

Using the eNANOS Low-Level Support in the GRMS Framework

I. Rodero, F. Guim, J. Corbalan

`{irodero, francesc.guim, julita.corbalan}@bsc.es`

*Barcelona Supercomputing Center
Jordi Girona 29, 08034 Barcelona, Spain*

A. Oleksiak, K. Kurowski, J. Nabrzyski
`{ariel, kikas, naber}@man.poznan.pl`

*Poznan Supercomputing and Networking Center
Noskowskiego 10 61-704, Poznan, Poland*



CoreGRID Technical Report
Number TR-0079
March 16, 2007

Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence
URL: <http://www.coregrid.net>

Using the eNANOS Low-Level Support in the GRMS Framework

I. Rodero, F. Guim, J. Corbalan

{irodero, francesc.guim, julita.corbalan}@bsc.es

Barcelona Supercomputing Center
Jordi Girona 29, 08034 Barcelona, Spain

A. Oleksiak, K. Kurowski, J. Nabrzyski
{ariel, kikas, naber}@man.poznan.pl

Poznan Supercomputing and Networking Center
Noskowskiego 10 61-704, Poznan, Poland

CoreGRID TR-0079

March 16, 2007

Abstract

The eNANOS is an execution framework developed in the Barcelona Supercomputing Center. One of its main objectives is to provide a framework to execute multilevel parallel applications with low-level support. It is also able to provide information about the execution behavior of applications in run time. This information can be used by a Grid Resource Broker or metascheduler to improve its scheduling and resource strategies and the execution platform can improve the execution time of applications and resource usage as well. In this paper we discuss the steps that we have to follow to integrate the eNANOS execution environment into the GRMS infrastructure, developed by PSNC. In particular we are interested in the mechanisms to allow the integration of the different components and how to use the information provided by eNANOS to improve the scheduling strategies in the GRMS system.

1 Introduction

The eNANOS is an execution framework developed in the Barcelona Supercomputing Center. One of its main objectives is to provide a framework to execute multilevel parallel applications with low-level support. Furthermore the eNANOS architecture is based on the idea of coordination between the different layers [9]. Currently the eNANOS Execution platform uses the eNANOS Grid Resource Broker [8] which manages the jobs from the Grid layers in coordination with the local resource environments.

The eNANOS System is also able to provide information about the execution behavior of applications in run time, such as its progress or the obtained performance in a given moment. This information can be used by a Grid Resource Broker or metascheduler to improve its scheduling and resource strategies and the execution platform can improve the execution time of applications and resource usage as well.

The main effort of the PSNC in the Grid resource management is the GRMS resource broker [2]. GRMS is an open source meta-scheduling system for large scale distributed computing infrastructures. Based on the dynamic resource selection, mapping and advanced grid scheduling methodologies, it has been tailored to deal with resource management challenges in Grid environments, e.g. load-balancing among clusters, setting up execution environments before and after job execution, remote job submission and control, files staging, workflow management and more.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

In this paper we discuss the steps that we have to follow to integrate the eNANOS execution environment into the GRMS infrastructure. In particular we are interested in the mechanisms to allow the integration of the different components and how to use the information provided by eNANOS to improve the scheduling strategies in the GRMS system.

In section 2 we present the eNANOS approach and its main characteristics and in section 3 we introduce the GRMS system. In section 4 we study the possibilities of integration between the eNANOS and GRMS systems, and, finally, in section 5 we present the conclusions and future lines of work.

2 eNANOS Execution Environment

The eNANOS project aims at developing an execution platform for parallel applications on Grids. The objective of this platform is to provide support to implement and evaluate resource management and scheduling policies on a computational Grid. The eNANOS project is based on the idea of having a good low level support for performing a good high level scheduling taking into account these ideas: a fine grain control between scheduling levels, dynamic allocation (MPI+OpenMP jobs) to improve system performance, detailed information about current scheduling and performance to improve future scheduling decisions, and, based on this accurate information, efficient scheduling (in terms of slowdown) based on performance prediction.

The scheduling strategies are based in the coordination between the different layers involved in the execution of a job: Grid scheduling, cluster scheduling and processor scheduling. The scheduling decisions are known by the other elements in the system and are taken based on direct information, not based neither on estimations nor just observations.

The idea is providing well defined API between levels to both forcing scheduling decisions (for instance specific allocations) and getting detailed information (for instance the real performance reached by a job in run time). A general overview of the system architecture is shown in Figure 1.

