Monitoring and Performance Analysis of Grid Applications*

Bartosz Baliś[1], Marian Bubak[1,2], Włodzimierz Funika[1], Tomasz Szepieniec[2], and Roland Wismüller[3,4]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] LRR-TUM – Technische Universität München, D-80290 München, Germany
[4] Institute for Software Sciences, University of Vienna, A-1090, Wien Austria
{bubak,funika,balis}@uci.agh.edu.pl, wismuell@in.tum.de
*p*hone: (+48 12) 617 39 64, *f*ax: (+48 12) 633 80 54 *p*hone: (+49 89) 289-28243

Abstract. This paper presents the main ideas and the design details of a performance analysis tool - G-PM and a Grid application monitoring system - OCM-G for applications running on the Grid which are under development within the EU CrossGrid project. Besides of the operation of G-PM's components, we overview its internal interfaces. G-PM enables not only standard measurements, but also comprises application-specific metrics and high-level measurements. The OCM-G is aimed to provide services via which tools supporting application development are enabled to gather information, manipulate, and detect events that occur when applications are running. The functionality of the OCM-G is available via a standardized interface, On-line Monitoring Interface Specification (OMIS).
Keywords: Grid computing, monitoring, performance analysis, measurement tools, interactive applications, instrumentation

1 Introduction

A new EU project – CrossGrid [6] – extends existing Grid technologies by *interactive applications*. Besides providing the necessary Grid services and the testbed, four interactive Grid applications are developed in the CrossGrid: simulation of vascular blood flow, flooding crisis support tools, data mining in High Energy Physics, and meteorology / air pollution simulation.

Applying the Grid applications requires to exactly know the performance behavior of the applications. However, even with a good knowledge of the application's code, its detailed runtime behavior in a Grid environment is often hard to figure out because of the dynamic nature of this infrastructure. To support this task, a tool is being developed, named G-PM, which besides of the the standard performance metrics, allows to determine higher-level performance

* This work was partly funded by the European Commission, project IST-2001-32243, CrossGrid [6]

properties and application specific metrics, like e.g. the response time and its breakdown.

There exist a number of performance tools, which are already adapted to the Grid [1]. They are based on an off-line analysis of event traces so they are not helpful for interactive applications, since they cannot present the performance data concurrently with the end-user's interactions with the application. On-line tools for the Grid are available mainly for infrastructure monitoring, used for resource management. An example is the Network Weather Service [14]. A notable exception is the monitoring system GRM [2], which is in its structure very similar to OCM-G. GRM is adapted to the Grid within the DataGrid project and offers on-line performance data of Grid applications. However, GRM is mainly based on event traces. Autopilot [12], a distributed performance measurement and resource control system, exploits a concept called *sensors* corresponding to our probes. User-defined instrumentation is used in the TAU performance analysis environment [9]. The SCALEA tool [11] supports application specific instrumentation via directives inserted into the source code. The APART Working Group[1] has developed a specification language ASL [7] which allows to specify performance properties at a high level of abstraction.

Paradyn [10] is one of the few performance tools that strictly follow the on-line approach, where even the instrumentation is inserted and/or activated during run-time. Another such tool is PATOP [4], which is the predecessor of G-PM.

The G-PM tool will use three sources of data: performance measurement data related to the running application, measured performance data on the execution environment, and results of micro-benchmarks, providing reference values for the performance of the execution environment. The OCM-G is meant to enable investigation and manipulation of parallel distributed applications running on the Grid and to provide a basis for building tools supporting development of parallel applications for the Grid (see Fig. 1).

2 Basic features of the performance analysis tool

The tool should provide performance data meaningful in the application's context, including application-specific data ("amount of disk I/O for a specific user's interaction", "detailed breakdown of an interaction's response time", or "convergence rate of the numerical solver". The performance analysis of an interactive application should be carried out in on-line mode. This allows to correlate the performance data with the end-user's interaction patterns. Besides the application's performance data, the tool should display data on the performance of the computing environment. This data will be used for steering an application and for assessment of an application's performance. Finally, the tool must well be integrated with the Grid infrastructure, especially the job submission services. Submitting a job with the performance analysis tool attached to it should be as simple as a normal job submission.

