John von Neumann Institute for Computing

eNANOS: Coordinated Scheduling in Grid Environments

I. Rodero, F. Guim, J. Corbalán, J. Labarta

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 81-88, 2006.

http://www.fz-juelich.de/nic-series/volume33

eNANOS: Coordinated Scheduling in Grid Environments*

I. Rodero[a], F. Guim[a], J. Corbalán[b], J. Labarta[b]

[a]Barcelona Supercomputing Center, Jordi Girona 31, 08034 Barcelona, Spain

[b]Universitat Politècnica de Catalunya, Jordi Girona 1-3, 08034 Barcelona, Spain

One of the current challenges in Grid computing is the efficient scheduling of HPC applications. In the eNANOS project we propose a 3-layer coordinated scheduling architecture for the execution of HPC applications, from the Grid level to the processor scheduling level. In this paper we propose an architecture that allows that the resource broker schedules in coordination with the local resource manager and its local processor scheduler. We are interested in enabling the broker to obtain information about the dynamic behaviour of applications that are running on the local computational resources. Since the Grid technology area is very wide, this project is targeted to High Performance Applications (OpenMP, MPI and MPI+OpenMP) executed on a Grid composed of parallel machines, shared-memory architectures (SMP and CC-NUMA) with a medium to high number of processors. We also show some preliminary results obtained from real workloads which demonstrate the advantages of our coordinated execution environment.

1. Introduction

Grid Computing has emerged in recent years providing a way to perform parallel computing in distributed computational resources. Furthermore, the Grid allows executing jobs in different administrative domains and sharing their existent HPC (High Performance Computing) resources. In order to perform job scheduling and resource management at Grid level, usually it is used a meta-scheduler or a Resource Broker. Typically, the Resource Broker is on top of the Grid infrastructure and it tries to respect the local computational resources policies of each administrative domain and its autonomy. Therefore, the HPC centres usually have local management systems. So, in the execution of Grid applications the meta-scheduler or Resource Broker loses the control of them. We want to take the scheduling decisions being aware of the information reached on lower levels. We think this is the way to achieve an effective scheduling using efficiently the Grid resources. Despite the attempts to define a Grid scheduling architecture [20], that topic is not still fixed and there is no standard available. Even so, it seems reasonable to think that these scheduling layers should interact to each other.

We propose a coordinated scheduling of every component involved in the execution of HPC Grid applications, from the Grid level to the processor scheduling level. We not only want to coordinate the Grid scheduler with the local resource management systems, we also want to perform the scheduling being fully aware of every level which is involved. In particular we are interested in enabling the broker to obtain information about the dynamic behavior of applications that are running on the local computational resources. We consider three basic scheduling levels, from the heterogeneous and dynamic of a Grid to the efficient execution of processes and threads in one or more CPUs of a computer or cluster. This work is part of the eNANOS project [3] whose main aim is the autonomic resource management, one of the required tasks of autonomic computing [11]. Since the Grid

*This research has been supported by the Spanish Ministry of Science and Technology under contract TIN2004-07739-C02-01, and the European Union project HPC-Europa under contract 506079.

technology area is very wide, this project is targeted to High Performance Applications (OpenMP, MPI and MPI+OpenMP) executed on a Grid composed of parallel machines, shared-memory architectures (SMP and CC-NUMA) with a medium to high number of processors.

To carry out an efficient scheduling and for achieve a coordinated scheduling it is also very important the monitoring of both resources and applications. In the eNANOS project the idea is monitoring the jobs in every scheduling level but especially the at the two lowest levels where we want to provide more specific information about the applications behaviour, for instance the reached speedup. This kind of dynamic information can be useful to take decisions about scheduling, as can be dispatching more applications or migrating certain processes or threads. This information can also be very helpful for the researcher because it allows us to see the workload behaviour and precise information in order to improve and adjust the scheduling policies. For this reason we have implemented the NANOS Job Monitor as is shown in section 3.5.

Currently we have implemented the basic mechanisms that coordinate the different layers of the infrastructure and we are working in adding advanced functionalities. Although the complete execution environment is more complex, this paper is centered in the scheduling issues.

