Experiences in Designing a Modular Resource Monitoring System

Augusto Ciuffoletti
Dipartimento di Informatica – Univ. di Pisa
Tiziana Ferrari
INFN/CNAF – Bologna

Abstract

Resource monitoring plays a vital role in the Grid. One of its basic features is the capability to deal with with the heterogeneous resources available in the Grid, and, consequently, the possibility to support both new and existing monitoring tools in a modular way. We start from this observation to discuss what are the components of a Grid Resource Monitoring System and the requirements they need to satisfy. We then show, as a case study, how the NWS *monitoring tool can be integrated with a* LDAP*-based directory service. We use the results of this case study to outline a modular architecture for a generic Resource Monitoring System. A proof of concept implementation is finally presented.*

1 Introduction

The deployment of a distributed computing platform requires the availability of a *resource monitoring service*: both the middle-ware and user applications should be able to query such service, and adapt their behavior to resource availability.

We study an architecture that integrates various monitoring tools whose observations are made uniformly available to a number of heterogeneous applications in a Grid environment.

As proposed in the GGF Working Document [1], the architecture of a Grid Resource Monitoring System is based on three main components:

- a **consumer** that uses resources availability observations,
- a **producer** that collects and provides resource availability observations,
- a **directory service** which is in charge of granting access to **producers** selected according to the attributes specified by the **consumer**: **consumers** credentials are checked during the data search phase, and the address of the searched **producer** – if any – is delivered in the response message.

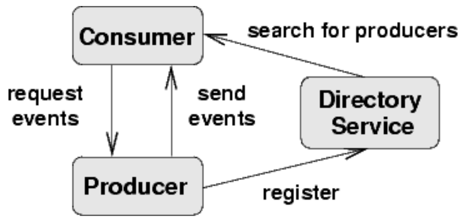


Figure 1. A monitoring architecture ([1])

This architecture emphasizes the distinction between management of frequently changing observational data, in the **producer**, and of slowly changing organizational data, in the **directory service**.

A Grid Resource Monitoring System should:

- provide a facility to supervise a complex set of resources, and in particular to start, stop, and track monitoring activities on remote sensors;
- exhibit fault tolerance properties, for instance to survive after the partition of the system, or the failure of a control node;
- minimize the impact of the monitoring activity on the monitored resource;
- organize the outcome of the monitoring activities in an organized and searchable structure, enforcing the required security levels;
- provide users with significant statistics, not bare measurements;
- allow seamless integration of heterogeneous monitoring tools;

- scale well with the extension of the monitored system and with probe frequency;

NWS [2] is a monitoring tool that already meets many of the above features. We suggest an approach aiming at strengthening scalability, fault tolerance and security properties of NWS, which is based on the integration with LDAP. The proposed method integrates the remote control and query capabilities of NWS with the expandibility and robustness typical of LDAP-based directory services like GLOBUS MDS [3]. We illustrate our idea through a "proof of concept" implementation.

The paper is organized as follows: Section 2 illustrates the most important characteristics of NWS and LDAP, the two tools for our case study, and shows how a collection of sub-systems can be used to reliably monitor the Grid; in Section 3 we discuss the organization of the LDAP tree needed to represent such Resource Monitoring System; Section 4 introduces an abstract layout aiming at integrating heterogeneous tools; Section 5 illustrates a *proof of concept* implementation for our case study; Section 6 concludes the paper by summarizing the lessons learned while implementing the prototype, and the design characteristics of a general software architecture for Resource Monitoring.

2 Complementary aspects of NWS and LDAP

The NWS resource monitoring tool [2] is designed to supervise resource availability in a Grid environment.

A NWS **system** consists of **sensors**, that produce resource availability observations, **memories**, that store resource availability observations, and **forecasters**, that process resource availability observations. Individual components - that are implemented as a Unix process - are distributed among NWS system hosts. Each NWS system is supervised by a **nameserver**.

We have identified the following weaknesses in the NWS design:

- a single point of failure and a communication bottleneck, namely the **nameserver**, that limits data availability. In fact access to the resource availability observations relies on the correct operation of this component, that cannot be replicated;

- no support for an authentication mechanism that controls the registration of a new NWS component: in particular, any host can register itself as a source of resource availability observations.

