An Asynchronous Middleware for Grid Resource Monitoring

Vivien Quéma, Renaud Lachaize, Emmanuel Cecchet
SARDES project, LSR-IMAG, INRIA Rhône-Alpes
{Vivien.Quema,Renaud.Lachaize,Emmanuel.Cecchet}@inrialpes.fr

1 Introduction

Resource management in a Grid Computing environment raises several technical issues. The monitoring infrastructure must be scalable, extensible, flexible and configurable to support thousands of devices in a highly dynamic environment where operational conditions are constantly changing. The french VTHD project [1] offers a Grid Computing testbed to experiment innovative solutions.

The problem we address in this paper is the *scalable, dynamic* and *flexible* monitoring of resources in a Grid infrastructure such as VTHD. To reach our goal, we propose a Grid monitoring service on top of a J2EE (*Java 2 Enterprise Edition*) application server that relies on the DREAM (*Dynamic REflective Asynchronous Middleware*) middleware. DREAM combines both the use of asynchronous communications and the use of component-based approaches. Message-oriented middleware are recognized as one means to achieve the scalability-extensibility-openness objectives. Component-based technologies provide flexibility and configurability, whereas J2EE technology is a fundation for Web services.

The rest of this paper is organized as follows: section 2 introduces the DREAM middleware. The monitoring service is presented in section 3. Section 4 discusses related work and we conclude in section 5.

2 DREAM

DREAM is a component-based asynchronous distributed services platform. It allows the configuration, deployment and administration of distributed services communicating through asynchronous message passing.

Each node hosting the DREAM platform runs a local instance made up of three entities : a message channel, an administration service and a set of additional services.

The **distributed message channel** provides an asynchronous messaging service between the different services using message queuing. It is implemented as a set of components interacting together and implementing several non-functional properties like reliability, ordering and security of message transfers.

The **administration service** is in charge of configuring, deploying and administrating the message channel and the set of deployed services. It is itself a distributed service that is represented by an *Administration Service* on each node. Each administration service includes an *architecture repository* which is used to store information about the local DREAM instance's internal architecture.

Finally, DREAM allows the deployment of **several services** providing various features: a naming service implementing the JNDI (*Java Naming and Directory Interface*) specifications that helps registering and locating other services; an event-based service that provides a means to deploy autonomous software entities called agents. Agents are active objects that can run concurrently and behave according to an "event → reaction" model. DREAM also offers a service implementing the JMS (*Java Message Service*) specifications.

3 Anatomy of a resource monitoring service

3.1 Monitoring service overview

The monitoring service we present in this paper aims at giving access to a global view of all resources available in a Grid. Our goal is to monitor Grid resources and not applications. We want to maintain a knowledge base about devices and their state.

Observing events in a distributed system like a Grid raises several problems: events are distributed at different parts of the system, the volume of observed events may be such that it overwhelms the observer, observed events occur at a rate which cannot be easily used by the observer, observed events appear in a form which is not suitable for immediate use, etc.

To overcome these problems, the monitoring service we propose in this paper is a distributed activity that performs three tasks in parallel: the first task is to **collect** appropriate indicators and alarms from devices; the second task is to **process** this data by aggregation, filtering, forwarding; finally, the third task is to **store** and **deliver** the processed data.

3.2 Monitoring service architecture

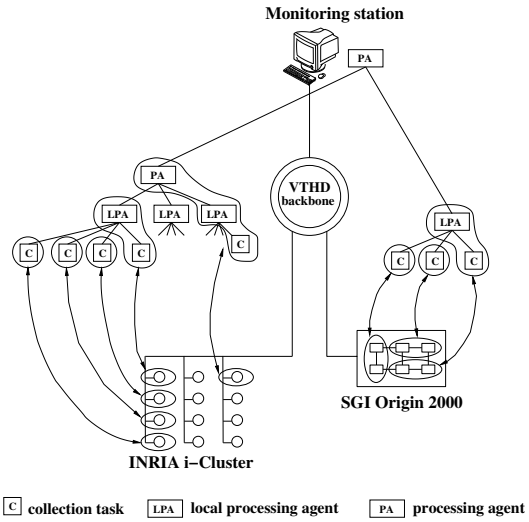

Figure 1. The VTHD network.

