available hosts which is the best suited for a task. This selection favours reducing the total execution time, which is computed not only as the estimated execution time of the task in the host but also includes the time that will be spent to transfer all files required by the task to the host. This allocation policy exploits the file locality, reducing the total number of file transfers. Those tasks that do not have any data dependence can be run on parallel on the Grid. This process is automatically controlled by the GRID superscalar runtime, without any additional effort for the user, as well as all Grid related actions (file transfer, job submission, end of task detection, results collection), which are totally transparent to the user. The GRID superscalar is notified when a task finishes. Next,

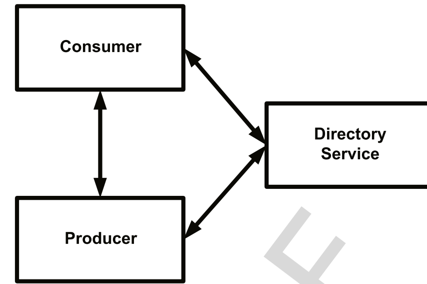

Fig. 1. GGF Grid Monitoring Architecture (GMA).

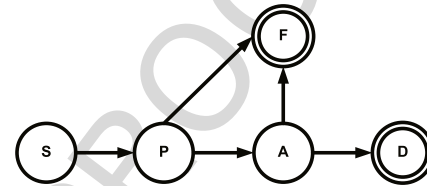

Fig. 2. State transition diagram for grid tasks.

the data structures are updated and any task than now have its data dependencies resolved, can be submitted for execution. GRIDSs assigns an ID number to each task, when the dependence graph allows it to be executed.

When the application has finished, all working directories in the remote hosts are cleaned, application output files are collected and copied to their final locations in the local host. Summarizing, from the user point of view, an execution of a GRIDSs application looks as an application that has run locally, with the exception that the execution has been hopefully much faster by using the Grid resources.

3. Design and implementation issues

The proposed system falls in the framework of the OGF monitoring scenario 6, Job Status and Progress Monitoring [18]. This scenario corresponds to the use of monitoring for determining the state and time of a large number of executing tasks. In addition, the monitor conforms to the OGF Grid Monitoring Architecture (GMA) [19], see Fig. 1. Here, we observe the relationships among the three types of components of a Grid monitoring system: Producers, consumers and directory service. The producers generate the different events (timestamped data) that are accessible through the directory service under query by the consumers. In the approach here proposed, the generation of events (by producers) and its use (by consumers) are asynchronous processes.

As a monitor, GSTAT provides on-line (during the execution) and off-line (after the execution, by implementing a persistent directory service) information of the application and its component tasks. The monitor identifies the possible status of a task, namely [40]: Start (S), Pending (P), Active (A), Failed (F) and Done (D). Fig. 2 shows the state transition diagram for the states the monitor identifies.

Fig. 3 shows, in a conceptual diagram, the logical model of the GSTAT monitoring and steering system. Here, we observe that the user interface is formally separated from the functional component. In this way, we allow for independent adaptation to different interaction schemes, i.e., command-line interface or GUI. GSTAT follows a layered organization, see Fig. 3. The first layer corresponds to the monitoring, steering and GSTAT help capabilities. In turn, the monitoring and steering subsystems are divided in a second layer corresponding to the system, application and task levels. It can be seen that system monitoring allows getting information on the status of the running application on the Master node and on

UNCORRECTED PROOF

Please cite this article in press as: S. Reyes, et al., Monitoring and steering Grid applications with GRID superscalar, Future Generation Computer Systems (2009), doi:10.1016/j.future.2009.12.002

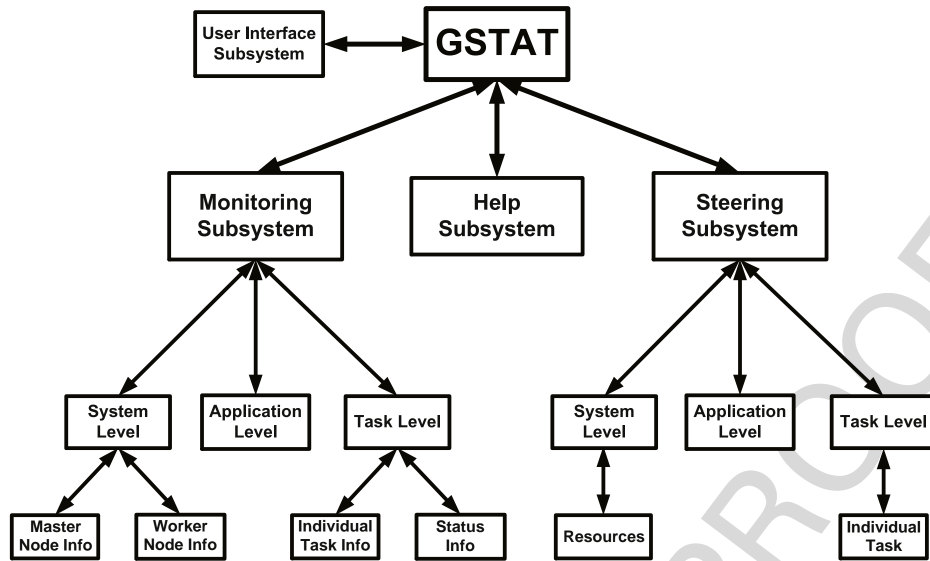

Fig. 3. The GSTAT logical model.

