CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

GMarte: Grid Middleware to Abstract Remote Task Execution

J. M. Alonso, V. Hernández and G. Moltó *,†

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, Spain.

SUMMARY

Grid Computing technologies are now being largely deployed with the widespread adoption of the Globus Toolkit as the industrial standard Grid middleware. However, its inherent steep learning curve discourages the usage of these technologies for non experts. Therefore, to leverage the usage of Grid Computing, it is important to have high level tools that simplify the process of remote task execution. In this paper we introduce a middleware, developed on top of the Java Commodity Grid, which offers an object-oriented, user friendly Application Programming Interface, from the Java language, which eases remote task execution for computationally intensive applications.

KEY WORDS: Grid Middleware, Scheduling, Grid Computing, High Performance Computing

1. INTRODUCTION

Grid Computing has emerged as the solution for the computational problems of organisations, by enabling the usage of remote resources for executing computationally expensive tasks. There are many scientific fields in which time consuming applications are very common and therefore, they could greatly benefit from Grid Computing. However, the complexity of the underlying middleware very often discourages scientists from porting their applications to the Grid.

Among all the Grid middlewares available, the Globus Toolkit [**?**] represents the *de facto* standard for deploying large-scale computational Grids. However, the steep learning curve of this middleware prevents the exploitation of its features until a training period with this

*Correspondence to: Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022, Valencia, Spain. Tel: +34963877356, Fax: +34963877359.
†E-mail: {jmalonso,vhernand,gmolto}@dsic.upv.es
Contract/grant sponsor: This work has been funded by the Spanish Ministry of Science and Technology, partially supported by the Structural Funds of the European Regional Development Fund (ERDF) under the project GRID-IT; contract/grant number: TIC2003–01318.

Received 1 November 2004
Revised 16 April 2005

software is fulfilled. In fact, in many engineering fields, scientists require tools to be able to exploit the benefits of the Grid, in their own applications, without worrying about the implementation details and the burden of the underlying Grid middleware. Therefore, in order to widespread the usage of Grid Computing, it is crucial that these technologies get closer to the user.

In order to reduce this gap, from the Grid to the end user, some approaches have been taken into account. The Commodity Grid Kits offer a high level of abstraction that allow computational scientists, who are not expert in Grid programming, to use the Grid services as part of the scientific problem-solving process. In fact, the Java Commodity Grid (CoG) Kit 1.2 [**?**] provides an excellent gateway, from the Java programming language, to employ the Globus Toolkit services, such as GridFTP, the Monitoring and Discovery Service (MDS) or the Globus Resource Allocation Manager (GRAM).

However, a user of Java CoG still has to be aware of the main components and services offered by the Globus Toolkit. Even more, these services must be adequately employed and combined, like building bricks, to achieve the intended purpose, which, in general, represents remote task execution. This involves dealing with resource information discovery, file staging, job execution and status control, among others. Entirely new approaches to software development and programming are required for Grid Computing, so that it becomes accessible to ordinary scientists [**?**].

In this paper we introduce GMarte, a middleware developed on top of the Java CoG 1.2, which provides an object-oriented view of the Grid, offering a high level API that enables the transparent execution of already existing computationally intensive sequential or parallel applications in a Globus-based Grid environment, with fault-tolerant metascheduling capabilities. Emphasis is stressed on the integration of High Performance Computing techniques and Grid Computing, allowing to perform concurrent parallel executions on multiprocessor machines of a Grid deployment.

The remainder of the paper is structured as follows: First, section **??** introduces the related work and justifies the development of a new middleware. Then, section **??** presents the main functionality of GMarte, providing partial disclosure about the design and introducing the important topic of application portability. Next, section **??** details the most important features offered by this middleware. Later, section **??** addresses the metascheduling functionality of GMarte. Finally, section **??** describes two practical examples that reveal the ease of usage of this software and section **??** summarises the main achievements pointing to future development.

2. MOTIVATION AND RELATED WORK

There are currently several research actions that provide a framework for simplifying the usage of distributed resources. The Grid Application Framework for Java (GAF4J) [**?**] is a simple framework of classes that abstracts the essentials of interfacing with a Grid infrastructure, assumed to be Globus Toolkit 2.0. Javelin [**?**] is a Java-based infrastructure for Internet-based parallel computing, which provides a self-designed framework for achieving execution using *Remote Method Invocation* for communications. Also, ProActive [**?**] is a Java library

for parallel, distributed and concurrent computing, by means of the creation of active objects which can be distributed and communicate through the interfaces provided.

All these tools aim at simplifying the process of either porting or developing a Java application to be executed on a distributed environment. However, GMarte is a middleware, developed in Java, that easily allows execution on Globus-based distributed resources for already existing applications, that might have been written in any programming language, not only Java. Of course, the application must be able to run on the remote machine without any modification. The user would create a small Java application using the API of GMarte, which would be in charge of performing the task allocation procedure for successful execution.