[1] see `http://www.fz-juelich.de/apart/`

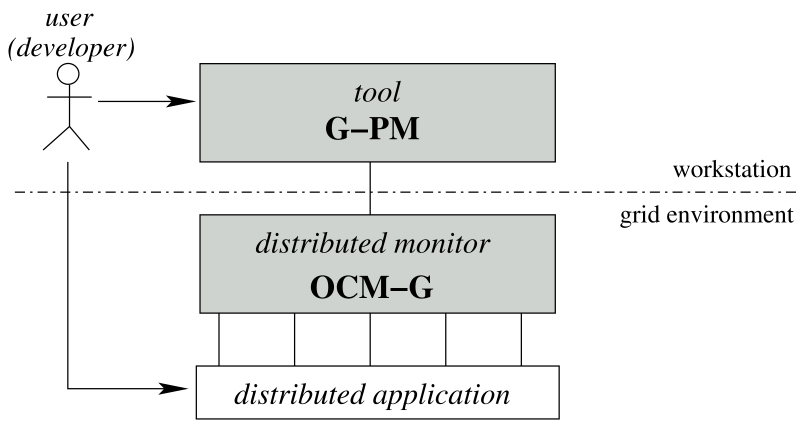

Fig. 1. General overview of G-PM/OCM-G monitoring infrastructure

According to these requirements, the tool consists of three main components (see Fig. 2):

1. a performance measurement component (PMC),
2. a component for high level analysis (HLAC),
3. a user interface and visualization component (UIVC).

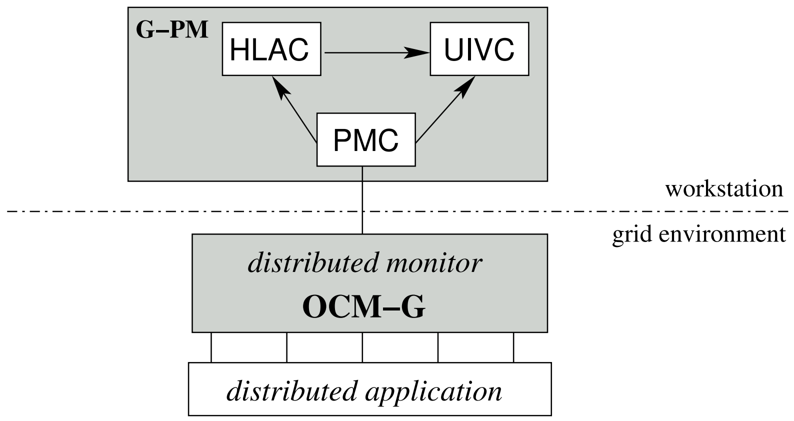

Fig. 2. G-PM architecture in monitoring infractructure

The PMC provides the functionality for standard performance measurements of both Grid applications and the Grid environment. The results of these measurements can be directly visualized by the UIVC component and they can serve as an input to the HLAC component. The HLAC provides application developers with more meaningful, application-specific performance data. This is achieved by providing a metrics specification language. Finally, the UIVC allows the user to specify performance measurements and visualizes the performance data produced by PMC and/or HLAC.

There are two major interfaces:

- Measurement Interface: This interface allows to define performance measurements and to read their results. Both the HLAC and the PMC provide the same interface.
- OCM-G Interface: The interface to the monitoring system is based on OMIS.

As the PCM and the HLAC implement the same interface, the UIVC (and in turn also the user) can handle measurements based on user-defined metrics in the very same way as those based on standard metrics. The interface is implemented in C++.

