A Performance Contract System in a Grid Enabling, Component Based Programming Environment*

Pasquale Caruso[1], Giuliano Laccetti[2], and Marco Lapegna[2]

[1] Institute of High Performance Computing and Networking, Naples branch,
National Research Council – via Cintia Monte S. Angelo, 80126 Naples, Italy
[2] Department of Mathematics and Applications,
University of Naples Federico II – via Cintia Monte S. Angelo, 80126 Naples, Italy

Abstract. In these years, grid computing is probably the most promising approach for building large scale and cost effective applications. However, this very popular approach needs a sophisticated software infrastructure to address several requirements. One of these requirements is the ability to sustain a predictable performance in front to the fluctuations related to the dynamic nature of a grid. In this paper we describe design and realization of a Performance Contract System, a software infrastructure that manages the computational kernel of a grid application with the aim to face such aspect of the grid computing, as well as the strategies and the experiences to integrate it in a grid-enabling component-based programming environment still under development.

1 Introduction

As stated in [7], a Grid is a system that "… coordinates resources that are not subject to centralized control, … using standard, open, general purpose protocols and interfaces,… to deliver non trivial qualities of services". That means that a Grid infrastructure is built on the top of a collection of disparate and distributed resources (computers, databases, network, software …) with functionalities greater than the simple sum of those addends [8]. The "added value" is a software architecture aimed to deliver good Quality of Service (QoS), so a stronger attention has been recently given on the technologies enabling it (see for example [10]). Inside this software infrastructure, a significant part, known as Performance Contracts System, is devoted to the aspects related to the response time and to the delivered performance. Grid topics related to performance contract systems have been widely studied in the last years, mainly in the GraDS project (see for example [1,2]). Other papers (see [15]) report studies about the forecast of the performances in distributed computing environment, by using algorithms simulating an ideal customer, opportunely defined by means of some rules of behavior, in terms of use of the resources. A Performance

* This work has been partially supported by Italian Ministry of Education, University and Research (MIUR) within the activities of the WP9 workpackage *"Grid Enabled Scientific Libraries"* , coordinated by A. Murli, part of the MIUR FIRB RBNE01KNFP *Grid.it* project *"Enabling Platforms for High-Performance Computational Grids Oriented to Scalable Virtual Organizations"*.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 982 – 992, 2005.

Contract of an application for a computational grid by means of the computational cost of the algorithm is defined in [19] and then it is checked and validated.

Approaches that make use of statistical data , on the other hand, are introduced in [13,22]. Results related to the development of performance contracts based on the fitting of runtime data are reported, finally, in [16]. With regard to the run time monitoring, several tools for distributed applications are available and they will be shortly described in Section 4 [18,20,27]. A statistical analysis that, instead of checking all the software modules of the application, uses only some meaningful sections of the application itself, is developed in [17].

This paper is therefore organized as follows: in Section 2 we outline our Performance Contracts System and its role in a grid application; in Section 3 we introduce the software environment in which the Performance Contract System will be integrated; finally, in Section 4, we show some computational results about the definition of the performance contract and the related monitoring of a parallel routine that is part of a medical imaging application.

2 The Role of a Performance Contract System in a Grid Application

One of the aspects of grid computing is the simple and transparent use of the computational resources of a distributed system [8]. To such aim it is "mandatory", in some way, the presence of several software units, that are side by side to the application. Among the tasks of such software modules there are, for example, the selection of the resources, the development of the performance contract, its monitoring and the management of possible violations of the contract itself. The module that manages all activities is the Application Manager (AM), whose outline (or workflow) is depicted in Fig. 1. Its main software component are:

1. Resource broker. This component selects the computational resources on the basis of information about the application (e.g. the dimension of the problem), the user (e.g. the time to solution, that is the maximum amount of time to complete the application), the state of the grid (e.g. resources available in that moment) and finally, information about previous executions (e.g. performances caught up on a machine already used). See [6,14] for an example of selection of the computational resources.
2. Contract developer. This component has in charge the definition of the Performance Contract on the basis of the resources selected in step 1 without other input from the user. These information are combined with those related to the computational features of the application (e.g. the computational cost) as well as to the performance of previous executions (e.g. stored in a "historical performance database"); more details related to these aspects are reported in the sequel.
3. Monitor of the application. This is a key software item for a reliable grid-enabled application, because the actual performance can be very different from that one specified in the Performance Contract. The dynamic nature of distributed resources not under the same centralized control, can do these values very different among them. Some existing tool for the monitoring of distributed application are shortly described in the next Section.