Figure 1 depicts a snapshot of the VTHD Grid. The 225 nodes Beowulf cluster and a 6-processor SGI Origin 2000 supercomputer are remotely monitored through the VTHD backbone.

3.2.1 The collection task

This task is in charge of collecting the device's state. It is performed by the DREAM *collector service*. This service provides features to collect resource information from the device it is deployed on. A collector captures the current state of a device (e.g. CPU load, memory usage, ...). As every device is affected by the monitoring process, it is important to keep as low as possible the overhead induced by each collector. It is nevertheless important to keep the collection as flexible as possible given that the resources to be monitored may change at run-time: collectors must be able to dynamically change the set of indicators they are collecting.

Each collector depends on one *Local Processing Agent* which processes the collected data as described in the next section. Three interaction mechanisms between collectors and local processing agents are possible: *pull*, *push* and *alarm*. The pull mechanism follows the client-server paradigm: the local processing agent requests data from collectors. The push and alarm mechanisms refer to events produced by collectors and sent to local processing agents. These events can either be produced periodically (push) or only when bounds are raised (alarm).

One benefit given by the DREAM platform is that collector services can be dynamically loaded, which makes the monitoring service flexible. Indeed, if the observer needs to monitor a new resource on one node, he only has to request the deployment of the appropriate collector from the administration service.

3.2.2 The processing task

This task is in charge of processing the monitoring data received by Local Processing Agents. It is performed by a set of hierarchically distributed Processing Agents deployed on event-based services. Processing depends on the interaction mechanism and on the monitoring facilities required by the observers. Currently available processing tasks are event filtering and aggregation, alarm forwarding, ...

The depth of the hierarchy and the distribution of processing agents are totally free and are done according to the monitoring service criteria: response time, network traffic, processing unit capabilities, ... The more processing can be performed close to the source, the more scalable the solution is. Indeed, it lowers the amount of non-processed data that have to cross the network.

For the VTHD network, the cluster is monitored by a 2-level hierarchy of processing agents (see fig. 1). Each branch of the cluster owns a local processing agent (LPA) that is in charge of data collected by nodes of this branch. These LPAs represent the first level of the hierarchy. Collected data is processed and then forwarded to a processing agent (PA) hosted by one node of the cluster. This processing agent is at the second level of the hierarchy. Concerning the SGI Origin 2000 machine, it is monitored by a 1-level hierarchy of processing agents. For each pair of processors, a collection task is in charge of collecting monitoring data that are then aggregated by a LPA. Finally, monitoring data from both the cluster and the SGI machine are forwarded to a PA located on the monitoring station. This PA introduces a new hierarchy level that is useful to process data from the cluster and the SGI machine before delivering them to the presentation task.

3.2.3 The presentation task

This task is in charge of the *storing* and *reporting* logics. These logics are usually performed by the JMS service. JMS is itself integrated in the J2EE framework. The J2EE specification defines 2 containers: the EJB container that hosts the business logic of the service and the Web container (usually a servlet server) where the presentation logic is executed. J2EE is also a fundation for Web services and allows the resource monitoring information to be exported as a Grid service as specified in the OGSI specification. A J2EE server accesses the monitoring information through JMS and export it by several interfaces (email, HTML, SOAP, ...). Figure 2 shows how the Grid monitoring service sits on top of

a J2EE application server that relies on JMS to access DREAM's provided information.