There are also open projects that aim at harnessing the execution on distributed resources, thus providing *service middleware* functionality. Legion [**?**] is an object-based middleware that aims at creating a single virtual supercomputer from a set of distributed resources. While Legion provides an API for resource management, as well as a for managing executions, it is based on its own self-designed framework, thus being restricted to Legion-based distributed deployments. Condor [**?**] is a software system that effectively utilises the computing power of a set of resources. However, although Condor provides a gateway to interface with Globus (Condor-G) [**?**], it only offers a set of command-line applications without an API.

GMarte combines three important aspects. First, it provides a framework to easily port an already existing application to the Grid. Second, the framework is developed on top of the industrial standard Globus Toolkit middleware, that guarantees its functionality on all the largely deployed Globus-based Grids. Third, it provides an intuitive, object-oriented high level API, rather than a closed software tool, that enables an extremely versatile behaviour.

There are many advantages of providing a high level API for the Globus Toolkit. First of all, the complexity of the underlying middleware is completely hidden, providing the user with a natural interface, who no longer has to know the internal services provided by Globus. Next, as an additional functionality, the API offered by GMarte can be employed to easily develop Java applications which interact with Globus-based resources. For example, a monitoring application, that supplies information about the state and features of the resources of a Grid infrastructure, can be developed using the API for resource management available in GMarte. Thus, the user does not need to know the low level details of the MDS service. Finally, once the foundations of the API are established, it is quite straightforward to develop a Grid Portal [**?**] which provides a graphical user interface, accessible from a web browser, in order to run applications as well as accessing the result data. Even more, the functionality offered by GMarte can be exposed as a Grid Service that provides metascheduling capabilities, thus enabling bindings to the latest releases of Globus. The GMarte middleware was originally designed for scientists, to abstract the process of remote execution on a Grid. However, its implementation as a layered API enables to use a subset of this functionality in order to develop another Grid-related applications.

3. OVERVIEW OF THE MIDDLEWARE

GMarte can be described as a Globus-based object-oriented client-side Grid middleware, developed in Java, that provides a high level API, which mainly enables the transparent

execution of an already existing self-contained parallel computing-based application in a Grid environment, offering fault-tolerant metascheduling capabilities for the allocation of tasks to computational nodes.

The analysis of this definition clearly states the goals of GMarte. The underlying Grid middleware employed is the Globus Toolkit 2.4, also known as pre-web services, because it is the current *de facto* standard in computational Grids, thus guaranteeing compatibility with all the Globus-based Grid deployments. GMarte has been developed as an object-oriented middleware because it aims to provide a very natural view of Grid Computing technologies, something that can only be achieved through object abstractions. The choice of the Java language to implement GMarte enables rapid interoperability with Grid services, thus guaranteeing future extensibility. The middleware has been designed for the execution of self-contained applications, that is, applications that do not rely on runtime dynamic dependences, therefore enlarging the range of machines that can be used for execution.

As stated, the middleware supports the execution of parallel applications implemented with the MPI library, thus exploiting the capabilities of the multiprocessor nodes of a Grid infrastructure. Also, GMarte offers metascheduling functionality for the allocation of a set of tasks on a Grid environment, providing fault tolerance via application-dependent checkpointing or simply by restarted execution.

The term metascheduling or superscheduling is commonly employed when the scheduling decision (which task has to be executed on which resource) involves machines from multiple administrative domains. GMarte allocates tasks to computational resources and, if this machine is a multiprocessors or a cluster of PCs, its local queue manager will finally assign the task to the internal computing nodes. Throughout the paper, the scheduling and metascheduling terms will be employed with the same meaning.

It is important to point out that GMarte is a client-side middleware which enables the user to describe *what* has to be executed on the Grid via a high level API, without detailing the procedure involved in remote task execution. Then, GMarte interacts with the Globus-based resources of existing Grid deployments, in a transparent manner for the user, to allocate the tasks and achieve successful execution. Besides, regardless of GMarte being developed in Java, the user application may have been coded in any language. The GMarte high level API is employed to describe the user's execution requirements but, in the end, the RSL (Resource Specification Language) is used, which imposes no requirements on the user application. The golden rule can be summarised like follows: If the user application is able to be executed without modifications on the remote resource, then it can be employed under the GMarte framework.

The middleware is currently available[†] upon request in order to track the usage and interest on its features. It currently consists of more than 10000 lines of Java code, with 70 classes organised in four general packages: Tasks, Resources, Execution and Scheduling, which encapsulate all the functionality related to these large areas. Figure **??** summarises the

[†]GMarte - http://www.grycap.upv.es/GMarte

Figure 1. Class diagram outlining the object-oriented design of GMarte. Most of the methods have been omitted for brevity.

most important classes involved in the object-oriented design of GMarte. In the diagram only the most relevant methods of each class are exposed for clarity.

In the center, the *GridTask* class arises as the abstraction for a task that has to be executed in the Grid. The *GridTaskStudy* represents a collection of GridTasks, enabling the definition of computational studies where a set of tasks must be executed, for example, in parameter sweep applications. In the right hand side of the diagram, the *GridResource* appears as the abstraction for a computational resource in a Grid, being a *TestBed* just a pool of GridResources.

