GAMoSe: An Accurate Monitoring Service For Grid Applications

Thomas Ropars[†*], Emmanuel Jeanvoine[†‡], Christine Morin[†#]
[†]IRISA/Paris Research group, [*]Université de Rennes 1, [‡]EDF R&D, [#]INRIA
{*Thomas.Ropars,Emmanuel.Jeanvoine,Christine.Morin*}*@irisa.fr*

Abstract

Monitoring distributed applications executed on a computational Grid is challenging since they are executed on several heterogeneous nodes belonging to different administrative domains. An application monitoring service should provide users and administrators with useful and dependable data on the applications executed on the Grid. We present in this paper a grid application monitoring service designed to handle high availability and scalability issues. The service supplies information on application state, on failures and on resource consumption. A set of transparent monitoring mechanisms are used according to grid node nature to effectively monitor the applications. Experiments on the Grid'5000 testbed show that the service provides dependable information with a minimal cost on Grid performances.

Keywords: Distributed Systems, Grid Middleware, Application Monitoring

1 Introduction

Computational Grids [5] bring together a great number of computing resources possibly distributed on several administrative domains. They can provide the amount of resources needed for High Performance Computing (HPC). But executing and managing distributed applications on grids is a challenge since grid nodes are heterogeneous and volatile. Inherently to the grid nature, grid services must provide the user with dependable and consistent services to ease grid use.

In this paper, we present GAMoSe, a grid application monitoring service. The goal of this service is to facilitate the execution and the management of applications on grids by providing information on application execution. The architecture of GAMoSe has been designed to handle grid specificities, i.e. node volatility, node heterogeneity and grid scale. The monitoring mechanisms of GAMoSe can provide dependable information on the state of the applications, resource utilization and failure detection. Furthermore GAMoSe is totally transparent for applications and for the operating system of grid nodes.

The rest of the paper is organized as follows. In Section 2, we present the context of our work. Section 3 exposes the requirements and the information GAMoSe should provide. The architecture and the communication model of the service are presented in Section 4. We also outline how we ensure the high availability of the service. In Section 5, we present implementation details on monitoring mechanisms. GAMoSe has been integrated into the Vigne Grid Operating System [8, 11]. An evaluation of the service is presented in Section 6. To show how GAMoSe can help in executing and managing applications in a grid, we detail in Section 7 some possible uses of the information provided. Section 8 describes related work. Finally, we draw conclusions from this work in Section 9.

2 Context

In this section, we expose the context of our work and the assumptions we made. We also define some terms we use in this paper.

2.1 Grid Model

Computational Grids bring together a great number of computing resources distributed over many administrative domains. In this study, we consider Grids with an uncountable number of nodes. A grid node can be a Linux PC or a cluster which is also seen as a single node from a grid point of view. A cluster can operate with a Single System Image (SSI) or with a batch scheduler. As the goal of an SSI is to give the illusion of a single SMP machine, we consider that a SSI cluster is equivalent to a Linux PC at grid level: we call them Linux nodes. A grid is a dynamic system in which nodes can join or leave at any time and where failures

Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07)
0-7695-2936-4/07 $20.00 © 2007 IEEE

can occur. Failures can be node failure or failure of a communication link.

Each node of the grid has its own local Operating System (OS). To provide services at grid level, a Grid Operating System (GOS), Vigne for instance, is executed on each node on top of the local OS. A GOS provides services like resource discovery and allocation or job management. This paper focuses on an application monitoring service.

2.2 Application Model

A distributed application can make use of resources provided by the Grid since it can run on several nodes. Our service targets this kind of application.

We call application components the different parts of a distributed application. An application component runs on a single grid node, as illustrated on Figure 1, and can be composed of one or many processes. These processes may be created dynamically during the execution. Several application components that belong to different applications can be executed concurrently on the same node.

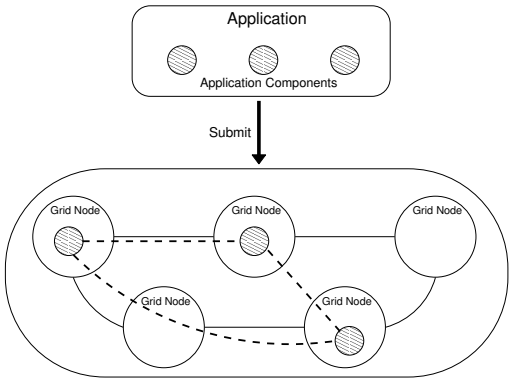

Figure 1. A distributed application executing in a grid environment

3 Requirements

To have a global view of GAMoSe, we first detail the information it provides. In the second part of the section, we deal with the properties of the service. In this paper we don't deal with security issues. In a last part, we explain why we want the monitoring mechanisms to be transparent.