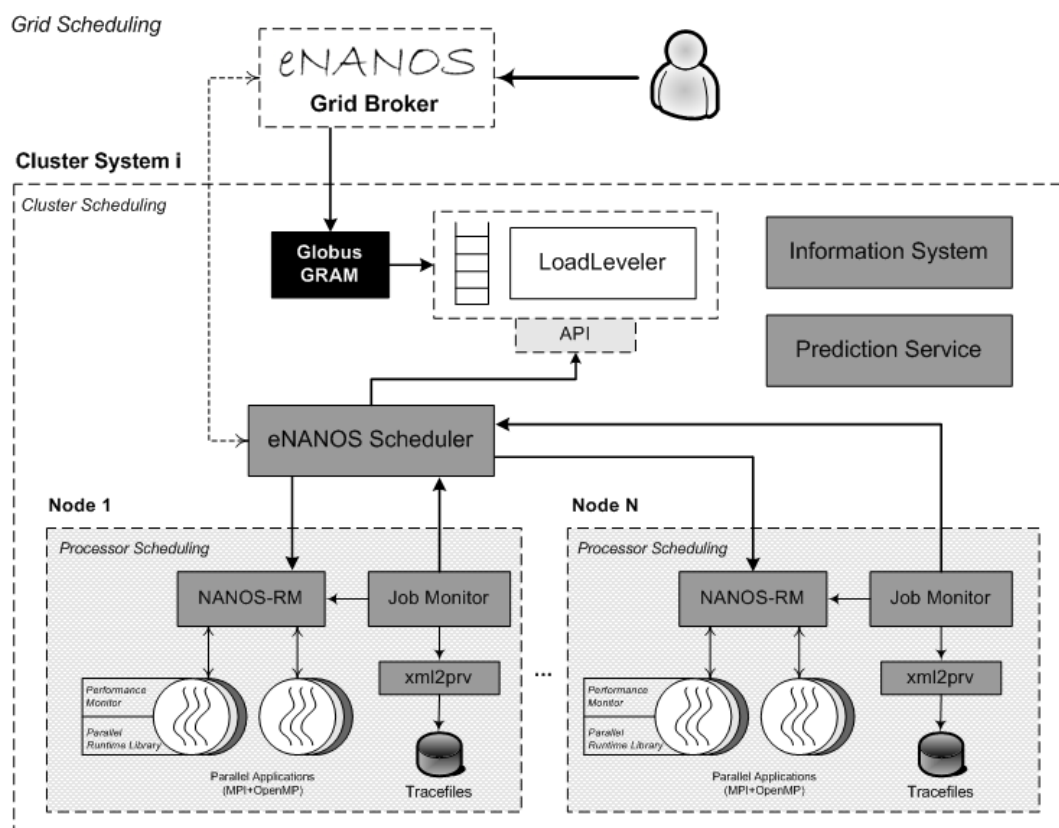


Figure 1: eNANOS Overall Architecture

The Grid jobs are managed by the eNANOS Resource Broker and submitted to the local HPC resources through the Globus infrastructure. The eNANOS Resource Broker is developed on top of Globus Toolkit 3 as a Grid service and it is compatible with both GT2 and GT3 services. It implements the resource discovery, selection and monitoring, the job submission, and the job monitoring. Moreover, the eNANOS Broker provides a set of Grid Service interfaces and a Java API that can be used from command-line clients, applications or portals. The eNANOS broker has been extended to provide more functionality and supporting JSDL 1.0 with a gateway implemented as a GT3 Grid service as well. More detailed information about this broker can be found in [8].

For the cluster management eNANOS uses LoadLeveler as a queuing system and the eNANOS Scheduler as an external scheduler to manage the local jobs. It implements scheduling policies based on FCFS and backfilling (in progress) guided by the information regarding the behavior of applications and performance obtained in runtime from the processor scheduler. So, the scheduler centralizes the information from the runtime about the allowed multiprogramming level of multiple computational nodes and also some information regarding the applications and hardware (e.g. load of nodes or CPUs used by each job). It also communicates with the eNANOS Broker in order to provide information to the upper level.

The CPU scheduling is performed by the NANOS-RM. Its main goal is to efficiently distribute a set of processors between a set of applications that are under its control. It implements scheduling policies based on dynamic allocation in a two-phase fashion (multilevel). The first phase is between applications and implements a FIFO policy, and the second phase is between processes of a MPI+OpenMP application and implements Equipartition [7] and Dynamic Processor Balancing [3]. The idea is sharing the required information for improving the whole system performance without penalizing the applications performance independently.

