

INTEGRATING THE PALANTIR GRID META-INFORMATION SYSTEM WITH GRMS *

Francesc Guim, Ivan Rodero, Julita Corbalan

*Computer Architecture Department
Universitat Politècnica de Catalunya*

fguim@ac.upc.edu

irodero@ac.upc.edu

juli@ac.upc.edu

Ariel Oleksiak, Krzysztof Kurowski, Jarek Nabrzyski

Poznan Supercomputing and Networking Center

krzysztof.kurowski@man.poznan.pl

ariel@man.poznan.pl

naber@man.poznan.pl

Abstract Grids allow large scale resource-sharing across different administrative domains. Those diverse resources are likely to join or quit the Grid at any moment or possibly to break down. In such environments there is a wide range of types of resources to be monitored, with different nature, characteristics and so on. These issues make the task of monitoring complex to treat, and it is difficult to provide a general way for accessing to all this information.

In this paper we describe the Palantir meta-information system that has been designed for uniforming the access to different monitoring and information systems and abstracting the complexity of the underlying systems. We present the API that Palantir provides to the end users, a general architectural description of its components, and the description of how GRMS providing job monitoring information can be integrated as a part of this meta-Information System.

Keywords: Job Monitoring, Resource Monitoring, Information Systems, Mercury, Ganglia, MDS, GRMS, Palantir

*The integration work published in this paper has been supported by "CoreGrid" network of excellence in "Foundations, Software Infrastructures and Applications for large scale distributed, Grid and Peer-to-Peer Technologies"; by the "HPC-Europa" project and by the Spanish Ministry of Science and Education under contract TIN2004-07739-C02-01.

1. Introduction

In the eNANOS [1] architecture we needed a new component for accessing to all the information related to the entities that are involved in our system, i.e: Grid jobs, local processes, resources and so on. This component would integrate information from the lower level of the Grid, coming from the local components of the centres, such as the NANOS scheduler, with information coming from the top components of our architecture, in our case from the eNANOS Broker. However, the nature of the information that it would provide could be very diversified. It not only would provide monitoring information from the monitoring system, or job monitoring system, it would provide information from other nature, for instance performance predictions. Using our recent experience in grid monitoring and information systems obtained in [3][2], we designed a meta-information system implements all the needed functionalities. Something important to remark is that the Palantir system is not intended to substitute other existing monitoring or information tools. Furthermore, it is intended to uniform the access to the information provided by them providing easy and powerful mechanisms for gathering data from different sources, and providing mechanism for implement new information modules in case that it is needed.

The goal of this paper is to provide a description and characteristics that our meta-information system has in terms of functionalities and architecture, and present how the GRMS [13] can provide job monitoring information as an information producer of the Palantir.

The paper is organized in two main parts: the first part contains the description of the Palantir system: we present its architecture and the application programming interface that is provided to the clients, its semantic and functionalities. The second part of the paper present how the GRMS system could be integrated as a part of this monitoring system.

2. Related Work

In the literature the most commonly used and discussed monitoring and information systems are: Globus Monitoring and Discovery Service (MDS)[4][5], GridLab Mercury[6], Network Weather Service (NWS) [7], Crossgrid OMIS Compliant Monitoring service for the Grid (OCM-G) [8][5] and Ganglia [9]. MDS is the Grid Information Service provided in the Globus Toolkit [10]. It supports monitoring sensors to register to the Information Service. It is composed of two types of components: information providers and data aggregation services. The information providers are present at the host end of the monitoring hierarchy whereas the data aggregation services are the internal nodes. Like MDS, Mercury is a hierarchal monitoring service, organized with Local Monitors at the host end, and Main Monitors. Many metrics are provided, al-

lowing an accurate resource-oriented monitoring. Mercury is the only tool to provide fine-grain access policies.

Network Weather Service (NWS) is a monitoring system designed for short-term performance forecasts. NWS provides a set of system sensors monitoring end-to-end TCP/IP performance, available CPU percentage, and available memory.