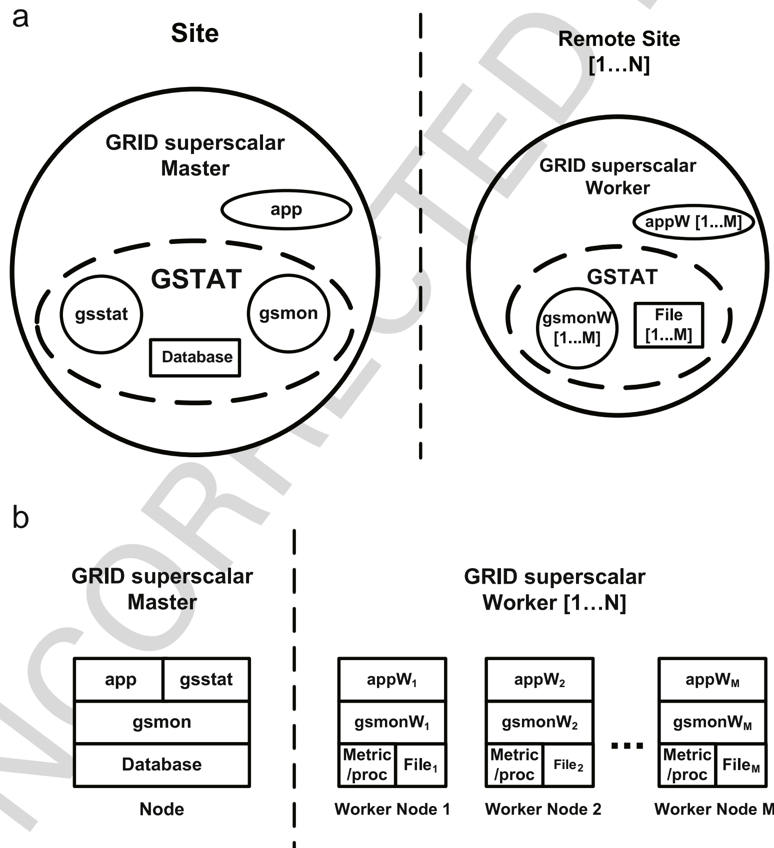

Fig. 4. The GSTAT architecture: (a) Static vision, (b) Dynamic vision.

the Workers. On the other hand, application monitoring provides information on the whole application. Finally, task monitoring provides information on a single task or on the tasks in a given status.

With respect to the steering subsystem, Fig. 3 shows that at the Grid system level we allow for the modification of the resources associated to the application (number of Grid nodes or number of processors in each node). On the other hand, at the application level, it is possible to act on the whole process. At this level, we incorporate the ability to stop all the individual tasks in execution ending the application. This last can later be restarted from that point, using the checkpointing capabilities of GRIDSs. With respect to an individual task, GSTAT allows stopping and rescheduling it thanks to the fault tolerance ability of GRIDSs.

The architecture of the system is shown in Fig. 4. Fig. 4(a) shows the static vision, whereas Fig. 4(b) shows a dynamic vision of GSTAT. It can be seen that GSTAT is organized in four modules distributed among the Master (Client) and the Worker (Server) nodes (see Fig. 4(a)). These modules are: (a) *gsmon*. This is an agent that supervises the execution of the application. It is initiated by GRIDSs in the Master. In order to separate the GSTAT

Please cite this article in press as: S. Reyes, et al., Monitoring and steering Grid applications with GRID superscalar, Future Generation Computer Systems (2009), doi:10.1016/j.future.2009.12.002

UNCORRECTED PROOF

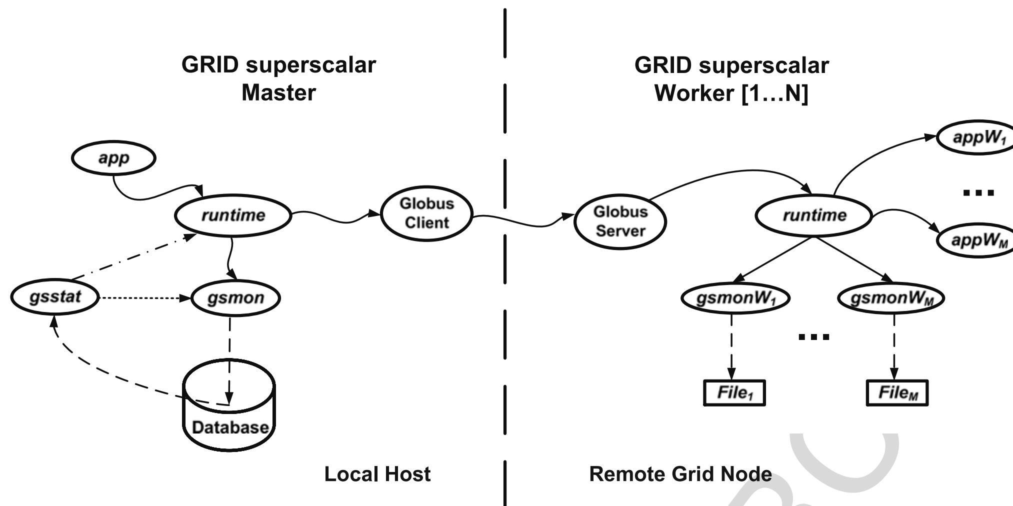

Fig. 5. Interaction diagram between components in GRIDSs. The continuous line shows the calls sequence with GRIDSs. The discontinuous line shows the GMA Consumer/Producer relationship. The dotted line shows the monitoring events. Finally, the dot-dashed line shows the steering events.

monitor from the GRIDSs implementation, *gsmon* is not included in the GRIDSs runtime. Periodically, the agent receives one event (per task) with three data from the application (*app*): the task identifier, the Worker used and the status of each task. *gsmon* starts counting time with the first event associated to a task. The data are inserted or updated in the *Database*, see Fig. 4. *gsmon* is a producer according to the GMA architecture.

