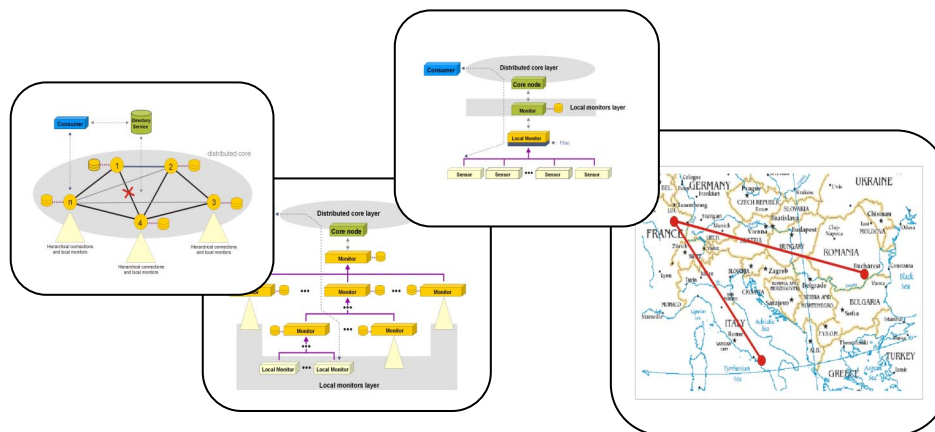


MONITORING METHODS AND PERFORMANCE ANALYSIS CRITERIA IN GRIDS FOR PHYSICAL PROCESSES CONTROL



Author:

Alexandru IOSUP

Supervisors:

SUPÉLEC

Ph.D. Prof. Eng. **Stéphane VIALLE**

U.P.B.

Ph.D. Prof. Eng. **Nicolae ȚĂPUȘ**

2004

U.P.B.
Universitatea Politehnica Bucureşti

SUPÉLEC
École Supérieure d'Electricité

MONITORING METHODS AND PERFORMANCE ANALYSIS CRITERIA IN GRIDS FOR PHYSICAL PROCESSES CONTROL

Alexandru IOSUP

Supervisors:

SUPÉLEC Stéphane VIALLE
Ph.D. Prof. Eng.

U.P.B. Nicolae ȚĂPUȘ
Ph.D. Prof. Eng.

Preliminaries of this work were presented Friday, the 28th of May, in Metz, France.
This work was presented Sunday, the 4th of July, in Bucharest, Romania, in order to
fulfill the requirements for obtaining the title of Master of Science in Computer Science
from the Politehnica University of Bucharest.

This research has been supported in part by the Region Lorraine (France) and the
ACI-GRID ARGE project (France). Excerpts from the Grid computing survey chapter
have been included in the INFOSOC MON-I project proposal (Romania).

Abstract

Monitoring applications in complex environments such as control Grids is not a trivial task. Techniques for monitoring cluster environments already exist and methods for monitoring computing Grids have already been developed. Yet, it has not been established whether any of these two, or a combination, can be used in the case of control Grids, especially with physical processes with soft time constraints.

This work addresses monitoring Grids for physical processes control issues, key issues being that:

- We survey the current Grid monitoring tasks, techniques, and trends;
- We present a monitoring tools taxonomy oriented towards control Grids;
- We introduce a novel monitoring architecture, named Toytle.

The main achievements are the establishment of a monitoring tools taxonomy, based on structural properties, the design of a novel monitoring architecture for control Grid environments, and the selection of monitoring criteria for a robot control Grid.

Keywords Toytle; monitoring architectures; control Grids monitoring.

*For Ana Lucia,
with all my love.*

Chapter i

Acknowledgements

I can hardly imagine this project getting finished without the help of so many people. People I know, people I worked with, people I made friends with, they all contributed to my efforts to conclude on this project. It is unlikely that the space would allow me to even mention them all, so this canvas will host just a few of my special thanks.

First of all, many thanks go to my two project coordinators:

Stéphane Vialle, professor and researcher at Supelec, for letting me experiment with ideas and for giving me great advice on how to present my findings. Being part of the *ERSDIP* research group was a rewarding adventure, both in professional and personal planes. Thank you, Stéph!

Nicolae Țăpuș, professor and researcher, head of the Computer Science Dept. at the Politehnica University of Bucharest, for his witty advice and for sharing glimpses of his research experience with me. The research group around him was both challenging and resourceful, and it is him who got me searching for a research career in the first place. Knowing him for the past four years has been a great privilege to me.

Their combined efforts assertively improved the results of my work, in general, and this thesis, in particular. The remaining set of errs and mistakes still is, however, the sole property of the author. Wish it was the other way round.

By establishing and maintaining an exchange protocol between the Politehnica University of Bucharest and Supelec, two other persons eased my work considerably. They are mr. **Constantin Iliescu**, professor at the Politehnica University of Bucharest, and mr. **Yves Tanguy**, research coordinator at Supelec. Thank you very much! This really could not have happened without your kind help.

I would also like to thank a person with whom I've had bennefic and enjoyable discussions: prof. **Irina Athanasiu**, from the Politehnica University of Bucharest. Many of her questions proved to be interesting advice and many of her observations words of wisdom, for which I cannot thank enough.

I could not omit people from Supelec, Metz, who made my life easier (thank you **Hervé, Patrick, Cristian, Radu, Anca, Laurent, Rashid, Jacques, Gilles, Eric** and **Claudine**). Special thanks should also go to **Fabrice**, for having the time and willingness to help me in numerous occasions, some very important in my work (do you remember the UDP 500 port, Fabrice?).

(Almost) in the end, a mention to my friends. It was their presence which kept me smiling, to say the least. The necessary space for mentioning them, even briefly, is missing, but they are in my mind and heart. However, I can mention some of their efforts: though my time away was not long, they insisted on cheering me up every single day (sometimes even more). My writing skills on French keyboards also developed during this time (no relation to previous phrases, whatsoever)...

Last, but not least, I would like to thank to my family, all of them very encouraging and caring. Thank you **Ana Lucia, Andrei, Ana-Maria, Andrea** and **Olga**.

Alexandru Iosup
Bucharest, 2004.

Contents

Abstract	ii
i Acknowledgements	iv
ii Contents	vii
iii List of Figures	viii
1 Introduction	1
1.1 Motivations	1
1.2 Project framework and goals	1
1.3 Difficulties	2
1.4 Executive summary	3
2 Grid computing	4
2.1 A brief history of Grid	4
2.2 Two Grid views: research and industry	5
2.3 Different kinds of Grid	7
2.4 DIET, a generic NES environment	8
2.5 Physical processes control Grid environments	9
3 Grid monitoring	10
3.1 Monitoring tasks	10
3.1.1 General monitoring tasks	11
3.1.2 Grid-only monitoring tasks	12
3.2 Monitoring techniques	15
3.2.1 Characteristics	16
3.2.2 Measurements	16
3.2.3 Hierarchies of characteristics	17
3.3 Monitoring tools structure	21
3.3.1 Overview	21
3.3.2 Flat, dispersed	23
3.3.3 Flat, connected	24
3.3.4 Flat, near cross-bar	24
3.3.5 Hierarchical, 1-level	26
3.3.6 Hierarchical, N-level	26
3.3.7 Hybrid connected core/N-level hierarchical connections	28

3.3.8	Hybrid near cross-bar core/N-level hierarchical connections	29
3.3.9	Related work	30
3.4	Monitoring trends and downsides	31
3.4.1	Monitoring trends	31
3.4.2	Monitoring downsides	31
3.5	Monitoring control Grids	32
3.6	Summary	33
4	Toytle: a monitoring architecture for control Grids	34
4.1	Premise	34
4.2	Toytle components	34
4.2.1	The distributed core layer	35
4.2.2	The hierarchical connections layer	37
4.2.3	The local monitors layer	37
4.3	Target applications	39
4.4	Toward implementing the Toytle architecture	40
4.4.1	The <i>inexpensive</i> approach	40
4.4.2	Experiments toward implementing local monitors layer	40
4.4.3	Experiments toward implementing hierarchical connections layer	42
4.4.4	Summary	42
4.5	Related work	43
5	Toytle test case: a mobile robotic application	45
5.1	Introduction	45
5.2	Project history	45
5.3	Robotic application general framework	45
5.4	Robotic system	46
5.4.1	Hardware	46
5.4.2	Software	47
5.5	Selection of monitoring criteria for a robot control Grid	49
5.5.1	Monitoring requirements for the robot control Grid	49
5.5.2	Host characteristics	49
5.5.3	Network characteristics	49
5.5.4	Middleware characteristics	49
5.5.5	User applications characteristics	50
5.6	Toytle deployment in the test-case environment	51
5.7	Summary	52
6	Conclusions and future work	53
6.1	Conclusions	53
6.2	Future work	54
	Index	56
	Bibliography	56
	A Memos, reports, and dissemination	61

List of Figures

2.1	DIET components	9
3.1	Butterfly Grid architecture	11
3.2	The GGF Grid Monitoring Architecture	12
3.3	The GGF Grid Monitoring Architecture producer/consumer pipe	13
3.4	Grid characteristics hierarchy top level	18
3.5	Host characteristics hierarchy	19
3.6	Network characteristics hierarchy	20
3.7	Middleware characteristics hierarchy	21
3.8	User applications characteristics hierarchy	22
3.9	Monitoring tools, flat and dispersed structure	23
3.10	Monitoring tools, flat and connected structure	24
3.11	Monitoring tools, flat and near cross-bar connection structure	25
3.12	Monitoring tools, hierarchical, 1-level structure	26
3.13	Monitoring tools, hierarchical, n-level structure	27
3.14	Monitoring tools, hybrid: connected core, N-level hierarchical connections structure	28
3.15	Monitoring tools, hybrid: near cross-bar core, N-level hierarchical connections structure	30
4.1	Toytle architecture	35
4.2	Toytle architecture: distributed core layer	36
4.3	Toytle architecture: hierarchical connections layer	38
4.4	Toytle architecture: local monitors layer	39
4.5	Ganglia vs. NET-SNMP speed comparison	41
4.6	Ganglia vs. NET-SNMP generated traffic comparison	41
5.1	Our complete mobile robotics application	46
5.2	The Grid infrastructure deployment	47
5.3	The test case Grid software architecture	48
5.4	Selected characteristics hierarchies for robot control Grids	50
5.5	Toytle deployment within the test case environment	51

Chapter 1

Introduction

This work looks into the field of monitoring, in general, with special emphasis on Grid monitoring, all in the context of dynamic physical applications, such as mobile robotics. This chapter discusses the significance of this field of research and presents the thesis statement.

1.1 Motivations

Controlling physical processes with software applications requires a great amount of monitoring, if only for the very nature of these applications (to respond to the state changes of the physical applications and to direct them toward a user-defined goal). Integrating control software in Grid environments adds numerous Grid monitoring tasks, like tracing the application execution on the Grid infrastructure and dynamical observation of the system's status.

With these tasks in mind, there are numerous monitoring tools for clusters and/or Grid environments, but it has not been established whether these tools can support adequate monitoring in the case of control Grids environments and applications, particularly in the care of robot control Grids.

Establishing the needs of control Grids monitoring must be done in a rigorous way, given that such projects are often med- and long- term interdisciplinary projects, with participants coming from different organizations. Focus should be put, of course, on data sharing and data visualization protocols, but automatic installation and maintenance should not be disregarded as well.

1.2 Project framework and goals

This project focuses on proposing a monitoring architecture for control Grids. The main target of this architecture is to be able to accurately monitor Grid applications for controlling dynamic physical processes.

We have chosen a mobile robotic application, viewed as an example of soft time-constrained application, as the test case for monitoring tools. The application has computing intensive

components, such as the self-localization, time-critical components, such as the obstacle avoidance, and communication-intensive components, such as the navigation and self-localization, and has been integrated in a control Grid environment [60].

Thesis statement Monitoring control applications for dynamic physical processes in Grid environments can be done only through the use of specific monitoring architectures and by the use of adequate monitoring criteria.

Problem(s) statement

1. The specific requirements of monitoring physical processes control applications in Grid environments must be established.
2. The ability of nowadays cluster and Grid monitoring tools, either separate or in combinations, to fulfill the requirements established at the previous step must be inquired. For that, a monitoring tools taxonomy should be developed.
3. Possibly, specific architectures for monitoring physical processes control applications in Grid environments must be designed, implemented and tested.

1.3 Difficulties

This project developed in a rather sinuous way. Numerous difficulties, both research and engineering, not to mention administrative, occurred and have been dealt with, only to give way to others. Many of them, from the first two categories, are presented in the following list:

Research tasks

- Getting accustomed with Grid computing history;
- Checking current Grid computing research and industry trends;
- Having a glimpse on GridRPC, with DIET as an exponent;
- Performing a survey on monitoring tasks, techniques, and trends;
- Proposing a taxonomy of monitoring tools, based on their hierarchical attributes;
- Performing a overview on monitoring trends;
- Proposing a monitoring architecture, named Toytle, suitable for monitoring dynamic physical processes in control Grids environments (and even more);
- Designing ways to use the proposed monitoring system for a real application (selecting a set of monitoring criteria for a robot control Grid and proposing a deployment strategy);

Engineering tasks

- Evaluating three of the most used monitoring tools nowadays: NET-SNMP, Ganglia and MonALISA.

1.4 Executive summary

This chapter (**Chapter 1**) has introduced the goal of this report to the reader, while pointing out some of the difficulties encountered in the fulfilling of this goal. The rest of this report is structured as follows:

Chapter 2 proposes to the user an introduction to the key concepts and features of Grid computing. It also overviews the characteristics of Grids used to control dynamic physical processes.

Chapter 3 performs a state-of-the-art survey in the domain of Grid monitoring. For this, tasks, techniques and trends are thoroughly reviewed. A Grid monitoring tools taxonomy, based on two structural attributes of monitoring system and focused on issues concerning control Grids monitoring, is also presented in this chapter.

Chapter 4 introduces a novel monitoring architecture for dynamic physical processes control Grids, named Toytle.

Chapter 5 presents a test case for the Toytle architecture, as well as a selection of monitoring criteria for a robot control Grid and a deployment strategy for a Toytle implementation in the test case environment.

Chapter 6 presents our conclusions and predicted future work.

Reading guide Following the chapters ordering ensures that the reader is faced with all intended information, in the order designated by the author. However, each chapter is, presumably, readable independently of the others. For the unaccustomed with the field of Grid monitoring, chapters 2 and 3 are required for the understanding of this report. For persons interested in the practical aspects of this work, chapter 5 proposes a real test case. For reviewers, the abstract, followed by the introductory chapter (chapter 1) and the final chapter (chapter 6), gives a first glance to the problems treated in this work.

Chapter 2

Grid computing

This chapter aims at introducing Grid computing to the reader. The first two sections, *A brief history of Grid* and *Two Grid views: research and industry* present the very basics of this research discipline. The other two, *GridRPC* and *Physical processes control Grid*, offer brief insights on two Grid computing aspects relevant to our project's goals (see section 1.2 for more info on the latter).

It must be noted that this chapter is a continuation of the author's previous work on the matter, recorded within his diploma project report [37]. Since then, another Grid computing generation has emerged (section 2.1), the Grid views of Ian Foster and Andrew Grimshaw (section 2.2) have received big companies' acceptance and support, and DIET has been used as a GridRPC environment (section 2.5) for a mobile robotics application [60]. Long story short: in only one year, many things have happened in the Grid computing world, and this chapter needs to correct and improve on the one previously written.

2.1 A brief history of Grid

The idea of Grid is linked with the evolution of Internet [2]. It is generally accepted that Grid is to computing what Internet was (and still is) for information: a mean to make it accessible to anyone, under various presentation conditions. Like in the Internet's case, the dynamic field of Grid computing grew rapidly, and has already produced three generations in fifteen years [17].

The *zeroth* generation (~1970) time-slice computers are cited as the first attempts to what we now call Grid computing [66]. Back then, computing was more about having multiple users using a mainframe, than having multiple users using multiple machines, but the concept of sharing spare cycles is what inspired most of the initial work in the Grid computing research. Therefore, the zeroth generation is worth mentioning for bringing forward some of the important ideas in current Grid research.

The *first* generation (1990-1995) contains the forerunners of Grid computing, embodied in projects like I-WAY [18] and FAFNER. Both projects were examples of *metacomputing*, that is, exploitation of computational resources within a national network ¹. The

¹a nation with the size of a continent

metacomputing setting involved the use of *metacomputers*, networks of heterogeneous, computational resources linked by software such that they can be used similarly to a personal computer in terms of easiness. While I-WAY was set to use supercomputers, FAFNER involved anonymous personal computers with slow computing capabilities. Both were demonstrated at the *Supercomputing'95* conference.

The second generation (1996-2003) The two leading projects of the previous generation evolved into what we call today middleware Grid computing: FAFNER into JXTA (1998) and Seti@home and I-WAY into Globus (1996) and Legion (1997). *Middleware* is generally considered to be the layer of software standing between the operating system and the user applications, providing a variety of services required by an application to function correctly [7]. As several middleware systems were made publicly available, integrated systems emerged. An example is the European DataGrid project conducted by CERN. Issues like heterogeneity, scalability and adaptability are tackled within this generation, in an attempt to establish general standards.

The third generation (2003-today) A *third generation* is currently taking over, with focus on distributed global collaboration, a service-oriented approach and information layer crystallization [2], all embarked in the flagship Grid distribution, the Globus Toolkit (now version 3).

It is a fact that we are currently dealing with the realities of the third generation Grids, but many people are already wondering about the future generation of Grid. Issues like the convergence of Grid computing and major computing technologies (e.g. *peer-to-peer* and Grid computing [21]) are believed to be instrumental into building next-generation Grids.

2.2 Two Grid views: research and industry

There are many different definitions currently available in the Grid research community. Problems of these definitions lie in the nearness to *distributed systems*, *metacomputing*, *networked systems* and *peer-to-peer computing* definitions.

Two definitions dealing with Grid systems stand apart:

- Ian Foster's *3-point checklist* definition [20]:

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 nontrivial qualities of service.

- Andrew Grimshaw's *Grid hardware and user perspective* definition: (the descriptions were combined in phrases instead of point lists as used in mr. Grimshaw's article [28]):

From a *hardware perspective*, a Grid is a collection of distributed resources connected by a network.

From a *user perspective* a Grid gathers together resources (e.g., CPU, data, and applications) and makes them accessible in a secure manner to users and applications.

These definitions represent the academic view (Foster) and the market-oriented view (Grimshaw) of the **Grid** phenomenon. Note that these definitions do not deal with the definition of *the Grid* - one **Grid** interconnecting all the others, much like the Internet did with all proprietary local networks -, the final target of the **Grid** systems research. *The Grid* is mentioned in [20], with redefinitions coming from every major company involved in *the Grid* marketing campaign, such as IBM, Sun Microsystems, Hewlett-Packard or Oracle.

Bote-Lorenzo et al. [9] extract 10 main characteristics from the most cited **Grid systems research views**:

1. *large scale system*, dealing with an important number of resources,
2. over a *wide geographical distribution*
3. connecting *heterogenous* software and hardware resources
4. allowing *sharing* between these resources,
5. *multiple administration policies*, such as different security and control requirements,
6. but with *resource coordination* between the different organizations using the Grid.
7. Access to these resources should be transparent, e.g. the user must perceive the Grid a single, virtual computing system,
8. *dependable*, e.g. access falls under many types of Quality of Service rules,
9. *consistent*, e.g. accessing resources should be done according to open, publicly available protocols,
10. and, finally, *pervasive*, e.g. the Grid should be seen as a fail-safe system, with the best possible resource failure recovery.