The NANOS Job Monitor provides the monitoring information about the execution of workloads in XML that can be translated to trace files format to visualize and analyze them [1].

The information system (Palantir) can be seen as a meta-information system that can collect a very large kind of different information, providing a uniform access to it. The Predictor service provides predictions about the job performance that can be used by both the eNANOS Scheduler and the eNANOS Broker. It implements two kinds of prediction techniques: those that are based on statistical approaches that use estimators; and those that are based on data mining algorithms. Both information and predictor systems are not integrated yet but their services are available.

3 Grid Resource Management System (GRMS)

GRMS is an open source meta-scheduling system for large scale distributed computing infrastructures. Based on the dynamic resource selection, mapping and advanced grid scheduling methodologies, it has been tailored to deal with resource management challenges in Grid environments, e.g. load-balancing among clusters, setting up execution environments before and after job execution, remote job submission and control, files staging, and more. For our tests we have used version 2.x of GRMS, which is based on the GT4 and makes use of low-level Globus Services deployed on resources located in various academic institutions in Europe and USA. GRMS connects to the core services through a set of Java and C APIs. In particular, GRMS uses GRAM, GridFTP and GRIS/GIIS services. As a persistent service, GRMS provides a set of well-defined GSI-enabled Web Service interfaces for various clients, e.g. applications, command-line clients or portals. Moreover, GRMS is able to take advantage of middleware services, e.g. the GridLab Authorization Service or Replica Management Services as well as to interoperate with the infrastructure monitoring tools such as the Mercury Monitoring System. Therefore GRMS is in fact one of the main components of a grid middleware layer that can be organized in many different ways depending on particular infrastructure and applications. The architecture of GRMS together with a set of its internal modules, namely Job Queue, Job Registry, Job Manager, Resource Discovery and a central unit called Broker Module is presented in Figure 2. The aim of the Broker Module is to control the whole process of resource and job management. The broker was designed in such a way that it allows us to implement various scheduling and policy plug-ins. One of plug-ins, called Reschedule plug-in, is responsible for jobs migration and rescheduling within GRMS. Worth mentioning is also the Resource Discovery Module, that monitors a status of distributed resources. It uses a flexible hierarchical access to both central (GIIS) and local information services (GRIS).

GRMS uses the multicriteria decision support methods for scheduling. Based on user preferences it evaluates and selects convenient resources. The multicriteria techniques used in Grid resource management were described in details in [4]. Examples of use of this model for specific Grid resource management problems were presented in [5][6].

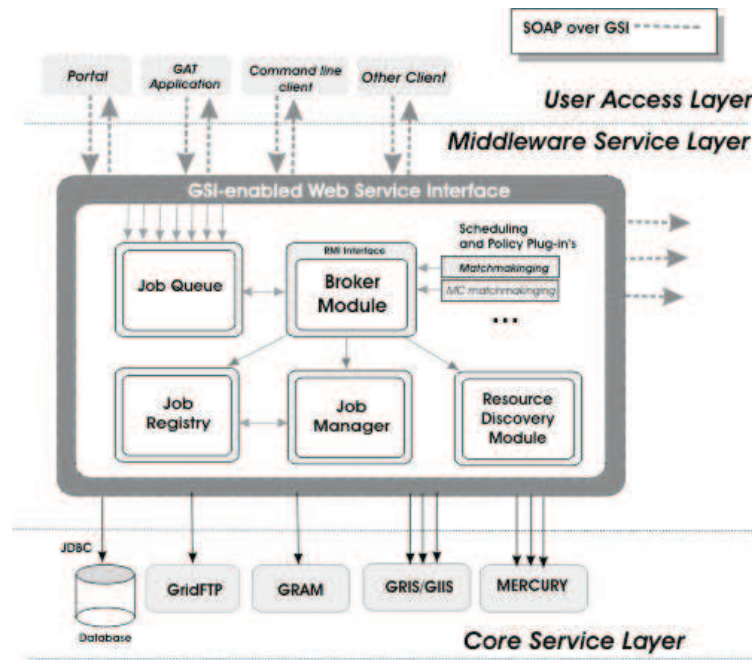


Figure 2: General GRMS architecture

4 Integration Issues

To start with the integration of the eNANOS execution environment with the GRMS system we need to identify the common features and what are the functionalities provided by eNANOS that can be useful for the GRMS.