A GridTask may provide a list of requirements (*GridTaskLoR*), which must be satisfied by the GridResource in order to proceed execution. A GridResource can also specify a set of execution policies (*ExecutionPolicies*), which describe the limitations that the resource can impose, regardless of its available capabilities. The *Scheduler* abstract class defines the interface that all the schedulers must implement. A Scheduler is in charge of performing the allocation of GridTasks to GridResources. For example, the implemented *AdvancedScheduler* performs fault-tolerant task allocation with the help of four concurrent threads. With the component oriented design, different scheduling policies can be implemented and plugged at the middleware level in order to assess their performance.

The rest of the classes, which will be mentioned through the rest of the paper, give proper support to the conceptualisation and design of the middleware, providing a natural mapping from the real world of the Grid to an object-oriented programming environment.

3.1. Portability: the key for Grid execution

GMarte is devoted to the execution of already existing self-contained applications and thus, it does not require their source code modification before execution under a Grid infrastructure. However, provided that the Grid is an heterogeneous pool of resources, it is important to achieve some level of portability to the application that guarantees successful execution on a wide range of machines. Therefore, generating a portable application represents one important key for success in the execution on a Grid infrastructure.

A resource must provide an appropriate execution environment for the application to be run and therefore, either the dynamic libraries dependences are guaranteed to be present at execution time or the application must be self-contained. Given that one of the features of the Grid is site autonomy [**?**], we may not have access to the remote resource in order to install the required libraries, so, in general, we are restricted to adapt our application to the Grid.

Static linking techniques enable to generate a self-contained executable file, which no longer depends on any dynamic library. Besides, in order to take into account minor architectural changes, all the architecture-dependent compiler optimisation flags, such as *-march* or *-mcpu*, must disappear from the compilation process, in order to guarantee the generation of portable computer code. Also, platform-dependent numerical kernel libraries, such as the BLAS [**?**] and LAPACK [**?**] provided by the Intel Math Kernel Library, should be avoided, as they may provoke the execution of illegal instructions on the remote resource if both the compilation and the execution platforms do not match. For parallel applications based on MPI, the communication layer may also be introduced into the executable. In a related work [**?**], we have employed the MPICH [**?**] implementation specially configured to disable shared memory communication between processes on a multiprocessor machine (i.e. using socket-based communication), since it is known to potentially introduce memory leak problems because of relying on the System V IPC facilities [**?**].

Following these compilation recommendations, it is possible to generate a self-contained portable parallel application that can be executed on a wide variety of Linux machines, which is one of the principal platform targets employed when using the Globus Toolkit. This has been ensured by parallel executions in different platforms such as Pentium III, Pentium Xeon and Intel Itanium 2, with different Linux flavours, such as Red Hat Linux Advanced Server, Red Hat 8.0 and Debian GNU/Linux.

In fact, GMarte expects the executable application to be self-contained and properly compiled for its execution on the remote machines. Additionally, the user can generate several executable versions for the different architectures (IA32, IA64, etc) with the purpose of taking advantage of their performances. This way, the middleware would choose the best executable file prior to execution on the computational node.

4. MIDDLEWARE FEATURES

As mentioned previously, GMarte's API is structured into four different areas: Tasks, Resources, Execution and Scheduling capabilities. In this section, the main features of the middleware are exposed.

4.1. Task management

The *GridTask* class introduces the abstraction required to support the definition of tasks. Through the usage of the *GridExecutableFile*, a GridTask may specify distinct executable files for different architectures. When a GridResource is selected, the information supplied by the GridTask class will be used to decide the most appropriate executable archive to exploit the computational capabilities of the remote machine. A GridTask also specifies the application-dependent arguments which will be supplied to the application before starting execution.

As applications typically rely on a set of input data files, and generate output data archives during execution, a GridTask can specify both a *GridInputFileSet* and a *GridOutputFileSet*. These classes provide convenient methods that accept the widely used wildcard mechanism to specify the files. This aspect is of paramount importance for specifying the output data files, given that their names may be unknown prior to execution.

The GridInputFileSet indicates all the GridFiles that the application requires for execution, which will be staged to the remote resource prior to execution. At the moment, the input data files are expressed only via pathnames. If an application uses a data file that already exists on a remote machine, the user will express it via the *setIsRemote* method of the GridFile class.

A GridTask is also associated to a local container (i.e. a folder in the local machine) where all the output data files generated during the remote execution will be hosted.

4.1.1. Requesting capabilities

A GridTask can demand some requirements in order to be executed, defined in the middleware through the usage of the *GridTaskLoR* class. This implies that a GridResource must accomplish all these capabilities to execute the GridTask. The user expresses the application requirements by creating an instance of the GridTaskLoR class which is modified via its interface to enumerate these requirements and then it is assigned to a GridTask. The capabilities that can be specified range from static characteristics, such as the Instruction Set Architecture or the operating system, to dynamic features, such as the minimum number of available processors or the amount of available RAM in the GridResource. For example, a computationally intensive simulation may require hundreds of MBytes of RAM as well as at least four processors in order to be computed in an acceptable time for the user.

