Integrating Resource and Service Discovery in the CoreGrid Information Cache Mediator Component

Giovanni Aloisio[1], Zoltán Balaton[2], Peter Boon[3], Massimo Cafaro[1], Italo Epicoco[1], Gábor Gombás[2], Péter Kacsuk[2], Thilo Kielmann[3], and Daniele Lezzi[1]

[1] Center for Advanced Computational Technologies ISUFI,
University of Lecce and
National Nanotechnology Lab/INFM&CNR,
Lecce, Italy
{giovanni.aloisio, massimo.cafaro,
italo.epicoco, daniele.lezzi}@unile.it
[2] MTA SZTAKI
Computer and Automation Research Institute
Hungarian Academy of Sciences, Hungary
{balaton, gombasg, kacsuk}@sztaki.hu
[3] Vrije Universiteit Amsterdam, The Netherlands
{pboon, kielmann}@cs.vu.nl

Abstract. In this paper we describe how the CoreGrid application-level information cache mediator component will benefit from the integration of resource and service discovery mechanisms available in iGrid and Mercury. The former is a novel Grid Information Service based on the relational model. iGrid has been initially developed within the GridLab project by the ISUFI Center for Advanced Computational Technologies (CACT) at the University of Lecce, Italy. It provides fast and secure access to both static and dynamic information through a Globus Toolkit GSI (Grid Security Infrastructure) enabled web service. Besides publishing system information, iGrid also allow publication of user's or service supplied information. The adoption of the relational model provides a flexible model for data, and the hierarchical distributed architecture provides scalability and fault tolerance. The latter, which has also been initially developed within the GridLab project by MTA SZTAKI, has been designed to satisfy requirements of grid performance monitoring: it provides monitoring data represented as metrics via both pull and push access semantics and also supports steering by controls. It supports monitoring of grid entities such as resources and applications in a generic, extensible and scalable way. It is implemented in a modular way with emphasis on simplicity, efficiency, portability and low intrusiveness on the monitored system.

1 Introduction

The CoreGrid partners of task 7.2 are actively developing a suite of components that mediate between applications and system software [1]. These Mediator Components will be derived from the ongoing developments of the partner institutions involved in this task. The research is focused on the definition and inplementation of a novel grid component architecture; the aim is to foster integration and linking of different grid components with minimal effort, by providing a simple, well defined glue layer between all kind of components. The envisioned architecture also includes a tools framework, consisting of an integrated components Grid platform, a component support toolkit, and a generic problem-solving toolkit, as in Figure 1.

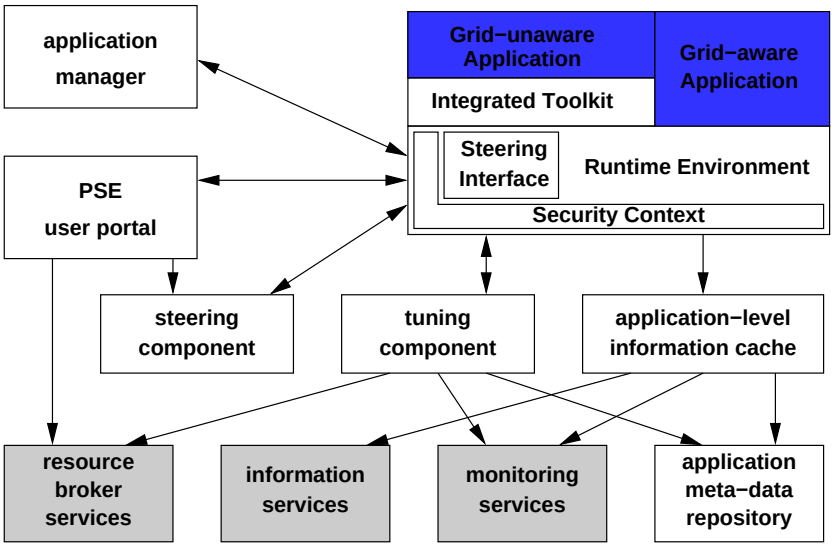

Fig. 1. Grid component architecture

The proposal for a mediator component toolkit includes an application-level information cache mediator component. This component will be designed to provide a uniform interface to access several kinds of different data originating from information services, monitoring services and application-level meta-data. Moreover, a caching mechanism will allow delivering the information to applications and/or components that need it really fast.