Major companies seem to have agreed on 5 main issues, in order to promote a **market Grid view**:

1. *Efficient use of already installed computing power*. Over 75% of the total time of the nowadays computers' companies remains unused (IBM suggests that 90% of the servers' and 95% of the PC workstations' time is unexploited [35]). Therefore, near optimal employment is more than necessary, leading to a rapid return of investment (*RoI*).
2. *Ease of resource partitioning* (sharing, scheduling utilization, and proper use). Grid distributions currently focus on offering adequate technological support for resource partitioning. This leads to significant management costs reductions, due to a smaller number of needed administrators and to their required qualification. In addition, ease of resource sharing leads to many more resources getting shared, thus ensuring *service persistence*.
3. *Processing larger data volumes*. The number of Grid resources is much higher than the same for companies, even for large ones. This happens because the total computing power of a number of sharing participants overcomes the sum of computing powers of the same participants, taken as individuals. Securing large volumes of data is also an important issue in this context.

4. *Sharing software licenses.* The acquisition cost of most important commercial software distributions, no matter the area of use, is often of an order of magnitude higher than that of the computing systems actually using it. Grid computing offers ways of sharing the software licenses between large groups of users, with the sole restrictions of having no more simultaneous users than the number of available licenses. Members of virtual organizations can take part in this sharing of license, whether they might be part of a single organization, such as the case of a company with branches, or of different organizations, such as the case of many companies collaborating on a certain project.
5. *Ready-to-use solutions.* Large companies interested in Grid computing have entered the market with a number of *ready-to-use* solutions. Several important commercial areas, like financial services, automotive/aerospace design & assembly lines, governmental storing/database systems, simulations for agricultural chemicals and life sciences, data visualization for petroleum fields, and virtual systems with massive simultaneous uses, may benefit from it.

2.3 Different kinds of Grid

As shown in the previous chapter, within the current context of lacking a globally accepted Grid definition, grouping all kinds of Grid systems into a complete taxonomy is a mere claim. *Krauter et al.* [42] present three Grid types, depending on the resource management systems design objectives: *computational*, *data* and *service* Grids.

Computational Grids are built around the idea of using together many computers, in order to achieve an aggregate computational power that exceeds that of any single machine in the system alone. These systems are further divided into distributed supercomputing and high throughput. For *distributed supercomputing Grids*, applications are redundantly executed on several machines, to reduce the total time and also to ensure reliable execution of a certain job. Highly computationally demanding with reliability issues applications, such as weather modeling or laser simulation are examples of suitable applications for distributed supercomputing Grids. *High throughput Grids* focus on increasing the execution rate for streams of jobs, usually on specialized architectures. Typical applications include those who require same operations to be run over and over, such as parameter range sweep in Monte Carlo simulations.

Data Grid systems provide an infrastructure for all the aspects of distributed high-throughput databases. Typical applications are large-scale data merging or specialized data mining from multiple sources.

Service Grids focus on providing the user services, which would be unavailable on one's regular working environment. Depending on the specific service delivery mode, service Grids span into on-demand, collaborative and multimedia. *On-demand* Grids aggregate, upon request, various resources, for limited periods of time. Visualization systems that increase their image output quality, given that more machines are added, are a typical example. *Collaborative* Grids focus on incarnating the term virtual organizations, i.e. globally spread humans and applications interacting in the same virtual environment. *Multimedia* Grids offer

support for real-time multimedia applications. Quality of service and high-speed streaming are main issues upon building such systems. *Control* Grids allow local and distant users alike to control systems such as mobile robots or other physical processes. Obeying soft and hard time-constraints, as well as offering protection to common system faults, are of most interest in *control Grids*.

2.4 DIET, a generic NES environment

With so many types of Grid computing, and the relative standardization of the physical infrastructure, it has become increasingly important the way actual *ease-of-use* of such systems [50]. Network Enabled Servers (short, NES), provide a solution for this desiderate. The NES paradigm enables writing Grid applications through the use of remote calls to procedures/libraries/programming environments located on the Grid [55]. The ability to use computational servers at distance, through the use of available network connections, is complemented sometimes with fault-tolerance, load-balancing, or advanced user interface design [10].

The *Distributed Interactive Engineering Toolbox* (DIET) project is a generic NES environment (i.e. it is adapted to any kind of problem to which the NES/GridRPC paradigm applies). DIET is based on a Client/Agent/Server paradigm, i.e. the servers register to the agent, which is contacted by the client and, in turn, establishes a connection between the client and the most appropriate server, with CORBA used for communication.

The DIET components are: Clients, Master Agents, Local Agents and Server Daemons. The DIET components can be used to provide a hierarchical agents system (see fig.2.1), with a scattered scheduler. Network Weather Service (NWS) sensors are placed on every node of the hierarchy to collect resource availability status, which are then used by Fast Agent's System Timer (FAST) [58] performance prediction tool and converted into suitable information for the scheduler.

Client A client is any application that uses DIET to access computational resources.

Master Agent (MA) A Master Agent advertises services and is responsible for searching, upon request, the best server that can perform one of its listed services.

Local Agent (LA) Local Agents are low-level agents, with the sole purpose of simplify the master agent's task of selecting the best computational server for a specific request. To dispatch requests, Local Agents contact the directly connected resources or Local Agents under them, issue the request, then select the valid servers and transmit the information above.

Server Daemon (SeD) A Server Daemon encapsulates the services offered by a computational server.

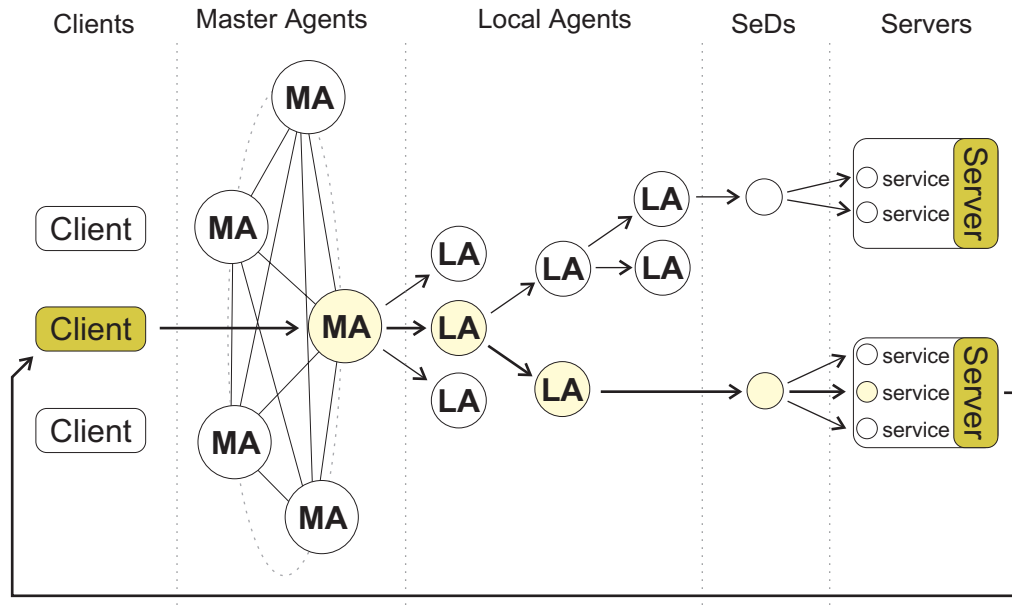


Figure 2.1: The DIET components. The client contacts the master agent (MA), which in turn makes use of the local agents (LA). The local agent contacts a computational server daemon (SeD). A virtual connection is established between the client and the computational server.

2.5 Physical processes control Grid environments

Physical processes control in Grid environments refers to the actual control of physical processes, such as having mobile robots wandering around or detecting intruders through the use of proximity sensors, with the help of software applications running in Grid environments. Such an approach requires that in the Grid environments:

- Programming environments are user-friendly, that is, it allows the user to create powerful applications with as few new coding practices as possible (due to the interdisciplinary nature of such applications);
- Quick response is ensured for control commands, usually below the second and, in general, above a tenth of a second (the average network time between local computers);
- Some level of fault-tolerance is offered transparently to the user, with the hope (often tested empirically) that it will suffice for general control applications.

DIET allows for quickly integrating applications into Grid-like environments through an intuitive, yet powerful API [15]. It's structure is very light (just a few local agents, a master agent and as many services) and its base is a high-speed metacomputing core (CORBA). An API providing fault-tolerance for DIET also exists [60]. In conclusion, DIET is well suited for physical processes control.

Chapter 3

Grid monitoring

3.1 Monitoring tasks

Monitoring distributed systems, be they local clusters or global Grids, serves at only one purpose: to gather relevant data that can be used to make the monitored systems perform better. Data is required for performance analysis, performance tuning, performance prediction, fault detection, fault tolerance, establishing user profiles, accounting, security and scheduling. Of these, the first five are of utmost importance, as they are responsible for much of the possible optimized management of the Grids.

As an example of how monitoring can help, let's take the case of massive multiplayer games over a wide-area Grid [36]. Once the players are authenticated, they are automatically assigned to one of the game servers (see figure 3.1). After logging in, a player must always have visual information about the environment, but this information has to be obtained dynamically from the server. If the player's server gets too crowded, some players must be automatically migrated over the network to another server. Of course, decisions like when the server is too crowded and where to relocate the player are taken by the Grid management system, but, to ensure performance, they are based on local and global monitoring (e.g. how many users are at the moment on the supposedly crowded server and what is the best server to migrate the player to, based on criteria like utilization and distance). If network and CPU load monitoring show that a server performs as if it was crowded, but the number of players or other objects on that server is small, there must be something wrong with that server, and this might range from security breach to physical problem. It is clear that the monitoring system must face a daunting task, having not only supervise the Grid infrastructure (network, CPU load), but also the application data (number of players, database load), all in real-time.

For a Grid monitoring system, there are two types of monitoring tasks: monitoring tasks that are common to all monitoring systems (with some minor particularities, given the Grid environment), the *general monitoring tasks*, and monitoring tasks that are specific to Grid systems monitoring, the *Grid monitoring tasks*.

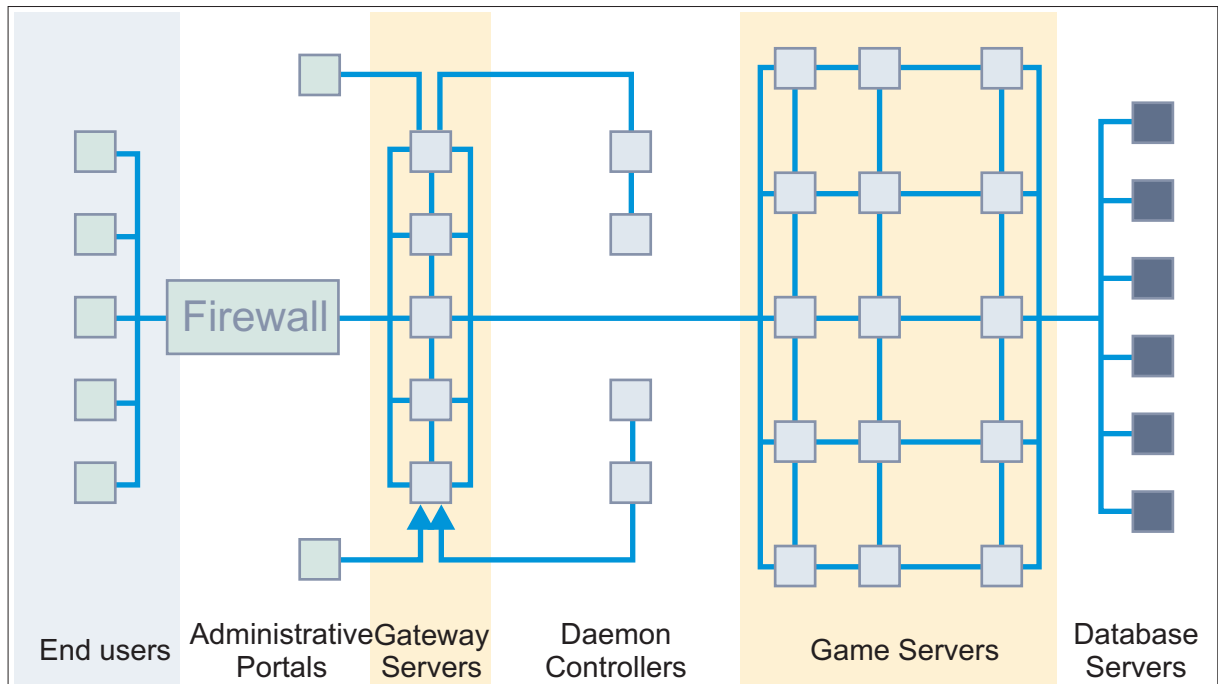


Figure 3.1: The *Butterfly.net* Grid components. Users connect to the Grid via a *firewall*. Administrators can interact with the game world using several *administrative portals*. Then it is relocated to the appropriate server by the *gateway servers*. *Daemon controllers* handle game parts that are not directly influenced by the player. *Game servers* are responsible for offering the player the gameplay experience. *Database servers* store all the needed information.

3.1.1 General monitoring tasks

3.1.1.1 Data acquisition

Gathering data raises many research and engineering problems. Data must be acquired dynamically, on time and with needed accuracy. Data must be relevant for the monitored system and describe it correctly. Probing for data might perturb the monitored system and change the outcome of the monitoring process itself. Once acquired, it is seldom that systems use the results automatically. Results analysis is a difficult and error-prone assignment, and often human intervention is required. Also, after a decision upon the meaning of the monitoring process results is made, it is almost always the humans that enforce the changes for the monitored distributed systems.

3.1.1.2 Visualization

Data visualization greatly facilitates fault detection and intervention on behalf of human system supervisors. As Grid systems tend to increase in size (as more and more clusters and/or other resource are added), monitoring data soon piles up, which makes the effect of visualization even more important. Indeed, without appropriate methods for representing the enormous amount of data in meaningful ways for the human supervisors, the data

could become useless. Visualization techniques refer to virtual organizations representation, resource status (e.g. machine or cluster load and types of load), hierarchical representations of resources, topology-aware displays and much more.

3.1.1.3 Archiving

All the data coming from monitoring various parts of the Grid systems should be also stored for some amount of time, until it either becomes obsolete or it is too much for the storing system, in which case the archiving system provides either a way of adjusting its size (for example by taking up more storage space), or a way of self-controlling the granularity of the stored data (for example by storing only the average of a minute of monitoring data, instead of each single second measurements).

3.1.2 Grid-only monitoring tasks

3.1.2.1 Standardization and openness

It is difficult, if not impossible, to ensure that different organizations install the same software. Therefore, a unique monitoring tool is not an appropriate solution within Grid environments. Instead, we should look for tools that follow an open path in their operation. This path has been laid down by the Global Grid Forum's Grid Performance Working Group, within their work on a general Grid Monitoring Architecture [70] (short, GMA (see fig.3.2)).

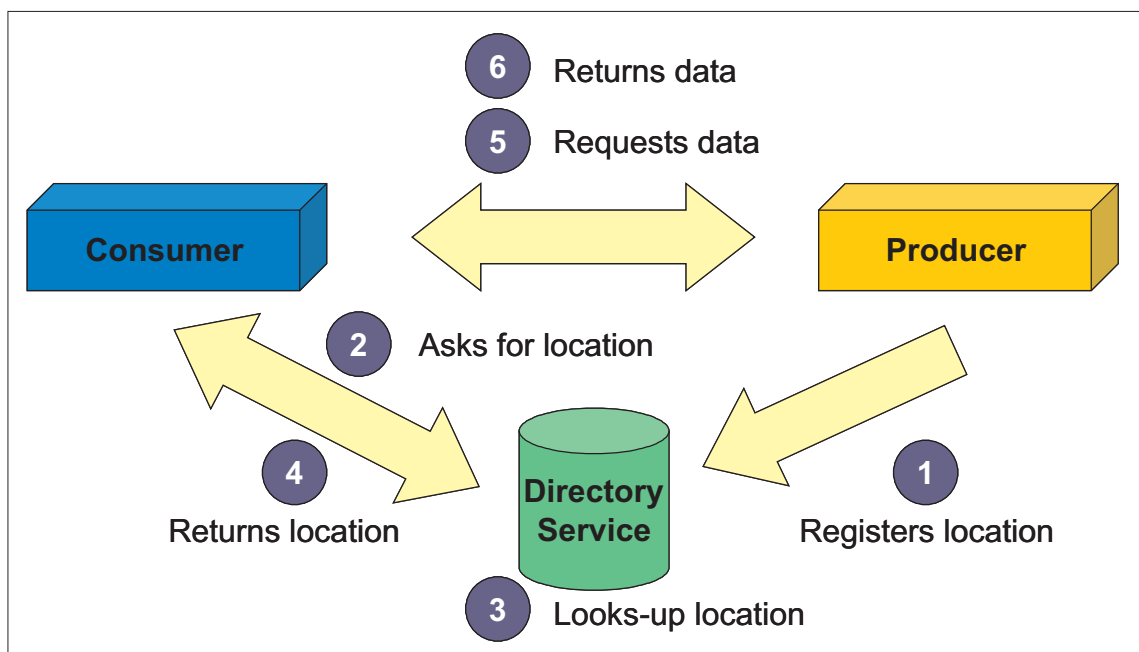


Figure 3.2: The GMA components: producers, consumers and directory services.

The GMA components are the producers, the consumers and the directory service.

Producers are responsible for making data available. In order to do so, producers acquire event data from sensors and wait for consumers to register.

Consumers are responsible for requesting and/or accepting data. Data is obtained from producers. Consumers have no direct contact with producers until they obtain the producer's location within the Grid framework.

Directory services are used to publish information about the event data available and on which producers can be contacted to obtain specific types of data. The directory services also have the means to advertise the event types that can be monitored.

Several additional components may be added / defined, such as event gateways, producer/consumer pipes (see fig.3.3) or repositories (archives).

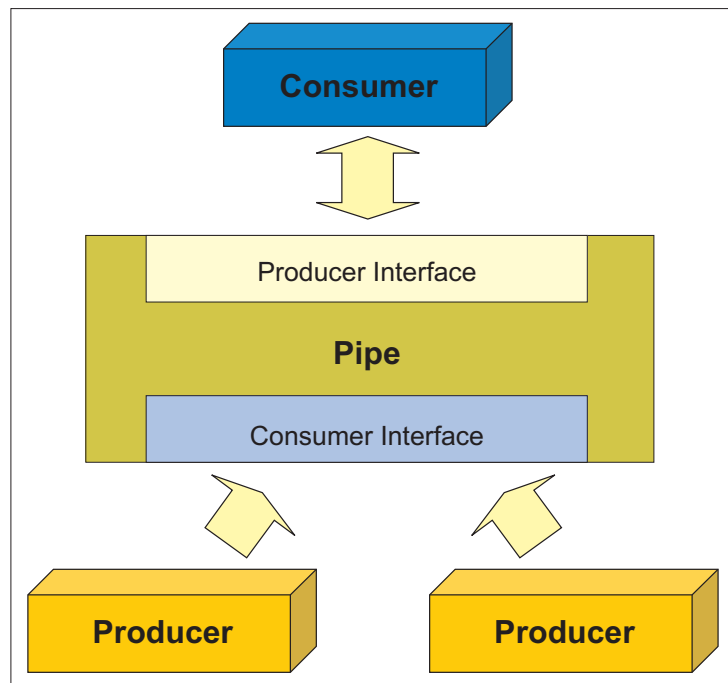


Figure 3.3: A producer/consumer pipe can be obtained by coupling the consumer interface of a monitoring service with a producer interface of the same monitoring service. This way, data from multiple sources can be gathered, filtered and offered transparently to other consumers.