The OMIS (On-line Monitoring Interface Specification, [11]) is an API aiming at defining a standard interface for communication between middleware applications. The OCM-G (OMIS-Compliant Monitoring system for the Grid) is an application monitoring tool, which is part of the Crossgrid project. Finally, Ganglia is a distributed resource-oriented monitoring system for high-performance computing systems such as clusters and grids.

3. The Palantir Data Model

The entities are the core of its data model. They represent the conceptual elements of the systems that can contain information suitable to be requested. They are not required to be physical resources. For example, hosts, jobs or applications are considered to be entities. Each entity has associated a set of metrics that contain specific information. For instance the metric elapsed time is a metric that can be associated to the entity job.

Each entity type has a set of instantiations: for example the entity host may have the instantiations “host1.bsc.es”, “host2.bsc.es” etc.

4. Palantir System Architecture

This section provides a general description of what is the architecture of the Palantir meta-information system that we designed. Figure 1 provides a general view of the system architecture. As can be observed there are three top architectural components: Palantir Access Point, the Palantir Gateways and finally the information modules. In the following subsections their main characteristics are presented.

4.1 The Palantir Access Point

The first layer is the Palantir Meta-Information System access point. It manages the queries that are done by the users or by the applications. They basically redirect such queries to the appropriate Palantir Gateway (see the following subsection). The protocol used from the application to this access point is based on the generic protocol presented in the following section. Thanks to this generality, the client has not to be aware to which IS the final queries are done (f.e: to the MDS or to the NWS etc.) neither to where this information is gathered (f.e: if the query has to be done to the IS that is built on the host

pcmas.ac.upc.edu, or to hostx.bsc.edu etc.). The main goal is to abstracting of the complexity of the underlying systems to the user.

4.2 The Palantir Gateway

The second layer of components that are located on the local centers, are the Palantir Gateways. They are responsible to carry out the queries to the information systems and monitoring system modules (see the following subsection).

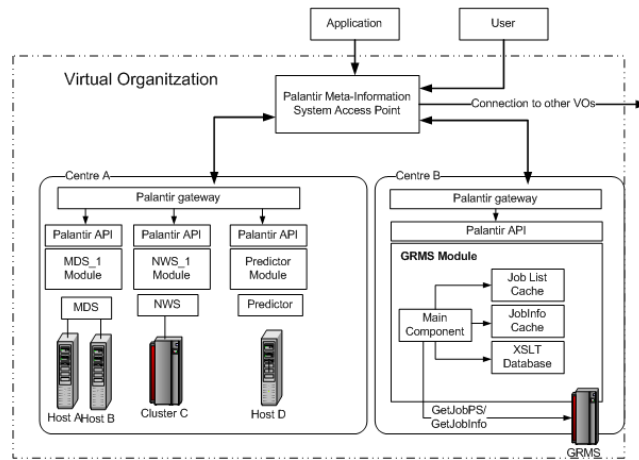


Figure 1. Palantir general architecture

The main task of the gateway is choosing the appropriate module that will process the query. It stores a data base with all the entities that are available on the underlying system (centre, VO etc.) containing: type of entity, which IS provides its information, metadata etc. When entity information is required it searches in such database where it can be retrieved. The gateway distinguishes two kinds of entities:

- Persistent entities. Entities that are non-volatile and that are likely to remain available and alive for a long time. Examples of this kind of entities are: host, network, cluster etc. For these entities the database stores among other data in which information system are located. For example in the figure 1 the Palantir Gateway of the Center A will know that the entity "Host A" can be queried in the module with id "MDS1". Regularly the data base is updated with the creation or destruction of entities in the information systems.
- Non-persistent entities. Entities that are volatile or with a limited amount life time. For example: jobs. These kind of entities are identified with a

composed key that contains information about where they can be found. Basically, a subset of its key must identify a persistent entity that will be used to find out in which module the query has to be done. For example entity "job1@hostD" has the key job1+hostD, then the gateway knows that the persistent sub key is hostID and can found in which module the information about the entity can be found.