4.1.2. Application-dependent checkpointing

A very common functionality implemented in iterative programs is application-level checkpointing [**?**], a technique that allows an application to be suspended and afterwards resumed from the latest checkpoint. This technique is employed to achieve fault tolerance for long-running executions, specially for those where the running time exceeds the mean-time-to-failure of High Performance Computing platforms [**?**]. Indeed, in a decentralised environment like a Grid, any fault-tolerance mechanism is crucial in order to shield the global execution from failures in resources.

Therefore, the GridTask class provides the capability to specify a set of checkpoint files, which are assumed to be periodically generated during the execution of the application. These

files will be periodically transferred from the remote host to the local machine so that, in case of failure, execution can be resumed on another resource from the latest checkpoint. This is an optional capability that may not be supported by the user application and thus, in case of failure, the execution will be restarted from the beginning on another machine.

4.2. Computational resources management

Resources play a fundamental role in a Grid deployment because they provide the execution capabilities that applications require. In this framework, we only focus on Computational Grids [**?**]. However, there are strong research activities being taken on data intensive applications (Data Grids) as well as the analysis of shared large-scale databases [**?**].

The GridResource class enables to abstract a computational resource, offering high level methods in order to access the information supplied by the Monitoring and Discovery Service of Globus (MDS). Therefore, the user no longer interacts with the resources via LDAP (Lightweight Directory Access Protocol) queries to obtain information. Instead, the user utilises methods, such as *getRAMAvailable* or *getNumberOfProcessors*, which return the appropriate information as strongly typed data. The middleware also introduces caching functionality for static information in order to minimise the slowdown introduced by multiple LDAP queries through high latency networks. Of course this high level interface enables to access a subset of all the information exposed by the MDS server of a resource. In fact, only the relevant and most important information required to achieve execution is offered via these high level methods. This enables to keep a simple interface to access the information of a computational resource. Although not very useful, it would be possible to offer full expressivity by providing a one-to-one mapping from each feature from the MDS server to a method of the GridResource class.

The *TestBed* abstraction allows for the programmatical definition of several execution environments involving different computational resources. This allows to easily plug various sets of machines in order to execute a set of computationally intensive tasks (defined by a GridTaskStudy) under different resources without modifying the definition of the study.

A very important functionality implemented in GMarte is the possibility to use separate identities for different resources. This allows the concurrent usage of several Globus-based Grids within the same middleware, employing different credentials for each one. Therefore, being able to span executions through different Grids enlarges the total available computational power and enables to access different computational deployments that would otherwise be unaccessible from the same middleware.

4.3. Execution policies

Regardless of the available computational nodes of a multiprocessor or cluster of PCs, we can specify, for a GridResource, the maximum number of processors involved in a parallel execution. This allows the user to impose a restriction on the execution in the GridResource, to implement, for example, an execution policy that avoids to employ all the computational nodes of the remote resource within a single execution.

We can also decide the maximum number of processes per node involved in a parallel execution. This is an important feature that enables to control the execution on a cluster of PCs composed by multiprocessor machines. On the one hand, we may be interested in using several processes per node with the purpose of using more processors at the tradeoff of reducing the available memory (various processes sharing the same computing node). On the other hand the user can choose to execute only one process per node, thus having available all the RAM of the internal computing node. These configurations may cause dramatic differences in terms of performance and thus, it is important to provide the user with the ability to decide. All these capabilities can be specified using the *ExecutionPolicies* class.

4.4. Interfacing with the LCG middleware

The GMarte middleware has also been designed to support executions on computational resources based on the LCG middleware. This middleware was developed in the framework of the LHC Computing Grid Project (LCG) [**?**], which leverages the development of the required computing infrastructure for the simulation, processing and analysis of the data provided by the Large Hadron Collider, the most powerful particle accelerator which is being constructed at CERN. The participating sites are mainly universities and research laboratories, which contribute more than 10000 CPUs and a total 10 million Gigabytes of storage capacity [**?**]. This represents the largest international scientific Grid, involving 100 sites in 31 countries worldwide.

The LCG middleware is the starting point of EGEE (Enabling Grids for E-sciencE) [**?**], a European Union funded project that aims to develop a service Grid infrastructure which is available to scientists 24 hours a day. Initially, the set of resources available in the EGEE testbed are those from the LCG project.

Currently, GMarte can fully interact with the nodes of an LCG 2.4.0 compliant testbed. In fact, the GridResource abstraction provides the same API regardless of the underlying middleware, being LCG or the standard Globus, enabling the usage of different existing testbeds from the same middleware. This functionality offers potential access to the LCG testbed.

5. METASCHEDULING: ABSTRACTING MULTIPLE TASK EXECUTION

Parameter sweep applications are embarrassingly parallel tasks that typically generate lots of independent parametric executions, which can greatly benefit from the usage of Grid Computing. As a general extension, scientific studies often require the execution of a set of computationally intensive independent tasks. This way, GMarte provides the required software infrastructure to perform automatic fault-tolerant task scheduling in a Grid deployment.