The producer/consumer/directory service architecture is similar to the client/server/agent architecture described in subsection 2.5. For consumers, the process of obtaining data includes 6 steps (see fig.3.2). When the monitoring system is set-up, producers register their location and capabilities with the directory service (*step 1*). Through the directory services, consumers can locate producers that can offer them specific information. Consumers ask for specific types of data (*step 2*) and directory services look up for the adequate producer's location (*step 3*), which they return to consumers (*step 4*). Upon success (an appropriate producer has been

located within the directory service), consumers may directly communicate with the producer. They request information (*step 5*) and retrieve it (*step 6*) from the producer.

It must be noted that producers may have any internal architecture as needed. This includes flat approaches, with data simple producers registered in the directory service, hierarchical approaches, with low-level data producers coupled with several levels of producer/consumer pipes and only one (top of the hierarchy) producer registered with the directory service, and other hybrid architectures.

It is also very important for monitoring tools to be open; that is, to allow other tools to use them or to use other tools for their usual tasks. This is due to the large number of resources types, but also to the number of monitoring system requirements.

In conclusion, Grid monitoring applications should implement the Grid Monitoring Architecture, which includes at least three components: producer, consumer, and directory service, and to be open, allowing other tools to plug-in into the monitoring system or to make use of it.

3.1.2.2 Scalability

As for any distributed system, scalability is a very important issue when developing monitoring applications for Grids. The resources are dispersed, often over wide-areas, which adds latency constraints to those of federating (storing and retrieving) large amounts of data. There is, however, another side to Grid scalability: the human factor. Indeed, it is not the same to deploy a distributed monitoring system on the owned infrastructure, and to do the same on a Grid that spans several organizations, often with their own monitoring support strategies. The main problem concerning scalability is to ensure a reliable working state for the monitoring infrastructure, even in the undesired case of one or some machines failing. As a simple speculation [21], the solution to this last part of the scalability problem may become available from the peer-to-peer computing research field, which focuses on issues like dispersed resources automatically locating other resources and the survival of the system in the absence of some of its members.

3.1.2.3 Security

Security is a key concern for every Grid system. Virtual organizations usually encompass different security policies, and monitoring in such environments must respond to this situation. Data may not be offered but to accredited users, with special attention on *single sign-on* (users having to authenticate only once in order to access all data they are entitled to, even if this data is located on several sites spanning different organizations) and *delegation* (users being able to pass their authenticated rights to other users and/or programs). The GGF GMA proposal [70] points at X.509 identity certificates [34] and the Secured Socket Layer (SSL) [33] authentication protocol to be used for authentication.

Monitoring systems should also dynamically adapt their acting strategy depending on their rights on a particular subsystem (e.g. a monitoring tool may have read rights on one subsystem, but no rights at all on some other, depending on the security policy of the virtual

organization the subsystems belong to; in such case the data should be obtained by the monitoring system with some other monitoring tool, or in some other way).

3.1.2.4 Data targets

There are four data levels for Grid monitoring systems: host, network, middleware and user applications.

Host The host level refers to the computing and storing infrastructure of the Grid systems. Mainframes, supercomputers, clusters made out of commodity PCs, *blade* centers, memory systems, I/O systems, sensors, data and knowledge bases are all part of this level, each with special monitoring needs.

Network The network infrastructure is a key component of the Grid computing infrastructure, therefore deserving its own level. Subnets, high-speed connections, Gateways, routers, and even switches are all subparts that must be monitored at this level.

Middleware The middleware Grid systems component acts, hence its name, as the middleman between the user's applications and the Grid infrastructure. Subparts of this level include security, resource allocation, and others, each with specific monitoring issues.

User applications The user applications encountered in Grid systems are of great variety (also see sections 2.3 and 2.5). Therefore, how and what to be monitored depends on the type of applications. However, a minimum subset of monitoring data, such as the running/stopped status of the applications, should be monitored.

For a Grid monitoring system to be complete, all four data levels must be successfully targeted. This includes general monitoring issues like acquisition, visualization, and archiving, but also adds proper monitoring data selection (due to the heterogeneity of the Grid components). Also, as the Grid systems expand, there may be the case that storing or sensors levels must be added as new stand-alone data targets.

3.2 Monitoring techniques

In the past few years a consensus seems to have established upon the basic monitoring elements. It is the *characteristics* of the monitored systems components that we are after. Monitoring tools must gather values of the required characteristics under the assumption of heterogeneous environments. They must provide filtering and analysis for the gathered data. They must reduce overheads to a minimum, in the context of seamless operation. Dynamic inclusion of machines as monitored hosts is also a necessary requirement. Monitoring techniques like logging, archiving or visualizing data have to be carefully considered.

With all these facts in mind, this section aims at defining what a characteristic means and which characteristics are typically put under surveillance in a Grid monitoring system. A clear distinction is made between two general techniques of performing the measurements: active and passive. Also, a number of issues concerning the monitoring process will be presented. A framework that tries to address the issues in the context of Grids is presented.

3.2.1 Characteristics

Extending the GGF NMWG definition [43], a characteristic is *an intrinsic property of an object, system or process that describes the way it appears or performs in some encompassing context*. For example, a *network characteristic* is *an intrinsic property of a portion of the network that are related to the performance and reliability of the Internet*. In this example, the object is a portion of the network, while the encompassing context is the Internet. In this paper, the encompassing context is the Grid system under monitoring, while the objects are the metacomputing environment's components, as described in [20, 22]: the network, the computing hosts, the middleware (that is, the software that allows for the user applications to use the network and the computing hosts) and the user applications.

3.2.2 Measurements

3.2.2.1 Principle

Data for a characteristic is obtained through a *measurement process*. Alternate names for this process include *probing* and *observing*. The process of measuring is conducted always (in order to obtain a reliable piece of information) after a measurement methodology, that is, a technique for estimating of measuring a characteristic. In this context, *observations* can be *atomic (singletons)*, gathered together (*samples*) or derived from a sample (*statistic*).

Specialized tools, named *sensors*, perform the observations. Depending on their domain of applicability, there are at least four types of sensors: network, host, middleware and *user applications* sensors. Besides the acquired data, sensors could add a time stamp to each recorded event. This allows for later, possibly statistical, processing of the acquired data. If used, timestamps have to be consistent; therefore means of ensuring a global clock, like having networked time servers (i.e. Network Time Protocol servers), are critical in such cases.

3.2.2.2 Active, passive, and lowly intrusive measurements

Depending on the means for gathering the data, there are two directions in monitoring: monitoring through *active measurements* and monitoring through *passive measurements*. Monitoring through active measurement implies that the monitoring system systematically generates probes that affects the whole system in order to estimate characteristics values, such as the cases of launching packets to estimate network bandwidth or using heart-beat signals [25] to detect alive hosts. Monitoring through passive measurement [63, 4] requires that no such systematically generated probes are used; instead, the monitoring system relies completely on the data gathered by the underlying software layer, such as the operating systems own performance measurements.

Given that both active and passive measurements would yield the same class of results, be it reliability or accuracy or both, it would be preferable to use passive measurement. Unfortunately, this is hardly the case, as the tools on which passive measurements are based may lead to severe problems [14]. Recent advances in networks (Gygabit, Myrinet, optical fibers) and computing power (Intel and AMD PC processors), however, make active measurements "affordable".

Using active measurements induces another delicate problem: the *probing effect*. The probing effect consists of perturbations on the functioning of the monitored system introduced by the system that monitors it. Therefore, the monitoring system starts registering not only information about the targeted system, but of itself too. This is most unwanted, especially if the perturbations cannot be kept within some limits [69]. Therefore, the current goal for researchers is how to obtain the monitoring data while keeping the probing as *lowly-intrusive* as possible [41].

3.2.2.3 Prediction engineering

A new methodology has emerged in the past few years, concerning the processing of monitoring data with the purpose of predicting its evolution. Performance prediction tools, such as the Network Weather Service (NWS) [75], are required to model time-sensitive, dynamic and heterogeneous performance information. The modeled data can be used to predict performance of distributed applications in a Grid environment.

3.2.3 Hierarchies of characteristics

3.2.3.1 Principle

A number of any Grid object's properties can be monitored at the same time. Therefore, grouping data relative to one object into a hierarchical form of representation makes sense, especially when dealing with complex objects, which are composed of other complex objects, with this arrangement being repeated for a number of times. The hierarchies formed using this approach can be extended until they encompass the Grid systems itself [44]. Note that, since the Grid definition is rather unclear, we have adopted the Grid Global Forum definition [29]:

"connected groups of computing resources that can work together as a single Grid"

With this approach, requesting data for a node in the hierarchy can return results from all the branches underneath it, which gives a complete view of that node's status. However, as the number of properties is rather high, it is not recommended that requests such as 'data from the complete Grid system' are issued.

3.2.3.2 A taxonomy of characteristics hierarchies

In a characteristics hierarchy that covers complete Grid systems (see figure 3.4), the root node is the Grid system itself. The first level underneath the root node defines the components of the Grid system, as seen by the monitoring systems (and, therefore, by the system administrator). There are four types of Grid components: hosts, network, middleware and user applications (also see section 3.1.2.4).

It is very important to have a standardized, complete hierarchy of characteristics for all four components of Grid systems. This work is far from being over, with much debate over the characteristics residing in each of the four components' hierarchies. A somewhat complete hierarchy has been proposed for the host and network components [43].

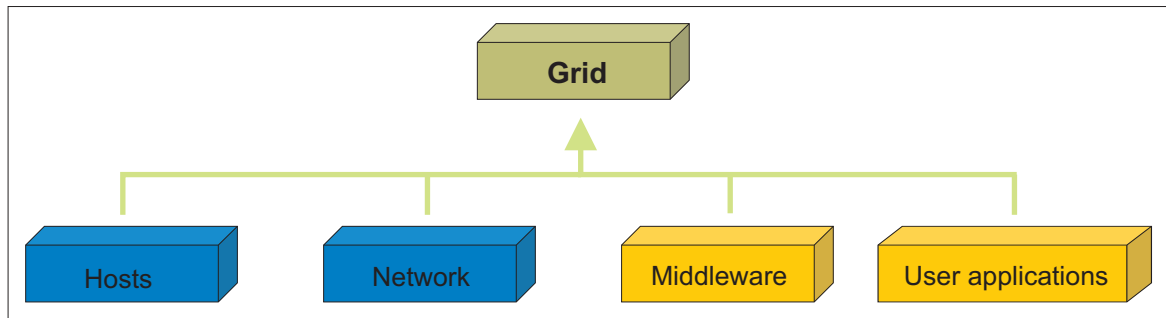


Figure 3.4: The top level of the complete Grid hierarchy of characteristics. Note the four Grid components: host, network, middleware and user applications.

Host characteristics

Host characteristics (fig.3.5) are used to describe the hosts within Grid environments (also see section 3.1.2.4). Such hosts may contain computing, data storage, and other resources, grouped into single or distributed systems. There are a number of characteristics in the host characteristics category, with the most important being [30]:

CPU Load the fraction of the CPU that is in use over a given segment of time; the smaller the time segment, the more accurate the information; usually, the time-frame for a CPU Load estimation is between one second and a couple of minutes;

System uptime the total time the system was not idle; this can be measured as a value over a period of time (much like the CPU Load, but with time frames from one minute to several hours) or since the last restart of the system;

Disk size the total space on a disk or on a disk-array;

Disk free the total available space on a disk or on a disk-array;

Available memory the total available volatile/non-volatile memory;

Host architecture the host system architecture;

Host OS the host system operating system.

Network characteristics

Network characteristics (fig.3.6) are used to describe the network elements interconnecting hosts within Grid environments (also see section 3.1.2.4). Amongst network elements there are routers, switches and other devices. There are a number of characteristics in this category, with the most important being [30]:

TCP available bandwidth the maximum available bandwidth, when using only the single-stream TCP communication protocol;

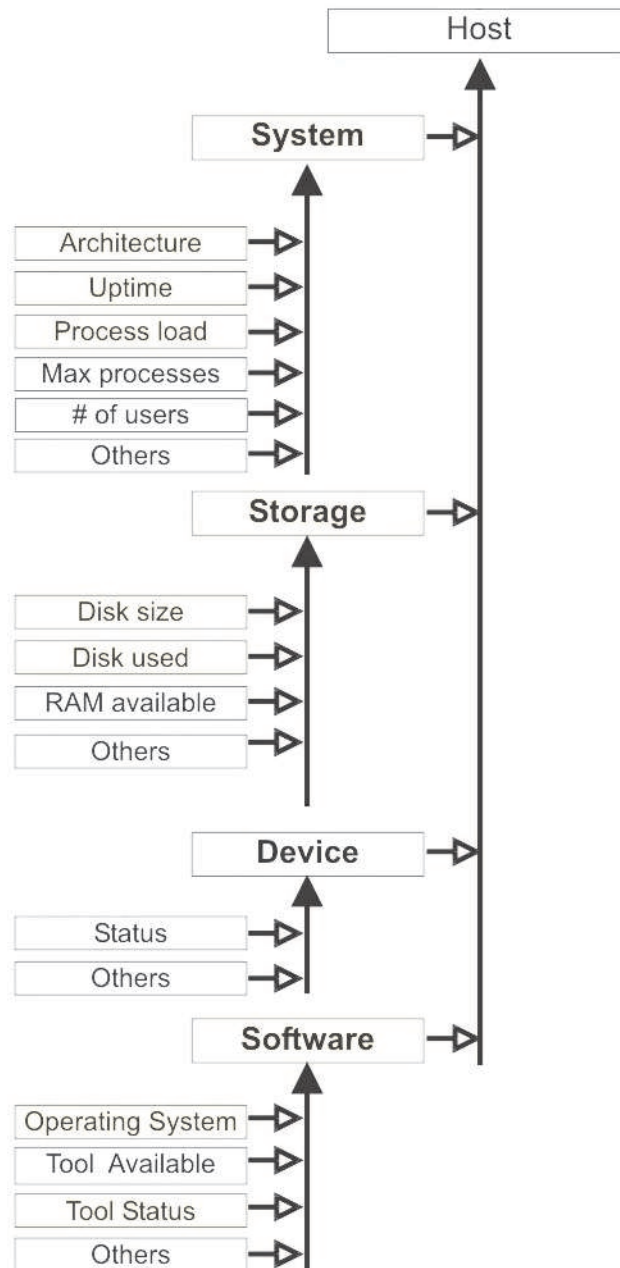


Figure 3.5: The host hierarchy of characteristics.

Round-trip time the total amount of time spent by a minimal message, sent using the ICMP protocol, in order to go to a given destination and to arrive back from where it started from (link round-trip);

Number of hops the number of hops (nodes that define a link for a given ISO OSI layer are called hops for that layer) between source and destination on a network link;

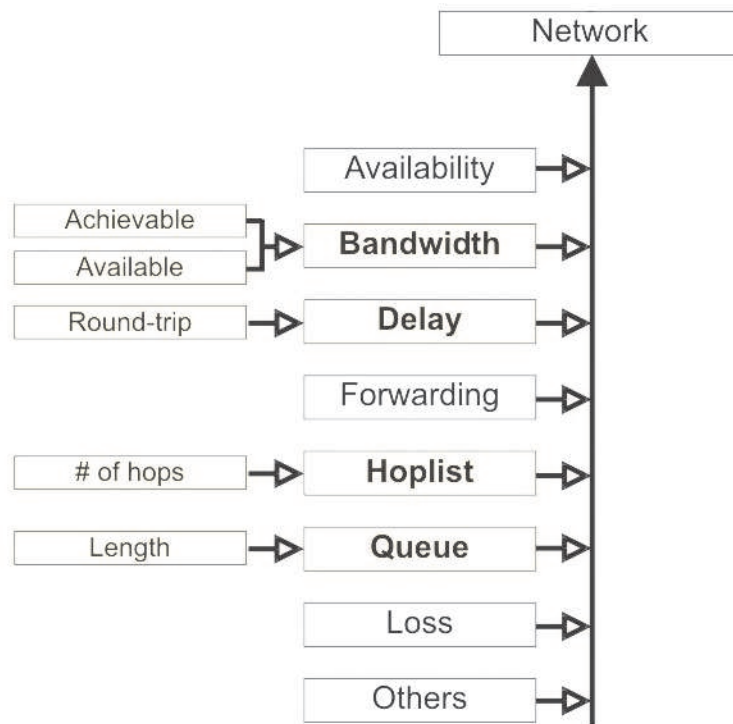


Figure 3.6: The network hierarchy of characteristics.

Packet loss the percentage of sent packets that do not reach the destination (also called one-way packet drop ratio).

Middleware characteristics

Middleware characteristics (fig.3.7) are used to describe the middleware acting within Grid environments (also see section 3.1.2.4). Amongst middleware elements there are schedulers and security components. There are a number of characteristics in this category, with the most important being [30]:

Scheduler queue length the scheduling queue length;

Middleware component status the current status of a middleware component (e.g. running, stopped, and so on).

User applications characteristics

User applications (fig.3.8) characteristics are used to describe the user applications within Grid environments (also see section 3.1.2.4). There is no consensus about what characteristics

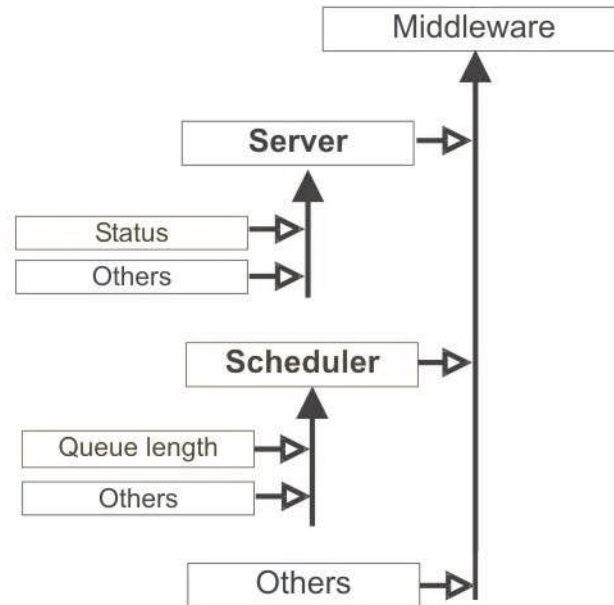


Figure 3.7: The middleware hierarchy of characteristics.

to be used at this level, many being left at the user's free will. There is one important characteristic in this category [30]:

Application running status the current status of a software application (e.g. running, stopped, and so on).

Other characteristics

As Grid systems components evolve, it is very likely that new monitoring requirements should appear in middleware and user applications components. This should conclude in additional characteristics, and the possibility of breaking the middleware component into Grid system middleware and general user helper middleware. The currently available hierarchies of characteristics should be completed with the new features.