3.1 Definition of the Service

The goal of a grid application monitoring service is to provide data on the execution of applications in the grid. GAMoSe supplies information about the state of the application, on failures and on the amount of resources utilized.

State of the Application Components The best way to give information on the state of a distributed application is to give the state of each application component. GAMoSe does not try to compute a global state for the application since it would not be pertinent.

The state of an application component qualifies its evolution on the node it has been submitted to. However, GAMoSe cannot give the percentage of computation executed because this evaluation can only be done by instrumenting the application code and this does not meet the constraint of transparency exposed in Section 3.3. The state of the application components enable the user to keep an eye on the evolution of the applications she has submitted to the grid.

Failure Detection Determining the state of application components also includes failure detection. As we saw in Section 2.1, failures of grid nodes or communication links can occur. These failures need to be detected because it can induce an issue for the global application achievement if they affect nodes where application components were running. Application components can fail for many other reasons, a bug in the application code or the reception of a signal like SIGKILL for instance.

Providing information on failures is mainly useful for grid application management. Section 7 details how accurate information on application failures can be used to improve application management.

Resource Consumption The resource consumed by an application during its execution must be evaluated. GAMoSe measures CPU time utilization and memory usage. Measuring resource consumption is useful for accounting. It can also help administrators to evaluate the level of use of their resources.

3.2 Service Properties

GAMoSe has to deal with the grid specificities presented in Section 2.1, to ensure a dependable service.

High Availability: Grid nodes are unreliable due to disconnections and failures. GAMoSe should be highly available and provide accurate information despite reconfigurations.

Scalability: A multitude of applications can potentially run simultaneously on the grid. Monitoring all these applications will generate a large amount of data. Dealing with this amount of data should impact neither the service performance nor the grid performance.

Accuracy: On each node of the grid, the monitoring of the application components must be as precise as possible according to the specificities of the local OS. The failure of one process can induce the failure of an application component. The application monitoring service has to monitor all application processes, even those created dynamically.

3.3 Constraint: Transparency

For several reasons, we have added a constraint to the monitoring mechanisms used: they should be transparent for the applications and for the local OS.

First of all, GAMoSe must be transparent for the applications. It should require no modifications in the applications and no relinking to be able to monitor applications that were not initially designed to work with the service.

Moreover GAMoSe must be transparent for the local OS since modifying an OS is a complex task that induces unwanted effects. First, resource owners will probably not share their resources on the grid if it has an impact on their operating system. Then, a resource in the grid can also be used by a local user outside the grid context. In this case, she probably does not care about grid features.

4 Design Principles

Requirements detailed in Section 3.2 have to be taken into account when designing GAMoSe. The service architecture and the communication model must provide support for scalability and high availability.

4.1 Architecture

Dealing with node failures is the most difficult case to ensure high availability of the service. Therefore, no node should have a central function to avoid a single point of failure issue that would compromise the entire service. Furthermore, GAMoSe must be able to manage efficiently large amount of data. Thus, a distributed architecture is the appropriate solution to overcome these issues.

The architecture of the service is presented in Figure 2. Two types of entity compose the service: application monitors and component monitors. In this example, two applications, each composed of two application components, are monitored. Component monitors are not represented on nodes hosting application monitors, even if one is present.

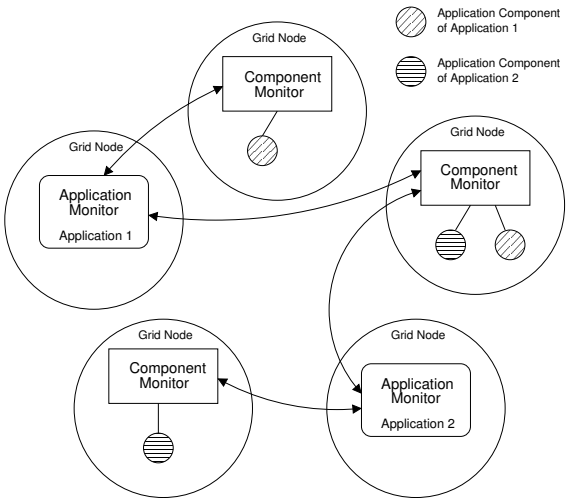

Figure 2. Architecture of GAMoSe

4.1.1 Application Monitor

Each application has its own application monitor. An application monitor is created when an application is submitted to the Grid OS and is destroyed after the end of the application execution. The application monitor collects the monitoring information about each component of the application and compute the application resource consumption. To collect information, application monitors have to interact with component monitors.