(b) *gsmonW*. This is a sensor that observes the execution of a single task. It is created by GRIDSs in every Worker node. At each Worker node we have an instance of *gsmonW* for each task (*appW*). These two entities are implemented separately because *appW* depends on the application being considered. Therefore, Node *M* of Worker *N* executes an instance of the Workers application, *appW*, and of *gsmonW*. The sensor obtains performance metrics (CPU and Wall times for *appW*, see Fig. 4(b)) of each task by accessing the/proc File System. In fact, what we do is to access the information of the individual process (at the Worker node level) associated to the task. The sensor stores the information in a *File*. According to the GMA architecture *gsmonW* is a producer.

(c) *Database*. Here, the data generated by *gsmon* and *gsmonW*, i.e., information about the state and time corresponding to each task, is stored and updated. The database, a relational one, is kept on disk after execution in order to implement off-line query capabilities. This component is a directory service according to the GMA architecture.

(d) *gsstat*. This is the user interface to the GSTAT system. Therefore, it is in charge of supplying the user, upon request, with a report about the tasks in the Grid system. In addition, it allows the user to alter the characteristics of the system and of the executing application. This component is a consumer according to the GMA architecture.

Fig. 4(b) also shows that one instance of the agent *gsmon* runs in the Master, whereas *M* instances of the sensor *gsmonW* are in each one of the *N* Workers (an instance for each task, *appW*, in execution in that Worker node).

Fig. 5 shows in a behavior diagram, the interactions among the components of GSTAT and GRIDSs. In the Master, the user executes the application *app*. The application *app* starts the *gsmon* agent using the Master runtime library. The Master runtime library uses the Globus Client services for submitting tasks to the workers in the Grid. In each worker, the Globus Server job-manager calls the Worker application, appW. In addition, the GRIDSs Worker runtime library calls the *gsmonW* sensor once by task. The *gsmon* agent and the *gsmonW* sensor are synchronous with *app* and *appW*, respectively. Therefore, ending *app* or *appW* ends *gsmon* or *gsmonW*, respectively.

gsstat is the user interface that provides the monitoring information needed for the steering of events, see Fig. 5. In this case, *gsstat* calls (sends a signal to) *gsmon*, asking for the needed monitoring information following a user request. With respect to the behavior of the steering events, Fig. 5 also shows that *gsstat* calls (sends a signal) the runtime library Master of GRIDSs, allowing specific actions such as termination of the *app* application (terminate) or of some specific task (cancel). Modification of the resources associated to the job is allowed by *gsmon* rewriting the GRIDSs configuration file. Then, *gsmon* sends a signal to the GRIDSs Master runtime to reload the updated configuration file.

Fig. 5 shows that, as a producer, *gsmon* in the GRIDSs Master node updates the Database. On the other hand, *gsmonW*, as a producer in the GRIDSs Worker node, updates the File. When a task (*appW* in Fig. 5) finalizes, the content of the File is transferred by the runtime and updated by *gsmon* in the Database. In this way, it is possible to perform off-line monitoring.

The modules *gsmon*, *gsmonW* and *gsstat* are written in Perl [41]. The first two run simultaneously with *app* and *appW* as background processes. *gsstat* is implemented as a system command, and it runs at the user's discretion. Separation of the detection and collection of information from its user interface allows for easier adaptability of the GSTAT monitor.

4. Displaying information

Consider a typical computational Grid conformed by several "Grid nodes". In the general case these nodes are individual systems, usually clusters of computers, belonging to different administrative domains. When running applications on GRIDSs, one node acts as Master, whereas the others act as Workers. Once an application is started in the Master node, we will have a number of tasks running in the individual processors of the Worker nodes. These tasks can be in any of the different states identified by the GSTAT monitor, see Fig. 6. The display of monitoring information could be done in several forms namely, textual or graphical through a GUI. In this first version of GSTAT we use the command line as user interface.

Fig. 6 shows how to access the different monitoring capabilities through the user interface shown in Fig. 3. In particular, the on-line help system is accessible from the command line by calling *gsstat* with the *-h* switch.

```
==========================================================
         GRID superscalar Tasks STATUS
==========================================================
USAGE: gsstat [OPTIONS] [COMMANDS]
OPTIONS:
 -a                  display ACTIVE tasks
 -aw worker          display ACTIVE tasks in worker
 -c n                CANCEL task number n
 -d                  display DONE tasks
 -dw worker          display DONE tasks in worker
 -e n                display status of task number n
 -f                  display FAILED tasks
 -fw worker          display FAILED tasks in worker
 -h                  display help
 -k n                TERMINATE of task number n
 -p                  display PENDING tasks
 -pw worker          display PENDING tasks in worker
 -r                  modify RESOURCES
 -s                  display status of the tasks
 -t n                display TIMES of task number n
 -w                  display status in all workers
 -w worker           display status in worker
 without argument    equivalent to -s
COMMANDS:
 | more              display one screen at a time
 | tail              shows the last lines of the command output
==========================================================
```

Fig. 6. Monitoring and Steering options available in the GRID superscalar monitor (GSTAT). The figure shows the output of the help option.

```
==================================================
         GRID superscalar Tasks STATUS
==================================================
MASTER: bscgrid01.bsc.es
-----------------------------------
Total TIME (dd:hh:mm:ss) = 00:00:31:02
==================================================
DONE: 37 ACTIVE: 3 PENDING: 1 FAILED: 0
==================================================
Statistics for Tasks DONE
------------------------
Time(Min|Avg|Max)=(00:01:23|00:03:11|00:09:09)
==================================================
         GRID superscalar RUNNING
==================================================
```

Fig. 7. Information provided by GSTAT at the application level.

```
==============================================
GRID superscalar Tasks STATUS in WORKERS
==============================================
------------------------------------------
 hermes.inf-cr.uclm.es
---------------------
 DONE: 35 ACTIVE: 2 PENDING: 0 FAILED: 0
------------------------------------------
 aristoteles.inf-cr.uclm.es
--------------------------
 DONE: 85 ACTIVE: 2 PENDING: 0 FAILED: 0
------------------------------------------
==============================================
```

Fig. 8. Information provided by GSTAT at the grid node level.