We are now jointly collaborating to develop an application-level information cache mediator component. Our activities also include the integration of the iGrid information service [2] [3] and the Mercury monitoring service [4], to provide the envisioned grid component architecture with resource and service discovery capabilities. Indeed, grid environments require the availability of an

information rich environment to support resource and service discovery, and thus decision making processes related to dynamic adaptation. Distributed computational resources and services are sources and/or potential sinks of information; the data produced can be static or dynamic in nature, or even dynamic to some extent. Depending on the actual degree of dynamism, information is better handled by a Grid Information Service (static or quasi-static information) or by a Monitoring Service (highly dynamic information).

In this context, information plays a key role, therefore Grid Information and Monitoring Services are fundamental building block of a grid infrastructure/middleware. Achieving high performance execution in grid environments is virtually impossible without timely access to accurate and up-to-date information related to distributed resources and services: the lack of information about the execution environment prevents design and implementation of so called grid-aware applications. Indeed, an application can not react to changes in its environment if these changes are not advertised. Therefore, self-adjusting, adaptive applications are natural consumers of information produced in grid environments. However, making relevant information available on-demand to consumer applications is nontrivial, since information can be (i) diverse in scope, (ii) dynamic and (iii) distributed across one or more Virtual Organizations. It is worth noting here that obtaining information about the structure and state of grid resources, services, networks etc. can also be challenging in large scale grid environments.

The rest of the paper is organized as follows. We discuss resource and service discovery mechanisms available in iGrid in Section 2, and present the Mercury monitoring service in Section3. We give an overview of the application-level information cache in Section 4, and conclude the paper in Section 5.

2 iGrid

iGrid is a novel Grid Information Service initially developed within the European GridLab project [5] by the ISUFI Center for Advanced Computational Technologies (CACT) at the University of Lecce, Italy. An overview of the iGrid Information Service can be found in [2]; here we delve into details related specifically to resource and service discovery mechanisms available.

iGrid distributed architecture is based on iServe and iStore GSI [6] enabled web services. An iServe collects information related to the computational resource it is installed on, while iStore gathers information coming from trusted, registered iServes. The current architecture resembles the one adopted by the Globus Toolkit MDS, therefore iStores are allowed to register themselves to other iStores, creating arbitrarily complex distributed hierarchies. Even though this architecture proved to be effective to build scalable distributed collections of servers, neverthless we are already investigating peer-to-peer overlay networks based on current state of the art distributed hash table algorithms in order to improve iGrid scalability. The implementation includes system information providers outputting XML, while trusted users and/or services can publish information simply calling a web service registration method. Resource discovery

using the iGrid Information Service is based on the availability of the following information (not exaustive):

System operating system, release version, machine architecture etc;

CPU for CPUs, static information such as model, vendor, version, clock speed is extracted; the system also provides dynamic information such as idle time, nice time, user time, system time and load;

Memory static information such as RAM amount and swap space is available. Dynamic information related to available memory and swap space is published too;

File Systems static as well dynamic information is extracted, such as file system type, mount point, access rights, size and available space;

Network Interfaces network interface names, network addresses and network masks;

Local Resource Manager the information belonging to this category can be further classified as belonging to three different subclasses: information about queues, jobs and static information about Local resource Management System (LRMS). Some examples of extracted information are: LRMS type and name; queue name and status, number of CPU assigned to the queue, maximum number of jobs that can be queued, number of queued jobs, etc; job name, identifier, owner, status, submission time etc. Currently information providers for OpenPBS and Globus Gatekepeer are available, with LSF planned;

Certification Authorities certificate subject name, serial number, expiration date, issuer, public key algorithm etc.

Virtual Organization information related to VO can be used to automatically discover which resources belong to a given VO; we have VO name, resource type, help desk phone number, help desk URL, job manager, etc.

Of course, this set of information is not meant to be static, the iGrid schema will continue to evolve and will be extended to support additional information as required by the GridLab project or iGrid users.

One of the most important requirements for grid computing scenarios is the ability to discover services and web/grid services dynamically. Services in this context refers to traditional unix servers. The iGrid system provides users and developers with the following functionalities: register, unregister, update and lookup. More than one instance for each service or web service can be registered. The following information is available for services: logical name, instance name, service description, default port, access URL, distinguished name of the service publisher, timestamps related to date of creation an date of expiration of the published information.

For web services, relevant information includes logical name, web service description, WSDL location (URL), web service access URL, distinguished name of publisher and timestamps related to date of creation and date of expiration of the published information.