Currently, in the complete infrastructure of the eNANOS Execution Framework the Grid resource management is performed by the eNANOS Resource Broker. The idea was to implement a customized environment to execute efficiently multilevel parallel applications, but one of the desirable features for one of this kind of systems is to provide generic interfaces. Therefore, the local execution system should be able to be integrated to other Grid Resource Brokers or metaschedulers such as GRMS.

Since the eNANOS system supports both traditional jobs and multilevel parallel jobs, the resources supporting eNANOS can be seen as a normal resource or as a specific case for those jobs that require support from the local environment. The idea of integrating this execution environment into a Grid is to improve the behavior of the Grid in general, taking advantage of the complementary functionalities provided by eNANOS. The improvement is though in terms of improving the execution time of applications and the resource usage.

On the other hand, one of the important capabilities of the eNANOS System is the coordination between the different layers involved in a job execution and, in particular, the detailed information that is able to provide to the Grid level regarding the behavior of the applications in run time. The eNANOS offers some libraries and tools to obtain detailed information in run time about the progress of the applications and the performance as well.

As a Grid Resource Broker, GRMS performs the scheduling of jobs and the resource management following some particular strategies. Hence, the GRMS system can consider the information offered by eNANOS regarding progress, performance and fine-grain monitoring to improve the scheduling strategies implemented into the scheduling engine of the GRMS.

Therefore, we have identified two different possible integration issues: using eNANOS as a new execution platform, and using eNANOS as a new information provider.

4.1 eNANOS as a new execution framework

Even though GRMS has been designed as an independent set of components for resource management processes, the core services used by GRMS are included into the Globus Infrastructure. As the eNANOS system is based on Globus, the integration of both components should be possible without changing the interface to the local resources.

In the eNANOS Execution Framework Globus is used as a middleware and a customized LoadLeveler jobmanager in local resources. Since the GRMS expects to find a Globus-based resource with a particular jobmanager for the queuing or fork system, the resources with eNANOS system installed should be presented as a new resource for the scheduling process. But we have to take into account certain particularities of the eNANOS-enabled systems, performing a special treatment for this kind of resources.

Since the main approach of the eNANOS system is the low level support for multilevel applications, the GRMS system can perform a special use of the eNANOS resources. In particular the most important consideration regarding this issue is the specification of parallel jobs.

The eNANOS system supports JSDL 1.0 for the specification of jobs. Moreover, in the eNANOS context is being implemented an extension of the JSDL 1.0 for multilevel parallel jobs [10]. In Figure 3 it is shown the XML schema of this extension for the JSDL.

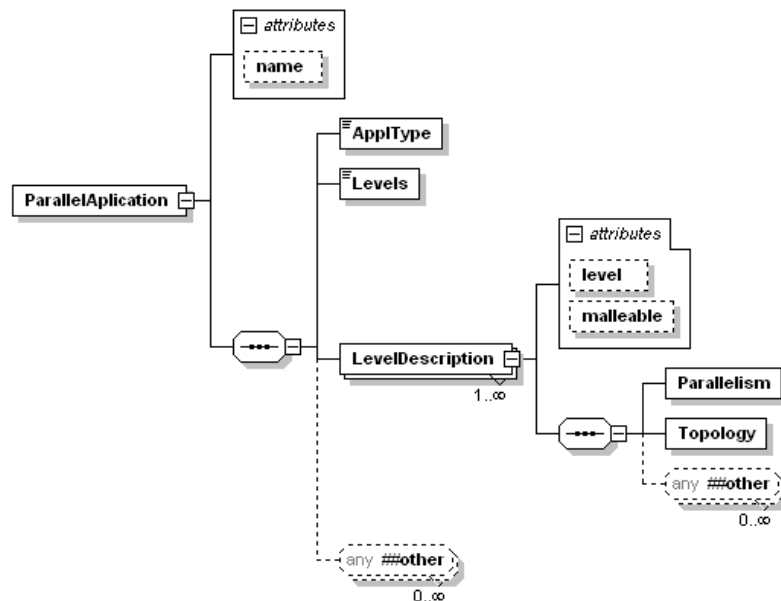


Figure 3: JSDL 1.0 extension for parallel jobs schema

The most important attributes for the local execution environment are:

- Application type. It is an enumeration type specifying the kind of application regarding its programming model (MPI, OpenMP, etc.).
- Number of MPI processes. It is just a positive integer value.
- Number of OpenMP threads. It is just a positive integer value.
- Malleability. It is a boolean that indicates if the application is malleable or not.
- Topology. It is an enumeration type specifying the topology of the application. This value can be taken into account by the LRMS to decide the number of processes or threads that will be spawned for the application (specific, power of 2, etc.).