- no support for authentications mechanism to protect resource availability observations: any user can obtain such information using the appropriate communication protocol from any host;

These points justify the conclusion that NWS is not fully scalable, and exhibits both fault tolerance and security leaks.

To address some of these problems, we propose to split the monitoring domain into distinct NWS **systems**: each subsystem is controlled by a **nameserver**, and an LDAP-based directory service integrates the access to the **nameservers**. The integration is carried out with an eye to the peculiar capabilities of the tools of choice:

- LDAP is appropriate to implement fault tolerance and security of seldom updated information;

- NWS is appropriate to provide access to continuously changing data (through **memories**), to control the monitoring activity (through the **nameserver**), and to pre-process observations (through the **forecasters**).

In detail, the LDAP directory contains references to available observations, which consist of a description of the observed characteristic and of a pointer to the location where the observation is stored.

The rationale behind this design strategy is to decouple data access issues from monitoring issues: monitoring activity is searched through the LDAP directory, but access to observations is through the monitoring tool itself. Since data recorded in the LDAP directory are seldom updated, its adoption is well motivated. In addition, introducing directory replicas improves the robustness of the Resource Monitoring System.

This strategy especially fits the integration of complex monitoring tools like NWS, that offer a efficient observation retrieval interface (like the one provided by the **forecasters**), but can also be adopted for less complex tools: the advantage is that with a single directory structure, observations from different monitoring tools can be accessed.

The**producer** in Figure 1 corresponds to a NWS **subsystem**, which represents the monitoring activity carried out by the NWS components in the NWS system.

Some of the components of a NWS system have the role of **sensors**. The execution of a number of monitoring processes generates different kinds of resource availability observations. Such observations are stored by **memory** components, which may run on possibly distinct hosts. In order to obtain these observations, the **consumer** addresses its request to the NWS **nameserver**, which transparently provides the client with a binding to the appropriate **memory**. Finally, the

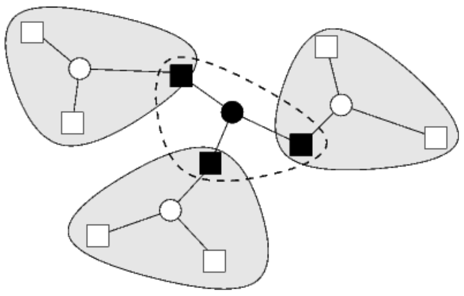


Figure 2. A hierarchy of NWS systems

memory returns the desired observations to the **consumer**.

Distinct NWS systems can be organized hierarchically to reduce the traffic produced by monitoring tools and consequently increase scalability. In Figure 2 we depict such case: peripheral NWS systems (shaded regions) have at least one sensor (black squares) in a backbone system.

Sensors within the same NWS system run a full mesh of internal monitoring sessions; in particular, backbone monitoring only involves sensors belonging to the backbone and to a peripheral system. The **directory service** will route measurement requests from **consumers** to the appropriate nameserver (circles in Figure 2). This model can be further nested, as suggested in a similar approach (although based on the R-GMA [4]), implemented in the framework of the DataGrid project [5].

Such organization exhibits an acceptable level of fault tolerance, as the single point of failure represented by the **nameserver** is removed. In fact, instead of implementing a single NWS system, the set of **sensors**, **memories** and **forecasters** is split into distinct NWS **(sub)system**. Seamless access to the information, independently provided by the **nameservers**, is achieved through a directory service, which enforces security policies. An authorized *consumer* is provided with a reference to the **nameserver** that handles the access to the resource availability observations. The LDAP directory can be easily replicated, since its contents are mostly static, while the failure of a NWS nameserver rules out the availability of the observations of the related sub-system.

3 LDAP Directory Information Tree for NWS data access

The **directory service** in Figure 1 provides the references needed to access resource availability observations stored in the NWS system, and consists of a distributed LDAP Directory Information Tree (DIT) that is designed following the GGF guidelines specified in [1]. The DIT contains a node for each NWS system. The corresponding objectclass is called `NWSeventProducer`, and includes attributes that indicate the **nameserver** transport address.

The `eventInstance` objectclass referenced in GGF document roughly corresponds to an **activity** in the NWS architecture. We introduce a subclass `NWSeventInstance` that extends this class. This kind of node is an immediate subordinate of a `NWSeventProducer`, and contains an attribute describing the **name** that NWS associates to an **activity**. The attributes of this kind of entry in the LDAP tree should be consistent with the corresponding registrations kept by the NWS **nameserver**.