2. Related Work

To the best of our knowledge, any developed systems have yet been developed implementing coordinated scheduling between Grid level and other lower levels managing MPI+OpenMP applications as we propose in this paper. An emerging research group at GGF is working in the Grid scheduling architecture and it is introducing some concepts of coordinated scheduling but it is only a preliminary work. There are different initiatives that deal with the problem of scheduling in the three levels that are proposed in this paper. On one hand, we can meet some problems of meta-schedulers or resource brokers for Grid as can be: AppLes [1], Condor-G [7], EZ-Grid [6], GridBus [15], GRMS [9], or GridWay [10]. These brokers implement several policies, for instance policies based on economic models, based on the monitoring information of the resources, or based on prediction mechanisms. On the other hand there are some local job schedulers which are suitable for HPC applications and they work externally from the queuing system, for example MAUI scheduler [13], EASY [19] or OAR Scheduler [17]. They implement several scheduling policies and support some mechanisms as the advanced reservations. There are other projects that are currently implementing systems that include both queuing system and scheduler, and that also advanced reservation mechanisms, for instance, the OAR scheduler, but they are difficult to extend for achieve our requirements: incorporate mechanisms to interact between the CPU and Grid scheduling levels in HPC environments.

3. System Architecture

3.1. General Overview

The system architecture is presented in Figure 1. The Grid Jobs are managed by the eNANOS Resource Broker and submitted to the local HPC resources through the Globus infrastructure. We use LoadLeveler as a queuing system and the NANOS Scheduler as an external scheduler to manage the local jobs. The CPU scheduling is performed by the NANOS-RM and the NANOS Job Monitor provides the monitoring information about the execution of workloads. The information system 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 Figure 1 also shows the information flow between the main components of the system architecture. This information is needed to perform a coordinated scheduling.