4. Manager of the violation policy. This is the software item aimed to take the suitable actions in case of violation of the Performance Contract. Typical actions are the migration of the application on other resources, redefinition of the terms of the contract or the addition of other computational resources. See [23] for an example of migration strategy.

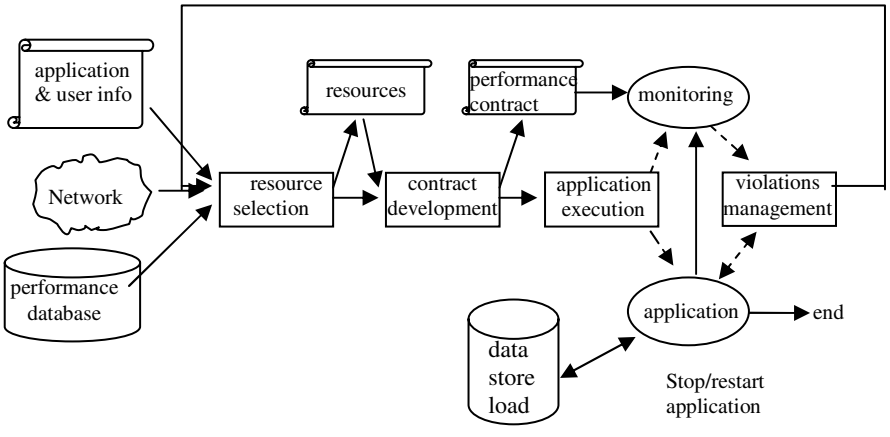

Fig. 1. Workflow of the Application Manager

A Performance Contracts System is the set of all software units of the Application Manager related to the performance contract and its monitoring. The definition of a Performance Contract is not a new one (see for example [26]), but in a grid environment it assumes a key role. A performance contract can be defined as a forecast of the performances of an application on given computational resource. More precisely, assigned:

- some computational resources (e.g. processors, memories, networks...)
- with given capability (e.g. computation speed, memory bandwidth...)
- and an application with given characteristics (e.g. dimension of the problem, amount of I/O, number of operations..)

a Performance Contract states

- the achievement of a fixed performances

There are several way to express a Performance Contract depending on the kind of application. Typical examples are the attainment of F operations/sec, the execution of I iterations/sec, transfer of B bytes/sec or the time necessary to compute a computational step (e.g. one frame in a image reconstruction problem). Obviously the choice among them depends on the features of the application.

Once selected the computational resources, the definition of the performance contract is essentially based on the following information:

1. use of a performance model based on the features of the application and of the selected computational resources. To be realistic, the definition of the model must take into account the computational cost of the algorithm, as

well as the workload of the resources, the fraction of peak performance actually obtainable, values of benchmarks, and so on. Such approach can be defined *Performance Model Approach*, and it is used, for example, in [19].

2. use of data related to the performances of previous executions. As an example, it is possible to use a database in which, for every computational resource selected in the time, average performances actually obtained, and the standard deviation (that can be used as estimate of the eventual deviation from the average value), are stored. The described approach can be defined *Historical Approach Performance*, and it is used, for example, in [22].

As previously said, because of the dynamic nature of a computational grid, during the execution of the application it is necessary to periodically check the actually obtained performances in order to compare them with those ones stated in the contract. Such monitoring is carried out by means of a suitable tool defined as *process monitor*: a sufficiently frequent check allows to take suitable actions (e.g.. migrating from a resource overloaded to another one, with the definition of a new contract; adding new computing resources, …). Tools like Automated Instrumentation and Monitoring System (AIMS) [27], Autopilot [20], Paradyn [18] or the commercial tool Vampir make this control. We note that all of them are based on the concept of *instrumentation* of the code, that is on the insertion of calls to library functions able to capture given information from the running code and to transmit them to a process monitor or a visualization tool.

3 A Grid-Oriented Component Based Programming Environment

Component programming model is a well known paradigm to develop applications. This approach, that can be considered as an evolution of the object-oriented model, that allow to build applications by binding independent software modules (the components) that interact with other components means of well defined interfaces (the ports) according a set of specific rules of the programming environment (the framework). The separation of the support code implemented into the framework from the application code into the components allows to the user to focus the attention on the application, avoiding to deal with environment dependent details [5].

Because the components describing the application can be implemented onto separate hardware and software environments, the component programming model is also a very promising approach to develop grid oriented programming environments [11,12]. So a new grid oriented component based programming environment is one of the goals of *Grid.it* Italian research project [24], where we are currently working to integrate a Performance Contracts System into the programming environment.