4.3 Monitoring and Information Systems modules

They are the components present in the bottom layer of the architecture. Modules are responsible of carrying out the final queries to the information or monitoring systems that they represent. The modules receive queries related to the entities based on the generic format presented in the following section and they take the appropriate actions for answer them.

Obviously, at this level modules must know how the queries have to be done to the IS that they represent. For each monitoring or information system that can be queried through the Palantir, a module must be implemented.

It is important to remark that all the modules must provide and implement a set of common functionalities for support the requirements explained in the following section.

5. Palantir Interfaces

This section provides the description of the protocol and the application programming interface that we have designed for accessing the meta-information system. As explained before, the main objective has been to provide mechanisms for allowing carrying out queries to different kind of resources or Grid entities using a generic and uniform format. Something important to remark is that the functionalities that implement such mechanisms are provided by the gateways and can be accessed by the end-user and applications through the XML queries over the TPC/IP stack or using the access point. Some of the presented methods have also to be implemented by the information modules, mainly those methods responsible to provide information, in this kind of methods user/application will call to the gateway and it will redirect the query to the module, the once that will really implement the functionality.

The functionalities are divided in three main parts. The first one is a set of methods that allows starting and ending communications between authorized components and the meta-information system. The second set of methods allows discovering which type of features and methods are available. The system is intended to be extensible, so it is important if new features are added the user-end or the applications that are using it would be able to discover and learn how to use them. And finally, the third set of methods is oriented to

retrieve the information from the system. In this paper for lack of space we provide a general description for the second and three kind of methods.

5.1 Discovering the available features

The meta-information system is providing a wide range of information of a very diversified type of entities. It must provide ways to discover how to query them and what information can be gathered. Below are presented a list of all the methods that have been defined for provide these functionalities.

- **GetResources:** Returns the list of the types of resources available on the system that underlies the gateway. It is important to clarify that this list will be a set of abstract resources not a specific physical resources, for instance it would return the resource type "host" instead of host "host1.bsc.es". This method returns an XML document containing the list of resources available on the system (for example, Host, Clusters, Jobs). Each resource will have associated a human readable description, and a XML Schema describing how the resource is identified, and the list of the metrics associated to the entity.
- **GetResourceInstantiation:** Returns the list of instances of a provided entity type. For instance, user may want to know the list of entities of type "host". Each instantiation will be returned following the XML Schema format that describes how to identify such resource.
- **GetMetricInfo:** Returns information about a particular metric of a particular entity or resource. This method returns an XML document that will contain a human readable description of the metric, parameters that must be provided when querying it, the description of the available filters that can be used applied to the given metric, and the available notifications.

The methods *GetResources* and *GetMetricInfo* presented in this subsection are implemented only by the gateways since no runtime information is needed from the modules. The gateways store all the required information for provide this discovery information. On the other hand the functionality *GetResourceInstantiation* must be provided by the modules due to the different entities available on the modules can be volatiles and may change during the time.

5.2 Getting the entity information

The methods presented in this subsection are the set of methods that allows accessing to the real information of the underlying systems. All the methods must be implemented by the modules.

- **GetMetricValue** Requests the current value of a given metric for a given entity. User/application must be able to filter the information that wants to be retrieved, for instance knowing only the uptime of a host this in a given virtual organization. Thereby, when carrying out a query for a metric of a given entity, user/application can specify using filters which information has to be gathered and which not. In our system, this can be done using the filters parameters of the metrics.
- **Subscribe/UnSubscribe**: Requests or cancel for a periodic update of the metric.
- **AddNotification/DeleteNotification**: Requests or cancels a notification on a specified event.