4.1. Monitoring capabilities

(a) *Information on the whole application and the Master node.* This case is accessed just by invoking the *gsstat* monitor. The information is presented in the format shown in Fig. 7. Here, MASTER identifies the Grid node (hostname) where the application was started. The information provided includes the state (Done, Active, Pending and Failed) of the entire set of tasks handled by the application, plus the whole application execution time. In addition, for the finished, "DONE", tasks the monitor shows the minimum, average and maximum execution times. With this monitoring capability the user can verify the state of his/her application.

(b) *Information at the Grid node level.* This option is accessed by invoking *gsstat* with the *-w* switch (*gsstat-w*). The result is depicted in Fig. 8. Now, information about the status of the tasks running at the workers (Grid nodes) level is displayed. All the workers where individual tasks are allocated are considered. Information about a specific worker can be requested invoking *gsstat-w worker*.

(c) *Information by task Status.* This option is accessed by invoking *gsstat* with the *-d*, the *-a*, the *-p*, or the *-f* switches, see Fig. 6. The switches request information on the DONE, ACTIVE, PENDING or FAILED tasks, respectively. In this case, the Grid time of each task (the time between the submission of the task until reception of the results), and its hosting Grid node, are presented in the output, see Fig. 9(a). Information on the ACTIVE, DONE, PENDING or FAILED tasks in a single worker can be obtained by invoking *gsstat-aw worker*, *gsstat-dw worker*, *gsstat-pw worker* or *gsstat-fw worker*.

(d) *Information on an individual task.* Also, it is possible to visualize information regarding a single task, see Fig. 9(b). To such an end, we invoke *gsstat-t N*, where *N* is the number of the task (assigned by the Master runtime library, a task identifier). In addition, it is possible to obtain the Grid, Wall, CPU and the Round Trip Grid (RTG) times of a finalized task (status DONE), see Fig. 9(c). Here, RTG time refers to the time used in the scheduling, communication, and queueing processes.

a
```
===============================================
        GRID superscalar Tasks ACTIVE
===============================================
Task   Time       Running On
------ --------   --------------------------
 30  00:04:02     hermes.inf-cr.uclm.es
 32  00:02:20     aristoteles.inf-cr.uclm.es
 33  00:00:38     hermes.inf-cr.uclm.es
 34  00:00:35     aristoteles.inf-cr.uclm.es
-----------------------------------------------
4 Tasks ACTIVE
===============================================
```

b
```
=========================================================
     GRID superscalar STATUS for Task number 30
=========================================================
Task   Time      Status      Running On
------ --------  ----------  --------------------------
 30  00:07:09    DONE        hermes.inf-cr.uclm.es
=========================================================
```

c
```
=========================================================
     GRID superscalar TIMES for Task DONE number 1
=========================================================
 GRID Time       WALL Time       CPU Time       RTG Time
 ------------    ------------    ------------   ------------
  00:05:05        00:04:51        00:02:03       00:00:14
 ------------    ------------    ------------   ------------
=========================================================
Executed  in hermes.inf-cr.uclm.es    (node compute-0-2.local)
=========================================================
```

Fig. 9. The information provided by GSTAT at the task level. (a) Information for tasks in ACTIVE status. (b) Information for an individual task. (c) Temporal information for task DONE.

4.2. Steering capabilities

Fig. 10 shows the steering options available to the user through the interface of GSTAT, in accordance to the design in Fig. 3.

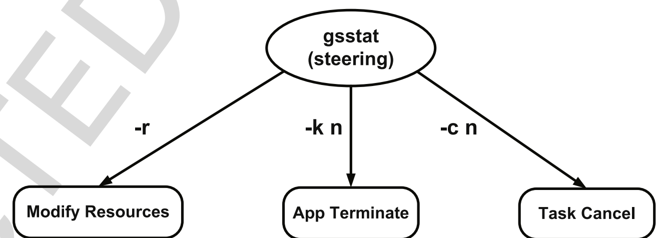

Fig. 10. Steering capabilities included in GSTAT.

Fig. 10 shows that the user can add or remove resources and modify the characteristics of the defined resources with the command *gsstat-r*. With this option, the user can add or remove workers and/or modify the number of processors assigned. In addition, it is possible to finish the application invoking *gsstat-k N*. Here, *N* represents a key task. Previous tasks are allowed to finish whereas later tasks, if scheduled, are cancelled. When needed it is possible to restart the application from the task $N + 1$. Finally, the user can cancel an individual task invoking *gsstat-c N*, *N* being the task number.

UNCORRECTED PROOF

Please cite this article in press as: S. Reyes, et al., Monitoring and steering Grid applications with GRID superscalar, Future Generation Computer Systems (2009), doi:10.1016/j.future.2009.12.002

Table 1
Number of executed tasks and summary of average times for the GHyper application as supplied by the database of the GSTAT monitor.