As already mentioned, the key role of the framework is to shade the details of the programming environment, by exposing only the services required to implement the components. In a grid oriented programming environment, therefore, beyond to the classical services related to the cycle life of the components (instantiation, resource allocation, …), the framework has to provide more sophisticated grid oriented services like resources discovery, remote data access as well the actions concerning the application structuring and rescheduling. As already said in Section 2, in a grid enabled application, these services are in charge of the Application Manager, that in

this context has a natural implementation in the framework. It is important to note that the middleware Globus [9] will be integrated in the framework in order to address the problems related to the access of the geographically distributed resources. In the *Grid.it* programming model, the components are supplied with several types of ports [24]:

1. Remote Procedure Calls (RPC) interfaces. These are the classical CCA-like ports mainly for client-server applications [5]. These interfaces define the kind of services that the component provides or uses.
2. Stream interfaces. This kind of interfaces allows the unidirectional communication of a data stream between two components. This kind of interfaces allows a better use of the bandwidth in case of high latency networks.
3. Events interfaces. These interfaces are used for the interaction of the components with the framework. The asynchronous events of a computational grid (e.g. the failure of resources) can be communicated to the components through these interfaces in order to take eventual actions for the rescheduling of the application on different resources.
4. Configuration interfaces. These interfaces allow to the Application Manager to access and to modify information and data inside the components and can be used for the reconfiguration steps (e.g. stop and restart of the application on other resources).

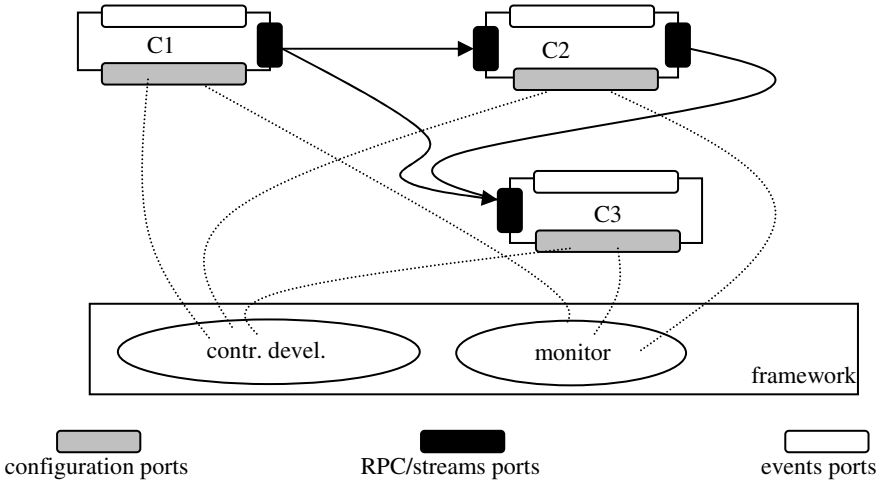

Fig. 2. Integration of the Performance Contract System in to the *Grid.it* environment

In order to integrate the Performance Contracts System, described in Section 2, into the *Grid.it* programming environment we firstly note that, while the application can be developed assembling the components directly by using the RPC or the streams interfaces, the software units composing the Performance Contract Systems (monitor and contract developer) and all related files and data structure can be di-

rectly implemented in the framework interacting with the application through the configuration interfaces

In Fig. 2 is shown an example of application with three components (namely C1, C2 and C3). The components exchanges their data by means of the streams port (black line) while the monitor and the contract developer access the data into the components by means the configuration ports (dotted lines).

More precisely, referring to the integration of the monitor in the environment, it is possible to add the components with proper scripting annotation, specific for the application (e.g. number of floating point operations in each iteration, number of communications,…), reporting which data have to be monitored. These information are accessed by the monitor through the configuration ports and are combined with the information directly acquired from the middleware implemented in the framework (e.g. number of processors to be use, kind of networks,…) and/or from the performances database, in order to define the Performance Contract. Through the same ports, the components provide to the monitor the run time values of the data to be monitored, in order to realize the monitoring process. In such a way the monitor can be based on a general purpose and application-independent template depicted in Fig. 3. An analogous approach can be used for the Performance Contract Developer.