3.3 Monitoring tools structure

This section presents a monitoring tools taxonomy, based on two of their structural attributes: the organizational structure and the same-level components interconnection type.

3.3.1 Overview

Monitoring tools, in general, and Grid monitoring tools, in particular, have a number of distinctive attributes, like the targeted user community, intrusiveness, scalability approach, validity of information guarantees, communication model and many others.

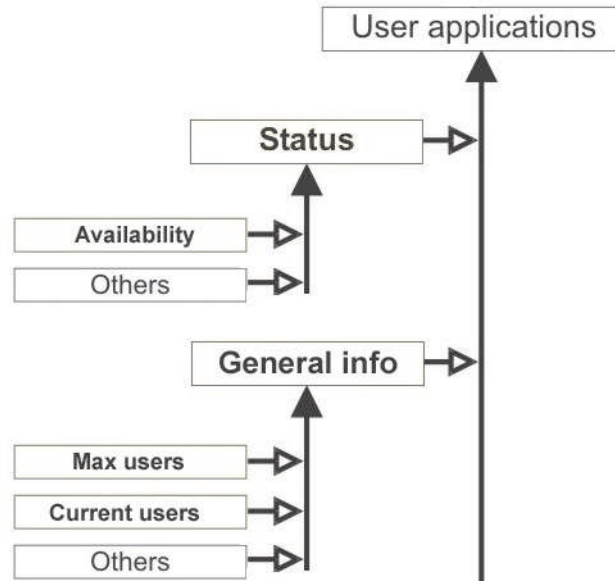


Figure 3.8: The user applications hierarchy of characteristics.

As a reminder, this whole chapter introduces a taxonomy of monitoring tools. Given the variety of attributes according to which to categorize the various monitoring tools, we have chosen *five* objectives for this taxonomy:

- (i) it should be intuitive (*intuitive*);
- (ii) it should include just a small number of categories (*clear*);
- (iii) it should be constructed around at least two distinctive attributes of monitoring tools (*comprising*);
- (iv) it should allow categorizing the most important monitoring tools of today (according to [8, 24, 63, 68]) or their components (*useful*);
- (v) it should try to propose at least one category for monitoring tools that are to appear (*guiding*);

For the sake of the first objective, we have selected focusing on the structure of monitoring tools. The structure also gives many information about the scalability and the maintenance easiness of the approach. With some exceptions, the structure of a monitoring tools can also give hints about the validity of information within the monitoring system. Note that a directory service has been added by default to the structure of these monitoring tools, to make them *Grid-aware*.

To make the taxonomy clear, yet comprising, we have chosen two structural attributes, namely the *organizational structure*, for which flat, hierarchical 1-level, hierarchical N-level and hybrid types are covered, and the *same-level components interconnection*, for which no

interconnection, loose interconnection and near cross-bar interconnection types are covered. The simple taxonomy consists of seven types (the other possible ones being left out because of the lack of monitoring tools falling under these categories, and their likelihood of being filled with new monitoring tools in the near future):

1. Flat, dispersed
2. Flat, connected
3. Flat, near cross-bar
4. Hierarchical, 1-level
5. Hierarchical, N-level
6. Hybrid connected core/N-level hierarchical connections
7. Hybrid near cross-bar core/N-level hierarchical connections

Examples selected for each category of this taxonomy are an attempt to prove that objective (iv) holds, while chapter 4 presents an architecture that we believe is going to become the next step in monitoring control Grids, thus fulfilling objective (v).

3.3.2 Flat, dispersed

This category (see fig.3.9) includes monitoring tools that comprise only sensors residing on each of the monitored hosts, with no communication happening between the sensors. A consumer trying to obtain data from a sensor must connect to that sensor and follow the communication protocol spoken by that sensor.

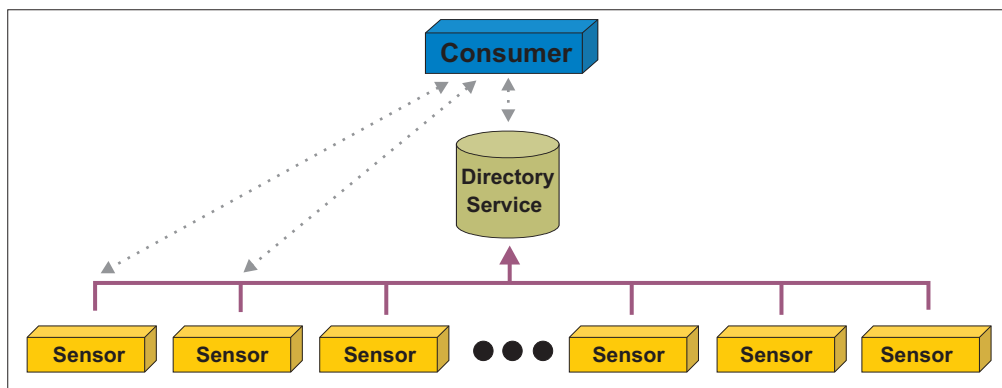


Figure 3.9: The flat, dispersed structure of monitoring tools. Sensors cannot communicate with each other. A consumer issues direct connections with each sensor, in order to obtain monitoring information.

As sensors reside on the very systems they monitor, they cannot usually use complicated filters on the acquired data and they cannot store too much data, lest they should be too intrusive. This leads to a constant requirement of consumers taking the data from the sensors.

An example of this category is SNMPv1 [11]. The first stable release of SNMP offers a simple administrative model, based on only three basic commands: *get*, *get-next*, and *set*. The services are offered via a client/server model, while the data is stored and exchanged according to data collections called Management Information Bases (short, MIBs). The MIB-II [51] data collection is supported (thus, maintained) by every SNMP-compliant device and contains device descriptions, as well as interfaces and protocol descriptions and statistics.

3.3.3 Flat, connected

This category (see fig.3.10) includes monitoring tools which rely on sensors residing on each of the monitored hosts, with some form of communication happening between the sensors. In some cases, sensors know how to cooperate with other sensors, in order to solve specific user requests. For example, a request to gather data from all sensors is dealt in a cooperative manner, with the first contacted sensor sending the request to another and so on. A consumer trying to obtain data from a sensor must connect to that sensor, and follow the communication protocol spoken by the sensors. Acquiring data from all sensors can be done by connected to all the sensors or to just one, given that the delegation mechanism is available.

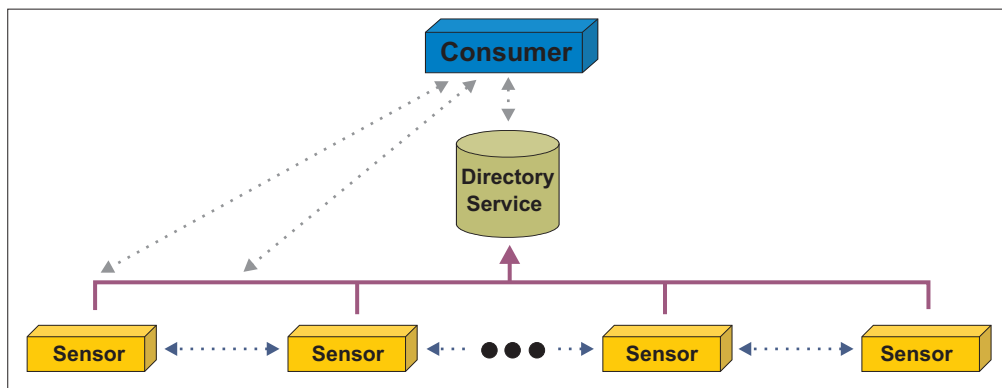


Figure 3.10: The flat and connected structure of monitoring tools. Sensors can communicate with each other. A consumer issues direct connections with each sensor or with only one (then delegates requests through that connection; the loose connection means that the delegation mechanism may generate a *domino effect*), in order to obtain monitoring information.

An example of this category is the HP Open View [32]. This tool is built on top of SNMPv2c [12] and uses the Extensible SNMP Agent [31], allowing, amongst others, for extra cooperation between agents. The HP EMANATE Subagent Development Kit is offered for those who want to programmatically create their cooperative agents.

3.3.4 Flat, near cross-bar

This category (see fig.3.11) includes monitoring tools which rely on sensors residing on each of the monitored hosts. The sensors can communicate with each other through the use of a high-speed, highly interconnected network, similar to cross-bars (every resource may communicate with every other resource through a dedicated channel). This allows sensors to

cooperate in order to solve specific user requests and also to add some degree of fault-detection and autonomous self-management. For example, by using a heart-beat signal, sensors can detect the falling of one sensor and attempt to restart it. The cross-bar communication structure can be physical or logical.

Consumers trying to obtain data from one sensor must connect with that sensor, or use some delegation mechanism, and follow the communication protocol spoken by the sensors. Acquiring data from all sensors can be done by connected to all the sensors or to just one, given that a delegation mechanism is available.

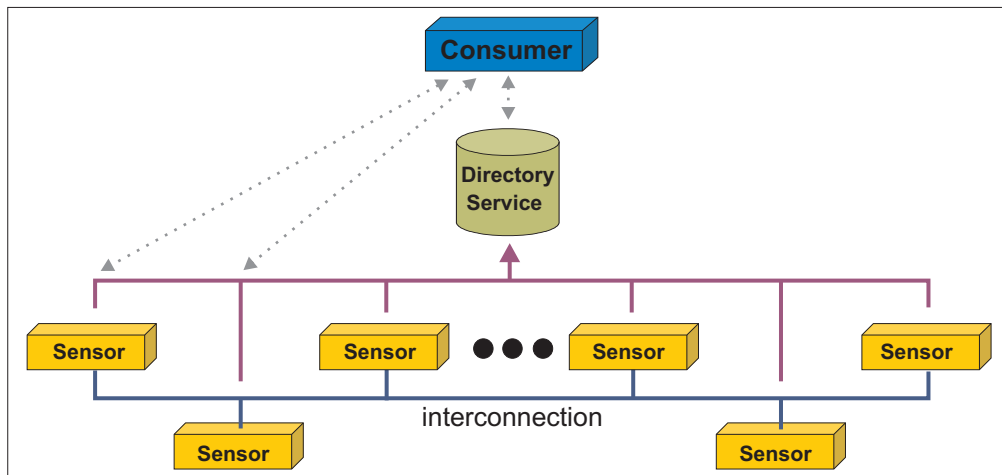


Figure 3.11: The flat and near cross-bar connected structure of monitoring tools. Sensors can communicate with each other via a near cross-bar connection. In order to obtain monitoring information, consumers issues direct connections with each sensor or with only one (then delegates request through that connection).

The main difference between the monitoring tools flat, near cross-bar and the flat, connected categories is the connection between the sensors. While for flat, connected monitoring tools the connection between sensors has a reduced number of communication channels, the flat, near cross-bar monitoring tools employ a strategy that offers a high number of communication channels. The obvious result is that with the near cross-bar connections, replicating and even broadcasting information is very efficient, with direct impact in auto-maintenance and high availability, but at the cost of more communication resources being used.

An example of this category is the Ganglia cluster level, namely the gmond [49]. The tool uses the multicast protocol [5] in order to offer a broadcast medium for all the participants within the cluster. Sensors publish acquired data in the multicast channel, which greatly reduces the risks of losing data due to failure, at the expense of supplementary network traffic. Heart-beat signals are emitted by sensors at regular times, thus allowing for easier failure detection.

3.3.5 Hierarchical, 1-level

This category (see fig.3.12) includes monitoring tools which include a centralized monitoring point on top of the sensors. This centralized monitoring point, named *group sensor*, gathers data from all bottom-level sensors, and makes this data accessible to the users. Data filtering or storage can also occur at this point.

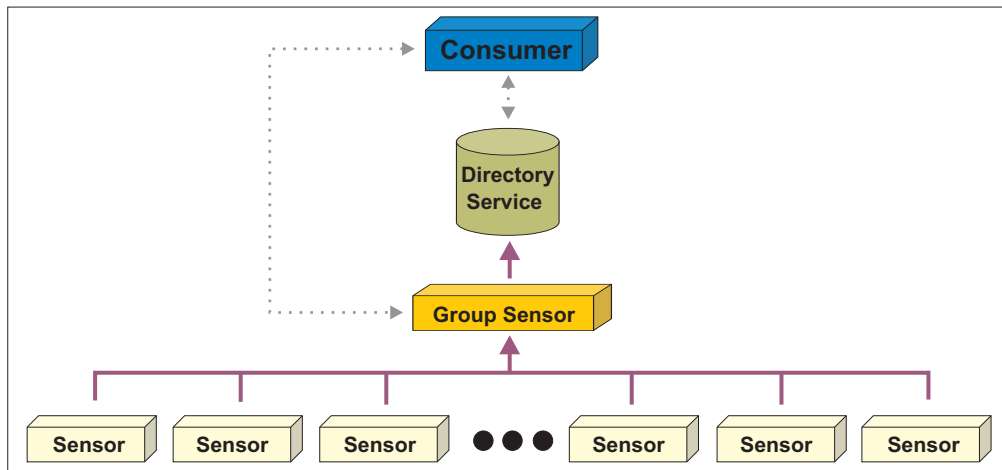


Figure 3.12: The hierarchical, 1-level structure of monitoring tools. A group sensor centralizes data from all sensors. Consumers connect with the group sensor, in order to obtain monitoring information.

In order to obtain monitoring information, consumers connect with the group sensor. The latter either fetches some data from its already available database, or polls it from the sensors on-demand. Issues like data freshness data summarization can also be dealt with at this level.

The main difference between the hierarchical, 1-level and the flat categories is the existence of a group sensor, on top of the sensors. The group sensor automates the flat categories user's job (e.g. the user was forced to perform by itself the sensors polling).

An example for the hierarchical, 1-level category is NetLogger v1 [71]. This tool can monitor network-, host- and user application- related events. It is designed to facilitate non-intrusive instrumentation of distributed computing components. Its structure includes a number of sensors, be they sensors of the operating system or instrumented points within user applications, and a single, centralized, group sensor, called *netlogd*. Though this approach is hardly scalable, this tool has been proven useful in monitoring Grid applications, with sensors being complex monitoring tools themselves [72].

3.3.6 Hierarchical, N-level

This category (see fig.3.13) extends the hierarchical, 1-level category, described in the previous section. In this category, the bottom level consists of sensors, grouped under a group

sensor. Several group sensors may be grouped under another group sensor and so on, forming a hierarchical (or tree) structure, with as many levels as needed. The group sensor is responsible for federating data from sensors that it dominates (sensors that cannot be reached starting from the root of the whole system without passing through that group sensor; in other words, sensors that belong to the sub-tree having that group sensor as the root). Data filtering or storage can also occur on every group sensor.

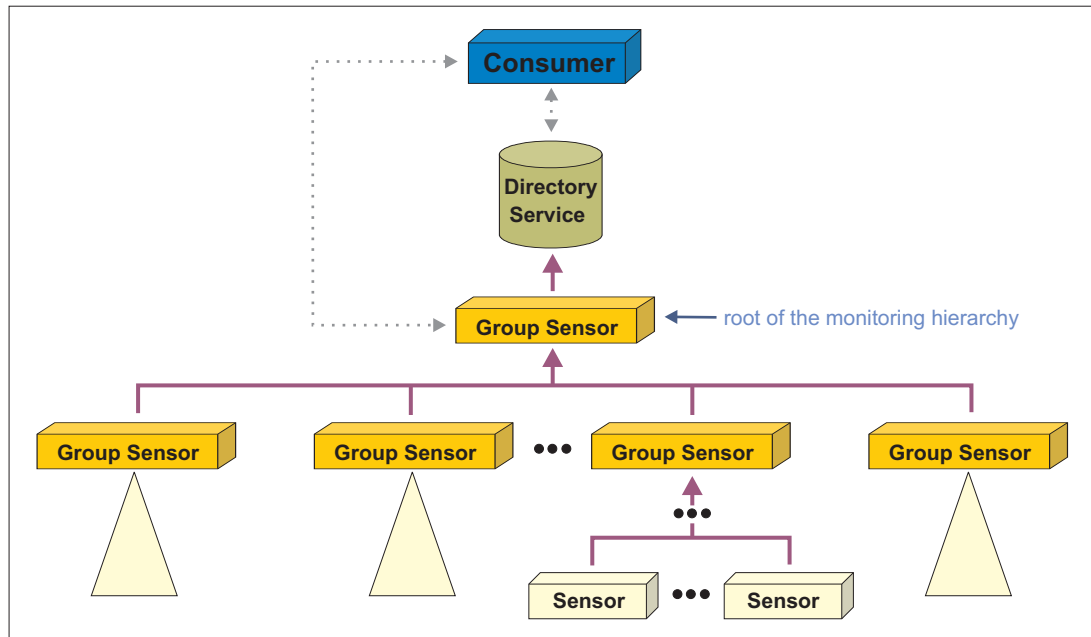


Figure 3.13: The hierarchical, 1-level structure of monitoring tools. A group sensor centralizes data from all sensors/group sensors underneath it. Group sensors are organized in a tree-like structure, with one group sensor being the root of the hierarchy. Consumers connect to the root group sensor or to one of the group sensors, in order to obtain monitoring information.

In order to obtain monitoring information, consumers connect with the root group sensor, and sometimes with every group sensor in the hierarchy (though less likely). The latter either fetches some data from its already available database, or polls it from the (group) sensors underneath it on-demand. Issues like data freshness data summarization can also be dealt with at this level. Often group sensors offer a standardized method for communicating data, like XML [73] or XDR [16, 67].

The main difference between the hierarchical, N-level and the hierarchical, 1-level categories is the existence of any number of group sensor, as opposed to just one, for the 1-level hierarchy. This allows for better monitoring of large groups of resources (summarization gives an overview of big parts of, or even the whole, system). This approach also favours scalability, as a large number of nodes at the same level would induce data gathering bottlenecks.

It is important that using several tools from the hierarchical, N-level category is similar to using a tool implementing a hybrid category, with the core having the structure of a flat, dispersed category and the main body having a hierarchical, N-level structure. The core would consist of every root group sensor of the tools used, while the body will contain every hierarchy build underneath the core nodes. The lack of communication in these systems, however, makes data retrieval difficult and the system centralized at its core.

An example for the hierarchical, n-level category is the Ganglia monitoring system [23]. This tool can monitor network- and host- related characteristics. Its bottom-level sensors are well adapted for cluster monitoring, with features like auto-detection and simple data filtering. The group sensors have a design well-adapted for monitoring large structures, like wide-area multicluster systems [61]. For that, they follow an n-level hierarchical structure; they also offer some degrees of fault-tolerance, by using fixed-size data storage. This tool has been proven to work within large Grid environments and is included in the Grid3 project [26].

3.3.7 Hybrid connected core/N-level hierarchical connections

This category (see fig.3.14) is a hybrid between the flat, connected and the hierarchical, N-level approaches. The whole system would consist of several N-level trees, each with a root group sensor and a body (the rest of the group sensors and also the sensors). The core would be formed by all the root group sensors, that are able to cooperate in order to ensure some degree of fault tolerance, optimized system performance or some other feature(s). The body of this approach is hierarchical and structured on any number of levels.