6. Integrating GRMS in Palantir

In this section we present the module that we have designed for integrating the broker GRMS as an information producer in the Palantir architecture. This module will mainly provide monitoring information for the different jobs that have been submitted to it.

For carry out this integration two different actions are required:

- Implementing modules interfaces. As has been introduced before, the modules mainly have to implement two interfaces: `GetResourceInstantiation` and `GetMetricValue`.
- Defining the entities and metrics. When adding a new module to the system the syntax and semantics of all the entities and its metrics have to be defined. Once they have been defined their descriptions and have been added to the different gateways that provide access to all the GRMSModules accessible in the system.

The following subsection present all the entities that have been designed in the system and its metrics.

6.1 Entities monitored in the module

The GRMS Palantir module provides information about the jobs that the users has been submitted to the Grid using this broker. We have defined the following set of entities:

- 1 Entity **GRMSBroker** this entity provides information about the GRMS broker it self. Table 2 provides the description of the current metrics of this entity.

Metric	Description
Host	Indicates the hostname where the job has been submitted
Queue	Indicates the Queue name where the job has been locally allocated
Status	Indicates the current status of the job.
JobDefinition	Contains the definition of the job, including: Job type, jsld etc.
Submissiontime	Indicates when the job was submitted (in the local system and Grid).
Starttime	Indicates when the job has started.
Finishtime	Indicates when the job has finished (in the local system and Grid).
Performance	Indicates which is the progress of the application in seconds or Megaflops.
SubmittedBroker	Indicates in which broker has been submitted.
QueueingTime	Indicates the amount of time that the job has been waiting in the queue.
LocalId	Indicates the id that the job has in the local system.
StageingPropierties	Returns information about the stageing process of the job.
StageoutPropierties	Returns information about the stageout process of the job
LRM	Indicates the local resource manager used for allocating the job in the local system.
FileNameSubmitted	Script submitted to the broker.

Table 1. Metrics for the entity job

Metric	Description
Description	Provides a description of the GRMS version that is accessed by the module
Version	Provides the version of GRMS broker accessed by the module
JobList	Provides a list of jobs submitted to the broker. (see source 1)

Table 2. Metrics for the entity GRMSBroker

- 2 Entity **Job**. This is clearly the main entity of the module. It provides monitoring information about jobs that have been submitted using GRMS. The table 1 provides the list of the main metrics available for this entity.
- 3 Entity **FileTransfer**. This entity provides information about a file transfer associated to a job. This can be either associated to the stagein or stageout process. Table 3 provides the description of the current metrics of this entity.
- 4 Entity **File**. This entity provides information about a file associated to a job. As the previous entity, this can be either associated to the stagein or stageout process. This entity only provides basic metrics about the file, such as the host where stored, size and path. Table 4 provides the description of the current metrics of this entity.

It might seem that this number of entities is quite reduced, however as currently this module mainly provides information about job monitoring in our opinion it makes no sense to extend it with more definitions. However, in the future the module may provide information about more entities.

Metric	Description
Target	Provides a reference to an entity of type File that is the transfer destination.
Source	Provides a reference to an entity of type File that is the transfer source.
TransferStatus	Indicates the status of the transfer (DONE, FAILED or PROCESSING).
Reason	Indicates the reason for the current status.
TransferredBytes	If available, provides the number of bytes already transferred.

Table 3. Metrics for the entity TransferFile

6.2 Example of Interaction with GRMS: accessing to a Job entity Performance metric

In the figure 2 we present an example of interaction of an application/user for monitoring its submitted jobs to the GRMS broker with the Palantir system, and how the queries are internally managed by the different components of the system.