5.1. Phases of a single execution

For the execution of a GridTask in a GridResource, several stages must be accomplished. First of all, the *file stage in* phase must take place, where all the dependent data files, as well as the

Figure 2. State transition diagram of a RunnableGridTask during its normal life-cycle.

most suitable version of the executable file, are transferred, via GridFTP, to the GridResource, creating an appropriate execution environment.

Second, the execution policies are taken into account in order to translate them to a valid RSL (Resource Specification Language) command, that can be understood by the GRAM component of the Globus resource, before the execution takes place with the number of processors specified. GMarte enquires the available job managers in the remote machine to select the most appropriate one, prioritising the use of queue managers (such as PBS or LSF). This ensures that a Grid execution does respect the execution policies of the remote resource. However, the user can alter this default policy to specify the jobmanager to be used, for a given resource.

Finally, once the execution has finished, the *file stage out* phase is performed, where all the output data files generated during execution on the GridResource are transferred, via GridFTP, to the local submission machine, and deleted from the remote resource.

5.2. MetaScheduling capabilities

In order to perform the execution of a GridTaskStudy in a TestBed, a task allocation process must be carried out [**?**]. The *RunnableGridTask* provides the abstraction that allows to perform the phases for a single execution. We have transformed metascheduling into a state transition problem where only the RunnableGridTasks that satisfy a specific state can be promoted to the next one, leading to a cleaner design and implementation. Figure **??** summarises the principal states that traverses a RunnableGridTask during its life-cycle.

The AdvancedScheduler class implements the scheduling policy, selecting the GridResource, from those specified by the user, with more available computing nodes which satisfies the list of requirements of the GridTask. For workstations or serial PCs, an estimation of the CPU usage during the last minute indicates the availability of the resource.

The AdvancedScheduler consists mainly of three threads, which are in charge of performing the promotion of states for the RunnableGridTasks. The *ThreadPreExecution* is responsible of promoting from the UNASSIGNED or FAILED state to the SCHEDULED one by selecting the current best resource and performing the file stage in phase, leaving the RunnableGridTask in the STAGED_IN state. Analogously, the *ThreadExecution* transfers a RunnableGridTask from the STAGED_IN state to the ACTIVE one, by starting the execution on the GridResource. When the execution finishes, the DONE state is automatically reached.

The *ThreadPostExecution* performs the file stage out phase for the RunnableGridTasks in the DONE state, being promoted to the STAGED_OUT state. The same thread performs the garbage erasing process, leading the RunnableGridTask to the end of its life-cycle reaching the COMPLETED state.

In the case of failure of the resource during execution, as well as in any of the data transfer phases, the RunnableGridTask is downgraded to the FAILED state so that it can be executed again, either from the beginning or from the latest checkpoint, on a different GridResource.

Being a multithreaded object, the AdvancedScheduler can concurrently handle an expensive stage in phase for a GridTask, while performing the execution of another one or carrying out the stage out phase, thus reducing the bottleneck of traditional sequential scheduling approaches.

5.3. Customising the scheduling policies

The design of GMarte allows for an easy modification of the strategy employed for the task allocation process. The Scheduler class is abstract, providing no implementation but a contract of the interface that should be implemented by the different schedulers. For example, it is completely feasible to create a subclass of the AdvancedScheduler class in order to override the *chooseGridResource* method to implement a different policy for selecting the computational resource that will execute a determined GridTask.

We could also be interested in overriding the *chooseNumberOfProcessors* method in order to modify the policy employed in selecting the number of processors involved in a parallel execution, knowing the GridTask and the GridResource that the scheduler has selected. This subclass of the AdvancedScheduler class would inherit all the already implemented functionality, which leads to a shorter development time. Therefore, the scheduling policies can easily be altered by using the object-oriented programming feature of class inheritance combined with method override.

These *pluggable* scheduling policies allow us to assess different task allocation alternatives under the same Grid deployment (TestBed abstraction), analysing certain parameters such as task distribution or total simulation time to decide the most suitable scheduling strategy.

5.4. Fault tolerance scheme

The GridTask class supports the definition of checkpointing capabilities to introduce execution recovery after remote resource failure. Therefore, if the user application supports checkpointing, the GridTask can optionally specify the checkpoint files that the application will generate during its execution. The AdvancedScheduler will detect the definition of these files and it will activate the checkpointing capabilities, periodically retrieving the checkpointing files, by the *ThreadCheckpointing* thread, from the remote GridResource to the local machine. In the case of failure of the remote resource, the latest checkpoint files which are available in the local machine will be transferred to the new GridResource so that the execution can be resumed from the latest checkpoint.

A failed GridTask that does not specify checkpointing capabilities will be started from the beginning at a new GridResource. With this simple, yet powerful, fault tolerance strategy, the execution of GridTasks is guaranteed as long as there are active GridResources in the TestBed.