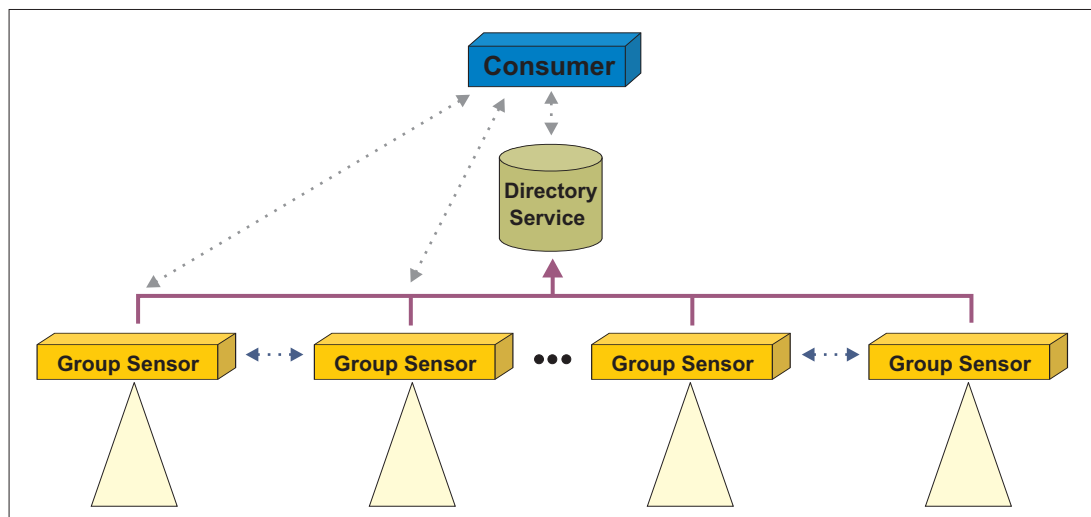


Figure 3.14: A hybrid structure of monitoring tools, with connected core and N-level hierarchical connections. A group sensor centralizes data from all sensors/group sensors underneath it. Group sensors are organized in a tree-like structure, with one group sensor being the root of the hierarchy. All root group sensors are grouped in a connected core. Consumer connect with the each or one of the root group sensors, in order to obtain monitoring information.

In order to obtain monitoring information, consumers connect with root group sensor that can offer it the needed information or, if caches exist at this level, to one of the root group sensors that has the needed information in its cache. The contacted root group sensors fetch the data from their cache or database, or start polling the (group) sensors underneath them for it. Data freshness is a very important issue at this level. Often root group sensors offer a standardized method for data transfer, like XML [73] or XDR [16, 67]. The group sensors are often implemented with features like advanced data summarization (through statistical means or even prediction methods), fault-tolerance (small caches or fixed-size databases).

Often monitoring systems implementing the hybrid, connected core/N-level hierarchical connections approach are an assembly of several monitoring tools. This can be done because of the openness of many monitoring tools (see more about this aspect of Grid monitoring tools in section 3.1.2.1. An example in this direction is the next version of MonALISA [52], coupled with Ganglia. The MonLISA tool is based on the Java JINI [39] distributed components technology, which allows for automated core nodes discovery and integration. It can monitor a wide range of host-, network- and user applications- related characteristics. The next release would provide an optimized data retrieval system, through the use of replicated data repositories [53]. This tool has been proven to work within large Grid environments and is included in the Grid3 project [26].

3.3.8 Hybrid near cross-bar core/N-level hierarchical connections

This category (see fig.3.15) is a hybrid between the flat, near cross-bar and the hierarchical, N-level approaches. The whole system would consist of several N-level trees, each with a root group sensor and a body (the rest of the group sensors and also the sensors). The core would be formed by all the root group sensors, that are able to cooperate in order to ensure some degree of fault tolerance, optimized system performance or some other feature(s). The interconnection at the core level is of very high-speed and includes a large number of available point-to-point links, like a cross-bar structure. The body of this approach is hierarchical and structured on any number of levels.

In order to obtain monitoring information, consumers connect with root group sensor that can offer it the needed information or, if caches exist at this level, to one of the root group sensors that has the needed information in its cache. The contacted root group sensors fetch the data from their cache or database, or start polling the (group) sensors underneath them for it. Data freshness is a very important issue at this level. Often root group sensors offer a standardized method for data transfer, like XML [73] or XDR [16, 67]. The group sensors are often implemented with features like advanced data summarization (through statistical means or even prediction methods), fault-tolerance (small caches or fixed-size databases).

We see no currently existing monitoring tool designed under this structural approach. However, the Toytle architecture, presented in the next chapter, is a solution for control Grids monitoring that falls under this category.

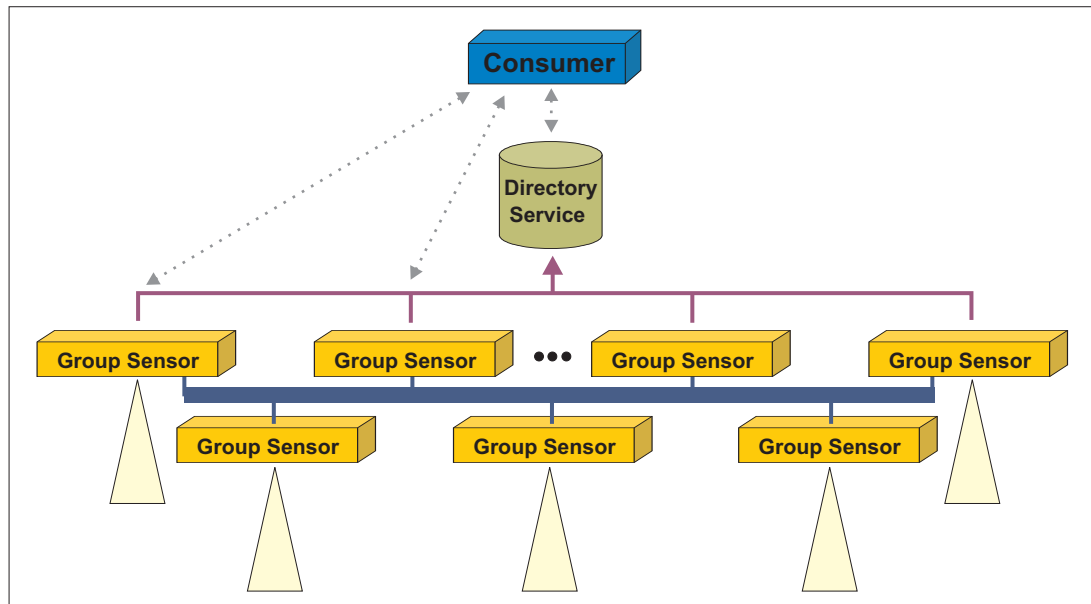


Figure 3.15: A hybrid structure of monitoring tools, with near cross-bar core and N-level hierarchical connections. A group sensor centralizes data from all sensors/group sensors underneath it. Group sensors are organized in a tree-like structure, with one group sensor being the root of the hierarchy. All root group sensors are grouped in a connected core, with near cross-bar connections. Consumer connect with the each or one of the root group sensors, in order to obtain monitoring information.

3.3.9 Related work

The large number of available monitoring tools, as well as the novelty of the Grid computing research field, lead to a number of monitoring tools surveys dedicated not only to Grid computing but also at distributed systems, particularly those with wide- or midum- physical spanning areas.

Martin-Flatin et al. propose two taxonomies of such systems, based on simple and complex organizational models [48]. Their approach with the simple taxonomy overlooks the hybrid organizational approaches, while the complex taxonomy looks into another direction, namely the tasks delegation paradigms. Our approach also consists of two directions. It adds the more categories to the organizational axis, and looks for the communication facilities, on the second, the interconnection, axis.

For Grid environments, *Balaton et al.* [8] make a good presentation of the representative monitoring tools. Their report covers issues like complexity, scalability, intrusiveness and others, pretty much like this paper does. However, they do not try to assess tools that do not fall under the Computing Grids monitoring category and they are not looking for a simple taxonomy. Our approach focuses on clarity and usefulness, on the expense of having only a small number of descriptive attributes for monitoring tools.

A much larger report has been published by *Gerndt et al.* [24]. This report covers many monitoring tools that can be used for Grid monitoring. Again, a large number of categorizing criteria are presented, amongst which target communities, scalability, intrusiveness, and others. This approach deals well with presenting an important number of tools. However, the MonALISA monitoring tool, which looks very promising and has been included in several important Grids projects, is not mentioned in this report. Also, the report authors did not intend to establish a simple taxonomy, but only to present existing solutions. Therefore, this report is a basis for our taxonomy, which complements it.

3.4 Monitoring trends and downsides

3.4.1 Monitoring trends

A number of trends concerning the development of Grid monitoring tools have crystallized in the past few years. There are several trends for these tools [8, 64]: openness, portability, scalability, standardization in data format, standardization in communication format, reduced intrusiveness, standardization in measurement, freshness of information, security, expandability, autonomy, self-advertisement and manageability.

Grid monitoring tools must fit inside the Grid infrastructure, as stated in section 3.1.2.1. They must be portable and scalable. Their output/input data must be formatted according to some international standard, and XML is regarded as an important option. Their communication protocols must be standardized, such as SOAP or XML-RPC. The sensors actually performing the measurements, as well as the data passed through, toward the Grid monitor, must reduce the monitoring tools interaction with the monitored system as much as possible. The measurements must be performed according to standards and results must be presented in the form of standardized metrics, such as percent for the CPU load. The information must be kept fresh for real-time monitoring systems or be declared old or even invalid, except for the systems involving some form of performance prediction engineering. Also, monitoring tools should provide the security provided by any normal Grid computing component. The monitoring tools must be expandable, that is, they must be conceived in such a manner that allows users to add their own, custom, sensors, filters, and other extensions to the original system. The monitoring tools must be autonomous, that is, autonomously start and stop the sensors and also automatically detect and recover from failures, according to some operator-assisted guidelines. The monitoring tools must advertise upon its monitoring capabilities, such that other tools, including monitoring applications, may use it. Finally, given the physical and human environment they must serve, monitoring tools should be easy to manage (e.g. install, upgrade and some repairs should be simplified).

3.4.2 Monitoring downsides

As Grid monitoring tools become bigger and bigger, one might forget what they are based upon: simple sensors. The SNMP standard [11] has been imposed for sensors regarding networked components. However, designing new sensors raises issues like usage profile estimation and systems coupling, while implementing even simple sensors brings up issues like adapted performance and optimal configuration. This leads to an increase in the development cost and time of such components.

The current lack of standards for the data format and communication protocols, vital for Grid monitoring tools, can lead to a slow development of Grid monitoring field; in addition, non-Grid-aware tools might still be considered as strong contenders for Grid monitoring while clear standards for monitoring in such environments are not set.

The middleware and user application monitoring levels are the least used and suffer from the lack of standardization. In this context, there have been no efforts so far to salvage the parallel computing monitoring and performance evaluation experience in the context of Grid computing. Indeed, it may prove beneficial to monitor supercomputers or large clusters with the parallel approach in mind, especially for very-demanding computing applications.

3.5 Monitoring control Grids

Monitoring control Grids presents numerous challenges. In addition to Grid-awareness, scalability and standards-based communication [24], several new problems emerge, like:

- gathering data from all of the Grid components (network, hosts, middleware and applications),
- dealing with large "bursts" of information,
- performing high-speed sampling (less than 0.5 s between per-variable samples and more than 100 per-node samples/s),
- ensuring low intrusiveness on the monitored machines,
- presenting relevant data in meaningful ways,
- ensuring some level of fault-tolerance.

Even more, as control Grids are, in general, interdisciplinary projects, with most of the people involved having low technical skills, if some at all, the installation and management of monitoring tools are extremely important.

We claim that nowadays monitoring tools cannot address these issues in full. Systems like GRM/Prove and NetLogger lack a lowly intrusive multi-level monitoring hierarchy. The NWS system cannot be used with "bursty" applications and is not suitable for fast changing data sets, such as control Grids applications. Systems like R-GMA, MDS and Nagios are too resource-demanding and slow-paced to be effective in this case. The Ganglia monitoring system is well-suited for such applications, but lacks a central monitor, as well as the appropriate visualization means. The MonALISA monitoring system is well suited for data visualization and high-speed sampling, but it is too resource-intensive to be used as-is for these purposes. Therefore, a complete architecture, specifically designed for addressing these particular problems in the context of monitoring control grids is needed.

3.6 Summary

This chapter has presented the state-of-the-art in the field of monitoring large distributed systems, with an emphasis on monitoring Grid computing systems. For that, we have presented the current tasks, techniques and tools in this field. Nowadays trends and downsides have been analyzed. The special case of monitoring control Grids has also been discussed.

This chapter has also presented the reader a simple taxonomy of monitoring tools, with separating criteria being targetted at what we consider of being the today's limits of control Grids monitoring: monitoring tools organization and internal interconnection. The taxonomy offers seven categories, namely *Flat dispersed*, *Flat connected*, *Flat near cross-bar*, *Hierarchical 1-level*, *Hierarchical N-level*, *Hybrid connected core/N-level hierarchical connections*, and *Hybrid near cross-bar core/N-level hierarchical connections*. The last category aims at what we believe to be structure of the next step in control Grids monitoring.

Chapter 4

Toytle: a monitoring architecture for control Grids

This chapter introduces a novel monitoring architecture that aims at satisfying the most important requirements of control Grids: the Toytle architecture.

4.1 Premise

The Toytle¹ architecture is specifically designed to address the specific needs of control Grids monitoring, such as high speed data gathering and federation, scalability, program lightness (in terms of monitored system's resource consumption), low intrusiveness (in terms of monitored system's performance degradation) and easy deployment.

4.2 Toytle components

This section introduces a new monitoring architecture, named Toytle, which addresses many issues of monitoring physical processes with soft time constraints. This new architecture is structured in three layers:

1. **Distributed core** - a complex core with near cross-bar connections between nodes that provides high-level monitoring services to the user;
2. **Hierarchical connections** - a fairly light hierarchical structure that connects the local monitors to the distributed core, through data federation and piping;
3. **Local monitors** - a very light set of monitors (i.e. not just sensors) that gather data from locally available sensors and perform some simple filtering on it.

The three layers cooperate for seamless monitoring services, in a hybrid, near cross-bar core/N-level hierarchical connections structure (see section 3.3). This approach ensures that, while the hierarchical connections lead to a scalable approach, the data obtained from multiple large geographical areas (thus, multiple hierarchical trees) can be centralized in a distributed core, with ease of access added to some degree of fault tolerance.

¹the name is only a reminder that even the slow and unintrusive beings can win even the foulest of races (see the rabbit v. tortoise contest [3, 74])

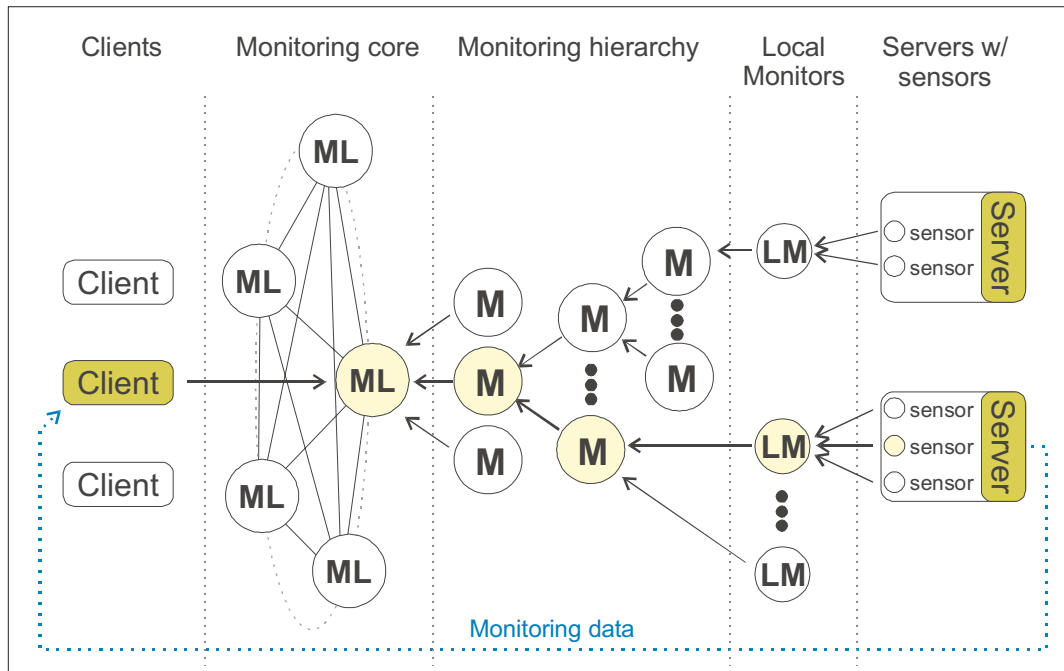


Figure 4.1: The Toytle architecture components: *distributed core*, *hierarchical connections* and *local monitors*. The path between data provided by the sensors and the monitoring system interface with the consumers is highlighted.

4.2.1 The distributed core layer

The *distributed monitoring core layer* deals with issues like data gathering, storage, replication, and visualization. Its structure is horizontal (see fig.4.2), with nodes distributed according to geographical or problem-wise needs. This module tackles issues like adaptive system balancing, fault-tolerance, self-maintenance, Grid data access, and user control.

The monitoring core nodes should be able to transparently balance the monitoring system use. First, when a user tries to connect to a core node, it will be automatically relocated to another core nodes, given that the first node was too crowded. Second, data summarization should be performed by the least charged core node that has access to that data. For both these tasks, the core should also be self-monitoring.

Fault-tolerance is ensured through data replication. Data storage is performed per-node or per-system in one or more repositories. Recent data is stored in memory buffers, for quick access. Recent data replication is performed transparently for the user between core nodes. Data visualization allows for both resource- and application- oriented views. The monitoring core is self-maintained, so that node insertion and failure are dealt with transparently to the user. A link failure between two core nodes (see the red cross in fig.4.2) can be transparently replaced by another link creation (see the thick blue link in fig.4.2).