- *Acquire from the components the data to be monitored through the configuration interfaces*
- *Acquire from the middleware the features of the resources to be use*
- *Acquire from the Performance Contract the values to be monitored*
- *Define the step time to get run time data from the components*
- *For each step time*
  - *Get run time values of the data to be monitored through the configuration interfaces*
  - *Test the values with the Performance Contract*
  - *if violation occurs apply violation policies*
- *endfor*

Fig. 3. Template for a general purpose monitor for grid applications

4 Computational Experiments

Our experiments were carried out on a preliminary version (ASSIST-CL 1.2) of the Grid.it environment [25]. The ASSIST programming model is based on a combination of the concepts of structured parallel programming and component-based programming. An ASSIST program is structured as a graph, where the nodes are the components and the edges are the component abstract interfaces, defined in terms of typed I/O streams. The basic unit of an ASSIST program is a component named *parmod* (parallel module), which allows to represent different forms of parallel computation. The user interface of the ASSIST environment is a coordination language, named *ASSIST-cl*.

A layered software architecture has been implemented to support the above programming model on the target hardware architectures, including SMPs, MPPs, and NOWs. An ASSIST-cl code is compiled and then it is loaded and run onto the target

architecture by a Coordination *Language Abstract Machine (CLAM)*. The CLAM is decoupled from the target hardware by a run-time support, named *Hardware Abstraction Layer Interface (HALI)*, which currently exports functionalities from the underling software layers. The ASSIST compiler translates ASSIST-cl source code into C++/HALI processes and threads, using predefined implementation templates. In running this code, the CLAM uses all the facilities provided by HALI, making no assumptions on the existence of other software running on the same nodes and competing to use the same resources. Finally the ACE library supplies standard routines to exchange data between processing elements with different architectures [21]. This is the layer of software that will be substituted with the middleware Globus in the Grid.it programming environment (see also the following Fig. 4).

To monitor the contract, we used the Autopilot library . This is a software environment for the adaptive run time control of geographically distributed applications. Such package is constituted by software items that allow to communicate data of programs in execution to a process monitor. Such software items are said *sensors*. Usually the sensors are used to capture the data related to the effective performance of the computational kernel to be monitored, in order to compare them with those stated in the Performance Contract. Further it is possible to use separate sensors in each process of a distributed application, so that the monitor is able to determine exactly which component of the application eventually causes the violation. Further, Autopilot is able to modify the value of variables of the executing applications, by means of the so-called *actuator* : the presence of the actuators is fundamental for example in a migration step. It is important to note also that Autopilot library does not introduce significant overhead in the software environment [20] and it uses the same Globus middleware that will be used in the Grid.it environment. In the following Fig. 4 the software architecture to realize our experiments is shown. In such an architecture it should be noted the role of the Autopilot library used to realize the monitor process, in accordance with Fig. 2.

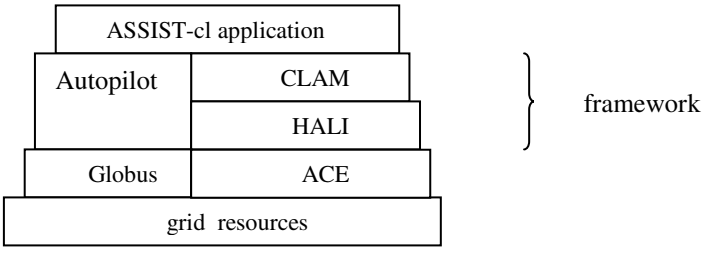

Fig. 4. Software architecture of ASSIST whit the Autopilot library

The computational kernel we used for our tests is based on the Coniugate Gradient (CG) algorithm, implemented in a routine of the the parallel library Meditomo realized to be used in the medical imaging application MediGrid [3,4], that reconstructs 64 independent bi-dimensional images,using 10 iterations of CG for every image, for a total of 640 iterations. Features of the matrices involved are: sparsity, not structure ness, order $n = 10^3$. For this problem we developed a parmod for the re-

construction of the 64 images, where the workload among the processors is distributed dynamically by using a *farm:* a parallel construct available in ASSIST. With this construct, each of the 64 images appearing on the input stream of the parmod is processed, independently from the other ones, by the first free processor.

As previously mentioned, and following a consolidate way, to define a contract it is necessary to know something about the past, in the sense of a historical database containing info regarding performances of previous executions.

Table 1. executions time for the reconstruction of the 64 images

	P=1	*P=2*	*P=4*	*P=8*	*P=12*
exec time without I/O (T_P)	2494	1261	629	313	234

Table 1 shows a very simple example of record of such database, reporting the execution time T_P, in seconds on P processors, of the computational kernel on a dedicated Linux Beowulf cluster with 12 Pentium 2 processors running at 550 MHz, connected by a Fast Ethernet switch at 100 Mbit/sec. We emphasize that such a times does not consider the I/O phases before and after each conjugate gradient. Such a values confirm however the natural parallelism of our problem, because we found that $T_1 \cong P \cdot T_P$.