GRMS uses its own job description language (GJD). It is a XML-based language, which allows specifying a description of the job executable and also the job resource requirements. There are available several parameters in the GRMS Job Description, including the location of files, arguments, file arguments, executable, environment variables, standard input/output/error, checkpointing definition, name of hosts for the job execution, operating system, required LRMS, network parameters, system paths, minimum memory required and so on. Regarding the specification of the parallelism details of the applications, the GJD allows specifying the type of the application with the `<executablei` tag and the number of required processors with the `<cpucounti` tag. It only allows the threads and MPI programming models for parallel applications. The `<executablei` tag contains 'count' and 'type' attributes that denotes the number

of executions of the executable and the way the job-manager submits a job respectively. For the 'type' attribute the following values are available: single (only 1 process or thread will be started), multiple (start count processes or threads), mpi (use the appropriate method to start the job compiled with a vendor-provided MPI library, the job is started with count nodes).

The description of a Grid job with the GJD is shown bellow. It is a MPI job with 4 processes, and it gets the configuration for submitting a NAS-BT from an input file.

```
1 <grmsjob appid="appid">
2   <simplejob>
3     <executable type="mpi" count="4">
4       <file name="exec-file" type="in">
5         <url>file:///home/bench/nas-mz/exec-bt</url>
6       </file>
7       <arguments>
8         <value>file.log</value>
9         <file name="file.conf" type="in">
10          <url>gsiftp://pcmas.ac.upc.edu/~ex/file.conf</url>
11        </file>
12      </arguments>
13    </executable>
14  </simplejob>
15 </grmsjob>
```

There are some attributes required by eNANOS not supported by the GRMS job description language. Thus, it is required to extend the specification of details about parallelism for applications in GRMS.

There are basically two different ways:

- Extending the GJD language.
- Using a mechanism to specify more details with the current semantic.

We have decided not extending the language used to describe jobs in the GRMS system. We have used some simple mechanism such as environment variables. We have chosen existing variable names from OpenMP for a good understanding and some other for other semantics. The complete list of considered environment variables is the following:

- OMP_NUM_THREADS. Indicates the number of OpenMP threads.
- OMP_SCHEDULE. Indicates the scheduling policy to follow by the OpenMP runtime.
- PAR_MALLEABLE. Indicates if the parallel application is malleable or not.
- PAR_TOPOLOGY. Indicates the kind of topology followed by the application.

These environment variables can be included externally to the GRMS system just by the job description file.

Another important issue to take into account regarding the definition of jobs is the identification of the jobs. Since this is a coordinated infrastructure and there are several layers involved in the architecture, the mapping of IDs into the different layers and the description of jobs is crucial.

Figure 4 shows the call procedure between the main eNANOS execution framework entities.