Information related to firewalls is strictly related to service information. As a matter of fact, before registering a service, developers will query iGrid to retrieve

the range of open ports available on a specified computational resource. This is required in order to chose an open port, allowing other people/services to connect to a registered service. The information available includes firewall hostname, open ports, time frame during which each port (or a range of ports) is open, the protocol (TCP/UDP) used to connect to these ports, the distinguished name of the firewall administrator, and timestamps related to date of creation and date of expiration of the published information.

iGrid uses a push model for data exchange. Indeed, system information (useful for resource discovery) extracted from resources is stored on the local database, and periodically sent to registered iStores, while user and/or service supplied information (useful for service discovery) is stored on the local database and immediately sent to registered iStores. Thus, an iStore has always fresh, updated information related to services, and almost fresh information related to resources; it does not need to ask iServes for information. The frequency of system information forwarding is based on the information itself, but we also allow defining a per information specific policy. Currently, system information forwarding is based on the rate of change of the information itself. As an example, information that does not change frequently or change slowly (e.g. the amount of RAM installed) does not require a narrow update interval. Interestingly, this is true even for the opposite extreme, i.e., for information changing rapidly (e.g., CPU load), since it is extremely unlikely that continuous forwarding of this kind of information can be valuable for users, due to information becoming quickly inaccurate. Finally, information whose rate of change is moderate is forwarded using narrow update intervals.

We have found that the push model works much better than the corresponding pull model (adopted, for instance, by the Globus Toolkit MDS) in grid environments. This is due to the small network traffic volume generated from iServe to iStore servers: on average, no more than one kilobyte of data must be sent. Moreover, we tag information with a time to live attribute that allows iGrid to safely removes stale information from the database when needed. For instance, when users search for data, a clean-up operation is performed before returning to the client the requested information, and during iGrid system startup, the entire database is cleaned up. Therefore the user will never see stale information.

Finally, it is worth recalling here that the performances of iGrid are extremely good, as reported in [2] .

3 Mercury

In a complex system as the grid, monitoring is essential for understanding its operation, debugging, failure detection and for performance optimisation. To achieve this, data about the grid must be gathered and processed to reveal important information. Then, according to the results, the system may need to be controlled. The Mercury Grid Monitoring System provides a general and extensible grid monitoring infrastructure. Mercury Monitor is designed to satisfy specific requirements of grid performance monitoring [7]. It provides monitoring

data represented as metrics via both pull and push model data access semantics and also supports steering by controls. It supports monitoring of grid entities such as resources and applications in a generic, extensible and scalable way. The architecture of Mercury Monitor extends the Grid Monitoring Architecture (GMA) [8] proposed by Global Grid Forum with actuators and controls. Mercury Monitor features a modular implementation with emphasis on simplicity, efficiency, portability and low intrusiveness on the monitored system.

The input of the monitoring system consists of measurements generated by sensors. Sensors are controlled by producers that can transfer measurements to consumers when requested, and are implemented as shared objects that are dynamically loaded into the producer at run-time depending on configuration and incoming requests for different measurements. It is also important to note that in Mercury measurements are performed only when requested by a consumer and data is only sent where it is needed. All measurable quantities are represented as metrics. Metrics are defined by a unique name such as *host.cpu.user* which identifies the metric definition, a list of formal parameters and a data type. By providing actual values for the formal parameters a metric instance can be created, representing a specific entity to be monitored. A measurement corresponding to a metric instance is called a metric value. Values contain a timestamp and the measured data according to the data type of the metric definition. Sensor modules implement the measurement of one or more metrics. Mercury Monitor supports both event-like (i.e. an external event is needed to produce a metric value) and continuous metrics (i.e. a measurement is possible whenever a consumer requests it, e.g., the CPU temperature in a host). Continuous metrics can be made event-like by requesting automatic periodic measurements.