	Aristoteles	*Carrasca*	*Hermes*	*Popocatepetl*
Executed tasks	140	165	65	64
Avg. latency (ms)	0.01	5	0.01	300
Avg. grid time (s)	154	131	341	326
Avg. wall time (s)	140	116	333	276
Avg. CPU time (s)	140	113	150	182
Avg. RTG time (s)	14	15	8	50

Table 2
Number of executed task and summary of average times for the block matrix multiplication as supplied by the database of the GSTAT monitor.

	Aristoteles	*Carrasca*	*Hermes*	*Popocatepetl*
Executed tasks	104	232	80	96
Avg. latency (ms)	0.01	5	0.01	300
Avg. grid time (s)	120	46	144	90
Avg. wall time (s)	104	32	134	43
Avg. CPU time (s)	45	31	58	37
Avg. RTG time (s)	16	14	10	47

5. Examples of application

GSTAT allows analyzing the behavior of typical high throughput and high performance applications in a computational Grid. Therefore, we have considered an example of both cases.

5.1. Monitoring examples

In order to carry out the examples, the following resources have been used: as Grid Master (*qcycar*) a PC biprocessor × 86_64 with 4 GB RAM, running on CentOS [42]. As Grid Workers we use four computer clusters: *aristoteles*, *hermes*, *carrasca*, and *popocatepetl*. *Aristoteles* is formed by 42 × 86_64 processors with 2 GB RAM. In turn, *hermes* consists of 12 × 86 processors with 1 GB RAM. *Carrasca* has 8 × 86_64 processors and 2 GB RAM. Finally, *popocatepetl* is a cluster with 8 × 86_64 processors and 1 GB RAM. All the clusters run the configuration and management cluster system Rocks [43]. *Aristoteles*, *carrasca*, and *popocatepet* use SGE as queueing system [44], whereas *hermes* uses PBS [45]. All the systems use Globus Toolkit 4 as Grid middleware [21]. Physically, *qcycar*, *aristoteles* and *hermes* are located in the QCyCAR group of the Castilla-La Mancha University at Ciudad Real (Spain), *carrasca* is in the Departamento de Astrofísica Molecular e Infrarrojo (DAMIR) of the CSIC at Madrid (Spain), and *popocatepetl* is at the Puebla University in Puebla (Mexico). Average latencies of 0.01, 5 and 300 ms were found for connections from the *qcycar* Master to *aristoteles* and *hermes* (they all in Ciudad Real), to *carrasca* (Madrid) and to *popocatepetl* (Mexico), respectively. For the experiments, eight processors have been used, two per Worker.

As a first example, we consider a parameter sweep problem of great interest in the physical molecular field: the evaluation of molecular potential energy hypersurfaces. These are needed for analyzing the dynamics of molecular motions, and for the spectroscopic characterization of molecular species. In this context, we have developed a methodology to determine the set of molecular structures that characterizes a given potential energy hypersurface. To such end we use a complete set of molecular coordinates. In addition, using GRID superscalar we have developed an application (GHyper) able to generate automatically the previous set of molecular structures, run the electronic structure calculations needed to obtain the values of the potential energy, and organize the results on a computational Grid [2,46,47]. This case represents a parameter sweep example in a high throughput computing environment. In particular, we will consider the computation of the potential energy hypersurface of protonated formaldehyde, a molecule of astrophysical interest [48].

We consider a partial resolution of the potential energy hypersurface of protonated formaldehyde. Thus, we use two internal coordinates (angles), which define a two-dimensional problem. This case involves 433 individual electronic structure calculations. Each calculation corresponds to one task in our parameter sweep problem. The GHyper application was monitored from the Master *qcycar* using GSTAT. The makespan for the application was 3 h and 12 min. The average time inverted in the resolution of each task has been 3 min and 20 s. Table 1 shows the number of executed tasks, the average latencies, the average Grid time, the average Wall time, the average CPU time and the RTG (Round Trip Grid) time per task in each Worker.

Table 1 shows that a smaller number of tasks were executed on the systems with less resources (processor, memory): *hermes* and *popocatepetl*. This fact is also reflected in the relationship between Wall time and CPU time. In fact, *aristoteles* and *carrasca* show no difference in practical terms, whereas *hermes* and *popocatepetl* exhibits an average Wall time that almost doubles the average CPU time. This can be attributed to the smaller amount of memory available in the last two: 1 GB RAM versus 2 GB in *aristoteles* and *carrasca*. Electronic structure calculations are input–output bound. Therefore, in the systems with less memory more use of the disk is needed, and this is reflected in a difference between the Wall and CPU time.

On the other side of the scientific computing spectrum we have high performance computing applications. Here, dependencies do exist among the distributed subtasks. Therefore, the need of synchronization appears. As an example, we have the block matrix multiplication. In this work, we consider two matrices of 512 × 512 factorized in eight blocks of size 8 × 8. The application is monitored from the Master *qcycar* using GSTAT. The makespan for the application was 1 h and 53 min for 512 tasks. The average time inverted in the resolution of each task has been 1 min and 24 s. Table 2 shows the number of executed tasks, the average latencies, the average Grid time, the average Wall time, the average CPU time and the RTG time per task in each Worker.

Table 2 shows that, as in the previous example, more tasks are allocated to the more powerful Grid nodes: *aristoteles* and *carrasca*. On the other hand, we also observe that the *aristoteles* and *hermes* nodes exhibit the largest difference between average Wall time and average CPU time. This behavior can be explained in terms of the scheduling policy used by GRIDSs. To exploit data locality, GRIDSs tries to allocate dependent tasks in the same workers. Thus, communication overhead is reduced. In the present case, *aristoteles* and *hermes* collect the dependent tasks and, consequently, they need additional time to resolve the dependencies.