3 New approach to monitoring on the Grid

Monitoring services are essential in Grid environments for several purposes. At least two of them can be distinguished — infrastructure and application monitoring.

3.1 Grid infrastructure monitoring

Monitoring of the Grid infrastructure is aimed to provide information about Grid components, such as computing elements or network connections. Measured parameters are for example current CPU load and network connection load. The information is useful for resource brokers, scheduling agents, etc., and is needed for such tasks as resource allocation or load balancing. Moreover, not only the current status but also statistic information can be of use, such as average network load, etc. Statistic data can be useful, e.g., for prediction purposes. For this reason, monitoring of infrastructure usually involves databases in which the information should be stored for statistical analysis.

3.2 Grid application monitoring

Monitoring of applications, in the sense as we speak of it, is quite different. The information of interest is what is happening inside applications, e.g., what subroutines are being called, what are the delays due to synchronization or communication between processes, how much data is sent between processes, etc. The purpose of this type of monitoring is mainly for tools such as performance analyzers or visualizers, i.e., for bottleneck detection, observation of the current status of processes, etc. This type of monitoring may also be required to provide manipulation services, such as to stop processes, read/write process' memory, etc. Manipulation services are essential for debuggers.

On the contrary to the infrastructure monitoring, the application monitoring information is only meaningful in the context of a particular monitoring session.

4 OMIS as a monitoring protocol

Two notions are important in our approach to monitoring: *services* and *objects*. Monitoring functionality of the OCM-G is exposed as a set of *services* which can be divided into three classes: 1. for collecting information, 2. for doing manipulations, and for 3. detecting events, i.e. information, manipulation and event services, respectively. *Objects* represent the entities in the target system, either real ones such as sites, nodes, and processes, or more abstract ones, such as counters. The concrete monitoring requests always consist of one or more monitoring services, and, for each service, a list of objects to which the service should be applied, and perhaps additional arguments.

The protocol of communication with the OCM-G, in which the monitoring requests are expressed is OMIS (On-line Monitoring Interface Specification) [8]. The objects in OMIS are identified by *tokens*, e.g., "p_1" (process), "n_1" (node), "app_1" (application), etc. Below is a handful of examples of OMIS services and requests.

- `proc_get_info` – information service for getting information about processes; the service expects a list of processes, and a specification of what kind of information should be returned for these processes.
- `:proc_get_info([p_1,p_2], 0x1)` – a concrete request which returns information about processes p_1 and p_2; the type of information is specified by the flag '0x1'; this is an example of a *unconditional request*, i.e., a list of actions which are to be executed immediately; thus, the effect of a unconditional request is a single piece of information.
- `thread_executes_probe` – an event service which reprents the event of an execution of a probe; the service expects list of processes in which the event should be detected, and a name of the probe.
- `thread_executes_probe([p_1], ''probe1''):`
  `pa_counter_increment([c_1])` – a request whose semantics is as follows: each time, the probe named "probe1" is executed in process p_1, increment the counter c_1; this is an example of a *conditional request*, i.e., a specification of an event, and a list of actions which are to be executed each time the event occurs; thus, the effect of a conditional request is a stream of information; note the colon sign which separates the event from the action list; in case of unconditional requests the event part is empty.

5 Design of the monitoring system

The OCM-G was designed to be a permanent Grid service and at the same time to satisfy the high requirements for scalability, efficiency, and security, and to enable monitoring of applications distributed across multiple sites. The OMIS specification, originally designed for monitoring applications in cluster environments, was extended with new objects and services, suitable for the Grid. The following sections describe these issues.