- 1 Client asks for the list of available entities in the system.
- 2 The Access Point (AP) gathers the list of the available entities to the different Palantir Gateways and forward the list to the client. The GW that proxy the queries to the GRMS broker would return **GRMSBroker**, **Job**, **FileTransfer** and **File**.
- 3 Let's suppose that the client decides to query information about the entity Job. It queries about all the job entities instantiation that matches the filter "user = fguim and host = rage3.poznan.pl" and decides to what job it wants to ask the information. (*user* is the grid unique name for the user).
- 4 The AP gathers the list of the available job entities of the job type to the each Gateway and forward the list to the client.
- 5 The client queries for the metrics list that the job entity has and decides which metrics it wants to ask. The AP gathers this list from the available Gateways and returns it.
- 6 When the user has chosen the entity instantiation and the list of metrics to be queried, it gathers all the metric data with the GetMetricData functionality. For example the client may ask for the metric *Performance* for the job entity with id *1@121.rage3.poznan.pl* (where this id would be a grid global id).
- 7 The AP asks to the Gateway about the metric data for the given instantiation. The Gateway based on the key that identifies the entity chooses the appropriate module and gathers the required data. The AP returns the provided data by the Gateway to the client.

Metric	Description
Host	Provides a reference to an entity of type Host where the file is or will be stored.
Path	Indicates the absolute path of the file.
Size	Indicates the size of the file.
Rights	If available returns the permissions of the file. (group, user and permissions).

Table 4. Metrics for the entity File

6.3 Gathering job monitoring information from GRMS

In the context of the HPC-Europa project we defined a job monitoring functionality that allows to the end-user, using Single Point of Access (SPA) portal, to monitoring its own jobs that he/she has submitted to the different centers. This monitoring information is gathered from the different centers using a common API designed for this goal (in the context of the HPC-Europa project). Each center must implement a plug-in for accessing to its job monitoring information. The plug-ins are components of the SPA architecture that allow accessing to the different functionalities of the center (monitoring, submission etc). Each center must implement all the methods specified in the common interface that plug-ins inherit [3]. One of the plug-ins that have been developed is the job monitoring plug-in for the GRMS broker.

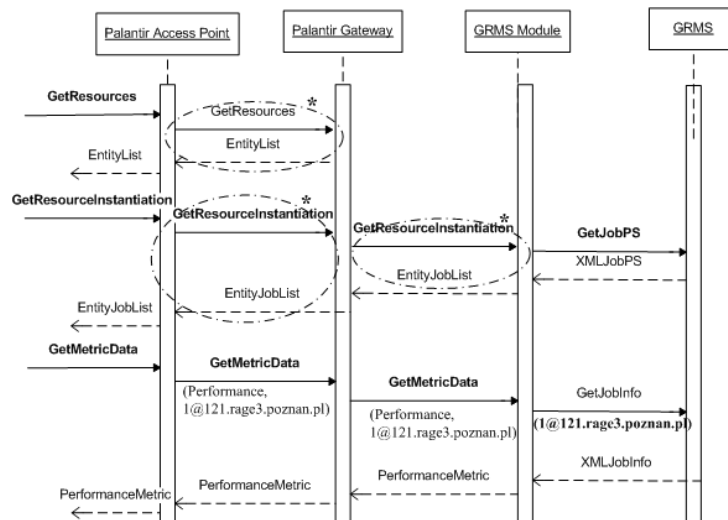


Figure 2. Example of interaction between the different Palantir components.

One of these functionalities is the GetJobPS, which given a user and its credentials, the plug-in returns an XML that containing the monitoring information

for all the jobs that this user has submitted. This XML is based on an XML Schema that we designed during the project (see [3]). The GetJobPS allows specifying filters that helps to user to filter the list of jobs to be retrieved based on set of criteria, for example the status of the jobs, their finish time etc (see source 1).

The entities and metrics that have been designed for accessing information of the GRMS in the GRMS Palantir Module are based on this schema.

As can be seen on the figure 1 the module would gather the information from the GRMS broker using the monitoring functionalities explained before. The module design has two different caches, the first one that stores information about the different jobs identifiers, and the other one stores the detailed information for each job. Obviously these caches must have TTL that indicates for how long the cached data is valid.