Failures during a data transfer are detected through the exception mechanisms provided by the client-side implementation of GridFTP (*jglobus*) from the Java CoG. Stalled data transfers are detected through the *performance markers* capability of such implementation. Upon failure, the operation is retried several times before notifying it to the upper software layers, thus reallocating the GridTask to another machine.

As opposed to other approaches [**?**], GMarte does not involve the migration of tasks prior to resource failure. The main reason is that typically a performance degradation model must be developed in order to decide if migration should occur, which is very dependent on the user application. As GMarte tries to be as independent of the application as possible this feature has not been incorporated.

6. PRACTICAL EXAMPLES OF USAGE

Even though the GMarte middleware capabilities have been thoroughly explained, the following examples expose its ease of usage.

6.1. Single remote execution

Consider the following scenario. The user application *app3D* must be executed with 3 processors on a cluster of PCs called *machine.domain.com*. The application requires two input data files, *infile1.dat* and *infile2.dat*, and generates a set of output data files with the suffix *.out*. Besides, the application requires at least 128 MBytes of RAM. Also, the task is invoked with the command-line arguments "*-x 10*". Then, assuming that all the security issues, such as valid credentials, are well established in both machines, the following Java code using GMarte, executed on the client machine, will achieve a parallel remote task execution of the app3D application:

```
GridTask gt = new GridTask("Sample GridTask: app3D");
GridExecutableFile gef = new GridExecutableFile("/home/user/app3D");
gt.setGridExecutableFile(gef);
gt.setArguments("-x 10");
GridInputFileSet gifs = new GridInputFileSet();
gifs.addGridFilesBySuffix("/home/user", ".dat");
gt.setGridInputFileSet(gifs);
GridOutputFileSet gofs = new GridOutputFileSet();
gofs.addSuffix(".out");
gt.setGridOutputFileSet(gofs);
GridTaskLoR gtl = new GridTaskLoR();
gtl.addRequirement(AVAILABLE_RAM, 128);
gt.addGridTaskLoR(gtl);
GridResource gr = new GridResource("machine.domain.com");
RunnableGridTask rgt = new RunnableGridTask(gt,gr);
rgt.setNumberOfProcessors(3);
```

```
rgt.stageIn(); rgt.run(); rgt.stageOut();
```

As the example has shown, there is a common usage pattern. An object is created, its properties are changed according to the functionality required and finally it is assigned to another high level object. The dependent input data files have been indicated with the wildcard mechanism for the simplicity. Also, the files are assumed to reside in the */home/user* directory.

For the sake of comparison, the user would have had to perform the following actions using the traditional mechanisms provided by Globus (the parentheses enclose the typical command that would be used to carry out the action):

1. To investigate if the resource is alive (*ping*), if the MDS is running (*grid-info-host-search*), if proper credentials are being used to access the resource and if the GRAM service is currently working (*globusrun*).
2. To ensure that the resource has at least 128 MBytes available, by issuing a customised LDAP query (*grid-info-host-search*) that only returns the value of the attribute *Mds-Memory-Ram-freeMB* by the MDS. Besides, to construct another customised LDAP query that returns the value of the attribute *Mds-Computer-Total-Free-nodeCount*. If there are several execution queues on the resource, to investigate the values for all of them in order to decide if there are at least 3 processors available.
3. To decide where are going to reside all the files on the remote filesystem and to transfer them via GridFTP or GASS by issuing as many *globus-url-copy* commands as files, as no wildcard mechanism is currently supported by Globus. As an additional problem, the GridFTP protocol does not maintain the attributes of the files and thus, the executable file in the remote resource will have the executable flag disabled, thus preventing its execution.
4. To decide if the execution is going to be batch or interactive and to start the execution (*globusrun*) with the number of processors specified. If the execution is batch, then periodically query the status of the job (*globus-job-status*) until the DONE state is reached.
5. To guess where the application will have written its output data files and to transfer them back to the local machine (*globus-url-copy*). To be polite and erase all the generated data files in the resource so that they do not waste disk space in a resource from, possibly, another organization.
6. To perform error detection along all the previous phases and to take appropriate actions to circumvent them or notify the user.

It can be noticed that, using the traditional mechanisms offered by the Globus Toolkit, a user faces a lot of complexity and several design decisions that impose a clear overhead in the process of porting an application to a Grid infrastructure.

The sequence of calls under the GMarte middleware internally involves the usage of several Globus Toolkit services such as GridFTP, for file transfer, MDS, for resource information, and GRAM, for job execution. However, it is clear that the user is not aware of all this burden, but instead is provided with an API that allows to declare *what* to execute and *where* to execute it, instead of dealing with the implementation details about *how* to perform the execution.