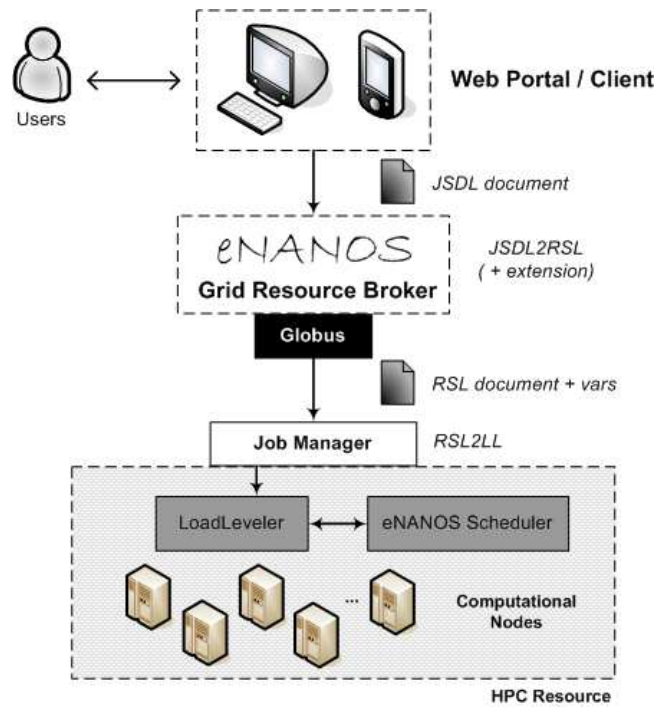


Figure 4: Data flow between the main components of eNANOS

In our current execution environment, the Grid Resource Broker (eNANOS Broker) receives a JSDL document from the Web Portal or from a client interface. Afterwards, the JSDL document is converted to RSL because the Grid broker is built on top of the Globus infrastructure. Not all the information expressed in a JSDL document can be covered in an RSL document, then we use some environment variables as a simple mechanism to solve this problem. Finally, in the local resource the appropriate job manager transforms the RSL document to a LoadLeveler script.

An example of an obtained LoadLeveler script is shown below.

```

1  #!/bin/sh
2  # Job command file created by GRAM/JobManager/loadleveler.pm
3  # @ job_type      = parallel
4  # @ initialdir    = /scratch/irodero/cpmd
5  # @ input         = /dev/null
6  # @ output        = cpmd.4.pwr4.out
7  # @ error         = cpmd.4.pwr4.err
8  # @ class         = short
9  # @ restart       = yes
10 # @ total_tasks   = 2
11 # @ node          = 1
12 # @ environment = COPY_ALL;\
13 #   MP_EULIB=ip;\
14 #   MP_EUIDEVICE=en0;\
15 #   PP_LIBRARY_PATH=/scratch_tmp/irodero/CPMD-3.9.1/PP_LIB;\
16 #   OMP_NUM_THREADS=16;\
17 #   PAR_TOPOLOGY=power2
18 #@ queue
19 #
20 /scratch/irodero/CPMD-3.9.1/cpmd.x /scratch/irodero/CPMD-3.9.1/inputs/small.inp
21 #
22 # End of job command file.
  
```

Since from both the Grid and the local environment it is required to know the mapping of a given job into the other layers (for example to ask for information to a local system), we need to include into the GRMS system another environment variable, but in this case it has to be assigned inside the GRMS system not by the user (because the Job ID is unknown before its submission).

- GRID_ID_ENV. Identification of the Grid job (given by the GRMS system).

4.2 eNANOS as a new information provider

In order to provide more details of job execution in the local environment (where eNANOS gives support for parallel applications) we have found two possible ways of integration as is shown in Figure 5:

- Provide an API for eNANOS functionalities that can be used directly from the GRMS system.
- Using an Information system (such as Mercury) and adding a new plugin to eNANOS collect the information provided by eNANOS (information queried with general mechanisms).

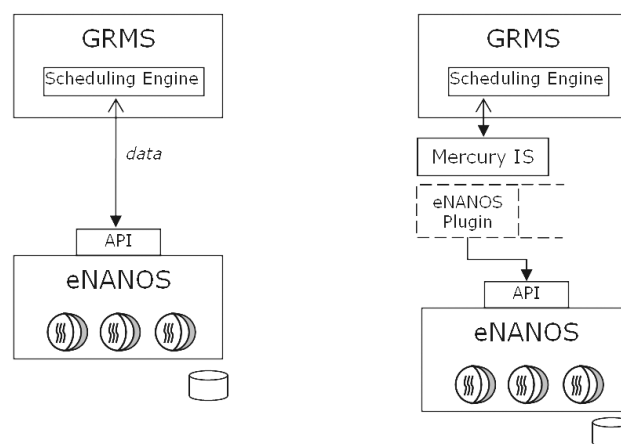


Figure 5: Two possibilities of integration: by API (left) and by an IS (right)

The second solution should be better in terms of extensibility and generic, but for the first steps of integration is more complicated since there is no plugin implemented yet for eNANOS and the GRMS system has to be modified to support the new data in the scheduling engine.

Therefore, in this first step we have implemented an API for the eNANOS Execution Framework that allows the GRMS system to obtain information related to the application behavior in run time. As it is discussed in the previous section, the identification of jobs is very important to map the Grid jobs into the local execution platform when getting data. The API has been implemented in Java and consists of the following basic functionalities:

- **geteNANOSProgressInfo** (JOB_ID, RELATIVE/ABSOLUTE)
- **geteNANOSPerformanceInfo** (JOB_ID, RELATIVE/ABSOLUTE)
- **geteNANOSLocalInfo** (JOB_ID)

The schema of the information returned by the API methods includes several data from the local execution systems. Actually, when a job is in the RUNNING state, the XML includes a set of applications with their respective processes and threads. For each of these processes and threads there is also included information about performance metrics (such as MIPS or MFLOPS). This information is very important to know the behavior of the job since our project is targeted to parallel HPC applications, MPI and MPI+OpenMP as well.