Also, in the event that the core node that was maintaining the connection with a lower-

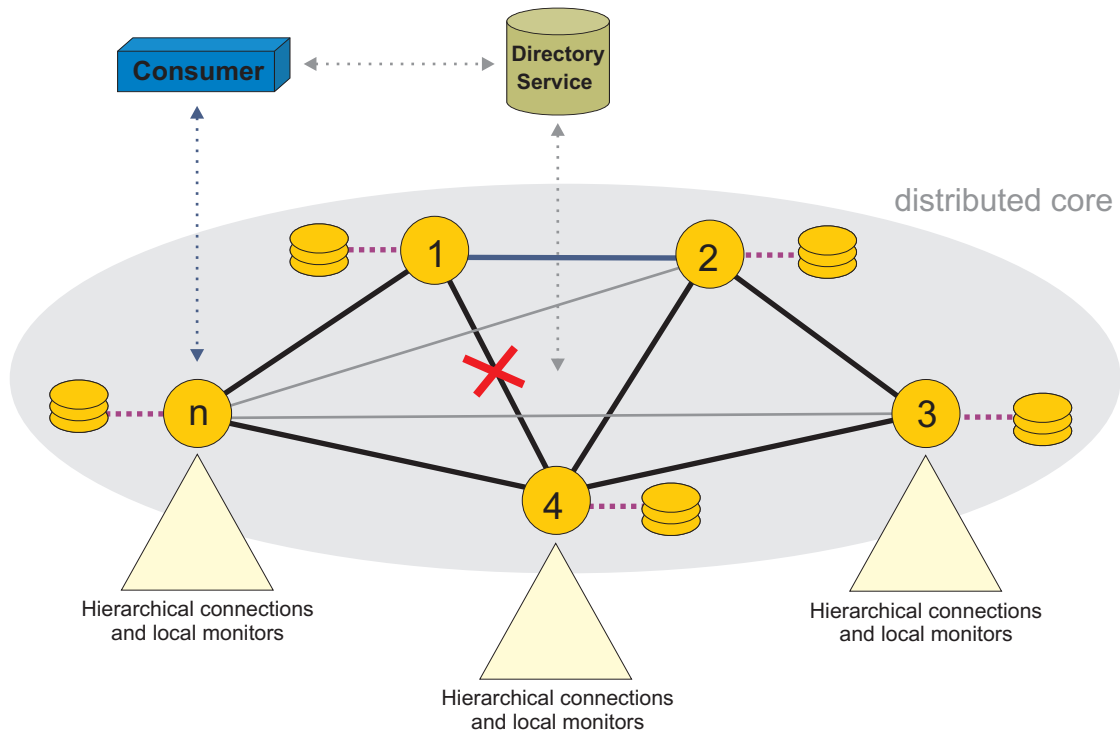


Figure 4.2: The Toytle architecture distributed core layer. The nodes within the core can maintain permanent contact (thick connection lines between nodes) or just be able to communicate with one another (thin connection lines between nodes). In the case that a permanent contact link falls down (red cross), another link may be established with other nodes of the core (think blue link). The consumer may use the distributed directory service to locate the adequate distributed core node.

layer node (a root monitor from the hierarchical connections layer), the core should transparently relocate that connection to another core node. In other words, as long as the core contains at least one node, data is available to the users, and as long as several nodes are active at the core level, data federation is optimized for speed issues.

User control of the monitoring system is done through the specification of monitoring levels for all four categories of Grid monitoring data: host, network, middleware and user applications. Every characteristic monitored by the monitoring system has a type and a monitoring level associated with it. Once the monitoring level for the characteristic's type has been surpassed, that characteristic becomes monitored and its data starts reaching the core, thus being available to the user. The higher the monitoring level the larger the number of data gathered and summarized at the core level.

Being given that the number of core nodes typically does not exceed 1 – 2% of the total number of resources in the monitored Grids, the near cross-bar interconnection at this level does not raise scalability issues.

The fact that the core nodes are not light does not interfere with the low intrusiveness requirement, as defined in the prologue of this chapter (see sec.4.1), as the core nodes do not actually run on the monitored system's machines. The number of core nodes, discussed earlier, makes this extra-resource usage affordable.

In order to obey to the GGF GMA consumer/producer/directory services, information about the monitoring core's data type and actual data location can be obtained from a monitoring directory. Also, standard data representations, both with size reduction (e.g. XDR) and scheme self-definition (e.g. XML) targets, should be offered.

4.2.2 The hierarchical connections layer

The *hierarchical connections layer* of the monitoring system performs data gathering and summarizing. It is also responsible for ensuring hierarchical monitoring connections, while keeping the monitored system's load at a minimum, and for transmitting changes in monitoring levels to the local monitors.

The vertical connection structure (see fig.4.3) allows for efficient data-summarization, possibly reducing the load of the monitoring system. Each monitoring hierarchy top node is linked with one or more monitoring core nodes.

As long as at least one of its nodes is active, the hierarchical connections layer should continue to perform as a link between the core and the local monitors. For this, a fallen link should be taken care of by the father of the dead node, if it was a *top-down* link (a link in which the node that just died was on a higher level in the hierarchy than the node that is still alive), or by the son of the dead node, if it was a *bottom-up* link (a link in which the node that just died was on a lower level in the hierarchy than the node that is still alive). This fault tolerance target should be performed transparently to the user.

Temporary data recovery, in the context of low intrusiveness, is offered through backups on fixed-size (e.g. round-robin) databases. Standard data representations, both with size reduction (e.g. XDR) and scheme self-definition (e.g. XML) targets, are offered.

4.2.3 The local monitors layer

The *local monitors layer* performs all the data acquisition and simple on-the-fly filtering. At this level, locally-close monitors are connected in horizontal structures (see fig.4.4), with benefits like node insertion and failure being dealt with transparently to the user. Data from all of the Grid components, network, hosts, middleware and applications, can be acquired at this level.

Fault-tolerance can be ensured through data advertisements. Each local monitor can advertise its data and send it to other local monitors, so that exact replicas of that data can be temporarily found on other places than the origin, at any time.

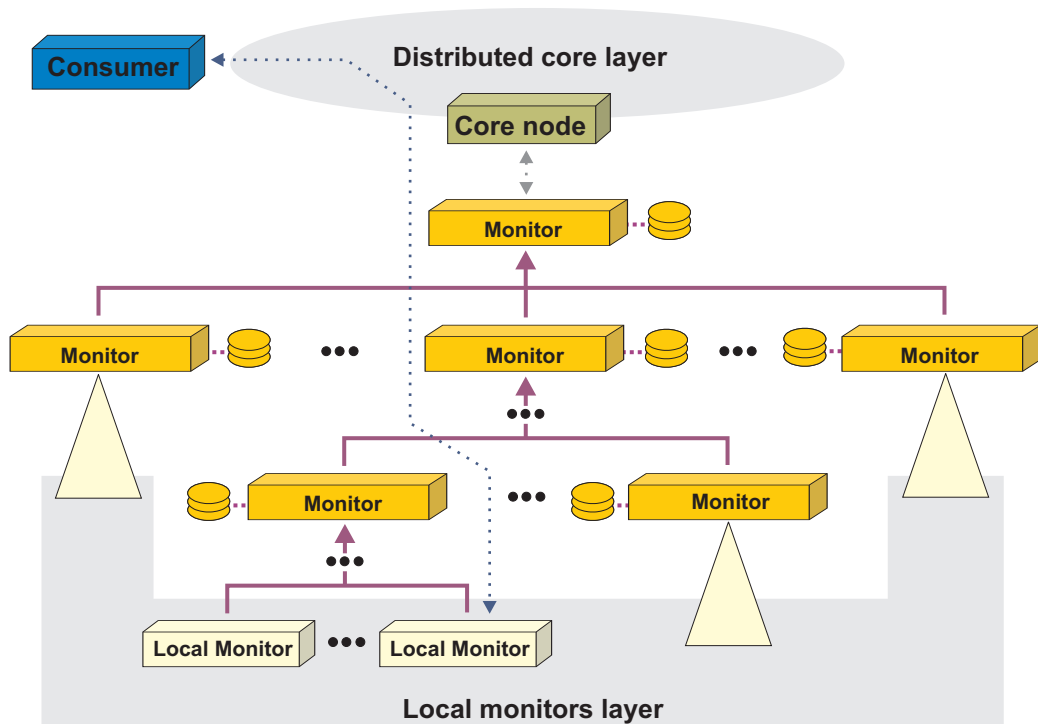


Figure 4.3: The Toytle architecture hierarchical connections layer. Monitors form an N-level hierarchical structure. Consumers cannot directly access nodes at this level. Monitors can store important data in local databases, often of fixed size.

Simple filtering should be performed at this level. By simple we understand filtering characteristics with thresholds and data correlation. Filtering with thresholds is done by setting certain limits, called *thresholds*, such that if a characteristic goes over the threshold, it gets immediately reported. Filtering by data correlation means not only defining thresholds for any given characteristic, but also combinations of thresholds for combinations of characteristics (e.g. condition10 is that CPU load is less than 50% *and* available RAM is under 10% of the total RAM). Filtering by any of these means should be done on all Grid data targets (i.e. network, hosts, middleware and especially user applications).

Two different data pruning methods should be used at this level: derived data pruning, and monitoring level pruning. Pruning derived data means that data that can be derived from other available data is not transmitted through the hierarchical connections layer. Pruning according to a specified monitoring level means that each characteristic must be gathered once the monitoring level reaches a certain amount. In any case, the low-intrusiveness requirement is that generated traffic for data transfers should be less than 1% of the theoretical bandwidth of the link (e.g. 0.1Mbps for a 10Mbps Ethernet connection, 1Mbps for a 100Mbps Fast Ethernet connection, and 10Mbps for a 1000Mbps Gigabit Ethernet connection).

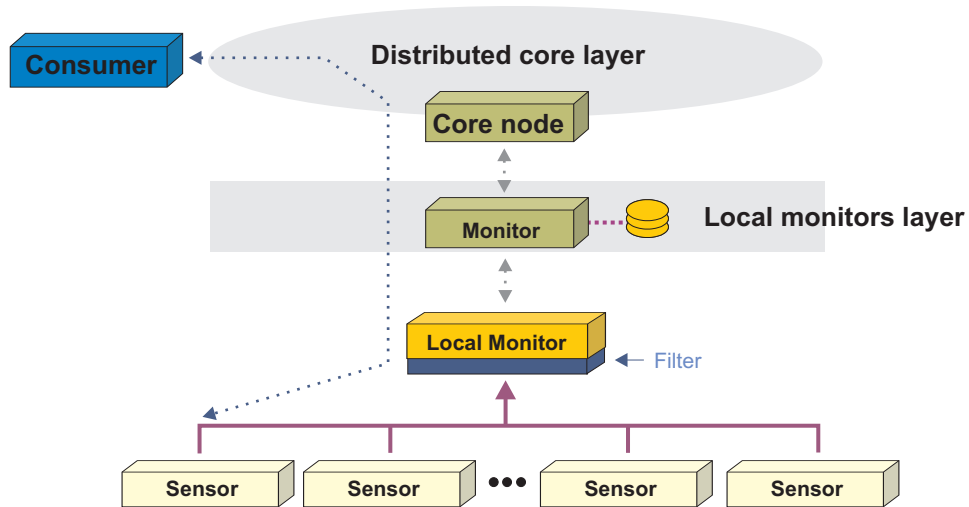


Figure 4.4: The Toytle architecture local monitors layer. Local monitors and sensors form a 1-level hierarchical structure. Consumers cannot directly access nodes at this level.

The sampling rate of this level should be of at least 10samples/second . This targeted sample rate has been derived from the normal local area network round-trip time (the time needed for a single packet to get from a source to a destination and back), which is nowadays just below 0.1s . A recommended sample rate should be, however, around 20samples/second .

This layer sends data to the upper layer in standard data formats, like XDR and XML. We emphasise that a compact and portable data transfer standard, like XDR, must be employed at this level, because of the sheer size of the monitoring data.

4.3 Target applications

The Toytle monitoring architecture is well-suited for dynamic application distributed over wide areas. Such applications require that the monitoring system is especially designed for fast incoming data, and has special requirements for data representation. The low-intrusive factor, regarding the monitored resources level, is critical. As such systems grow in size, scalability is a must also for the monitoring systems.

A special favoured case is the physical processes control in widely- and med- distributed environments, such as control Grids. In addition to requirements regarding scalability, data formats, intrusiveness level and some others, these applications require that the monitoring services are offered in the GGF GMA conceptual format. The Toytle monitoring architecture has been specifically designed with these application in mind, and fulfills these requirements.

4.4 Toward implementing the Toytle architecture

This section presents our early tries toward implementing the Toytle architecture. The selected approach is inexpensive, i.e. we try to use as much as possible existing modules or even complete monitoring tools, and bottom-up, i.e. we start from the bottom-most layers and continue to the top layers of the Toytle architecture.

4.4.1 The *inexpensive* approach

The large number of already available monitoring tools, many of high-quality with respect of their functional targets, and published under open-source licenses, makes attempts to adapt these tools to the needs of the Toytle architecture very convenient. Indeed, instead of building *from scratch* modules for monitoring Grid data targets (hosts, network, middleware and user applications), the focus could be shifted toward other important aspects (e.g. data transfer protocols).

We have selected two monitoring tools, namely Ganglia [23] and NET-SNMP [56], for tests regarding a possible integration of these tools within an implementation of the Toytle architecture. Ganglia has two complex components, namely the cluster component and the hierarchical component, which could be adapted, respectively, to the local monitors and hierarchical connections layers. The Ganglia monitoring tool has been extensively used in large cluster projects, with over 500 of them using Ganglia services now. NET-SNMP is an open-source implementation of the SNMP v1, v2 (many flavors, including v2c) standards and of SNMP v3 proposal (believed to be accepted as a standard in the coming years). The SNMP standards are implemented in a wide-range of network-connected tools, such as routers, switches, computers attached to the networks and many others, offering services that are located at the local monitors layer in the Toytle architecture.

4.4.2 Experiments toward implementing local monitors layer

Data targets The Ganglia monitoring tool cluster component can measure any of the four Grid monitoring targets: hosts, network, middleware, and user applications. Over 20 default characteristics are offered for the first two categories. A simple mechanism can be used to update values coming from all four data categories to the monitoring system. The NET-SNMP monitoring tool can also measure any of the four Grid monitoring categories. The default selection of characteristics is in fact the huge description set from the MIB-II standard, plus some vendor-specific extras. However, implementing extensions to the default sets is rather cumbersome.

Sampling rate The Ganglia monitoring tool cluster component speed was more than enough for both the required and the recommended sampling rates (also see section 4.2.3). The NET-SNMP monitoring tool did not always provide the needed performance for the required sampling rate, and we could not reach with it the recommended sampling rate at all (see fig.4.5).

Data federation The Ganglia cluster component can gather data from all other similar tools in the cluster, and federate the result on an *DTD + XML* format. This makes the

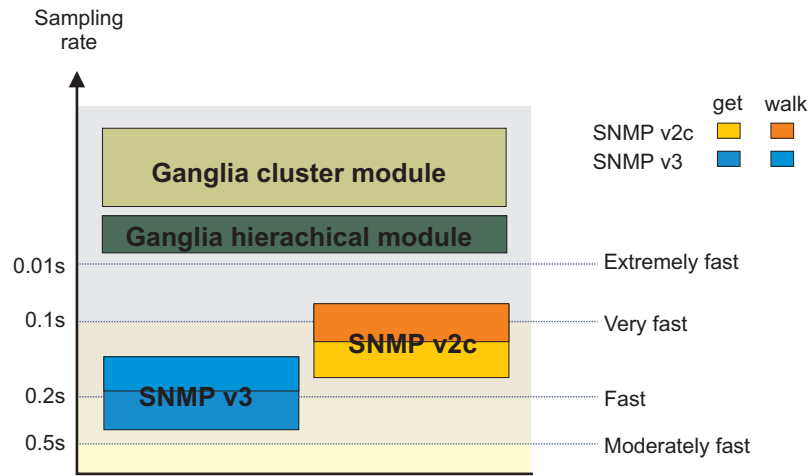


Figure 4.5: Ganglia vs. NET-SNMP speed comparison.

data transfer protocol standard, structured, and self-descriptive, but also generates increased traffic toward the hierarchical connections layer. The NET-SNMP monitoring tool operates in the same fashion with clients within and outside the cluster.

Intrusiveness The Ganglia cluster component is very lowly intrusive (see fig.4.6). The required bandwidth inside the cluster is lower than the lowly-intrusive limit for Fast Ethernet (therefore, for all LAN technologies with speeds higher or equal to 100Mbps). The NET-SNMP monitoring tool offers two methods of acquiring data: by SNMP get and walk commands. The get command can be seen as a *get-one* command, while the walk command can be seen as a *get-many-closely-related* command. Using the *get* command, the NET-SNMP monitoring tool perform marginally better than Ganglia’s cluster component and falls too in the Fast Ethernet lowly-intrusive class. Using the *walk* command, the NET-SNMP monitoring tool becomes too intrusive even for Gigabit Ethernet. The computing power required for both tools was minimal and can be disregarded.

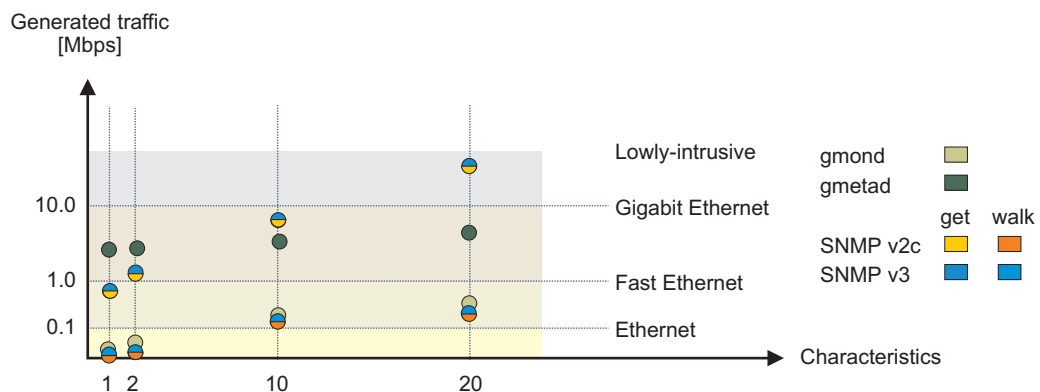


Figure 4.6: Ganglia vs. NET-SNMP generated traffic comparison.

Fault tolerance The Ganglia cluster component offers some degree of fault-tolerance, with the ability to share data between all cluster nodes and to temporarily store data in memory caches. The NET-SNMP monitoring tool offers no fault protection, but detects and reports failures.

4.4.3 Experiments toward implementing hierarchical connections layer

Sampling rate The Ganglia monitoring tool hierarchical component speed was more than enough for both the required and the recommended sampling rates (also see section 4.2.3).

Data federation The Ganglia hierarchical component can gather data from all nodes below it in the hierarchy, be they hierarchical or cluster components, and further federate the results. Data is acquired/sent in a *DTD + XML* format. Summaries can also be federated upwards, also in *XML* format.

Intrusiveness Due to the XML standard used for data transfers, The Ganglia hierarchical component is intrusive (see fig. 4.6). The required bandwidth inside the cluster is higher than the lowly-intrusive limit for Fast Ethernet (therefore, for all LAN technologies with speeds higher or equal to *100Mbps*). The situation changes for a Gigabit Ethernet connection, for which the Ganglia hierarchical component is lowly-intrusive. Also, the computing power required to parse the incoming data is important, more than 40% CPU Load being reported on a Pentium IV machine, with 3 lower-level nodes sending data to it (see [38]). To summarize, the hierarchical component of Ganglia cannot be considered lowly-intrusive.

4.4.4 Summary

The early experiments demonstrated that from the selected tools, only Ganglia seems suitable for an easy integration within a Toytle architecture implementation. Also, the experiments showed that:

- The data federation must employ a compressed, yet standardized, data format for the transfer part. We envision an XDR channel for each architectural level;
- Having restricted sets of monitored characteristics is critical; SNMP protocol implementations must conform to searching through a complete database of characteristics, which leads to severe performance hits;
- The user data definition must facilitate further data summarization and forwarding; Ganglia offers a very simple tool for updating user values, but has not been conceived for an extensive use of this feature and lacks many characteristics that would allow true application-oriented monitoring;
- The mechanisms adopted by Ganglia to facilitate fault-tolerance, self-managed multicast channel and transparent heart-beat signal, are indeed well-suited for our targets, at the local monitors level.