Objectclass `NWSeventInstance` is intended as a subclass of objectclass `eventInstance`. This type of entry does not contain volatile observations, but just a reference to the corresponding NWS **sensor**. They are the leaves of our DIT, since objectclass `elementInstance`, as defined in Table 3 of [1], does not appear in our architecture.

In Figure 3 we show part of the DIT associated to our experimental NWS system. For each `objectclass` only a few relevant `attributes` of the directory entry are displayed.

As outlined above, the `eventProducer` objectclass introduced in the GGF document should be complemented with further attributes, in order to provide the client with the information needed to reach the observations collected by NWS: in Figure 3 there is an instance of a `NWSeventProducer` entry. In the same figure we also show the content of a `NWSeventInstance` entry: such attributes should be consistent with the corresponding **activity**, whose registration is kept by the **nameserver**.

Figure 4 shows the evolution of the standard GGF architecture from Figure 1, when a NWS-based resource availability monitoring system is implemented by two NWS **systems** – NWS *system A*, and NWS *system B*. Two distinct **nameservers**, one for each NWS **system**, keep record of the resource availability observations provided by that **system**, and a single LDAP DIT provides a homogeneous access to all observations.

Consumers produce two kinds of queries, depending on whether they interact with an LDAP server or a NWS **nameserver** (see Figure 4):

- *ldapsearch* queries: they ask LDAP for the identity of the **nameserver** controlling the monitoring of a given resource – *freeMemory*, a NWS keyword, in the example of Figure 4 – on a given host.

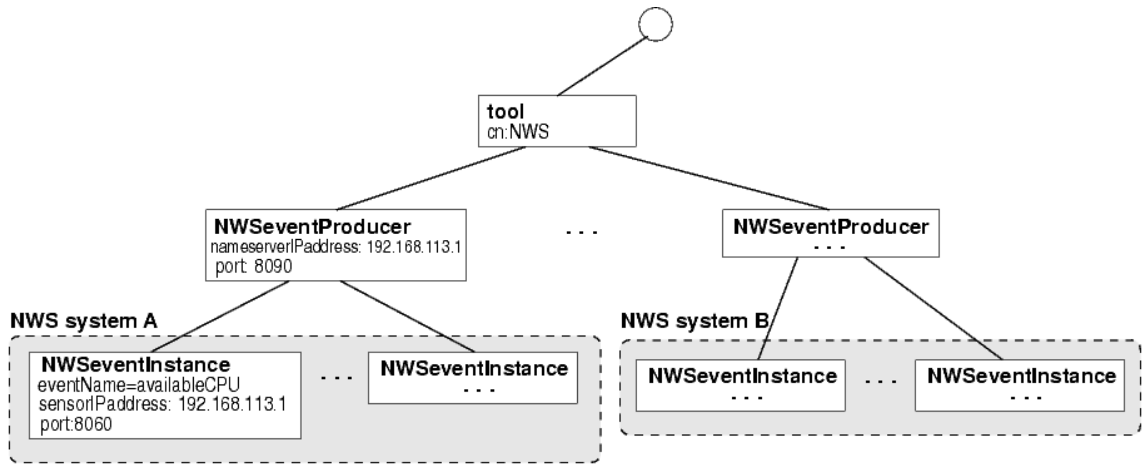


Figure 3. The Directory Information Tree of the Directory Service

In Figure 4 the `ldapsearch` command returns a table with the `nameserverIPaddress` attributes of the registration, presumably unique, that matches the pattern given as first parameter of the command.