An application monitor is also in charge of monitoring the nodes where the application is executed and has to detect node failures. Each application monitor has an heartbeat mechanism to check that the nodes used by the application components are alive.

In the system, the nodes are organized in a structured overlay network. Thus, each node has a location independent name. Thanks to key based routing (KBR), it is possible to communicate with any node if its logical name is known. Furthermore, a distributed hash table (DHT) service is built on the top of the structured overlay. This DHT service has several channels, particularly an application monitor channel. As a consequence, an application monitor can be reached from anywhere in the network only by knowing its logical name. The logical name is obtained during the submission of the application, is unique and is indepen-

dent of the application submission node. The overlay network features of the system are described in [8].

4.1.2 Component Monitor

There is a component monitor on each node of the grid. The function of the component monitor is to monitor all application components executed on the local node. So a component monitor is not dedicated to an application. The data collected by the component monitor are stored on the local node in a storage unit associated to the monitor. The storage is local to limit network communications because monitoring data are typically more frequently updated than read. Moreover, storing the data locally distributes the amount of monitoring data to store among all the nodes of the grid. The component monitor communicates with the application monitor through the communication model described in Section 4.2.

4.2 Communication Model

Network is a critical resource as components of distributed applications need to communicate with each other. Thus, the network bandwidth used by grid services must be minimal.

Notification To get information about a component, the application monitor could periodically send a request to the concerned component monitor. But this method induces useless communications if the information on a component has not changed since the last update. To reduce these useless communications, we introduce a communication model based on notifications. In this model, communications are initiated by the component monitors. When a component monitor detects an important change in the state of a component, it notifies this change to the application monitor and through the same message, it updates also the resource consumption of the component. The important changes that are notified are the beginning and the end of the execution and the application failures.

Query/Response Query/response is the communication model used between users and application monitors. The user can send a request to the application monitor that monitors her application. The application monitor forwards this request to the concerned component monitors to get the updated information about resources consumption of each application component and then sends the response to the user.

4.3 High Availability of the Service

GAMoSe must ensure a constant quality of monitoring despite the reconfigurations occurring in the grid. Thus,the service must be resilient to nodes failures induced by the dynamic behavior of the grid.

First of all, a failure may occur on a node hosting an application component. The failure implies losses of information concerning resources utilization. Such losses are not an issue, if we assume that a grid user should not pay for the utilization of resources that sustained failures. Otherwise, the losses can be avoided by regularly sending utilization reports to the application monitor. In this case, a trade off must be found between the frequency of sending local information to the application monitor and the bandwidth cost.

Then, a failure may occur on a node hosting an application monitor. Because the application monitor is a key component for GAMoSe, it must live through failures and thus must be replicated. As stated in Section 4.1.1, the application monitor is based on a structured overlay network with a DHT. A mechanism of group communication built on top of the structured overlay allows to send messages to a group of replicas. The messages sent to an application monitor are atomically multicast to the group of replicas of the application monitor. The application monitors are actively replicated. The consistency of the replicas is ensured since the application monitors rely on a deterministic automaton. The nodes hosting the replicas are automatically chosen using the DHT service. As a consequence, with the help of group communications and duplication facilities of the DHT, the application monitors are highly available. More details on the self-healing properties of the application monitor are presented in [11].

5 Implementation

We have implemented a prototype of GAMoSe and integrated it into Vigne. In this section, we first briefly present this prototype. Then we focus on monitoring of application components. We describe the monitoring mechanisms that we use according to the local OS.

5.1 Integration to Vigne

Vigne [8, 11] is a GOS for large scale Grids with heterogeneous and unreliable nodes. Vigne is based on a distributed architecture providing self-healing capabilities.

Vigne is designed to provide a simplified view of a grid for users. Vigne provides services like resource allocation or application management. All these services are built on top of structured and unstructured peer-to-peer overlays. Vigne provides a simple interface for users to enable them to submit jobs and to get information on the job they have submitted. Vigne does not require root privilege to be installed and does not rely on any other grid middleware.

We have integrated our service into Vigne so that users can get monitoring information on the applications through the user interface of Vigne. Thanks to the structured peer-to-peer overlay and KBR, we have been able to easily locate application monitors and to establish communications between application monitors and component monitors.