The data in Tables 1 and 2 also show that the largest difference between average Grid time and average Wall time appears in the Grid node with the largest latency (*popocatepetl*). The GSTAT data permit to quantify these differences as increases of 15% and 52% for the examples in Tables 1 and 2, respectively. It is observed, in Tables 1 and 2, that RTG time is related to the latency.

5.2. Steering example

The steering capabilities can also be illustrated using the hypersurface computation test case. The following resources have been used: as Grid Master (*bscgrid01*) a PC biprocessor × 86 with 1 GB RAM, running on Linux. Physically, *bscgrid01* is located in the Barcelona Supercomputing Center (BSC-CNS). As Grid Workers we use *aristoteles* and *hermes*, which are described in the monitoring examples section. For this application, Fig. 11(a) shows the tasks in ACTIVE state in a given moment (*gsstat-a*). Here, we observe one of the tasks, number 49, taking much more time (the average time is

UNCORRECTED PROOF

Please cite this article in press as: S. Reyes, et al., Monitoring and steering Grid applications with GRID superscalar, Future Generation Computer Systems (2009), doi:10.1016/j.future.2009.12.002

a
```
=============================================
      GRID superscalar Tasks ACTIVE
=============================================
 Task    Time       Running On
 ------  --------   -------------------------
   49  00:29:18   hermes.inf-cr.uclm.es
   74  00:04:21   hermes.inf-cr.uclm.es
   76  00:02:10   aristoteles.inf-cr.uclm.es
   77  00:01:01   aristoteles.inf-cr.uclm.es
---------------------------------------------
 4 Tasks ACTIVE
=============================================
```

b
```
=============================================
      GRID superscalar Tasks ACTIVE
=============================================
 Task    Time       Running On
 ------  --------   -------------------------
   76  00:02:29   aristoteles.inf-cr.uclm.es
   77  00:01:20   aristoteles.inf-cr.uclm.es
   78  00:00:01   hermes.inf-cr.uclm.es
---------------------------------------------
 3 Tasks ACTIVE
=============================================
```

c
```
=============================================
      GRID superscalar Tasks ACTIVE
=============================================
 Task    Time       Running On
 ------  --------   -------------------------
   77  00:01:53   aristoteles.inf-cr.uclm.es
   49  00:00:36   aristoteles.inf-cr.uclm.es
   78  00:00:34   hermes.inf-cr.uclm.es
   79  00:00:21   hermes.inf-cr.uclm.es
---------------------------------------------
 4 Tasks ACTIVE
=============================================
```

Fig. 11. Task cancel steering capability. (a) Before cancel task number 49, (b) After cancel task number 49, (c) New execution of task number 49.

3 min 11 s, see Fig. 7) than the others. Among other causes this can be due to problems in the Grid node where the task is running. The user can cancel this task (*gsstat-c 49*). After that, Fig. 11(b) shows that task 49 has disappeared from the system. Next the GRIDSs runtime reschedules the task (*gsstat-a*). The task is then rescheduled, tentatively, in the same Worker (Grid node). If the rescheduling fails, the task is sent to a different Grid node. In the example, Fig. 11(c) shows task 49 again in execution.

6. Conclusions

This paper presents the design and implementation of a monitoring and steering tool (GSTAT) for Grid applications, integrated in the GRIDSs programming framework. A general design conforming to the Grid Monitoring Architecture defined by the Open Grid Forum is presented. The system allows getting information about the task status at the whole Grid system or at individual Grid nodes. Information on individual tasks or tasks in a given status is also provided. The steering capabilities provide interaction with the Grid system to allow modifying the resources associated to the application. Restarting and rescheduling capabilities are also available. GSTAT provides on-line (during the execution) and off-line (after the execution) information of the application and its tasks. GSTAT has been designed and implemented in a way allowing integration with all the GRID superscalar versions. The present version of GSTAT uses the command line as user interface. Part of the ongoing work is to develop a web-based graphical user interface.

As a further extension, visualization of the tasks graph could be integrated in the GRIDSs task dependence dynamic graph monitor (GSM) [49].