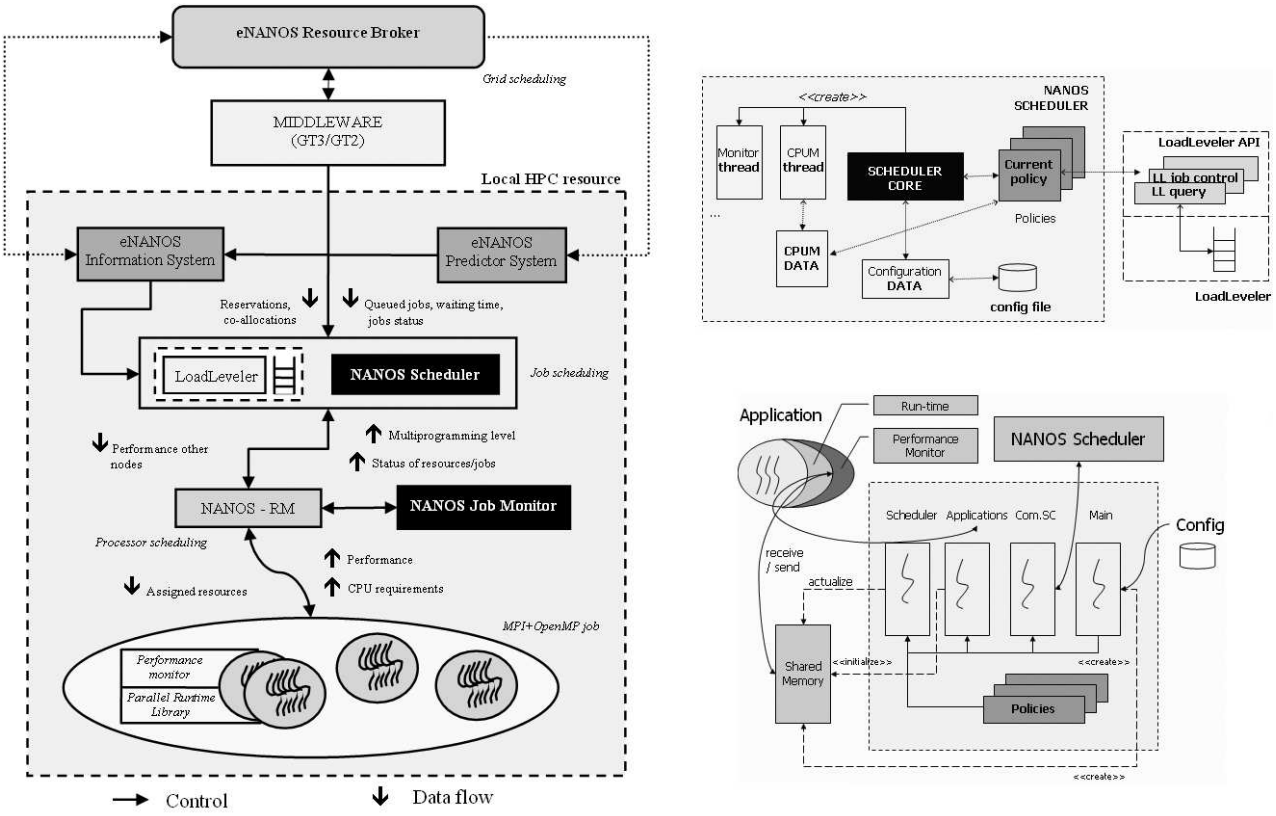

Figure 1. System Overall Architecture (left), NANOS Scheduler Architecture (right-up), NANOS-RM Architecture (right-down)

3.2. eNANOS Resource Broker

The eNANOS Resource Broker is developed on Globus Toolkit 3 as a Grid Service and it is compatible with both GT2 and GT3 services, and implement flexible mechanisms that allow it to become compatible with next Globus versions. It implements the Resource Discovery, Selection and Monitoring, the Job Submission, and the Job Monitoring. Furthermore, 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. Currently the eNANOS broker interacts with our execution environment through the GRAM service provided by the Globus middleware. The broker should obtain information from the Information System and it is also planned to implement scheduling policies based on the information provided by the Predictor Service. More detailed information about this broker can be found in [18].

3.3. NANOS Scheduler

The NANOS Scheduler is responsible to submit and control the jobs which are queued in the queuing system using the LoadLeveler API. The overall architecture of the scheduler is shown in the Figure 1. This scheduler communicates with the NANOS-RM to obtain information related to the applications and the local resources behaviour, and also related to the eNANOS Broker in order to provide information to the upper level. Additionally the scheduler is planned to be communicated to both a prediction service and an information system that are in development. As usual, the scheduler has an iterative behaviour; in each iteration the scheduler performs the following basic functions: searches jobs from the queuing system, updates the system information, evaluates the scheduling

policy, and dispatch the selected jobs

The communication with the other components of the architecture is done through 2 additional threads. The first one is in charge of the interaction with the NANOS-RM. This thread receives information from the runtime about the allowed multiprogramming level by the computational nodes and, also some information related to the applications and hardware, for instance the load of the nodes or the CPUs used by each OpenMP thread. The second one sends the job identifications to the NANOS Job Monitor. It is very important because the Scheduler centralizes the jobs identifications of each layer and is able to map this IDs with the PIDs of processes and threads. It is planned another thread to notify to the Grid level the relevant changes related with the job execution or with the resources. It also can receive instructions from the resource broker but currently, since this functionality is under development, the communication between these components is performed indirectly through the Globus jobmanager and environment variables.

At this moment the scheduler implements the next scheduling policies: FIFO, backfilling and CPUM-based. The policy based on backfilling is implemented with the execution time limits provided by the user. The CPUM-based policy uses the information provided by the NANOS-RM to determine when a node allows the execution of more applications. The runtime information and the multiprogramming level allow us to schedule the next application of the queue following a FIFO policy (depending of the submission time of the job). We are working in a backfilling policy implementation based on the prediction provided for an external prediction service.

3.4. NANOS Resource Manager

The NANOS Resource Manager (NANOS-RM) is a local processor scheduler. Its main goal is to efficiently distribute a set of processors between a set of applications that are under its control. The NANOS-RM offers an API to coordinate with the different components of the system, it interacts with: the parallel applications, the queuing system, the monitoring system, and the performance analysis system. It also includes a set of predefined processor scheduling policies, currently based on space-sharing approaches, to distribute processors. The required functionality to enforce the processor distribution is shared between NANOS-RM and the applications. The NANOS-RM collects the applications requirements, applies the scheduling policy and sends the processor distribution to the applications. We assume that parallel applications are executed in the context of some runtime (OpenMP, MPI) that can provide it the functionality to automatically manage their parallelism. The idea is to modify this runtime to include calls to the NANOS-RM API at the needed points to adjust the processor allocation to the NANOS-RM decisions. The NANOS-RM works in an iterative way (asynchronously) and performs the processors scheduling, the system load control, and the dynamic detection of multilevel applications.