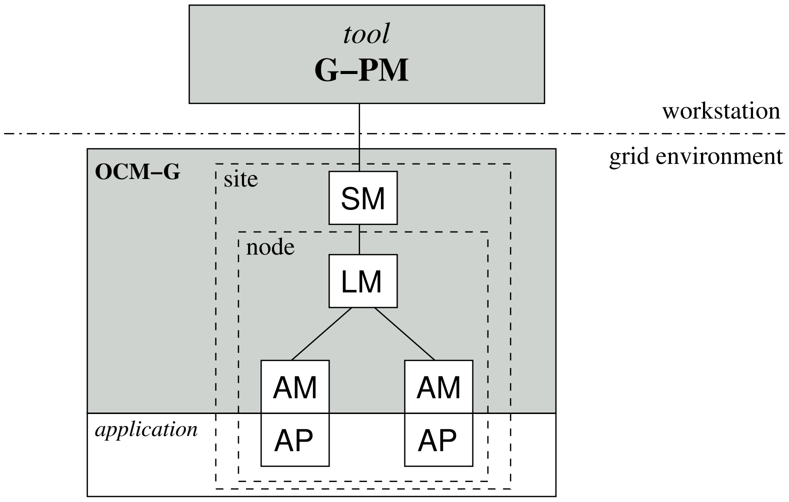

Fig. 3. OCM-G architecture in monitoring infrastructure

5.1 Structure of the OCM-G

The OCM-G is a composition of two types of modules: Local Monitors (LMs) and Service Managers (SMs). A LM resides on each host in the Grid on which there are application processes to be monitored. It accepts and executes OMIS requests only regarding local objects. A SM exposes the monitoring functionality to end users (tools); one SM runs on each site of the Grid. A tool, in order to begin a monitoring session, must connect to its SM (i.e., the SM on the same site). The task of the SM is to accept OMIS requests from the tool, split them into subrequests which will be forwarded to appropriate LMs within the same site and, possibly, to other SMs, if the request concerns objects on remote sites.

Local Monitors are only created when needed, i.e., when new processes to be monitored are created on a host; LMs can be terminated when the processes are gone. Service Managers are permanent; they are supposed to be started as part of the Grid middleware.

With the request distribution pattern described above, communication between two sites is only possible via the appropriate SMs. This in fact enables monitoring of applications distributed across multiple sites. As SMs are permanent, they can be assigned a well-known port numer making the communication behind firewalls possible provided that the port is open.

The described structure is shown in fig. 3. The additional component shown in the figure is the monitoring part linked to the application ("AM"). This part provides the code to initialize the communication with the OCM-G, the infrastructure to handle execution of actions in the context of an application process, etc.

5.2 Grid services

The adaptation of OMIS to the Grid required several extensions to the specification. First of all, the object hierarchy was extended by new types of objects — *sites* — which are in the top of the hierarchy. We feel that it better reflects the structure of the Grid which can be viewed as a collection of sites which are composed of individual nodes hosting processes.

OMIS must also be extended with new services. This includes:

- services related to the new site objects (e.g., to get information about a site);
- services for infrastructure-related metrics which were not necessary in cluster environments (e.g., return information about a network connection);
- services for handling multiple applications (return list of applications, return list of processes of an application, etc.);
- other services, not indispensable in a Grid environment but adding new functionality, such as services for handling *probes* — objects inserted by a user into the source code to define arbitrary events and user-defined metrics for performance analysis.

5.3 Security

Security problems occur on two levels. First, each LM is allowed to perform manipulations to the target system, e.g., it can read and write processes' memory. A user authorized to send a request to a LM could then acquire access to other user's processes. Furthermore, the LM is supposed to handle all processes for all users; thus, it would need priviledged user's rights which can be dangerous for the system. For these reasons, one LM is created on a host *for each user* who owns some processes to be monitored. In other words, LMs run with user priviledges. It means that there may be more than one LMs on one host, but the security is ensured.

Second problem occurs at the SM level, which handles requests from multiple users. This opens a possibility, that a user authorized to send any monitoring request, will try to gain access to other user's application. To prevent this, Grid authentication mechanisms should be incorporated: each request will be properly signed and a check will be perofrmed whether the user who sent a request is authorized to operate on objects the request is related to.