Acknowledgements

The authors wish to thank the Barcelona Supercomputing Center-Centro Nacional de Supercomputación (BSC-CNS), the Facultad de Ciencias Químicas of the Universidad Autónoma de Puebla (Mexico) and the DAMIR group of the Consejo Superior de Investigaciones Científicas (CSIC, Spain) for allowing the use of their systems.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling scalable virtual organizations, International Journal of High Performance Computing Applications 15 (2001) 200–222.
[2] S. Reyes, C. Muñoz-Caro, A. Niño, R.M. Badia, J.M. Cela, Performance of computationally intensive parameter sweep applications on Internet-based Grids of computers: The mapping of molecular potential energy hypersurfaces, Concurrency and Computation: Practice and Experience 19 (2007) 463–481.
[3] J. Díaz, S. Reyes, A. Niño, C. Muñoz-Caro, Derivation of self-scheduling algorithms for heterogeneous distributed computer systems: Application to internet-based grids of computers, Future Generation Computer Systems 25 (6) (2009) 617–626.
[4] R. Sirvent, J.M. Pérez, R.M. Badia, J. Labarta, Automatic Grid workflow based on imperative programming languages, Concurrency and Computation: Practice and Experience 18 (2006) 1169–1186.
[5] M. Gerndt, R. Wismueller, Z. Balaton, et al. Performance Tools for the Grid: State of the Art and Future, 30 of Research Report Series, in: Lehrstuhl fuer Rechnertechnik und Rechnerorganisation (LRR-TUM) Technische Universitaet Muenchen, Shaker Verlag, 2004. ISBN 3-8322-2413-0.
[6] R.L. Ribler, J.S. Vetter, H. Simitci, D.A. Reed, Autopilot: Adaptive control of distributed applications, in: Proceedings of the Seventh IEEE Symposium on High-Performance Distributed Computing, 1998, pp. 172–179.
[7] Z. Balaton, P. Kacsuk, N. Podhorszki, Application monitoring in the grid with GRM and PROVE, in: Proceedings of the International Conference on Computational Science, ICCS2001, Part I, in: Lecture Notes in Computer Science, vol. 2073, Springer-Verlag, San Francisco, CA, USA, 2001, p. 253.
[8] J. Joyce, G. Lomow, K. Slind, B. Unger, Monitoring distributed system, ACM Transactions on Computer System 5 (2) (1987) 121–150.
[9] S. Shende, A.D. Maloney, The TAU parallel performance system international, Journal of High Performance Computing Applications 20 (2) (2006) 287–311.
[10] B.P. Miller, M.D. Callaghan, J.M. Cargille, J.K. Hollingsworth, R.B. Irvin, K.L. Karavanic, K. Kunchithapadam, T. Newhall, The paradyn parallel performance measurement tool, IEEE Computer 28 (1995) 37–46.
[11] A. Morajko, O. Morajko, T. Margalef, E. Luque, MATE: Dynamic performance tuning environment, LNCS 3149 (2004) 98–107.
[12] S. Moore, D. Cronk, K. London, J. Dongarra, Review of performance analysis tools for MPI parallel programs, LNCS 2131 (2001) 241–248.
[13] S. Zanikolas, R. Sakellariou, A taxonomy of grid monitoring systems, Future Generation Computer Systems 25 (2005) 163–188.
[14] H. Garcia-Molina, F. Germano Jr., W.H. Kohler, Debugging a distributed computing system, IEEE Transactions on Software Engineering 10 (2) (1984) 210–219.
[15] R.T. Snodgrass, Monitoring distributed systems: A relational approach, Ph.D. Dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa, 1982.
[16] M.D. Harrison, Monitoring a target network to support subsequent host simulation. Res. Rep., Dept. of Computer Science, University of York, Toronto, Ontario, Canada, 1984.
[17] http://www.ogf.org. Last visit: 2009-02-23.
[18] R. Aydt, D. Quesnel, Performance Data Usage Scenarios (Draft 1). Technical Report, Global Grid Forum. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/, 2000.
[19] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski, A grid monitoring architecture, Global Grid Forum, http://www.ogf.org/documents/GFD.7.pdf, 2002 Last visit: 2009-02-23.
[20] Z. Balaton, G. Gombás, Resource and Job Monitoring in the Grid, in: Lecture Notes in Computer Science, vol. 2790, 2004, pp. 404–411.
[21] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, A directory service for configuring high performance distributed computations, in: Proceedings of the Sixth IEEE Symposium on High Performance Distributed Computing, IEEE Computer Society Press, 1997, pp. 365–375.
[22] http://www.globus.org. Last visit: 2009-02-23.
[23] M.L. Massie, B.N. Chun, D.E. Culler, Ganglia distributed monitoring system: Design, Implementation, and Experience, Parallel Computing 30 (2004) 817–840.
[24] R. Wolski, N. Spring, J. Hages, The network weather service: A distributed resource performance forecasting service for metacomputing, Future Generation Computing System 15 (1999) 745–755.
[25] M. Baker, G. Smith, GridRM: A resource monitoring architecture for the grid, Lecture Notes in Computer Science 2536 (2002) 268–273.
[26] S. Andreozzi, et al., GridICE: A monitoring service for Grid systems, Future Generation Computer Systems 4 (2005) 559–571.
[27] B. Tierney, D. Gunter, NetLogger: A toolkit for distributed system performance tuning and debugging, in: G.S. Goldszmidt, J. Schönwälder (Eds.), Proceedings of the IFIP/IEEE Eighth International Symposium on Integrated Network Management, IM 2003, Kluwer, vol. 246, 2003 pp. 97–100.
[28] B. Balis, M. Bubak, T. Szepieniec, R. Wismüller, M. Radecki, Monitoring grid applications with grid-enabled OMIS monitor, in: Proceedings of the First European Across Grids Conference, in: Lecture Notes in Computer Science, vol. 2970, Springer-Verlag, Santiago de Compostela, Spain, 2004, pp. 230–239.
[29] T.L. II, R. Wismüller, OMIS 2.0: A universal interface for monitoring systems, in: Proceedings of the Fourth European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, in: Lecture Notes in Computer Science, vol. 1332, Springer-Verlag, Cracow, Poland, 1997, pp. 267–276.
[30] F. Baude, L. Baduel, D. Caromel, A. Contes, F. Huet, M. Morel, R. Quilici, in: Jose C. Cunha, Omer F. Rana (Eds.), Programming, Composing, Deploying for the Grid in GRID COMPUTING: Software Environments and Tools, Springer Verlag, 2006.