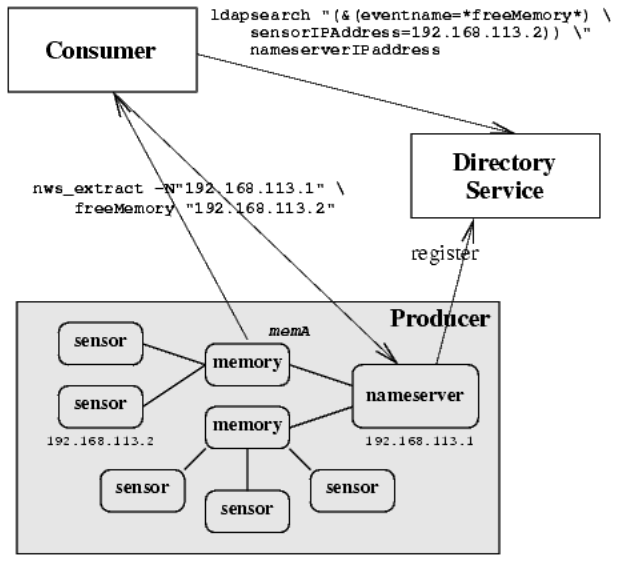
- *nws_extract* queries: they ask the NWS nameserver whose IP address has been obtained from the previous step, using as parameters the sensor address and the desired observation.

Commands in Figure 4 might be replaced by calls to functions of an API library for LDAP and NWS, respectively.

4 Monitoring software architecture

Figure 5 is a detailed view of the architecture from Figure 1 that describes the proposed internal structure of *consumers*, *producers*, and of the *directory service*. Note that such architecture does not target a specific tool, but it can be integrated with any tool that conforms to some standard interface. New monitoring tools are treated as "plug-in" modules, which offer a standardized API to both the *consumers* and the *directory service*.

An *application* makes use of resource availability observations. The lookup and retrieval of the observations is obtained using the functions offered by an API. The *interface library front end* implements the functions offered by the API, that return a reference to a running instance of a monitoring tool. The *interface library back end* submits a *query* to the referenced tool instance, using the protocol which is appropriate for the tool.



```
ldapsearch "(&(eventname=*freeMemory*) \
    sensorIPAddress=192.168.113.2)) \"
    nameserverIPaddress
```

```
nws_extract -N"192.168.113.1" \
    freeMemory "192.168.113.2"
```

Figure 4. Example of a monitoring architecture consisting in a NWS subsystems accessed through an LDAP directory service

The *monitoring tool controller* coordinates monitoring activities: in our case study this corresponds to a NWS system. The *monitoring tool controller* registers itself into the LDAP directory using a *resource registration protocol.*

We map the above architecture onto the one of our case study: the *data transfer* session involves the **nameserver**, which performs nws_extract operations that address a **memory** component and optionally a **forecaster**. The *monitoring interface library back end* is implemented by the commands that are offered as

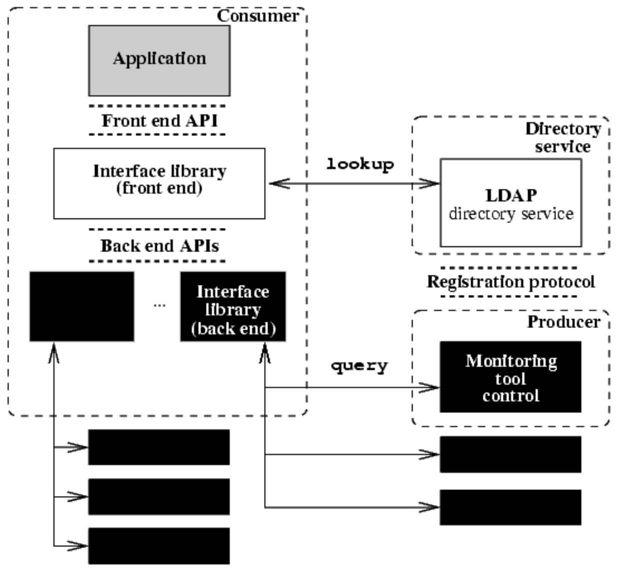


Figure 5. The software architecture of a Grid Resource Monitoring System integrating different Grid Resources Monitoring Tools

part of the NWS. The *monitoring interface library front end* is not described in our case study, since we consider only one tool.

Now we can group together some of the modules and identify three packages (indicated by different shades of gray in Figure 5) that can be reasonably implemented by distinct teams:

- *application* packages (shaded box), that contains a unique module, the *application*;
- *monitoring tool* packages (black boxes), that should contain both the *monitoring tool controller* and the *interface library back end*;
- the *directory service* package (white boxes), that should contain the LDAP *directory service* and the *interface library front end.*

One key role in the above architecture is played by interfaces: they provide standard hooks to link *applications* and *monitoring tools* through a unique *directory service.* The focus on module *interfaces* complies with OGSA intents [6], since it lays the basis for interoperability between heterogeneous platforms. In the next section we summarize the content of the interfaces introduced in Figure 5.