On the basis of these data, and referring to the definition given in Section 2, we can define the Performance Contract as follow:

- given P processors (the computational resources)
- able to execute the given application in *2494/P* secs (the capability of the resources)
- and an application based on 640 iterations of the Coniugate Gradient
- the Performance Contract, expressed in term of seconds for one iteration, establishes that
- one iteration of the conjugated gradient has to be be executed in *2494/640=3.9* seconds independently from the number of processors P.

The monitored data are those ones defined in the Performance Contract, that is the execution time (Wall Clock Time) of one iteration of the conjugated gradient. By integrating the Autopilot sensors in the routine, we were able to carry out a set of experiments on the Beowulf machine, accessing runtime by means of the monitor, the Wall Clock Time of the execution of one iteration of the conjugated gradient every 40 seconds. After 150 seconds, one of the four processors (Node 1) has been overloaded by a process, stranger to the application, that engages the CPU for approximately 120 seconds before ending. Such overload is aimed to simulate the dynamical nature of the computational environment, in order to check if the performance contracts system, in this case, is able to finding the violation of the contract.

In Figure 5 the results of our test are reported, where on the x-axis is reported the time and on the y-axis is reported the execution time for one iteration of the Conjugate Gradient as caught by the monitor in 4 processors. Further is reported the value of the Performance Contract (PC). It can be view that, when the nodes of the Beowulf are not overloaded with other applications, the monitored values of the execu-

tion time agree with those stated in the Performance Contract. Moreover it is possible to observe that when the Node1 is overloaded with other applications, the actual value of the execution time is very different from that received from the monitor. Such first experiments confirm that our performance contracts system is able to define a realistic contract, able to preview the performance of the application in normal situation, and also to find violations of the contract itself. Such results are encouraging for future developments of the aspects related to the performance contracts system, as for example, the definition and implementation of suitable strategies to face violations of the contract itself.

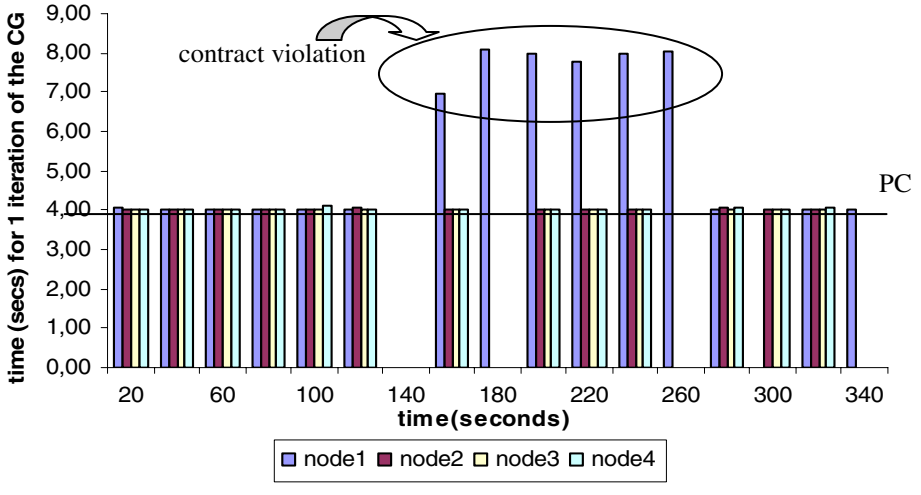

Fig. 5. Results of the monitoring process

References

1. R. Aydt, C. Mendes, D. Reed, F. Vraalsen - Specifyng and Monitoring GRADS contracts - available to the URL http://hipersoft.cs.rice.edu/grads/publications/grid2001.pdf
2. F. Berman, To Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnson, K. Kennedy, C. Kasselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolsky - The Grads Project: Software support for High Performance Grid Applications - *Int. Journal on High Performance Applications*. Vol 15 (2001), pp. 327-344.
3. M. Bertero,P.Bonetto, L.Carracciuolo, L.D'Amore, A.Formiconi, M.Guarracino, G.Laccetti, A.Murli, and G.Oliva – A Grid-Based RPC System for Medical Imaging - Parallel and Distributed Scientific and Engineering Computing: Practice and Experiences, Advances in Computation: Theory and Practice, vol. 15, Y.Pan and L.Yang (eds.), 2004, pp. 189-204.
4. P. Boccacci, P. Calvini, L. Carracciuolo, L, D'Amore, A. Murli - Parallel Software for 3D SPECT imaging based on the 2D + 1 approximation of collimator blur – *Ann. Univ. Ferrara, sez. VII, Sci. Mat. Vol. XLV, 2000*
5. CCA Forum Home page. http://www.cca-forum.org