5.2 Monitoring of Application Components

The information on application components must be as precise as enabled by the local system. In this section, we detail monitoring mechanisms used by the component monitors on clusters with batch schedulers and on Linux nodes.

5.2.1 With Batch Schedulers

An application component is submitted as one job to a batch scheduler. Every batch scheduler provides information on jobs. They give the state of the jobs, including fail state, and most of them also give data on resource utilization. So, the only work for the component monitor is to retrieve the information provided by the batch scheduler. Due to DRMAA [12], we can have the same interface for every batch scheduler and also get uniform information.

Many batch schedulers only monitor the first process of a job. So the monitoring can be inaccurate, especially for failure detection, if application components create new processes dynamically. However, we do not want to modify batch schedulers to improve monitoring because of the constraint of transparency expressed in Section 3.3.

5.2.2 On Linux Nodes

On Linux nodes, i.e. Linux PC or SSI clusters, the component monitor of the node must know at any time the identity of each process belonging to an application component to be able to give accurate information on resource utilization or to detect failures. The problem is that, from an external point of view, creation and termination of processes are not governed by any rule. This means that new processes can be created or terminated at any time during the execution of the application component.

Using the process group ID of Unix systems can be a simple solution to identify every process of an application component but this solution is not reliable because programmers can change the process group ID of a process with the `setpgid` system call.

By intercepting the systems calls implied in process creation and termination, we can get the information we need for process monitoring:

`fork` is called to create a child process. Intercepting `fork` enables to know the new processes created.

`exit` terminates the current process. Intercepting `exit` enables to know the processes terminating.

`waitpid` waits for the state of a child process to change, i.e. the child process has terminated or the child process was stopped by a signal. Intercepting `waitpid` enables to know processes ending and how they end.

Using `LD_PRELOAD`, we wrap the `libc` functions corresponding to these three system calls with monitoring code. This code transfers useful information to the component monitor when one of these system calls is used by an application component. We can see this mechanism as the generation of an event on the execution of these system calls. Thus the component monitor is aware of each dynamic process creation and can detect when and how processes terminate. The main limitation of this solution is that `LD_PRELOAD` can not be used with statically linked applications.

As `LD_PRELOAD` is set just before executing an application component, our code is used only by the processes of the application component. So the parent process of the first process of an application component does not use the wrapped functions. `fork` and `waitpid` are not intercepted for the first process since they are called by its parent process: creation and failure of this process are not detected. To overcome this problem, we use a `start process`. The `start process` fork to execute an application component. `LD_PRELOAD` is set before executing the `start process` so that `fork` and `waitpid` can be intercepted for the first process of the application component.

6 Evaluation

The main aspect we want to evaluate is the impact of GAMoSe on performances since we target HPC. Evaluations focus on Linux nodes since the role of the component monitor is very limited with batch schedulers.

We first measure the impact of system calls wrapping on application execution time. Then we present two experiments made on the Grid'5000 testbed to evaluate the bandwidth consumed by GAMoSe. The aim of the first experiment is to determine if the cost of the communication between GAMoSe entities can limit the scalability of the service. In the second experiment, we study the behavior of GAMoSe when failures occur.

6.1 Influence of System Calls Interception on Execution Time

The wrapping of `libc` functions with `LD_PRELOAD` used to intercept system calls can induce an overhead on the applications execution time. We evaluate this overhead. Our measurements are made on a workstation with a 3.6 GHz Pentium 4 processor and a 2.6.14 Linux system.

We execute an application which creates 100 processes. Each of these processes terminates with `exit` and is waited with `waitpid`. Each process simulates a computation time with a sleep of 100 seconds. We measure the execution time of the application with and without the wrapping of `libc` functions.

The standard execution time of the application is 100.415 seconds and the execution time with wrapped functions is 100.482, i.e. an overhead of only 0.07%. We can conclude that the overhead induced by system calls interception is negligible.

6.2 Communication Cost

The cost of the communications between the entities of GAMoSe could be an issue for its scalability. To evaluate this point, we make an experiment on the Grid'5000 testbed. Vigne is deployed on five Grid'5000 sites. The application we use for the experiment is composed of three application components. These application components need between 20 and 100 seconds to execute. Component monitors have to notify to application monitors the beginning and the termination of the application components, as well as their resource consumption.

For this experiment, the grid includes 396 nodes. We make 4 submission cycles. In the first cycle, we submit 128 applications and we wait for all of them to be finished. Then in each cycle, the number of applications submitted is doubled.