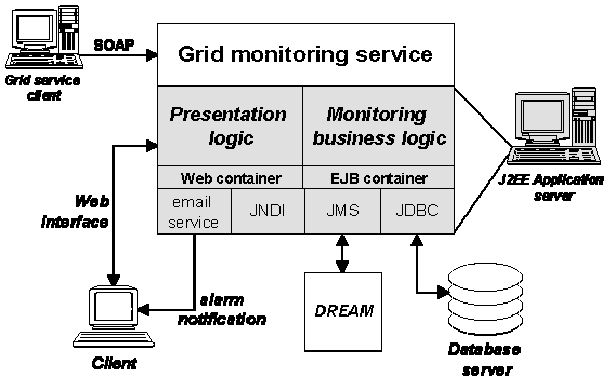

Figure 2. DREAM integration in a J2EE application server environment.

This bridge between DREAM and the J2EE application server provides the necessary business logic components to control a monitoring service through JMS. Using the same business logic, we can have several presentation implementations running in parallel. The Grid monitoring service integrates in the Open Grid Software Architecture (OGSA) [2], which is more likely to become the standard way to access Grid services.

4 Related work

DREAM has similarities with the JAMM [3] architecture, an agent-based system that automates the execution of monitoring sensors and the collection of data events. The reflective component-based architecture of DREAM eases its dynamic reconfiguration, thus providing the means to modify the monitoring service's architecture at run time.

The Network Weather Service (NWS) [4] is a monitoring system aimed at forecasting short-term network performance. NWS was specifically targeted for network performance study and thus, heavily focuses on limited intrusiveness. Although we are naturally concerned with controlled intrusiveness, DREAM was designed at a conceptually higher level than NWS (services deployed on top of a message channel) and for more general purpose monitoring needs.

GridRM [5] is a resource monitoring architecture for the Grid based on the Grid Monitoring Architecture. Our system shares common goals with GridRM but has significant architectural differences: we are concerned with providing a distributed, hierarchical and dynamically reconfigurable event processing logic. Moreover, DREAM relies on the use of a message-oriented middleware that eases event-filtering, communications with remote sites, fault-tolerance and dynamic reconfiguration.

Ganglia [6] is a toolkit aimed at monitoring clusters and clusters of clusters. It relies on the use of monitor daemons (gmond) and meta daemons (gmetad). Gmonds are running on every monitored node to multicast relevant changes. Gmetads merge data transmitted from gmonds over unicast routes. DREAM provides flexibility concerning data aggregation and processing functions. Moreover Ganglia is a cluster-centric approach while DREAM also targets embedded systems and ubiquitous heterogeneous computing contexts.

5 Conclusion

Grid resource monitoring is a difficult task since it raises many technical issues: scalability as thousands of devices are involved; extensibility as new devices are joining and leaving the Grid dynamically; flexibility and configurability as the monitoring application is facing the difficult problem of supporting simultaneously a wide range of devices and changing operational conditions.

We think that asynchronous communications and component-based infrastructures provide a way to achieve the aforementioned objectives. These two techniques are currently being applied in the DREAM middleware to build an asynchronous distributed service platform.

This paper has discussed how the services provided by this platform and its dynamic configurability help constructing monitoring services for large-scale distributed environments like Grids. A *scalable, dynamic* and *flexible* monitoring service of the VTHD Grid resources has been presented. Dynamicity and flexibility are provided by DREAM, whereas scalability is provided by the hierarchical distribution of data processing.

Finally, J2EE application servers provide multiple interfaces to access the monitoring business logic provided by DREAM. This allows a smooth transition from the current solutions (HTML, email, ad-hoc, ...) to the new Open Grid Software Architecture standard.

References

[1] The VTHD Project, 2002. http://www.vthd.org.

[2] I. Foster et al. The Physiology of the Grid, 2002. http://www.globus.org/research/papers/ogsa.pdf.

[3] B. Tierney et al. A Monitoring Sensor Management System for Grid Environments. In *Proceedings of HPDC'00*, 2000.

[4] Rich Wolski et al. The network weather service. *Future Generation Computer Systems*, 15(5-6), 1999.

[5] M. A. Baker and G. Smith. GridRM: A Resource Monitoring System for the Grid. In *Proceedings of the 3rd International Workshop on Grid Computing*, 2002.

[6] Ganglia Toolkit 2.5.0. http://ganglia.sourceforge.net/.