In the Figure 1 is shown the NANOS-RM architecture and the interaction with the NANOS Scheduler. The scheduling policies supported by the NANOS-RM are based on dynamic allocation and have two phases (multilevel). The first phase is between applications, it implements a FIFO policy: one application has N MPI processes and requires a minimum number of N processors. The second phase is between processes of a MPI+OpenMP application and implements two possible policies: Equipartition [12] and Dynamic Processor Balancing [2]. Equipartition starts from the number of cpus allocated to the application and distributes equally among processes of the application. Dynamic Processor Balancing tries to balance the load between the MPI processes of an application. The NANOS-RM interacts with the NANOS Scheduler. It indicates the maximum number of jobs (or multiprogramming level) that the node allows to work without overload problems. The idea is sharing the required information for improving the whole system performance without penalizing the applications performance independently. For the performance analysis, the NANOS-RM detects

the iterative structure of the applications. The spend time to the execution and to the MPI management is measured internally. In order to avoid peaks of unbalance some measures are done and the appropriate iterations are discharged to avoid outliers. More information about the NANOS-RM and the load balancing techniques can be found in [2].

3.5. NANOS Job Monitor

The NANOS Job Monitor collects information about the job execution from both the NANOS-RM and the NANOS Scheduler. It also provides files in XML format describing the execution of the workloads in a computational resource. These XML files follow an schema which contains job information such as the workload name, the Grid job identification (JobID assigned by the eNANOS Broker), the local job identification (LoadLeveler StepID), the job name, the application topology (number of MPI processes and OpenMP threads), and a set of events.

The events describe the life cycle of the threads involved in the job execution. A process event is composed of: the timestamp, the process ID (pid), the thread ID (tid), the status (IDLE, RUNNING, STOPPED, etc.), and CPU (only if the thread status is RUNNING). It also include some events related to the LL actions. The mapping of the job identifications of each layer is very important to the monitoring system because it manages identifications from the grid layer to the process/thread layer. Additionally, we also have implemented a tool to analyze and visualize the workloads. This tool transforms the XML files of the job monitoring to a trace which can be analyzed with the visualization tool Paraver [4]. Another alternative to analyze the workloads previously used in our execution environment is using the IBM Aixtrace tool. With this tool we are able to obtain a binary trace with the system information during an interval of time. With the aixtrace2prv [4] tool the binary trace can be translated to the Paraver format to visualize and analyze it.

3.6. eNANOS Information System

Our system is composed by several entities that provide different kind of information, from the NANOS-RM that can provide information about the state of a process that is running on the system, till the predictor system that provides predictions about the jobs that would be executed in the future. The information system is intended to be a central point of access to all the information related to the system, providing a uniform way to access to the information. It abstracts to the client to which information system the final query is done, it just has to specify which information requires. The information system is not only intended to provide the explained information, it also is intended to provide ways to discover which kind of information is available and how a client can query it, basically providing the XML Schema that describes the format of the XML query. Something important that guided the designing of the IS was that, if it would be necessary, we wanted to add new information systems in an easy way, for instance providing a way to develop modules that could be added to the IS. We could say that this architecture is pretty similar to those architectures presented on monitoring systems as [8], [14], [16], however our information system has to seen as a meta-information system that can collect a very large kind of different information, providing a uniform access to it. Actually we have done a preliminary design of this information system, based on the experience that we have gained on a first stage that consisted on studying the applications information requirements and the characteristics of the some of the most relevant monitoring and information systems that have been published during these lasts recent years [8], [14], [16]. The current state of the work consist on a final design an implementation of the system.

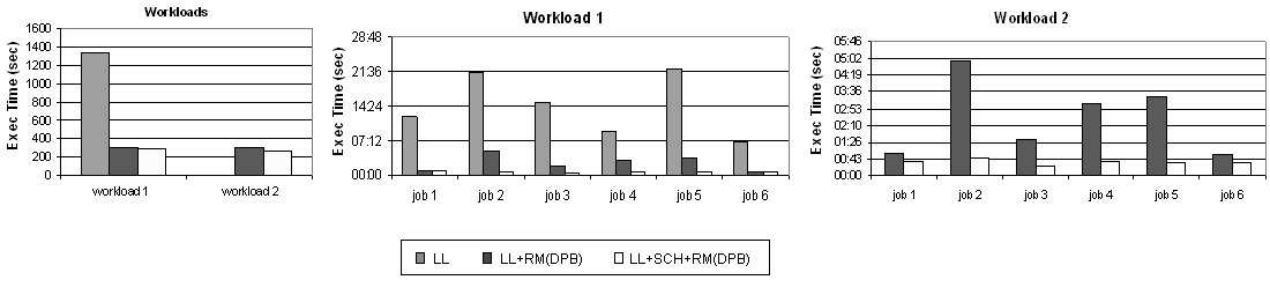

Figure 2. Execution time of the workloads (left) and the individual applications of the workloads