Interfaces

In Figure 5 we introduce three interfaces: their specifications has to be standardized to ensure the interoperability between *packages.*

The *front end API* of the *interface library* offers to the *application* a uniform access to the resource availability observations collected by the *monitoring tools.* This interface supports the retrieval of the observations, transparently from the *monitoring tool* that produced it. The interface provides a way to specify optional features (like push/pull modes), as well as security and reliability options. Observations are addressed by referencing standard metrics and other relevant parameters, like the application of estimators to historical data.

The *back end API* of the *interface library* is the specification of the interface offered by *monitoring tools.* It contains the tool-dependent functions to interact with the tool, and in particular to retrieve observations.

The *registration protocol* specifies the protocol that a *monitoring tool* follows to register itself and its components, as in the case of NWS, into the DIT.

Note that all interfaces have on one side a component of the *directory service* package: the *interface library back end* in the case of the *APIs*, and the *directory service* in the case of the *registration protocol.* The same team that implements and maintains the *directory service* package should deliver the specifications for these interfaces, in order to enforce interoperability between *applications* and *monitoring tools.*

5 Prototype implementation

The prototype is based on NWS and LDAP. Since it includes a single monitoring tool, the *interface library front end* of Figure 5 is not dealt with, while the *interface library back end* consists of the commands offered by NWS itself. We focus on the interface between NWS and the LDAP directory service, the *registration protocol* in Figure 5.

This interface is implemented by a process, running on the NWS **nameserver** host, so that each update in the registry kept by the **nameserver** triggers an update in the LDAP directory.

According to this concept, we envision two alternative implementations:

- updates on the NWS internal registry, are mirrored into the LDAP directory, or
- a NWS-oriented LDAP back-end interacts with the nws **registrations** file.

The latter solution is more complex to expand, since each *monitoring tool* should correspond to a different LDAP back-end, that is statically linked to the LDAP server code at compile time. The modification of the

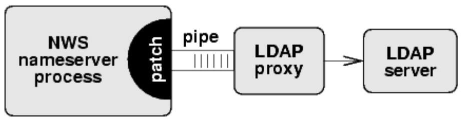


Figure 6. Implementation layout

code each time a new tool is integrated in the service, can cause undesirable reliability problems, if such modification affects an extremely sensible component such as the directory service. For this reason, we went for the former alternative.

The code we developed for our implementation can be split into two parts, as shown in Figure 6 [1]:

- a **patch** of the NWS code, that mirrors each registry operation (insert or delete of an entry) in a formatted message which is sent through a Unix pipe;
- a simple **proxy** that takes the formatted messages from the pipe and performs the corresponding function in the LDAP directory, invoking the appropriate commands.

Prototype code is available at `ftp://ftp.di.unipi.it/pub/Papers/ciuffoletti/nwspatch.tgz`.

6 Conclusions

We describe the implementation of a Resource Monitoring System that integrates a directory service with applications and monitoring tools. The design is inspired by the monitoring architecture introduced by the GGF, and aims at modularity and expandibility.

A *proof of concept* implementation based on NWS and LDAP is described. Integrating NWS into an LDAP tree improves NWS scalability and falt-tolerance features. The objectclasses needed to integrate NWS information into the LDAP tree are defined by borrowing from the GGF standard monitoring schema definition.

We do not plan a real scale deployement of such architecture, that was mainly used as a conceptual testbed. However, the experience gathered during such experiments provided valuable input to the GlueDomains initiative.

References

[1] Warren Smith and Dan Gunter. Simple LDAP schemas for grid monitoring. Technical Report GWD-Perf-13-1, Global grid forum - performance working group, June 2001.

[2] Rich Wolski. Dinamically forecasting network performance using the network weather service. Technical Report TR-CS96-494, University of California at San Diego, January 1998.

[3] 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.

[4] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–??, 2001.

[5] R. Harakaly, P. Primet, F. Bonnassieux, and B. Gaidioz. Probes coordination protocol for network performance measurement in grid computing environment. *to appear, Journal of Parallel and Distributed Computing Practices, Special Issue on Internet-based Computing*, 2002.

[6] Ian Foster, Carl Kesselman, Jeffrey Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[1] Patches are possible since NWS is **open source**.