UNCORRECTED PROOF

Please cite this article in press as: S. Reyes, et al., Monitoring and steering Grid applications with GRID superscalar, Future Generation Computer Systems (2009), doi:10.1016/j.future.2009.12.002

[31] F. Baude, A. Bergel, D. Caromel, F. Huet, O. Nano, J. Vayssière, IC2D: Interactive control and debugging of distribution, Lecture Notes in Computer Science 2179 (2001) 193–200.
[32] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolski, The GrADS project: Software support for high-level grid application development, International Journal of High Performance Computing Applications 15 (4) (2001) 327–344.
[33] D. Lorenz, S. Borovac, P. Buchholz, H. Eichenhardt, T. Harenberg, P. Mättig, M. Mechtel, R. Müller-Pfefferkorn, R. Neumann, K. Reeves, Ch. Uebing, W. Walkowiak, Th. William, R. Wismüller, Job monitoring and steering in D-Grid's high energy physics community grid, Future Generation Computer Systems 25 (3) (2009) 308–314.
[34] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi, M. Livny, Pegasus: Mapping Scientific Workflow onto the Grid, Across Grids Conference, Nicosia, Cyprus, 2004.
[35] I. Taylor, M. Shields, I. Wang, A. Harrison, The Triana Workflow Environment: Architecture and Applications, Workflows for e-Science, Springer, New York, 2007.
[36] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, B. Ullmer, The grid application toolkit: Towards generic and easy application programming interfaces for the Grid, Proceedings of the IEEE 93 (3) (2005) 534–550.
[37] I. Wang, I. Taylor, T. Goodale, A. Harrison, M. Shields, gridMonSteer: Generic architecture for monitoring and steering legacy applications in grid environments, in: Proceedings of the UK e-Science All Hands Meeting 2006, September 2006.
[38] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipning, J. Qin, M. Siddique, H.L. Truong, A. Villazon, M. Wieczorek, ASKALON: A grid application development and computing environment, in: In 6th International Workshop on Grid Computing, IEEE Computer Society Press, New York, 2005.
[39] R.M. Badia, J. Labarta, R. Sirvent, J.M. Pérez, J.M. Cela, R. Grima, Programming grid applications with GRID superscalar, Journal of Grid Computing 1 (2003) 151–170.
[40] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A Resource Management Architecture for Metacomputing Systems, Proceedings of the Workshop on Job Scheduling Strategies form Parallel Processing 1459 (1998) 62–82.
[41] http://www.perl.org. Last visit: 2009-02-23.
[42] http://www.centos.org. Last visit: 2009-02-23.
[43] http://www.rosckscIusters.org. Last visit: 2009-02-23.
[44] http://www.gridengine.sunsource.net. Last visit: 2009-02-23.
[45] http://www.openPBS.org. Last visit: 2009-02-23.
[46] S. Reyes, C. Muñoz-Caro, A. Niño, Desarrollo de aplicaciones fsico-moleculares en entornos grid, in: Proceedings of XVII JORNADAS DE PARALELISMO. A, 2006 pp. 259–263.
[47] S. Reyes, C. Muñoz-Caro, A. Niño, Grid computing. Applications in the computational chemistry field, Electronic Structure Principles and Applications, ESPA 2006, Santiago de Compostela, Spain. Communication. 2006 pp. 18–21.
[48] M.E. Castro, A. Niño, C. Muñoz-Caro, Theoretical Chemistry Accounts 119 (2008) 343–354.
[49] http://www.bsc.es/plantillaH.php?ca_id=150. Last visit: 2009-11-9.

S. Reyes is a Computer Science Engineer from the Granada University, Spain. He is currently a member of the Department of Information Technologies and Systems at the Castilla-La Mancha University (UCLM), Spain. He is a lecturer in Computer Networks. His present research interests are in clustering and Grid computing.

C. Muñoz-Caro received her Ph.D. degree from the Complutense University in Madrid, Spain. Her work covers Computer Science and Molecular Physics. Formerly, she worked at the National Research Council of Spain (CSIC), currently she is a member of the Department of Information Technologies and Systems at the Castilla-La Mancha University (UCLM), Spain. She has been a lecturer in Theoretical Chemistry, Management Information Systems and Object Oriented Programming. Her research interests are in the fields of clustering and Grid computing applied to Scientific Computing.

A. Niño received his Ph.D. degree from the Complutense University in Madrid, Spain. His multidisciplinary work involves Computer Science and Molecular Physics. Formerly, he worked at the National Research Council of Spain (CSIC) and is currently a member of the Department of Information Technologies and Systems at the Castilla-La Mancha University (UCLM), Spain. He has been a lecturer in Theoretical Chemistry, Software Engineering and Object Oriented Programming. His research interests are in the fields of clustering and Grid computing applied to Scientific Computing

R. Sirvent has a Ph.D. degree in Computer Science (UPC, February 2009). Priorly, he got a M.S. degree in Computer Science at the Facultat d'Informàtica de Barcelona (UPC, July 2002). He has been involved in research activities at the European Center of Parallelism of Barcelona (CEPBA) from 2002 to 2005. Currently, he holds a permanent position as Researcher since August 2005 at the Barcelona Supercomputing Center inside the Computer Sciences department (Grid Computing and Clusters). His main research interests are related to High Performance Computing, Grid programming models and tools, automatic workflow generation and fault tolerance mechanisms. He is currently involved in FP6 Grid projects BEinGRID and BREIN and is one of the founders of the GRID superscalar project.

R.M. Badia received the B.Sc. and Ph.D. degrees in computer science from the Technical University of Catalonia, Barcelona, Spain in 1989 and 1994. From 1989 to 2007 she has been lecturing at the Technical University of Catalonia on computer organization and architecture and VLSI design, both in undergraduate and graduate programmes. She held an Associate Professor position at the Department of Computer Architecture, Technical University of Catalonia from 1997 till 2007. Since May 2008 she has been a Scientific Researcher at the Spanish National Research Council (CSIC). Since year 2005 she has been the manager of Grid computing and clusters at the Barcelona Supercomputing Center, a position that she currently holds full-time. Her current research interests include performance prediction and modelling of MPI applications, programming models for multi-core architectures and programming models for Grid environments. She has authored around 80 publications in international conferences and journals.