The GMA proposal of the Global Grid Forum only describes components required for monitoring. It is often necessary however, to also influence the monitored entity based on the analysis of measured data. For example, an application might need to be told to checkpoint and migrate if it does not perform as expected, a service may need to be restarted if it crashed or system parameters (such as process priorities or TCP buffer sizes) might need to be adjusted depending on current resource usage. To support this, actuators have been introduced in Mercury. Actuators are analogous to sensors in the GMA but instead of monitoring something they provide a way to influence the monitored entity. As sensors are accessed by consumers via producers, actuators are made available for consumers via actuator controllers. As the producer manages sensors (start, stop and control sensors and initiate measurements on a user's request) the actuator controller manages actuators. Similarly to metrics implemented by sensors, actuators implement controls that represent interactions with either the monitored entities or the monitoring system itself. The functional difference between metrics and controls is that metrics only provide data while controls do not provide data except for a status report but they influence the state or behaviour of the monitoring system or the monitored entity.

Besides providing information about grid resources, Mercury contains two elements to aid application and service monitoring and steering: an application

sensor and an instrumentation library that communicates with this sensor. Together they allow to register application specific metrics and controls and to receive and serve requests for metrics and controls while the application is running. The application sensor is resposible for keeping track of processes of jobs as well as any private metrics or controls that the running applications provide. The application sensor forwards requests for application specific metrics or controls to the application process(es) they belong to. The request is then interpreted by the instrumentation library by performing the measurement or executing the requested control. The instrumentation library communicates with the application sensor using a UNIX domain socket, but it also has shared memory support to speed up transferring large volumes of data such as generated by fine-grained instrumentation. The application may also put certain variables into the shared memory area so they can be queried or modified without direct interaction with the application. This is useful for single-threaded applications or services that do not call the event handler of the instrumentation library for extended periods of time. Processing of application specific metric events is optimised inside the instrumentation library, i.e. they will not be sent to the application sensor if there are no consumers requesting them. This ensures that if an application is built with fine-grained instrumentation it can still run without noticeable performance degradation if the generated events are not requested.

4 Application-level Information Cache

The envisioned application-level information cache component [1] is supposed to provide a unified interface to deliver all kinds of meta-data (e.g., from a GIS like iGrid, a monitoring system like Mercury, or from application-level meta data) to a grid application. The cache's purpose is twofold. First, it is supposed to provide a unifying component interface to all data (independent of its actual storage), including mechanisms for service and information discovery. Second, this application-level cache is supposed to deliver the information really fast, cutting down access times of current Web-service based implementations like Globus GIS (up to multiple seconds) to the order of a method invocation. For the latter purpose, this component may have to prefetch (poll) information from the various sources to provide them to the application in time.

Such an application-cache component is currently being developed as a collaboration among the authors. Its API as presented to the application is inspired by GridLab's GAT [9]. The GAT specifies an API for monitoring purposes. The basis of this API is general and extensible enough to fit the current monitoring and information systems and possibly future ones too. Like Mercury and GAT, the mediator defines measurable quantaties as metrics. A metric is a container which holds the unique name of the metric and the properties needed to retrieve the information, like parameters. The result of a measurement is stored in a container called a metric value.

An application that requests information creates a metric which specifies the source host from which the information originates, the possibly required param-

eters for the metric, and a recommended frequency which indicates how often the mediator should update the corresponding metric value in its cache. Note that the frequency could be omitted if the underlying monitoring or information system is able to push the requested information.

From the moment the mediator receives the first metric value, the application will be able to get it from the cache without any delays. However, it could also choose to get notified if a value gets updated, this way both pull and push mechanisms are available to the application. Another option for applications is to request a metric value bypassing the cache. In this case, the mediator component will merely serve as a uniform interface between the underlying monitoring and information systems and the application.

In order to serve information from different sources, the mediator component uses an extensible 'plugin' system, where each plugin (metric provider) forms the link between the mediator component and the underlying monitoring or information system. The metric providers have to deal with the actual retrieval of information and present it to the (rest of the) mediator component. The mediator component will process the information further, possibly by caching it.

A metric provider instance will represent one running information system on a host. So multiple metric provider instances retrieving information from the same type of information system, only from different hosts can exist next to each other. Therefore, when an application does not specify a metric provider to use when retrieving a metric, it should at least specify the source host running the information system. That way, the mediator component is able to group metric providers retrieving information from the same host.

Some types of information may be retrieved from only one information system, while other types of information could be obtained from multiple. This latter type of information could be presented by the different systems in different ways, using different data types or even different measurement units. It is up to the metric providers to translate information presented by the underlying system into a format which the mediator component presents to the application.

If information is requested by an application, the application can either choose a certain metric provider, or leave it up to the mediator to decide which metric provider will be used to retrieve the information. If multiple metric providers are able to retrieve the same type of information, it is a matter of policy which one is chosen (lowest response time, most reliable).