6 Example of monitoring scenario

In this section, we present a scenario of an example monitoring session. Let's assume, that we have an application in which the algorithm is realized in two nested "for" loops, and the user wants to know the global volume of data sent in each step of the algorithm, where the step is meant as an interation of the outermost loop. For simplicity, let's also assume that only `MPI_Send` is used to send messages.

The user wants to see the result of monitoring on some form of a chart, e.g., on a bargraph chart in which one bar represents the amount of data sent in one step of the algorithm.

First, the application must be prepared to enable monitoring. An instrumented MPI library and a monitor library should be linked to the executeble. Additionally, in this case the user should manually insert a probe into the source code which will represent the event of and end of one algorithm step. The probe is inserted as a function call `ocmg_probe(''nextstep'')`, where the argument is the name of the probe, arbitrarily chosen by the user. The whole scheme of the application is shown below.

```
for ( ... ) {
  for ( ... ) {
    // computations and communication
    ...
  }
  ocmg_probe("nextstep");
}
```

Once the application has been prepared, it can be submitted in a usual way (`globus_run`, portal, etc.). The only additional requirement is that special command line parameters are needed, which specify the name of the application, and the location of one selected SM that should become the MainSM for this application.

In this case, three application processes are started, each one located on a different node, and spread across two sites (fig. 4). At the very beginning, each process calls a function to register in the OCM-G. The result of the call can be the creation of a LM, if one does not exist on the host; if this is the case, the newly created LM should register in its SM and from then it becomes a part of the OCM-G. The information about a new process is firstly delivered to the LM, which in turn passes it to its SM. The SM should then forward this information to the MainSM for the application, since the MainSM is supposed to always have an up-to-date knowledge about the application. When this step is finished, the MainSM possesses the complete information about the application; specifically, it knows about all sites on which the application runs. The SMs of all these sites and the underlying LMs form a Virtual Monitoring System for this application. The MainSM is the main information source for all other components of the VMS.

Once the application and the monitoring system is started up, the user can run the G-PM tool. G-PM connects to one of the SMs, and next it receives information about all running applications which can be monitored by the user who controls the the tool. The next step is to select one of the running applications and *attach* to it, i.e., join the Virtual Monitoring System for this application. As of this moment the tool can perform measurements on the attached application.

The user, by means of the graphical interface, defines the measurement and the visualization chart, and enables the monitoring. The defined measurement