This fine-grain information is provided by the NANOS Resource Manager (NANOS-RM) through its PS interface [11]. The NANOS-RM is the responsible of the CPU scheduling in the local system, so it is able to know the details of both the parallel applications and the CPU resources in real time. It provides for each application the number

of processes (e.g. the number of MPI processes), the number of threads per process (e.g. the number of OpenMP threads), the status of these threads and the CPU where the threads are allocated in case of running threads.

The eNANOS execution environment also provides information about the progress and performance of the applications in run time. This information can be queried as an absolute value (for example 1500 MFLOPS) or as a relative one (value from 0 to 100 percent). Currently there are two different ways to generate this information: using a library to instrument the application or just using the library for performance developed on top of the progress indicators infrastructure. In the local system, the progress information is managed by a dedicated daemon which provides an API to access to the indicators information. Furthermore, the NANOS Scheduler also queries to the daemon API to collect this kind of information.

The XML schema of the data provided by the daemon through the client interfaces is shown in Figure 6.

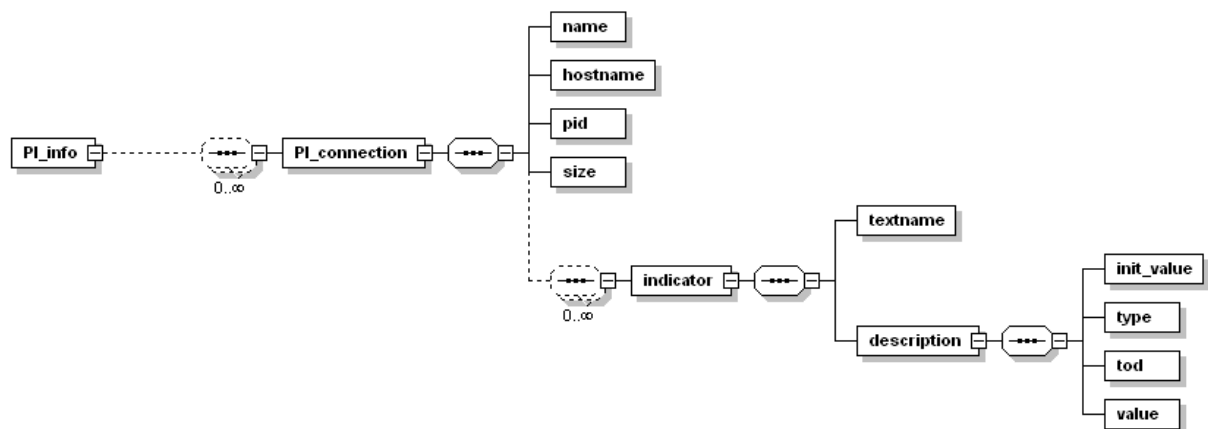


Figure 6: XML schema of the progress indicators data

GRMS can use various scheduling strategies depending on its configuration and available environment. In most cases it uses multicriteria models for Grid scheduling (e.g.[4][6]). Some of these models allow modifications of schedules in runtime based on dynamic information gathered from an environment. These methods can particularly take advantage of functionality provided by eNANOS since it allows getting information about performance of applications in runtime.

In particular, in [5] the dynamic rescheduling procedures implemented in GRMS are presented. They use checkpointing and migration mechanisms supported by GRMS. The scheduling model is also based on multicriteria methods. The complete description of the algorithm is available in the quoted paper. In a nutshell, GRMS reschedules automatically 'smaller' jobs if a 'large' job waiting in a GRMS queue cannot be executed due to lack of sufficient amount of resources (e.g. CPUs). By 'small' and 'large' jobs we mean here jobs that require small and large amount of resources respectively. A decision concerning which job should be migrated to which host is done based on multiple criteria.

We used in our experiment the following criteria for the evaluation of destination hosts: available memory, mean load during the last 1, 5 and 15 minutes, CPU count, CPU speed. The following set of criteria was applied to evaluate checkpointing and migration costs: the number of hosts a job can migrate to (to minimize the risk of a failure), the size of a migrating job (memory allocated by this job), the job's current runtime (in order to migrate jobs that are not to finish soon).