6. K. Cooper et al. – New Grid Scheduling and Rescheduling Methods in GraDS Project – available at URL http://citeseer.ist.psu.edu/697420.html
7. I. Foster - What is the Grid? A three point checklist - available at URL http://www-fp.mcs.anl.gov/~foster/Article/WhatIsTheGrid.pdf
8. I. Foster , C.Kesselman - The Grid: Blueprint for a New Computing Infrastructure - Morgan and Kaufman 1998
9. I. Foster , C.Kesselman - Globus: a metacomputing infrastructure toolkit - *Int. Journal on Supercomputing Application*, vol. 11 (1997), pp. 115-128
10. I. Foster, C. Kesselman, J. Nick, S. Tuecke – The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Global Grid Formum, 2002
11. N. Furmento, W. Lee, A. Mayer, S. Newhouse, J. Darlington - ICENI: An Open Grid Service Architecture Implemented with Jini – Supercomputing 2002
12. M. Govindaraju, S. Krishnan, K. Chiu, A. Slominski, D. Gannon, and R. Bramley : XCAT 2.0: A Component-Based Programming Model for Grid Web Services.. Technical Report-TR562, Department of Computer Science, Indiana University. Jun 2002.
13. J. Gehring, T. Reinefeld - MARS, framework for minimizing the job execution time in a metacomputing environment- *Future Generation Computer Systems*, vol. 12 (1996), pp. 87-99
14. M.R.Guarracino, G.Laccetti, A.Murli – Application Oriented Brokering in Madical Imaging: Algorithms and Software Architecture – this volume
15. N. Kapadia, J. Fortes, C. Brodley - Predictive Application Modeling Performance in a Computational Grid Environment - Eighth IEEE Int. Symp. On High Performance Distributed Computing (1999), pp. 47-54
16. C. Lu, D. Reed - Compact Application Signature for Parallel and Distributed Scientific Codes - *Proc. of Supercomputing 2002*, (SC2002), Baltimore
17. C. Mendes, D. Reed - Monitoring Large Systems via Statistical Sampling - Proc. LACSI Symposium, Fe Saint, 2002
18. B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Bruce Irvin, K. Karavanic, K. Kunchithapadam, T. Newhall - The Paradyn Parallel Performance Measurement Tools - *IEEE Computer* vol. 28 (1995) pp. 37-46
19. F. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, S. Vadhiyar - Numerical Libraries and the Grid: The GrADS Experiment with ScaLAPACK, - Technical report UT-CS-01-460, 2001
20. R. Ribler, J. Vetter, H. Simitci, D. Reed - Autopilot: Adaptive Control of Distributed Applications - *Proc. of High Performance Distributed Computing Conference*, 1998, pp. 172-179
21. D.C.Schmidt, T. Harrison, E. Al-Shaer – Object Oriented components for high speed network programming – in proc. of 1st conf. on OO technology and systems (1995)
22. W. Smith, I Foster V. Taylor. - Predicting application run times using historical information. - Proc. Of the IPPS/SPDP' 98 workshop on job scheduling strategies for parallel processing (1998)
23. S. Vadhiar and J. Dongarra – A performance oriented migration framework for the grid - Proceedings of the 3st International Symposium on Cluster Computing and the Grid, 2003
24. M. Vanneschi – High Performance Grid Programming Environments: The Grid.it ASSIST Approach , *invited lecture, ICCSA 2004.*
25. M. Vanneschi – The programming model of ASSIST, an environment for parallel and distributed portable applications – Parallel Computing, vol. 28 (2000), pp. 1709-1731

26. F.Vraalsen, R. Aydt, C. Mendes, D. Reed – Performance contracts: predicting and monitoring application behaviour – Proc. IEEE/ACM Second Intern. Workshop on Grid Computing, Denver, 2001, Springer Verlag LNCS, vol. 2242, pp. 154-165
27. J. C. Yan, M. Schmidt and C. Schulbach. "The Automated Instrumentation and Monitoring System (AIMS) -- Version 3.2 Users' Guide". *NAS Technical Report. NAS-97-001*, January 1999