Figure 3 depicts the amount of data exchanged between application monitors and component monitors with respect to the number of applications to monitor. The amount of data sent grows almost linearly with the number of applications. Furthermore, when GAMoSe has to monitor 1028 applications, each composed of three applications components, only 1008 KB are exchanged between the different entities of the service. These results prove the scalability of GAMoSe.

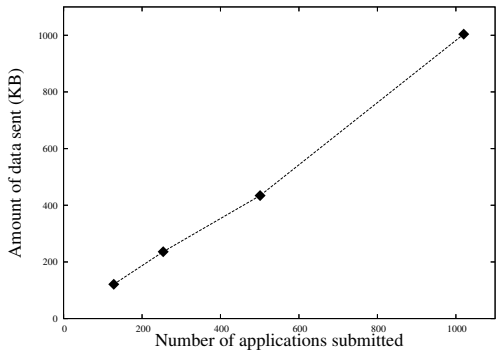

Figure 3. Amount of data exchanged between GAMoSe entities

6.3 Experiment With Failures

To see if GAMoSe is able to deal effectively with failures, we make a second experiment on the Grid'5000 testbed. This time, the grid is composed of 264 nodes and we submit 264 applications. The application submitted is the same as in the first experiment. To simulate failures, we kill some application processes randomly: every second we select one node of the grid and kill all the application components executing on this node. When the failure of an application component is detected, it is notified to Vigne by the application monitor and Vigne restart the execution of the component on another node.

Despite 344 failures of application components detected, every application have managed to finish their execution at the end of the experiment. The total amount of data exchanged between GAMoSe entities is 433 KB. We can compare this value with the results of the first experiment: for 256 applications, 236 KB had been exchanged. Since the failure rate in this experiment is very high, we can conclude that the additional communication cost induced by failures remains reasonable.

7 Practical Examples

GAMoSe provides the grid system with the ability to harvest application executions. In this section, we present through three examples the benefits for grid users to use GAMoSe.

7.1 Reliable Application Execution

GAMoSe provides two mechanisms to execute applications reliably. Indeed, it handles coarse-grained failures that are failures of nodes and it handles fine-grained failures that are process failures.

In case of a node failure, the application monitor is able to detect it (see Section 4.1.1). With this information, the GOS can apply a fault tolerance policy like restarting the application component on another node from scratch or from a checkpoint.

Several circumstances may lead to failures in application executions. Some issues are induced by errors in the application code so nothing can be done to achieve a correct execution. However, some issues are induced by the execution context. For instance, if an application uses a shared `/tmp` directory that becomes full, the application is killed if it tries to write a file. Thus, if this application is restarted to another node, its execution is likely to be correct.

The component monitor (see Section 4.1.2) is able to deal with this kind of failure. Indeed, it monitors the execution of every application processes and detects any abnormal termination. Thus, on detecting an abnormal termination, the component monitor notifies the application monitor that will make this information available for the GOS. Then the GOS will be able to apply a fault-tolerance policy as described above.

7.2 Problem Diagnostics

In the event of an application failure, some issues depend on the execution context and some other depend on errors in the code. In this latter case, the component monitor is useful because it can detect the reason for the failure, for instance, the signal sent to the application or the error code. Then, the grid user can be informed of the reason for the failure and possibly fix the code of her application.

7.3 Accurately Account the Resources Utilization

In grids where resources are leased to consumers, the accuracy of the resource utilization measure is a necessary condition for fairness. In this way, the component monitor provides fine-grained information about resource utilization. Indeed, by monitoring each process creation and termination, GAMoSe measures accurately the CPU and memory resources used by an application. If Service Level Agreement (SLA) are used in the grid, those measurements can help services that have to control the respect of agreements [9].

8 Related Work

The survey [16] presents an overview of grid monitoring systems. It identifies scalability as the main requirement for monitoring systems. That is why most services are based on a distributed architecture. As our service is dedicated to application monitoring, we have created application monitors which enable merging of data on a per application basis. This cannot be done in a generic monitoring architecture like the Grid Monitoring Architecture (GMA) [14] for instance. In this case, the user has to develop her own mechanism to aggregate information. Three models of interactions between producers, entities that produce monitoring data like the component monitors, and consumers, entities that receive monitoring data like the application monitors, are supported by GMA: publish/subscribe, query/response and notification. We have used two of these communication models: notification and query/response. Thus the network bandwidth we use for data collection is optimized in comparison with the periodical poll used in Ganglia [10] or in the XtremWeb grid middleware [3].