6.2. Multiple remote execution

In order to support the execution of a set of tasks, such as those arising in parameter sweep applications, the metascheduling capabilities of GMarte can be employed. This example shows how to use the metascheduling services, provided by the AdvancedScheduler, for the execution of an application-dependent case study in a testbed consisting of two machines.

```
GridTaskStudy gts = new MyCaseStudy();
String hosts = {"machine1","machine2"};
TestBed tb = new TestBed(hosts);
Scheduler sched = new AdvancedScheduler(tb, gts);
SchedulerConfiguration sconf = new SchedulerConfiguration();
sconf.setThreadCheckpointingPeriod(1800);
sched.setSchedulerConfiguration(sconf);
sched.start();
sched.waitUntilFinished();
```

In this example, *MyCaseStudy* is a subclass, declared and implemented by the user, of the GridTaskStudy class, which would contain the definition of all the GridTasks that are going to be executed in the Grid. For each GridTask, the user would specify the dependent input data files, the command-line arguments, the generated output data files, checkpointing capabilities and all other required information, as performed in the first example. For parameter sweep applications, typically one GridTask is defined and cloned into many instances, where only the application arguments are changed. In this particular example, the user application is assumed to support checkpointing and thus, the scheduler has been configured to retrieve all the generated checkpoint files from the remote resources to the local machine every 30 minutes.

Once the user has defined the GridTaskStudy, it can easily be executed under different machines (TestBeds), as well as using different scheduling configurations within the framework provided by GMarte. The plain Globus Toolkit does not expose any metascheduling capability so, in this example, we can not show any comparison between the interaction with Globus and with GMarte.

6.3. Real applications ported to the Grid under GMarte

The GMarte middleware is employed for batch applications, which do not require interaction with the user during their execution. This is the common behaviour of many simulation codes in engineering. Also, the middleware has proven to be very useful for the execution of parameter sweep applications on a Grid. The GMarte framework allows for an easy specification of a GridTask, which then can easily be duplicated to create a set of tasks (a GridTaskStudy). The different GridTasks must be independent, with no communication among them, in order to be executed via GMarte.

GMarte is currently being employed for the Grid support in two different engineering fields. First of all, cardiac simulations are computationally and data intensive tasks that require the execution of a large number of simulations for the study of the influence of drugs, in the

electrical behaviour of the heart, or to analyse certain pathologies [**?**]. With the help of Grid Computing technologies, it is possible to use a Grid infrastructure to concurrently execute all the independent simulations that form a cardiac case study, a parameter sweep application that has been greatly benefited by an enhanced productivity provided by the concurrent execution of simulations on the Grid.

Second, dynamic structural analysis, is one of the most time consuming stages in the design cycle of a building, where a lot of simplifications have been traditionally carried out. Memory and computing power requirements to efficiently manage 3D large structural systems requires the utilization of High Performance Computing techniques. Moreover, a large number of different structural alternatives, under the influence of multiple earthquakes, must be simulated in the design stage, in order to find the one which best accomplishes with all the economic limitations, aesthetical aspects or safety requirements [**?**]. However, architecture or engineering studios do not own multiprocessor resources, so thus, thanks to the use of Grid technologies, a high number of structural alternatives can be analysed in a realistic way.

In both of these fields, simulations may require hours of computation time, generating hundreds of MBytes of binary output data.

In a related work [**?**], we describe the execution of a cardiac case study on a subset of resources from the EGEE testbed across Spain, Italy, France and Taiwan. The usage of a Grid infrastructure reduced the total simulation time of the case study, which was composed of 21 cardiac simulations, from more than one day to less than two hours. This improvement has been fundamental to increase research productivity in such a time-consuming biomedical application.

7. CONCLUSIONS

By the application of object-oriented techniques to the Grid Computing field, it has been possible to design a middleware, developed on top of the Java CoG Kit 1.2, which encapsulates part of the functionality provided by the Globus Toolkit in an abstraction layer that simplifies the process of remote task execution in a Grid infrastructure. GMarte, in contrast to other approaches, exposes a framework based on the industrial standard Globus Toolkit, instead of a self-designed proprietary framework, that enables the middleware to be compatible with the numerous Globus-based Grid deployments. GMarte also provides capabilities to enable the usage of LCG-based resources, thus providing computational access to the EGEE testbed, the largest distributed Grid deployments in the world. In addition, a very natural high level API, from a Java environment, is offered to the user, thus combining versatility as well as ease of usage, as opposed to using a closed software tool.

GMarte's scheduling capabilities allow a transparent integration of High Performance Computing and the Grid in an object-oriented framework, by performing multiple concurrent parallel executions on a distributed deployment, an useful feature for parameter sweep applications.

Most of the implemented functionality can be found on other middleware approaches; however, the expressiveness, the ease of usage, extensibility and versatility, provided by an

object-oriented API, and the fact that the Globus Toolkit is the underlying middleware, provides the Grid community with a simple yet powerful tool.

The future works involve the development of a Grid Portal, developed on top of GMarte, to provide the simulation process with a Graphical User Interface, in the cardiac electrical activity and civil engineering fields. Besides, plans are to investigate new scheduling approaches, within the modular framework offered by GMarte, in order to assess the benefits of different task allocation strategies.

This middleware allows the exploitation of the Globus Toolkit functionality without requiring the knowledge of its provided services. Therefore, reducing the gap from Grid Computing technologies to the end user represents a step forward for the widespread adoption of these computational techniques, which would otherwise be restricted to computer scientists with a knowledge of Grid middleware services.