3.7. eNANOS Predictor Service

As has been presented on the previous section this service will be accessed though the information system. This service will provide predictions about the job performance that can be used by both the NANOS Scheduler and the eNANOS broker. It will have a set of prediction techniques, almost all of them will be based in prediction with historical information. We have implemented basically two kinds of prediction techniques: those that are based on statistical approaches that use estimators such as the mean, standard deviation, median and dispersion; and those that are based on data mining algorithms. All the predictors have the similar inputs, such as, user name, group name, number of processes that the job requires, job name and script name. The actual output is a prediction about the user time, system time and memory usage and an interval of confidence that tries to provide information about how accurate can be the prediction. The predicted values can be continuous or discrete; this can be constrained by the query. The prediction service will work basically as a hybrid predictor, depending of the job characteristics and some other input parameters, such as the user, group or system status, it will apply the prediction technique that fits better to it.

4. Preliminary Evaluation

In this paper we present our approach in coordinated scheduling and we are going to evaluate the two lower levels of the architecture. We have evaluated some MPI+OpenMP applications and workloads composed of them in order to show the benefits of this programming model in our execution environment. The evaluation has been performed in an IBM system RS-6000 SP with 8 nodes of 16 Nighthawk Power3 @375Mhz (192 Gflops/s) with 64 Gb RAM and 1.8TB of Hard Disk. The operating system is AIX 5.1 with IBM LoadLeveler queuing system. To evaluate our execution environment we have used the following MPI+OpenMP applications: the NAS Multi Zone (MZ) benchmarks BT, SP of class A, and the CPMD application . We are not going to evaluate scheduling policies; we want to show the potential of our proposed coordinated architecture. To demonstrate the benefits of the architecture we have executed two workloads with the following three configurations:

LL: is the default configuration of the IBM system in which is used a backfilling policy to schedule the queued jobs, there is not any coordination

LL+RM(DPB): the workload is managed by the IBM backfilling scheduler, but the applications are managed by the NANOS-RM with the DPB policy in order to avoid the overloading of the CPUs

LL+SCH+RM(DPB): as well as balancing the CPU load by the NANOS-RM, the job scheduling is done by the NANOS Scheduler in coordination with the NANOS-RM

The experiments are synthetic and are the configuration of them is shown in Table 1, with the

	job1	job2	job3	job4	job5	job6
workload1	BT.A 4x4	BT.A 2x8	SP.A 8x4	BT.A 4x4	SP.A 2x4	SP.A 2x4
workload2	BT.A 4x16	BT.A 2x16	SP.A 8x16	BT.A 4x16	SP.A 2x16	SP.A 2x16

Table 1. Composition of the workloads

application name and the number of MPI processes and OpenMP threads for each application.

In the Figure 2 the execution time of the workloads is shown for the three different configurations. In the first workload with the LL configuration we obtain an execution time higher than with the other two configurations (more or less six times higher). This is caused by an inefficient use of the CPUs. Then, overloading the CPUs of the node can cause undesired situations such as a high number of context switches between processes fighting for CPUs in a short time. In the other two configurations the workload execution time is lower and quite similar between them. With the load balancing in the applications we obtain quiet good results, but in coordination with the local scheduler the results are even better. These improvements are obtained through an efficient scheduling of the applications. We are able to avoid the CPUs overloading by the coordination between the scheduler and the NANOS-RM runtime. We also have to take into account that the LoadLeveler system do not have support for MPI+OpenMP applications, so the backfilling scheduler does not control the second level of parallelism and the taken decisions can be not enough effective. In the second workload every application has the same number of OpenMP threads as the number of CPUs of the node. Thus, we overload the node in a short time during the workload execution, and the second level of parallelism is exploited. Although the number of OpenMP threads is much higher in this workload, the execution time with load balancing is quite similar than the previous case. A good scheduling of the OpenMP level is reflected in a suitable behavior. In this case the improvement in the execution time is better when using the local scheduler and the NANOS-RM together. Due to the workload control is managed by the local scheduler the CPUs are not overloaded.