In conclusion, we envision that the local monitors layer of a first implementation of the Toytle architecture would rely on a modified version of the Ganglia cluster component, while the hierarchical connections layer of the same software milestone might be built on a modified version of the Ganglia hierarchical component.

4.5 Related work

Much work in the distributed and Grid monitoring area lead to many quality tools and architectures. However, we are not aware of any of these tools to fully qualify for the case of control Grids.

MonALISA + Ganglia We find this combination as a very good representative of the hybrid approach towards monitoring tools. The current MonALISA monitoring tool [52] has a flat dispersed structure, but the next version will have to be categorized, according to the authors, into the flat connected category. The monitoring service itself allows for high-speed data acquisition (as much as 3000 samples/second, according to the authors) and summarization. The MonALISA monitoring package offers the user, besides a reliable monitoring core, a set of tools for data visualization and monitoring control. The main concerns about MonALISA are that it is too resource-intensive and that data correlation cannot be easily done. This monitoring tool makes use of the existence of other, open, monitoring tools, such as Ganglia. The Ganglia monitoring tool [23] has hierarchical N-level structure. The leaf nodes within a cluster can be connected in a flat near cross-bar structure, through the use of multichannel protocol. The system is very fast and can monitor any of the four Grid data types. However, as users can connect to any of the hierarchical nodes, the XML data presentation format is employed, to the expense of increased data traffic. Also, the Ganglia system cannot filter nor prune the middleware or user applications data.

This set of tools inspired much of our work. Therefore, a list of main differences between this combined systems architecture and Toytle is presented below:

- fault-tolerance options (*one is enough to work* for Toytle core *v. by-hand* for MonALISA);
- different connection strategy at the core level (tighter for Toytle *v.* a possible loose connection for MonALISA);
- different strategies for monitoring process control (monitoring levels for each of the four Grid data types and monitoring characteristics definitions for Toytle *v.* monitoring modules dynamic loading);
- different choice of available data presentation formats (portable, standardized and compact or standardized at any level and XML/DTD self-defining standard at the core level for Toytle *v.* proprietary for MonALISA and XML/DTD self-defining standard for the hierarchical component of Ganglia, plus portable, standardized and compact for the cluster component of Ganglia);
- different low-level filtering strategy (configurable thresholds and correlated data for each of the four Grid data types for Toytle *v.* configurable thresholds only for network and hosts data for Ganglia).

- different user categories (control Grids administrators and applications developers for Toytle *v.* wide-area distributed systems, with long-term use applications).

NetLogger The Networked Application Logger (short, NetLogger) [57] has a hierarchical 1-level structure. It allows the monitoring of all four Grid data types, namely hosts, network, middleware and user applications characteristics. The NetLogger monitoring is based on time-stamped events, which makes this tool very important in profiling and tracing applications. The package also includes a visualization tool, which has been designed for data correlation, particularly performance bottlenecks. Monitoring process control can be done easily, through the use of a monitoring level, global for the whole Grid. The Universal Logger Message (short, ULM) [1] and the XML are supported as standardized data presentation formats, with other, proprietary, compact formats being supported as well. High-speed data-sampling and an interesting use of the concept of piping data make this tool a strong contender for control Grid application monitoring, tracing and profiling. However, issues like fault-tolerance, the lack of a cohesive monitoring core, and the absence of a portable, standardized and compact data presentation format make this tool less attractive for our target user categories.

GRM/PROVE [40] This combination has a hierarchical 1-level structure. The monitoring system can be used for parallel message-passing based applications tracing and profiling. The complete package, including the Mercury Grid monitoring system, can be used to trace applications on complete Grid systems, and also to identify performance bottlenecks and to visualize monitoring data. However, this system lacks a lowly-intrusive multi-level monitoring hierarchy, as well as a number of data filtering and presentation features. Therefore, its use in the control Grids context would be severely limited.

NWS The Network Weather Service (short, NWS) [75] monitoring tool produces short-term performance forecasts, based on statistical data. Periodical data describing end-to-end TCP/IP performance, and available computing and storage resources is obtained from system's sensors, in a hierarchical, 1-level structure. The NWS monitoring system, though very used for general Grid computing systems, cannot be used with *bursty* applications and is not suitable for fast changing data sets, such as control Grids applications.

Other systems generally used in the case of Grid computing, such as MDS [47], GridICE [27], R-GMA [59], Nagios [54] or MapCenter [46] are too resource-demanding and slow-paced to be effective for our target user categories.

Chapter 5

Toytle test case: a mobile robotic application

5.1 Introduction

The test case envisioned for the Toytle monitoring system comes from the partially autonomous robots research field. The specific control grid application deals with advanced mobile robot control. The robotic challenges include signal processing, path planning and movement, and obstacle detection [65]. The Grid computing challenges include high performance distributed computing, collaborative laboratory framework and software fault-tolerance. The monitoring challenges for such systems are plentiful, with the most important discussed in section 5.5.1.

5.2 Project history

1999-2003 This project started back in 1999, when Stephane Vialle, from Supelec, and Ali Siadat, from ENSAM, created the project general framework and established the first milestones. Several people were involved with developing the robotic application, amongst which the author of this report, who dealt with high-performance parallel and distributed computing solutions.

2003-present The project continued with a control Grid integration. For this, the DIET GridRPC solution was chosen, and the work was successfully completed [60] by Fabrice Sabatier, under direct guidance and supervision of Stephane Vialle.

5.3 Robotic application general framework

Consider robots navigating in industrial and office buildings. Because of the indoor environment, there are no guarantees for radio wave reception, so GPS usage is impossible. In addition, robots are being placed in changing environments: doors may be left open, chairs and other small objects may be moved, even furniture and decorative plants may have their locations changed. This is why an objects mapping database cannot be reliably used, though it could prove beneficial into estimating the robot position. Instead, artificial landmarks,

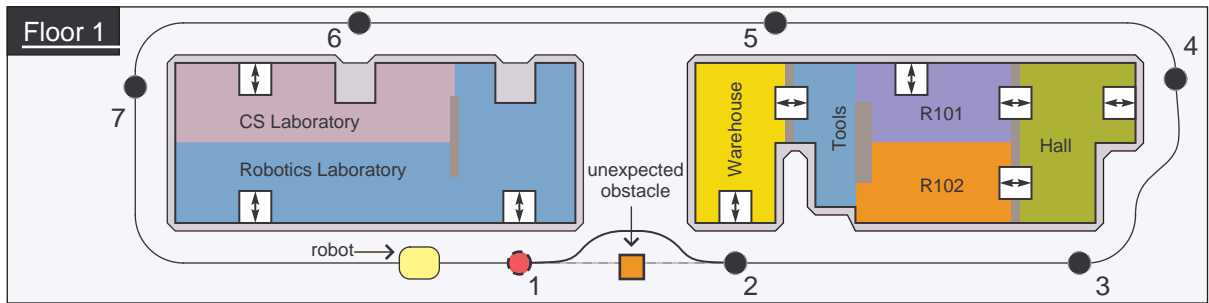


Figure 5.1: The complete mobile robotics application². An unexpected obstacle is observed (point 1). The old trajectory (dotted line between points 1 and 2) is replaced with a new one (curved trajectory between points 1 and 2). New self-localization routines are performed from time to time (points 3-7).

placed in the environment at known locations, are used. Robots can detect these landmarks and self-localize with an adapted triangulation algorithm [45]. The robot may then compute a theoretical trajectory to reach a final position and follow that trajectory. Checking the environment while moving is also an option. While following the theoretical trajectories, robots use infra-red sensors and edge detection in camera acquired images to dynamically detect unexpected obstacles (see figure 5.1, point 1). If an obstacle is detected, a new self-localization allows to accurately point out the obstacle and to upgrade the map. Then a new trajectory is computed (see figure 5.1, the curved trajectory between points 1 and 2 replaces the old trajectory, the dotted line between the same points), and the process repeats until the goal is reached (in a surveillance mission the goal would be to stroll around the building for as much time as possible, with some time used to recharge at a battery charging base). Other self-localizations allow to compensate error on final or intermediate positions, when following long trajectories (see figure 5.1, points 5 and 6).

5.4 Robotic system

5.4.1 Hardware

The hardware features of the working environments include the robot, the computer and the communication network. Due to its low on-board computing power, the robot is controlled by an external computer through a serial link. When performing the self-localization routine, several computers may be used to process the acquired data, through LAN, ATM or Internet communication networks. Being given that this system is to be used by people participating in this interdisciplinary and multi-organizational project, the heterogeneity and the complexity of the distributed hardware infrastructure must be covered by the overlaying Grid software.

The physical resources are currently distributed on two sites: one laboratory at Supelec, France, and another at the Salerno University, Italy. A third laboratory, situated at Politehnica University of Bucharest, Romania, is expected to be added to this project within a couple of months (see fig.5.2).



Figure 5.2: The Grid deployment. Note the three laboratories, with the central figure being the Supelec laboratory, and the two satellites being UniSa, Italy, and PUB, Romania.

5.4.2 Software

5.4.2.1 The robotic system modules

The robotic system makes use of several control components:

Navigation This module is responsible for computing an optimized path for the robot to follow. It must allow the robot to safely navigate between two given points, while avoiding sliding as much as possible

Self-localization This module is responsible for computing the robot position within the test area. It must deal with image acquisition, image filtering (in this case PSimilar landmarks [62] matching) and position adaptive interpolation. This module is very computing-, thus time-, consuming and must be executed in parallel on several machines for optimal performance.

Obstacle detection This module is responsible for detecting obstacles that block the robot. It must deal with image acquisition and processing.

Central control this module is responsible for employing all other modules, in order to control the mobile robot in order to fulfill user's requests.

5.4.2.2 The Grid software architecture

The Grid software architecture (see fig.5.3) consists of four components: the *Grid application*, the *high-level robot control Grid services*, the *low-level robot services* and the *Grid middleware services*.

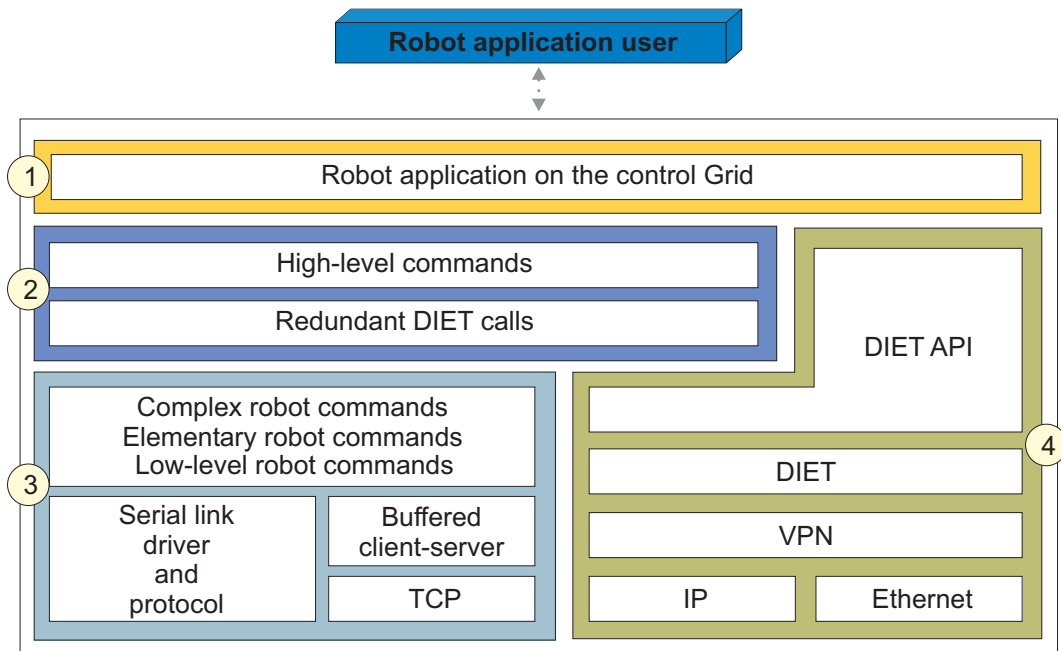


Figure 5.3: The test case Grid software architecture⁴. This architecture contains 4 software components: (1) the Grid application; (2) the high-level robot control Grid services; (3) the low-level robot services and (4) the Grid middleware services.

The *Grid application* level allows the user to issue high-level commands towards the physical robot. Depending on the user's commands, redundant calls to high-level services may be issued, which ensures a degree of fault-tolerance.

The *high-level robot control Grid services* implement the high-level robot commands, on a GridRPC approach [55]. The high-level commands can call other high-level commands or low-level services.

The *low-level robot services* offer different levels of control on the physical robot, as well as buffered robot access, through a client/server model. Communication protocols over serial links or TCP links are also located at this level. There are three levels for controlling the physical robot : the complex robot commands, the elementary robot commands, and the low-level robot commands, with the latter directly using the serial link control software.

The *Grid middleware services* are built on top of GridRPC implementation: the Distributed Interactive Engineering Toolbox (short, DIET) [19], which is, in turn, built on top of a CORBA bus. The secured communication layer is ensured through an IPSEC-based [6] Virtual Private Network (short, VPN) [13].

5.5 Selection of monitoring criteria for a robot control Grid

5.5.1 Monitoring requirements for the robot control Grid

There are two main components that need to be monitored within the robot control Grid: the hardware and the software.

The hardware parts that need to be monitored are the computing and the communication resources. For example, host systems must be thoroughly monitored, while the networks need adequate end-to-end monitoring.

The software monitoring is the most important target of a system monitoring the control Grid application. It is interesting to trace the application, and also to profile each of the different Grid software components (see also section 5.4.2.2).

There are also some mixed monitoring tasks, such as mapping the application's execution on the resources, or establishing a resource usage pattern given the user requests. Also, correlations between redundant service calls and network link latency or resource failure could give hints on improved versions of this application, and also on the somewhat lower-level redundant calls control Grid middleware.

Finally, minimizing the number of monitored characteristics, thus the monitoring system interference, should be an important target.

5.5.2 Host characteristics

The host characteristics selected for the monitoring task are depicted in fig. 5.4, point (a). They include characteristics for system overview, such as the system architecture, total uptime and process load, storage resources overview, such as the total available RAM and the disk used, and for software overview, such as the operating system and the system tools status.

5.5.3 Network characteristics

The network characteristics selected for the monitoring task are depicted in fig. 5.4, point (b). They include characteristics for bandwidth description, such as the the achievable bandwidth and the available bandwidth, delay overview, such as the round-trip time between various sites, and for communication network overview, such as the number of hops between the sites.

5.5.4 Middleware characteristics

The middleware characteristics selected for the monitoring task are depicted in fig. 5.4, point (c). They include characteristics for server description, such as the current status. Other characteristics should also be used, depending on the accesibility to the middleware libraries. We estimate that at least the high-level robot control Grid services should be instrumented in order to provide the much need application profiling and tracing information.

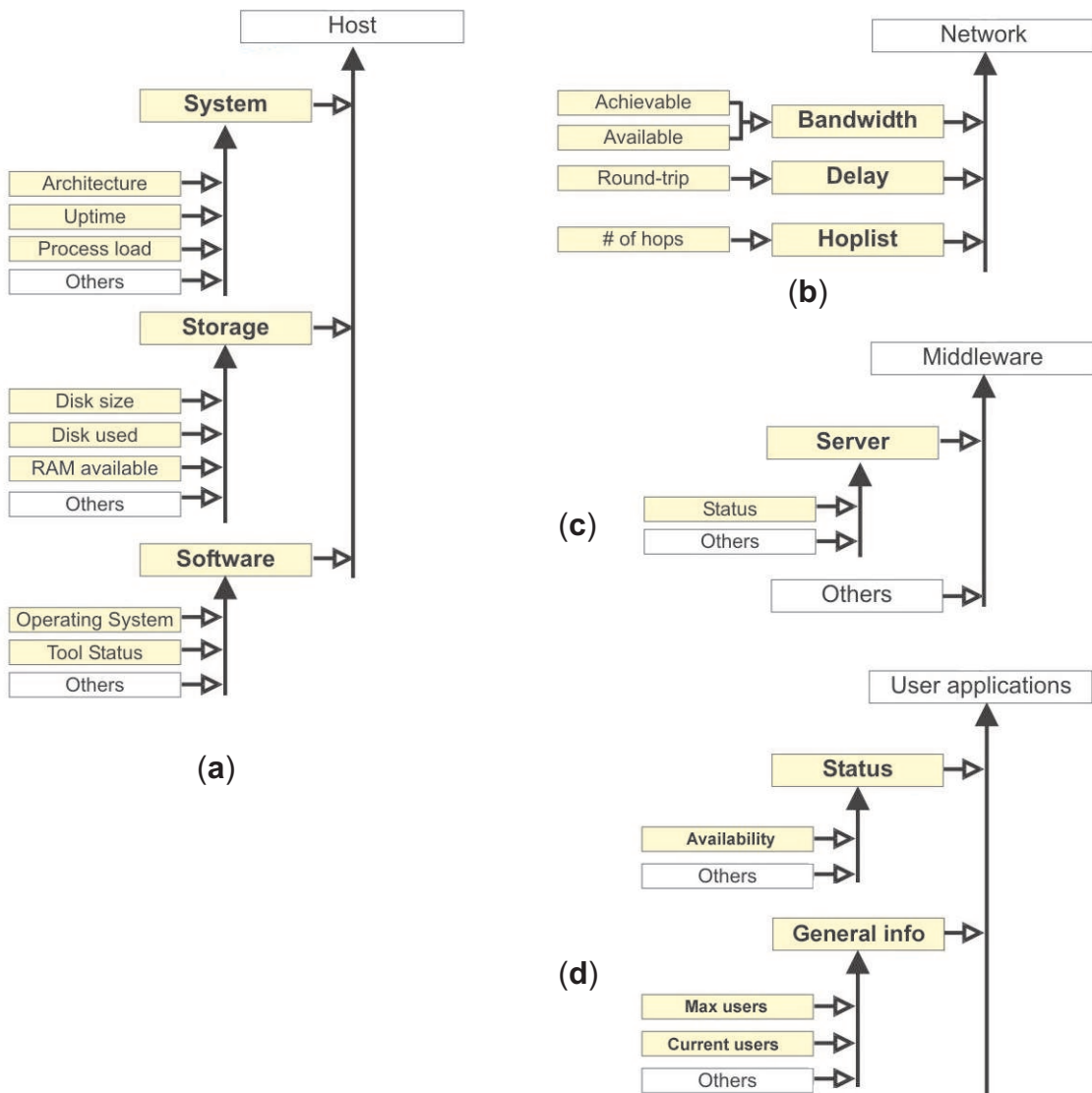


Figure 5.4: The selected characteristics hierarchies for robot control Grids: (a) the selected host hierarchy of characteristics; (b) the selected network hierarchy of characteristics; (c) the selected middleware hierarchy of characteristics; (d) the selected user applications hierarchy of characteristics.