Source 1 Sample of parameters for the metric joblist for the entity GRMSBroker

```
<params metric='joblist' entity='Broker'>
  <user>fguim</user>
  <MetricFilters>
    <FilterByDate>
      <BetweenDates>
        <StartDate>1136208170544</StartDate>
        <EndDate>1136380970544</EndDate>
      </BetweenDates>
    </FilterByDate>
    <FilterByState>
      <State>FAILED</State>
      <State>SUSPENDED</State>
    </FilterByState>
    <SubmissionDate>
      <BetweenDates>
        <StartDate>1138886570544</StartDate>
        <EndDate>1136380970544</EndDate>
      </BetweenDates>
    </SubmissionDate>
  </MetricFilters>
</params>
```

7. Conclusions and future work

In this paper we have introduced the architecture and main functionalities of the Meta-Information system Palantir. We have shown how the GRMS broker can be integrated as a part of the Palantir system: we have described the main entities that have been defined for be accessible in through the GRMS module

and its main metrics, and we have presented the main internal structure of this module and how this collect the information form broker.

Future work may include adding and defining new kind of entities and metrics that the GRMS can provide. The current design is mainly focused on providing job monitoring information. However, the Palantir system is intended to be a general information provided, so it is feasible to extend the module for providing other kind of information, for example information about the Broker capabilities or for accessing for other kind of functionalities, for example ask for the most suitable set of resources where a given job can be executed. On the other hand, using the GetJobPS and GetJobInfo is a first approach for accessing to the GRMS broker, however GRMS may provide more generic access to the module allowing more powerful mechanism for accessing its information.

References

- [1] I. Rodero F. Guim and J. Corbalán and J. Labarta. eNANOS: Coordinated Scheduling in Grid Environments In *ParCo* , 2005.
- [2] A. Oleksiak and A. Tullio and P. Graham and T. Kuczynski and J. Nabrzyski and D. Szejnfeld and T. Sloan. HPC-Europa: Towards Uniform Access to European HPC Infrastructures In *6th IEEE/ACM International Workshop on Grid Computing* , 2006.
- [3] F. Guim and I. Rodero and J. Corbalan and J. Labarta and A. Oleiksak and J. Nabrzyski. Uniform Job Monitoring using the HPC-Europa Single Point of Access In *International Workshop on Grid Testbeds 2006* , 2006.
- [4] Z. Balaton and F. Vajda Grid Information and Monitoring Systems 2004
- [5] K. Czajkowski and S. Fitzgerald and I. Foster and C. Kesselman. Grid Information Services for Distributed Resource Sharing. 10 th IEEE Symp. On High Performance Distributed Computing 2001
- [6] Z. Balaton and G. Gombas. Resource and Job Monitoring in the Grid Euro-Par 2003
- [7] R. Wolski and N. T. Spring and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing, *Journal of Future Generation Computer Systems* Volume: 15, 1999
- [8] B. Balis and M. Bubak and W. Funika and R. Wismuller and M. Radecki and T. Szepieniec and T. Arodz and M. Kurdziel. Performance Evaluation and Monitoring of Interactive Grid Applications. November 2004
- [9] M. L. Massie and B. N. Chun and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience *Parallel Computing* 2004
- [10] I. Foster and C. Kesselm. Globus: A metacomputing infrastructure toolkit, *J Intl - International Journal of Supercomputer Applications* 1997
- [11] T. Ludwig and R. Wismüller and V. Sunderam and A. Bode. On-line Monitoring Interface Specification Technische Universität München, 1997
- [12] B. Tierney and R. Aydt and D. Gunter and W. Smith and V. Taylor and R. Wolski and M. Swany. A Grid monitoring architecture. *Global Grid Forum* 2002.
- [13] GridLab, A Grid Application Toolkit and Testbed <http://www.gridlab.org>