The mechanisms used in existing grid systems to monitor applications executed on Linux nodes are not accurate or not transparent. The transparent solution which is mainly used is to scan the `/proc` pseudo file system to get information on the first process created when executing the application. End of the application is detected when the process disappears from `/proc`. Although this solution does not allow to monitor processes created dynamically and to detect failures, it is the solution used by the Globus Grid Resource Allocation and Management protocol (GRAM) [4], on which Condor-G [7] is based, by Nimrod/G [2], by XtremWeb and by Legion [6]. The OCM-G [1] monitoring system or Autopilot [15] enable to accurately monitor applications in grid systems but, in contrast with our approach, they are not transparent for applications since the application code needs to be instrumented.

9 Conclusion

We have described in this paper GAMoSe, a grid application monitoring service. This highly decentralized service has been designed to scale with the number of applications executed in a large grid and with the number of processes and components in an application. GAMoSe tolerates node joins and leaves and is highly available despite node or communication link failures. GAMoSe provides dependable information on application, i.e. on failure and resource utilization, especially on Linux nodes where it is able to monitor

every process of an application component even dynamically created processes. In contrast with other monitoring systems described in the literature, the accurate monitoring mechanisms of GAMoSe are completely transparent for the applications.

Due to the accurate information it supplies, GAMoSe facilitates the use of grids. It enables to execute applications reliably since node and process failures are detected. In the event of a process failure, GAMoSe reports the cause of the terminaison. The user can thus diagnose the reason for the failure and know if the error comes from the application code for instance. Finally, providing accurate information on resource utilization is very important if grid resources are billed to users.

GAMoSe cannot accurately monitor statically linked applications. It would be possible to monitor those applications if system calls were intercepted at system level. This issue will be further investigated in the framework of the XtreemOS European project [13] targeting the design and implementation of a Linux-based Grid operating system with native support for virtual organizations.

10 Acknowledgments

We want to acknowledge Louis Rilling for his works on the conception and the implementation of several services of Vigne on which this work is based. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https:// www.grid5000.fr)

References

[1] B. Balis, M. Bubak, W. Funika, T. Szepieniec, R. Wismüller, and M. Radecki. Monitoring Grid Applications with Grid-Enabled OMIS Monitor. In *European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 230–239. Springer, 2003.

[2] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. *The 4th International Conference on High-Performance Computing in the Asia-Pacific Region (HPC Asia 2000)*, May 2000.

[3] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Nri, and O. Lodygensky. Computing on large-scale distributed systems: Xtrem Web architecture, programming models, security, tests and convergence with grid. *Future Generation Computing System*, 21(3):417–437, 2005.

[4] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, London, UK, 1998. Springer-Verlag.

[5] I. Foster and C. Kesselman. Computational grids. *The grid: blueprint for a new computing infrastructure*, pages 15–51, 1999.

[6] A. Grimshaw, W. Wulf, and The Legion Team. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

[7] I. Foster J. Frey, T. Tannenbaum, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Cluster Computing*, 5(3):237-246, 2002.

[8] E. Jeanvoine, L. Rilling, C. Morin, and D. Leprince. Using overlay networks to build operating system services for large scale grids. In *Proceedings of the 5th International Symposium on Parallel and Distributed Computing (ISPDC 2006)*, Timisoara, Romania, July 2006.

[9] A. Keller, G. Kar, H. Ludwig, A. Dan, and J.L. Hellerstein. Managing Dynamic Services: a Contract Based Approach to a Conceptual Architecture. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, pages 513–528, 2002.

[10] M.L. Massie, B.N. Chun, and D.E. Culler. Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30:817–840, 2004.

[11] L. Rilling. Vigne: Towards a Self-Healing Grid Operating System. In *Proceedings of Euro-Par 2006*, volume 4128 of *Lecture Notes in Computer Science*, pages 437–447, Dresden, Germany, August 2006. Springer.

[12] The Global Grid Forum. Distributed Resource Management Application API. https://forge.gridforum.org/projects/drmaa-wg.

[13] The XtreemOS Project. http://www.xtreemos.eu.

[14] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. Technical report, GGF Technical Report GFD-I.7, January 2002.

[15] J. S. Vetter and D. A. Reed. Real-Time Performance Monitoring, Adaptive Control, and Interactive Steering of Computational Grids. *International Journal of High Performance Computing Application*, 14(4):357–366, 2000.

[16] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163–188, 2005.