The individual execution time of the applications that compose each workload is shown in the Figure 2. We can see how in both workloads the execution time of applications is lower when using the Dynamic Processor Balancing. Moreover, the execution time is even much lower when we use the schedulers with coordination. Therefore, with the coordinated scheduling in the local execution environment improve the execution time of the workloads, the throughput, and the response time of the applications. As well as the execution time of the workloads, it is also important the execution time of the applications individually because they are consuming resources during their execution. With the CMPD application we have obtained similar results but with a higher execution time (more than an hour per workload). These two kinds of applications are not problematic because they do not need an intensive use of the resources, the simultaneous execution of several applications do not overload the system. With other applications the load control would be much more important.

5. Conclusions and Future Work

In this paper we have presented our approach in coordinated scheduling of Grid jobs and we have described the main characteristics of the components involved in the eNANOS execution environment. We have discussed that the coordinated scheduling is needed to perform an efficient job scheduling on Grids composed of HPC resources. Furthermore we have presented the evaluation of the lower levels of the scheduling architecture with real workloads. We have seen how the MPI+OpenMP programming model can be a very good choice for the parallel HPC applications

execution on Grids. In the workload evaluation we have seen that the coordinated scheduling between the NANOS Scheduler and the NANOS-RM improve the execution time of the workloads, the throughput and the response time of the individual applications. The system behavior is quite better with the eNANOS execution environment.

As future work our main trend is finishing the eNANOS coordinated scheduling architecture. In general we need to finish and improve all the components of the presented architecture. For instance, at this moment we are designing a more generic and powerful job control interface for the NANOS-RM, to give the job scheduler clients the opportunity to take decisions such as stop running jobs or reduce the allocation of a submitted job. We are also working in the implementation of local scheduling policies for heterogeneous nodes in the NANOS Scheduler. These new policies could be exported to the Grid level taking into account the information of the lower levels. We also have planned to implement new scheduling policies based on prediction techniques (at local scheduler and Grid broker). Moreover, we have to implement the communication between the NANOS-RM and the eNANOS Broker through an information system which is under development. Some other new communication channels should be included or improved. Finally, with our coordinated execution environment it should be easier to implement in the future new functionalities such as checkpointing, coallocation or migration between resources.

References

[1] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao: Application-Level Scheduling on Distributed Heterogeneous Networks. Proceedings of the SC96. 1996

[2] J. Corbalán, A. Duran, J. Labarta: Dynamic Load Balancing of MPI+OpenMP applications. ICPP04, Montreal, Quebec. Canada. 2004

[3] J. Corbalán, R.M. Badia, J. Labarta: eNANOS Performance Tools for Autonomic Grid Resource Allocation Management. CEPBA-IBM Research Institute. 2002

[4] CEPBA Tools Web Site. http://www.cepba.upc.edu/tools_i.htm

[5] CPMD consortium Web Site. http://www.cpmd.org/

[6] EZ-Grid Resource Brokerage System Web Site. http://www.cs.uh.edu/ ezgrid/

[7] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke: Condor-G: A Computation Management Agent for Multi-Institutional Grids. 10th HPDC, IEEE Press. August 2001

[8] Globus Monitoring and Discovery Service (MDS) Web Site. http://www-unix.globus.org/toolkit/mds/

[9] GridLab, A Grid Application Toolkit and Testbed. http://www.gridlab.org

[10] GridWay Project Web Site. http://www.gridway.org

[11] IBM Research, Autonomic Computing Web Site. http://www.research.ibm.com/autonomic/

[12] C. McCan, R. Vaswani, J. Zahorjan: A Dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors. ACM Trans. on Computer Systems. 1993

[13] MAUI Cluster Scheduler Web Site. http://www.clusterresources.com/products/maui

[14] Mercury Grid Monitoring System Web Site. http://www.lpds.sztaki.hu/mercury/

[15] K. Nadiminti, S. Venugopal, H. Gibbins, R. Buyya: The Gridbus Broker Manual (v.2.0). Grid Computing and Distributed Systems (GRIDS). http://www.gridbus.org/broker

[16] Network Weather Service (NWS) Web Site. http://nws.cs.ucsb.edu/

[17] OAR scheduler Web Site. http://oar.imag.fr/

[18] I. Rodero, J. Corbalán, R.M. Badia, J. Labarta: eNANOS Grid Resource Broker. P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, Amsterdam. February 2005

[19] J. Skovira, W. Chan, H. Zhon: The EASY - LoadLeveler API Project. LNCS 1162. 1996

[20] R. Yahyapour, P. Wieder: Grid Scheduling Architecture-Requirements. GGF GSA Research Group. January 2005