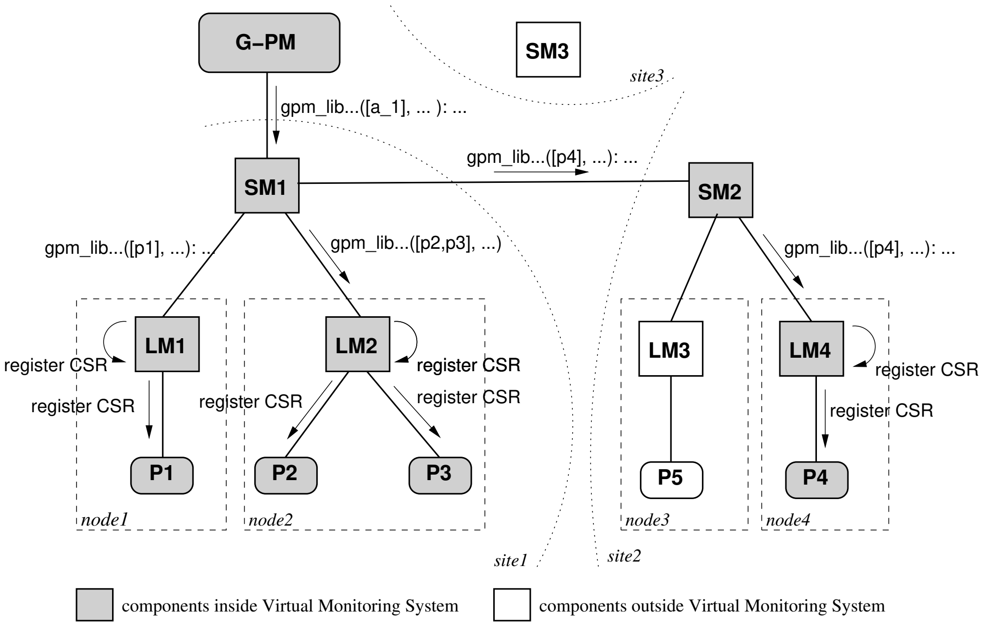

Fig. 4. Distribution of a conditional OMIS request

is transformed into a sequence of OMIS requests. In our case, the requests are essentially as follows:

```
1) gpm_lib_call_started([a_1], "MPI_Send"):
     pa_counter_increment([c_1], $len)
2) thread_executes_probe([p_1], "nextstep"):
     pa_counter_read([c_1])
3) :csr_enable([csr_1,csr_2])
```

The distribution of the first request across the components of the VMS is shown in fig. 4. The token `a_1` represents the whole application and is expanded to lists of processes in the subrequests. This is possible, since all the SMs of the VMS possess the knowledge about the whole application. Note that this mechanism could also work in such cases as process migration or a new process creation, since all the information data structures would be immediately updated and further events would include the changes due to migration/creation.

The semantics of the request is as follows. The first request tells the OCM-G that each time the `MPI_Send` function is called by whichever process of the application `a_1`, the counter `c_1` should be incremented by the length of the message sent ("$len" parameter).

The second request triggers the event related to the inserted probe. Its meaning is that when the probe is hit, the counter `c_1` will be read, its value will be returned and reset to zero.

The final request is used only to enable the two previous ones. The passed arguments `csr_1` and `csr_2` are tokens identyfing the requests, and are returned on their definition. Note that the final request, unlike the previous two, is an unconditional one.

Once the CSRs are enabled, the monitoring begins. The two events defined in the measurement by requests 1. and 2. are caught by means of instrumentation (the first one due to the instrumentation of the MPI library, the second one due to the inserted probe).

Monitoring is active until the user disables the measurement or the application has finished.

7 Summary

The main contribution of G-PM is its unique combination of Grid awareness, online measurement, and automatic instrumentation for standard metrics on the one hand with a support for manual instrumentation and user-definable metrics on the other. The OCM-G is designed as a Grid Service – it is permanent, being accessible via a well-defined interface, OMIS. The architecture of the monitoring system ensures a high scalability and efficiency of application monitoring.

The software design phase for the G-PM tool was recently finished. The first prototype of G-PM is going to be available at the beginning of 2003. This prototype will include some standard performance measurements, and some examples of higher-level metrics, but will not yet include fully user-definable metrics. The final version of G-PM will be ready by the end of 2004. At present the first prototype of the OCM-G is about to enter the test phase. The first prototype will support all services defined by the OMIS 2.0 specification and some new Grid extensions needed for the first prototype. This version will run only on one site and support one application and one tool. A fully functional version of the OCM-G will be available at the end of the CrossGrid project.

Acknowledgements. We would like to thank Mr. Tomasz Arodź, Marcin Kurdziel and Marcin Radecki from AGH as well as to Mr. Hamza Mehammed from TUM for their contribution. This research is partly funded by the European Commission the IST-2001-32243 Project CrossGrid.