REFERENCES

1. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *Intl. J. Supercomputer Applications* 1997; **2**:115–128.
2. von Laszewski G, Foster I, Gawor J, Lane P. A Java commodity grid kit. *Concurrency and Computation: Practice and Experience* 2001; **13**(8-9):643–662.
3. Kennedy K, et al. Toward a framework for preparing and executing adaptive grid programs. *Proceedings of NSF Next Generation Systems Program Workshop (International Parallel and Distributed Processing Symposium* 2002.
4. Jhoney A, Kuchhal M, Venkatakrishnan S. Grid application framework for Java (GAF4J). http://dl.alphaworks.ibm.com/technologies/gaf4j/GAF4Jwhitepaper1.2.doc [April 2005].
5. Neary MO, Brydon SP, Kmiec P, Rollins S, Cappelo P. Javelin++: Scalability issues in global computing. *Concurrency: Practice and Experience* 2001; **12**:727–753.
6. Caromel D, Klauser J, Vayssiere J. Towards seamless computing and metacomputing in Java. *Concurrency Practice and Experience* 1998; **10**(11-13):1043–1061.
7. Grimshaw AS, Wulf WA. The Legion vision of a worldwide virtual computer. *Communications of the ACM* 1997; **40**(1):39–45.
8. Thain D, Wright D, Miller K, Livny M. Condor - a distributed job scheduler. In *Beowulf Cluster Computing with Linux,* Sterling T. (ed.) MIT Press, 2001.
9. Frey J, Tannenbaum T, Foster I, Livny M, Tuecke S. Condor-G: A computation management agent for multi-institutional Grids. *Journal of Cluster Computing* 2002; **5**:237–246.
10. Thomas M, Dahan M, Mueller K, Mock S, Mills C, Regno R. Application portals: practice and experience. *Concurrency and Computation: Practice and Experience* 2002; **14**:1427–1444.
11. Schopf JM, Nitzberg B. Grids: The top ten questions. *Scientific Programming*, special issue on Grid Computing 2000; **10**(2):103–111.
12. Lawson CL, Hanson RJ, Kincaid D, Krogh FT. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Soft.* 1979; **5**:308–323.
13. Anderson E, Bai Z, Bischof C et al. LAPACK user's guide. Third Edition. *Society for Industrial and Applied Mathematics. Philadelphia, PA* 1999.
14. Alonso JM, Hernández V, Moltó G. Grid computing based simulations of the electrical activity of the heart. *Lecture Notes in Computer Science. Springer-Verlag* 2004; **3036**(1):482–485.
15. Gropp W, Lusk E, Doss N, Skjellum A. A high performance, portable implementation of the message passing interface standard. *Parallel Computing* 1996; **22**(6):789–828.
16. Gropp W, Lusk E. User's guide for MPICH, a portable implementation of MPI. *Mathematics and Computer Science Division, Argonne National Laboratory* 1996.
17. Vadhiyar S, Dongarra J. A metascheduler for the Grid. *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid* 2003; **1**:130–137.
18. Bronevetsky G, Marques D, Pingali K, Stodhgill P. Automated application-level checkpointing of MPI programs. *Principles and Practice of Parallel Programming* 2003; **1**:84–94.

19. Foster I, Kesselman C. The GRID: Blueprint for a new computing infrastructure. *Morgan Kauffman* 1999.
20. Hoschek W, Jaen-Martinez J, Samar A, Stockinger H, Stockinger K. Data management in an international Data Grid project. *Proceedings of the First IEEE/ACM International Workshop on Grid Computing. Lecture Notes in Computer Science* 2001; **1971**:77–90.
21. LHC Computing Grid Project Web Page. http://lcg.web.cern.ch [April 2005].
22. CERN. World's Largest Computing Grid Surpasses 100 Sites. *ERCIM News* **61**: 63–63.
23. EGEE Web Page. http://www.eu-egee.org [April 2005].
24. Schopf JM. Ten actions when superscheduling. *Scheduling Working Group* 2001; **8.5**.
25. Montero RS, Huedo E, Llorente IM. Grid resource selection for oportunistic job migration. *Lecture Notes in Computer Science. Springer-Verlag* 2004; **2790**(1):366–373.
26. Alonso JM, Ferrero JM, Hernández V, Moltó G, Monserrat M, Saiz J. Computer simulation of action potential propagation on cardiac tissues: An efficient and scalable parallel approach. *Parallel Computing: Software Technology, Algorithms, Architectures and Applications, included in series: Advances in Parallel Computing* 2004; **13**(1):339–346.
27. Alonso JM, Alfonso C, García G, Hernández V. Parallel and Grid computing in 3D analysis of large dimension structural systems. *Lecture Notes in Computer Science. Springer-Verlag* 2004; **3149**(1):487–496.
28. Alonso JM, Hernández V, Moltó G. Experiences on a large scale Grid deployment with a computationally intensive biomedical application. *Proceedings of the 18th IEEE International Symposium on Computer-Based Medical Systems*, special track *Grids for Biomedicine and Bioinformatics* 2005; **1**:567–569.