5.5.5 User applications characteristics

The user application characteristics selected for the monitoring task are depicted in fig. 5.4, point (d). They include characteristics for application status description, such as the current availability, and for general info overview, such as the number of current application users. Other characteristics should also be used, depending on the accesibility to the user application modules. We estimate that at least the application termination percentage should be provided

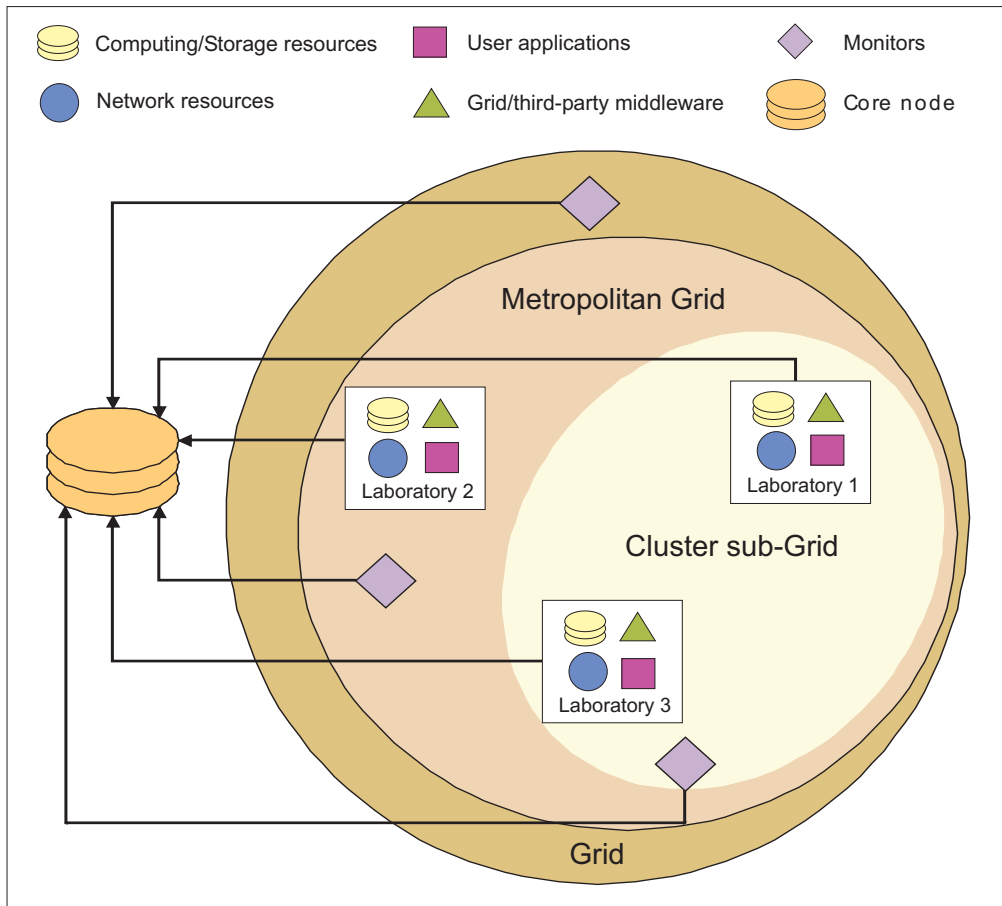


Figure 5.5: Toytle deployment within the test case environment. Note that one monitoring core node may be enough for the test case. Also, there should be monitors at responsible for cluster sub-Grids and one for each metropolitan group of cluster sub-Grids.

through adequate instrumentation, in order to provide the much need application profiling and tracing information.

5.6 Toytle deployment in the test-case environment

The Toytle architecture could be deployed in order to monitor the test case control Grid environment and application. The deployment strategy is the following (see also fig.5.5): the monitoring core should contain one node. In the case that more laboratories are added, or some local laboratory proves to be a metropolitan Grid, with several cluster-based systems, installing another core node in the metropolitan area could prove beneficial. Monitoring nodes should be installed for each cluster and for each group of clusters. Local monitors should reside on each control Grid machine.

Only core nodes require dedicated machines, while the others should run normally (that is, smoothly and according to the low intrusiveness policy) on machines also used for other tasks, or even on the control Grid machines.

5.7 Summary

This chapter has presented the reader a test case for the Toytle architecture. The test case is presented such as to emphasise the parts that require monitoring. The test case consists of a control Grid environment and one control Grid application. A selection of monitoring criteria is made to suit the test case. A deployment scheme for the Toytle architecture on the test case environment is also presented.

Chapter 6

Conclusions and future work

6.1 Conclusions

This work focused on four main issues: an overview of the current Grid computing views, a survey of the current Grid monitoring tasks, techniques, and trends, a presentation of a Grid monitoring tools taxonomy, based on their internal structure attributes, and an introduction of a novel monitoring architecture, named Toytle, aimed at monitoring control Grids applications.

The overview of the current Grid computing views presents a large number of information items, with an emphasis on the comparison of reasearch and industry view-points.

The state-of-the-art in Grid monitoring survey discusses seven key tasks and three important techniques, as well as identifying the current domain trends and downsides.

The Grid monitoring tools taxonomy uses two structural attributes of monitoring systems, namely the organizational structure and the same-level components interconnection. The taxonomy comprises seven main types, selected according to five rules governing the taxonomy. The seven types are: flat, dispersed monitoring architectures, flat, connected monitoring architectures, flat, near cross-bar monitoring architectures, hierarchical, 1-level monitoring architectures, hierarchical, N-level monitoring architectures, hybrid connected core/N-level hierarchical connections monitoring architectures, and hybrid near cross-bar core/N-level hierarchical connections monitoring architectures. The taxonomy is oriented at categorizing tools susceptible of being able to monitor control Grids. A test case from that field has been presented, and a selection of monitoring criteria for the test case has been performed.

The Toytle monitoring architecture is a hybrid near cross-bar core/N-level hierarchical connections monitoring architecture, that aims at satisfying the most important requirements of control Grids. High speed data gathering and federation, scalability, program lightness (in terms of monitored system's resource consumption), low intrusiveness (in terms of monitored system's performance degradation) and easy deployment are targetted by this architecture. The Toytle architecture's three layers, namely the distributed core, the hiarchical connection and the local monitors, ensure a good degree of modularity, while still offering the advantages of architectural compactness.

6.2 Future work

The logical step for this work is the med-term (six months to one year) development a full-implementation of the Toytle architecture. This would require exact specifications of the monitoring components operation and the development of a set of software tools.

Also for the med-term, but closing the long-term (two to four years), future, we estimate that a complete evaluation of this architecture should be performed. Since the test case presented in this report is a real-world application, all pre-requisites for this task, safe the development of the full-implementation of the architecture, have been met.

The long-term future should be a good period to seek for improvements to this monitoring architecture, both design- and development- related.

Index

- characteristic
 - definition, 16
 - network ~ definition, 16
- data
 - federation, 14
- DIET, 8
 - Client, 8
 - Local Agent, 8
 - Master Agent, 8
 - Server Daemon, 8
- federating data, 14
- Grid
 - perspective, academic, 5
 - perspective, hardware, 5
 - perspective, user, 5
 - type, collaborative, 7
 - type, computational, 7
 - type, control, 8
 - type, data, 7
 - type, distributed supercomputing, 7
 - type, high throughput, 7
 - type, multimedia, 7
 - type, on-demand, 7
 - type, service, 7
- Grid-aware, 22
- group sensor, 26
- link
 - bottom-up, 37
 - top-down, 37
- measurements
 - active, 16
 - lowly-intrusive, 17
 - passive, 16
- metacomputer, 5
- metacomputing, 4
- middleware, 5
- observation, 16
 - atomic, 16
 - sample, 16
 - singleton, 16
- sensor, 16
 - group ~, 26
 - host, 16
 - middleware, 16
 - network, 16
 - user applications, 16
- test case
 - Grid application, 47
 - Grid software, 47
 - high-level robot services, 47
 - low-level robot services, 47
 - middleware services, 47
- Toytle, 34
 - distributed core, 34, 35
 - hierarchical connections, 34, 37
 - local monitors, 34, 37

Bibliography

- [1] J. Abela and T. Debeaupuis. Univeral format for logger messages, November 1999. IETF Internet Draft.
- [2] S. Adcock. How does the Grid extend the Internet, and what is the future vision for this development? Research Support Article on CiteSeer, 2001. <http://citeseer.nj.nec.com/554105.html>.
- [3] Aesop. The hare and the tortoise, June 2004. <http://disneyshorts.toonzone.net/sources/hareandthetortoise.html>.
- [4] D. Agarwal, J.M. Gonzales, G. Jin, and B. Tierney. An infrastructure for passive network monitoring of application data streams. Report LBNL-51846, LBNL, 2003.
- [5] S. Armstrong, A. Freier, and K. Marzullo. Multicast transport protocol, February 1992. IETF NWG RFC 1301.
- [6] R. Atkinson. Security architecture for the internet protocol (IPSEC), August 1995. IETF NWG RFC 1825.
- [7] J. Bacon and T. Harris. *Operating Systems: Concurrent and Distributed Software Design*. Addison-Wesley, 2003. Chapter 29: Middleware. ISBN 0-321-11789-1.
- [8] Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajda. Comparison of representative grid monitoring tools, February 2000. MTA/SZTAKI Technical Report LPDS-2/2000.
- [9] M. Bote-Lorenzo, Y. Dimitriadis, and E. Gomez-Sanchez. Grid characteristics and uses: a grid definition, 2002. Technical Report CICYT TIC2002-04258-C03-02, Univ. of Valladolid, Spain.
- [10] H. Casanova and J. Dongarra. NetSolve's Network Enabled Server: Examples and applications. *IEEE Computational Science and Engineering*, 5(3):57–67, 1998.
- [11] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol (snmp), May 1990. IETF RFC 1157.
- [12] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Introduction to community-based snmpv2, January 1996. IETF NWG RFC 1901.
- [13] CISCO. Virtual Private Networks (VPNs), June 2004. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/vpn.htm.

- [14] K. C. Claffy and S. McCreary. Internet measurement and analysis: passive and active measurement, June 1999. CAIDA Tech Reports.
- [15] P. Combes. DIET user's manual, March 2004. v.1.0.
- [16] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design, 2nd Ed.* Addison-Wesley, 1994. p.102: XDR. ISBN 0-201-62433-8.
- [17] D. De Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt. The evolution of the Grid. Research Support Article on CiteSeer, 2002. <http://citeseer.nj.nec.com/535794.html>.
- [18] T. A. DeFanti, I. Foster, M. E. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide-area visual supercomputing. *The International Journal of Supercomputer Applications and High Performance Computing*, 10(2/3):123–131, Summer/Fall 1996.
- [19] DIET. Diet official web-page, June 2004. <http://graal.ens-lyon.fr/~rbolze/DIET/index.html>.
- [20] I. Foster. What is the grid? a three point checklist. *Grid Today ezine*, 1(6), July 2002.
- [21] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of Peer-to-Peer and Grid computing. -, (-), January 2003.
- [22] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
- [23] Ganglia development team. Ganglia official web-site, June 2004. <http://ganglia.sourceforge.net/>.
- [24] M. Gerndt, R. Wismueller, and Z. Balaton et al. Performance tools for the grid: State of the art and future. APART-2 White Paper on Grid performance analysis, APART WP3, 2004.
- [25] Globus development team. The globus heart-beat monitor (hbm), Jun 2004. <http://www.globus.org/hbm>.
- [26] Grid3 consortium. Grid3 official web-page, June 2004. <http://www.ivdgl.org/grid2003/>.
- [27] GridICE development team. Gridice, Jun 2004. <http://grid.infn.it/gridice>.
- [28] A. Grimshaw. What is a grid? *Grid Today ezine*, 1(26), December 2002.
- [29] D. Gunter. An architecture and protocols for grid performance monitoring. March 2001. DIDC Group Presentation.
- [30] D. Gunter and J. Magowan. An analysis of jtop n_i event descriptions, February 2003. GGF Discovery and Monitoring Event Descriptions WG, GGF-DAMED-01-I.
- [31] L. Heintz and S. Gudur. Definitions of managed objects for extensible snmp agents, January 2000. IETF NWG RFC 2742.
- [32] Hewlett-Packard. Hp openview extensible snmp agent 4.2 software, June 2004. Data Sheet, <http://www.managementsoftware.hp.com/>.

- [33] K. Hickman. The SSL protocol, 1995. Internet Draft RFC.
- [34] R. Housely, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure, Jan. 1999. IETF RFC 2459.
- [35] IBM. New to Grid computing panel, May 2004. <http://www-106.ibm.com/developerworks/grid/newto/>.
- [36] IDC. Butterfly.net: Powering next generation gaming with on-demand computing, March 2002. IDC e-Business case study.
- [37] A. Iosup. Parallelization and Grid integration of a mobile robotics application. Diploma Project UPB-Supelec, 2003. <http://prof.cs.pub.ro/~aiosup/>.
- [38] A. Iosup. Ganglia evaluation. Ersidp tech reports on grid monitoring, Supelec, 2004.
- [39] Jini Community. The community resource for jini technology, June 2004. <http://www.jini.org/>.
- [40] P. Kacsuk and N. Podhorszki. Security architecture for the internet protocol (IPSEC), June 2004. SZTAKI LPDS laboratory, <http://www.lpds.sztaki.hu>.
- [41] T. Kielmann. Lowly-intrusive network monitoring in the GridLab testbed, June 2003. Presented by B. Tierney at GGF Network Measurements Working Group, GGF8, Seattle.
- [42] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software - Practice and Experience*, 00:1-7, Winter 2001.
- [43] B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. White paper: A hierarchy of network performance characteristics for grid applications and services, January 2004. GGF Network Measurements Working Group, GGF8 Grid Working Document.
- [44] B. Lowekamp, B. Tierney, Les Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. Enabling Network measurement portability through a hierarchy of characteristics. *LBNL-53013 4th International Workshop on Grid Computing (Grid2003)*, 2003.
- [45] C.B. Madsen, C.S. Andersen, and J.S. Sørensen. A robustness analysis of triangulation-based robot self-positioning. *Proceedings of the International Symposium on Intelligent Robotic Systems, Stockholm*, 1997.
- [46] MapCenter development team. Mapcenter, Jun 2004. <http://mapcenter.in2p3.fr>.
- [47] MapCenter development team. Monitoring and discovery service (mds), Jun 2004. <http://www.globus.org/mds/>.
- [48] J.-P. Martin-Flatin, S. Znaty, and J.P. Hubaux. A survey of distributed enterprise network and system management paradigms. *Journal of Network and Systems Management*, 7(1):9–26, 1999.

- [49] M. Massie, B. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience, April 2004. Accepted for publication in *Parallel Computing*.
- [50] S. Matsuoka and H. Casanova. White paper for gf4 wg3: Network-enabled server systems and the computational grid, July 2000. GGF Performance WG, Grid Working Document GWD-GP-6-1.
- [51] K. McCloghrie. Management Information Base for network management of TCP/IP-based internets: MIB-II, March 1991. IETF RFC 1213.
- [52] MonALISA development team. Monitoring agents using a large integrated services architecture (MonALISA), June 2004. <http://monalisa.cacr.caltech.edu/>.
- [53] MSDN Data architecture patterns. Data replication, June 2004. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/ArcDataReplication.asp>.
- [54] Nagios development team. Nagios, Jun 2004. <http://www.nagios.org>.
- [55] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. GridRPC: A remote procedure call API for Grid computing, July 2002. GWD-I Advanced Programming Models Research Group.
- [56] NET-SNMP development team. NET-SNMP v5.1, May 2004. <http://net-snmp.sourceforge.net/>.
- [57] NetLogger development team. Netlogger, Jun 2004. <http://www-didc.lbl.gov/NetLogger>.
- [58] M. Quinson. Dynamic performance forecasting for network-enabled servers in a metacomputing environment. In *Proceedings of the Intl. Workshop on Performance, Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'02)*, 2002.
- [59] R-GMA development team. R-gma, Jun 2004. <http://www.r-gma.org>.
- [60] F. Sabatier, A. De Vivo, and S. Vialle. Grid programming for distributed remote robot control. *International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004)*, June 2004. (to appear).
- [61] F. Sacerdoti, M. Katz, M. Massie, and D. E. Culler. Wide area cluster monitoring with Ganglia. In *In Proceedings of the IEEE Cluster 2003 Conference*, pages 289–298, 2003.
- [62] D. Scharstein and A.J. Briggs. Real-time recognition of self-similar landmarks. *Workshop on perception for mobile agents*, June 1999.
- [63] R. Schoonderwoerd. Network performance measurement tools - a comprehensive comparison. Master's thesis, Vrije Universiteit Amsterdam, November 2002.
- [64] K. Shirose, S. Matsuoka, H. Nakada, and H. Ogawa. Autonomous configuration of grid monitoring systems, February 2004. Technical Report.

- [65] A. Siadat and S. Vialle. Robot localization, using p-similar landmarks, optimized triangulation and parallel programming. *2nd IEEE International Symposium on Signal Processing and Information Technology*, 2002. Marrakesh.
- [66] L. Smarr and C. E. Catlett. Metacomputing. *ACM SIGGRAPH'92*, 35(6):44–52, June 1992.
- [67] R. Srinivasan. XDR: External Data Representation standard, August 1995. IETF NWG RFC 1832.
- [68] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2 (3rd Ed.)*. Wiley, 1998. ISBN: 0201485346.
- [69] H. Thane. Monitoring, testing and debugging of distributed real-time systems, May 2000. MRTC PhD Thesis 00/15, ISRN KTH/MMK/R-00/16-SE.
- [70] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany. White paper: A grid monitoring service architecture, February 2001. GGF Performance WG, Grid Working Document GWD-GP-6-1.
- [71] B. Tierney and D. Gunter. NetLogger: A toolkit for distributed system performance tuning and debugging, 2002. Technical Report LBNL-51276, LBNL.
- [72] Brian Tierney, Brian Crowley, Dan Gunter, Mason Holding, Jason Lee, and Mary Thompson. A monitoring sensor management system for grid environments. In *Proceedings of the IEEE High Performance Distributed Computing conference (HPDC-9)*, pages 97–104, 2000. LBNL-45260.
- [73] W3C. eXtensible Markup Language (XML), June 2004. <http://www.w3.org/XML/>.
- [74] T. Wilkins. The tortoise and the hare, June 2004. 1935 Academy Award (Cartoons), <http://disneyshorts.toonzone.net/years/1935/tortoiseandthehare.html>.
- [75] R. Wolski and M. Swany. Network Weather Service (NWS), Jun 2004. <http://nws.cs.ucsb.edu>.

Appendix A

Memos, reports, and dissemination

Dissemination

- [1] **A.Iosup**, N.Tapus, *Toytile: A Monitoring Architecture for Control Grids*, IEEE Romania International Symposium on Automatic Control and Computer Science (SACCS), Iasi, Romania (to be presented in October 2004).
- [2] C.Cirstoiu and **A.Iosup**, *Monitoring and Control for Dynamic Processes using GRID Solutions* (in Romanian, original title *Monitorizarea si Controlul Proceselor cu Evolutie Rapida Folosind Solutii GRID*), 1st prize at SC-UPB'04 Symposium, May 2004.
- [3] **A.Iosup**, *Grid, the Final Frontier (2004)* (in Romanian, original title *Grid, ultima frontiera (2004)*, *Informatica* magazine (accepted for publication June 2004).

Technical memos

NET-SNMP install a technical memo that can be used as an installation-guide for the **NET-SNMP** monitoring tool.

Ganglia install a technical memo that can be used as an installation-guide for the **Ganglia** monitoring tool.

Technical reports

NET-SNMP tests a technical report on the performance of the **NET-SNMP** monitoring tool.

Ganglia tests a technical report on the performance of the **Ganglia** monitoring tool.

copyright ©2004 Alexandru Iosup