References

1. Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajda. Comparison of Representative Grid Monitoring Tools. Laboratory of Parallel and Distributed Systems (SZTAKI), LPDS-2/2000, 2000
   ftp://ftp.lpds.sztaki.hu/pub/lpds/publications/reports/lpds-2-2000.pdf
2. Z. Balaton, P. Kacsuk, N. Podhorszki,and F. Vajda. From Cluster Monitoring to Grid Monitoring Based on GRM. In: R. Sakellariou, J. Keane, J. Gurd, and L. Freeman (eds.), Euro-Par 2001 Parallel Processing, 7th International Euro-Par Conference, August 2001, Manchester, UK, pp. 874-881, vol. 2150, Lecture Notes in Computer Science,Springer-Verlag, 2001
   http://link.springer.de/link/service/series/0558/papers/2150/21500874.pdf

3. B. Baliś, M. Bubak, W. Funika, T. Szepieniec, and R. Wismüller. An Infrastructure for Grid Application Monitoring. In: Kranzlmüller, D. and Kacsuk, P. and Dongarra, J. and Volkert, J. (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting, September - October 2002, Linz, Austria, 2474, Lecture Notes in Computer Science, 41-49, Springer-Verlag, 2002
4. M. Bubak, W. Funika, B. Baliś, and R. Wismüller. On-Line OCM-Based Tool Support for Parallel Applications. In: Yuen Chung Kwong (ed.), Annual Review of Scalable Computing, vol. 3, ch. 2, pp. 32-62, World Scientific Publishing Co. and Singapore University Press, 2001
`http://www.wspc.com.sg/books/compsci/4663.html`
5. M. Bubak and W. Funika and R. Wismüller. The CrossGrid Performance Analysis Tool for Interactive Grid Applications. In: Kranzlmüller, D. and Kacsuk, P. and Dongarra, J. and Volkert, J. (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting, September - October 2002, Linz, Austria, 2474, Lecture Notes in Computer Science, 50-60, Springer-Verlag, 2002
6. CrossGrid Project: http://www.eu-crossgrid.org
7. T. Fahringer, M. Gerndt, G. Riley, and J. L. Träff. *Knowledge Specification for Automatic Performance Analysis.* APART Technical Report, ESPRIT IV Working Group on Autommatic Performance Analysis, November 1999.
`http://www.fz-juelich.de/apart-1/reports/wp2-asl.ps.gz`
8. T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. OMIS — On-line Monitoring Interface Specification (Version 2.0). Shaker-Verlag, 1997, Aachen, Germany, vol. 9, ISBN 3-8265-3035-7
`http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz`
9. A. Malony and S. Shende. Performance Technology for Complex Parallel and Distributed Systems. In: G. Kotsis and P. Kacsuk (eds.), Proc. Third Austrian-Hungarian Workshop on Distributed and Parallel Systems, DAPSYS 2000, 37-46, Kluwer, 2000
`http://www.cs.uoregon.edu/research/paracomp/papers/dapsys2k.ps.gz`
10. B. P. Miller et al. The Paradyn Parallel Performance Measurement Tools. In: IEEE Computer, vol. 28(11): 37-46, Nov. 1995
`http://www.cs.wisc.edu/paradyn/papers/overview.ps.gz`
11. H.-L. Truong and T. Fahringer. SCALEA: A Performance Analysis Tool for Distributed and Parallel Programs. In: B. Monien and R. Feldman (eds.) Euro-Par 2002 Parallel Processing, 8th International Euro-Par Conference, August 2002, Paderborn, Germany, vol. 2400, pp. 75-85,Lecture Notes in Computer Science,Springer-Verlag
`http://link.springer.de/link/service/series/0558/papers/2400/24000075.pdf`
12. J.S. Vetter and D.A. Reed. Real-time Monitoring, Adaptive Control and Interactive Steering of Computational Grids. In: The International Journal of High Performance Computing Applications, vol. 14, pp. 357-366, 2000
13. R. Wismüller, M. Bubak, W. Funika, and B. Baliś. *A Performance Analysis Tool for Interactive Applications on the Grid.* Workshop on Clusters and Computational Grids for Scientific Computing, September 2002, Le Chateau de Faverges de la Tour, France. To appear.
14. R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. In: Future Generation Computer Systems, vol. 15, pp. 757-768, 1999