Of course, the current runtime is not the best criterion for evaluation of migration costs of a given job. A job progress is much better metric for this. Taking advantage of this eNANOS functionality GRMS can better evaluate candidate jobs for migration and in this way improve efficiency of the whole schedule (for instance mean job completion time).

Another possibility of exploitation of eNANOS functionality for efficient dynamic rescheduling is use of information about job performance. If this performance is below a certain level (and progress is not advanced too much) then a better resource is found and a job is migrated to that resource (of course if checkpointing of this job is possible).

Ideally the information about a level of performance (or even better relative performance) below which a job should be rescheduled could be provided by end-users. They would have to specify it in a certain kind of agreement. Since currently such information cannot be passed in a GRMS job description we will consider these issues in the future.

5 Conclusions and Future Work

In this paper we have presented the eNANOS Execution Framework as a target infrastructure to be integrated with Grid services such as a Resource Broker or metascheduler as in the case of GRMS. We have studied how to integrate these two components and which of the required changes are realistic. We also have studied how to describe accurately the parallel jobs in the Grid layer and how it influences the whole architecture (specially the lower layers). We have solved this problem with a very simple solution based on the use of environment variables. We have implemented an API to allow the GRMS system to obtain specific information from eNANOS. This API has been implemented in Java and provides the functionality to get information about the progress, performance and accurate description of them in a run time. Finally, we have discussed the possibilities for improving the scheduling strategies of the GRMS system using the functionality provided by the eNANOS system.

As future work we should include the implementation of new plugin for the Mercury information system to allow GRMS to get the information provided by eNANOS with more generic mechanisms. However, some changes in GRMS are needed. We also expect to implement new scheduling strategies and policies into the GRMS system and, finally, evaluate some real workloads on a testbed composed by resources of both PSNC and BSC institutions.

References

- [1] CEPBA Tools Web Site. <http://www.cepba.upc.edu/tools.i.htm>
- [2] Grid Resource Management System (GRMS), <http://www.gridlab.org/grms>
- [3] Julita Corbalan, Alejandro Duran, Jesus Labarta. Dynamic Load Balancing of MPI+OpenMP applications. ICPP04, Montreal, Quebec, Canada. 2004.
- [4] Kurowski K., Nabrzyski J., Oleksiak, A., Weglarz, J., 2003, Multicriteria Aspects of Grid Resource Management, In Grid Resource Management edited by J. Nabrzyski, J. Schopf, and J. Weglarz, Kluwer Academic Publishers, Boston/Dordrecht/London.
- [5] Kurowski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A., Pukacki, J. Improving Grid Level Throughput Using Job Migration and Rescheduling Techniques in GRMS. *Scientific Programming. IOS Press*. Amsterdam The Netherlands 12:4 (2004) 263-273
- [6] Kurowski, K., Oleksiak, A., Nabrzyski, J., Kwiecień, A., Wojtkiewicz, M., Dyczkowski, M., Guim, F., Corbalan, J., Labarta, J., 2005, Multi-criteria Grid Resource Management using Performance Prediction Techniques, In Proceeding of the CoreGrid Integration Workshop, Pisa.
- [7] Cathy McCan, Raj Vaswani, John Zahorjan. A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors. *ACM Transactions on Computer Systems*. 1993.
- [8] Ivan Rodero, Julita Corbalan, Rosa M. Badia and Jesus Labarta. eNANOS Grid Resource Broker. P.M.A. Sloot et al.(Eds.): EGC 2005, LNCS 3470, Amsterdam. February 2005.
- [9] Ivan Rodero, Francesc Guim, Julita Corbalan and Jesus Labarta. eNANOS: Coordinated Scheduling in Grid Environments. *Parallel Computing (ParCo) 2005*, Malaga, Spain, September 2005.
- [10] Ivan Rodero, Francesc Guim, Julita Corbalan and Jesus Labarta. How the JSDL can Exploit the Parallelism?. 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid2006), Singapore, 16-19 May 2006.
- [11] Francesc Guim, Ivan Rodero, Julita Corbalan, Jesus Labarta, Ariel Oleksiak, Jarek Nabrzyski. Uniform job monitoring using the hpc-europa single point of access International Workshop on Grid Testbeds, in conjunction with CCGrid2006, Singapore, 16-19 May 2006.