The currently developed prototype is providing the following interface:

List getMetricDefinitions()
Gets a list of available MetricDefinitions.

MetricDefinition getMetricDefinitionByName(String name)
Gets a MetricDefinition given the metric name.

MetricValue getMetricValue(Metric metric)
Retrieves a MetricValue from the cache, if available. Throws an exception if the value is not stored in the cache.

void startProviding(Metric metric)
Tells the cache to retrieve the given metric. The cache tries to keep the metric

value updated according to the frequency set in the metric parameter. No guarantees can be given, however, whether the (underlying) system is capable of providing the information at this rate.

void stopProviding(Metric metric)
Tells the cache that the metric described by the parameter does not need to be updated anymore.

void addMetricListener(MetricListener listener, Metric metric)
Adds a MetricListener. The listener object will be called whenever an updated value becomes available for the metric.

void removeMetricListener(MetricListener listener, Metric metric)
Removes a MetricListener. No notification through the listener will be sent any more.

MetricProviderManager getMetricProviderManager()
Returns the MetricProviderManager for advanced configuration.

MetricValue getMetricValueFromProvider(Metric metric)
Retrieves a MetricValue directly from a MetricProvider, bypassing the cache. The MetricProvider should be specified in the Metric object.

MetricListener (Interface)
When a metric value is updated in the cache, applications can retrieve events through objects implementing the MetricListener interface.

void processMetricEvent(MetricValue val)
An instance of a class implementing this interface receives MetricValues through calls to this method when it is registered to receive such events.

5 Conclusion

We have described the iGrid information service and the Mercury monitoring service, focusing our attention to the integration of these and other sources of useful information in an application-level information cache mediator component we are jointly developing in the context of the European CoreGrid project. We gave an overview of the forthcoming mediator component, including details related to its API. This component will provide a basis for dynamic adaptation in grid environment, as envisioned in the CoreGrid grid component architecture. As its implementation is ongoing work, quantitative evaluations are not available yet.

Acknowledgements

This work is partially funded by the European Commission, via the Network of Excellence *CoreGRID* (contract 004265).

References

1. CoreGRID Virtual Institute on Problem Solving Environments, Tools, and GRID Systems: Proposal for mediator component toolkit. CoreGRID deliverable D.ETS.02 (2005)

2. Aloisio, G., Cafaro, M., Epicoco, I., Fiore, S., Lezzi, D., Mirto, M., Mocavero, S.: igrid, a novel grid information service. In: Proceedings of Advances in Grid Computing - EGC 2005. Volume 3470., Lecture Notes in Computer Science, Springer-Verlag (2005) 506–515
3. Aloisio, G., Cafaro, M., Epicoco, I., Fiore, S., Lezzi, D., Mirto, M., Mocavero, S.: Resource and service discovery in the igrid information service. In: Proceedings of International Conference on Computational Science and its Applications (ICCSA 2005). Volume 3482., Springer-Verlag (2005) 1–9
4. Gombás, G., Balaton, Z.: A flexible multi-level grid monitoring architecture. In: Proceedings of the First European Across Grids Conference. (2003)
5. Allen, G., Davis, K., Dolkas, K.N., Doulamis, N.D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Radke, T., Russell, M., Seidel, E., Shalf, J., Taylor, I.: Enabling Applications on the Grid – A GridLab Overview. International Journal on High Performance Computing Applications **17**(4) (2003) 449–466
6. Foster, I., Kesselmann, C., Tsudik, G., Tuecke, S.: A security architecture for computational grids. In: Proceedings of 5th ACM Conference on Computer and Communications Security Conference. (1998) 83–92
7. Németh, Z., Gombás, G., Balaton, Z.: Performance evaluation on grids: Directions, issues, and open problems. In: Proceedings of the Euromicro PDP 2004, A Coruna, Spain, IEEE Computer Society Press (2004)
8. Fisher et al., S.: R-gma: A relational grid information and monitoring system. In: Proceedings of 2nd Cracow Grid Workshop, Cracow, Poland (2003)
9. Allen, G., Davis, K., Goodale, T., Hutanu, A., Kaiser, H., Kielmann, T., Merzky, A., van Nieuwpoort, R., Reinefeld, A., Schintke, F., Schütt, T., Seidel, E., Ullmer, B.: The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. Proceedings of the IEEE **93**(8) (2005) 534–550