

Ubiquitous Grid Resource Monitoring

Mark Baker and Garry Smith

Distributed Systems Group, University of Portsmouth, UK

Abstract

In any wide-area distributed system there is a need to communicate and interact with a range of networked devices and services ranging from computer-based ones (CPU, memory and disk), to network components (hubs, routers, gateways) and specialised data sources (embedded devices, sensors, data-feeds). In order for the ensemble of underlying technologies to provide an environment suitable for virtual organisations to flourish, the resources that comprise the fabric of the Grid must be monitored in a seamless manner that abstracts away from the underlying complexity. Furthermore, as various competing Grid middleware offerings are released and evolve, an independent overarching monitoring service should act as a corner stone that ties these systems together. GridRM is a standards-based approach that is independent of any given middleware and that can utilise legacy and emerging resource-monitoring technologies. The main objective of the project is to produce a standardised and extensible architecture that provides seamless mechanisms to interact with native monitoring agents across heterogeneous resources.

1. Introduction

GridRM [1][2] is a generic open-source resource-monitoring framework that has been specifically designed for the Grid. The framework is used to harvest resource data and provide it to a variety of clients in a form that is useful for their needs. Our philosophy is to take a standards-based approach that is independent of any given middleware technology and that can utilise legacy and emerging resource-monitoring technologies. As various competing Grid middleware offerings are released and evolve, an independent overarching monitoring service should act as a corner stone that ties these systems together. In particular we see the need for a robust, generic system that does not require additional software components to be installed, but instead can utilise existing local monitoring infrastructure.

While most resources use common Internet protocols for underlying communication, they also use a diverse range of application-level protocols to provide resource specific interaction. Furthermore, not only do the local agents differ in the way in which they can communicate, but also by the type and format of the data that they produce. The heterogeneous data that can be collected from these resources can only be truly useful if it can be normalised, so that a homogeneous view of the data can be produced. This transformation of raw data into marked-up metadata creates the possibility of equipping the data with semantic meaning, that can then be used by a range of higher-level tools

for tasks such as intelligent monitoring, policing SLA or QoS agreements, or for generating higher-level knowledge for a range of other tools.

GridRM is not intended to interact with instrumented wide-area applications; rather it is designed to monitor the resources that an application may use. The main objective of the project is to produce ubiquitous and seamless mechanisms to communicate and interact with heterogeneous data sources using a standardised and extensible architecture.

In section 2 we provide a detailed overview of the GridRM architecture, highlight the mechanisms that abstract clients from underlying resource heterogeneity, and outline management and events features. In section 3 we compare GridRM to related projects. Finally in section 4 summarise and conclude the paper and outline future work.

2. Architecture

GridRM provides an extensible architecture that operates over the wide area and supports a Grid sites' autonomous control of local resources. GridRM Gateways are used to coordinate the management and monitoring of resources at each site. GridRM consists of two layers, a Global and Local Layer.

The Global Layer, which provides inter-grid site, or Virtual Organisation (VO) interaction, is interconnected with a lightweight implementation of the Grid Monitoring

Architecture (GMA), known as jGMA [3]. At the Local Layer each gateway provides an access point to local real-time and historical resource data within its control. Data is collected from native agents and normalised into annotated client resource information that is tailored to a client's needs.

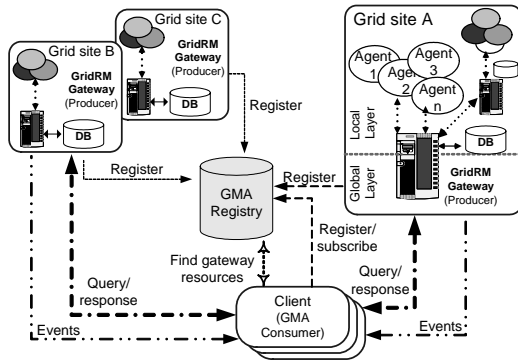


Figure 1: The GridRM Architecture

2.1 Normalisation

Normalisation is the process of collecting and transforming raw data from diverse native agents into information that is defined and described in a consistent and standard way. Normalisation is used to provide a homogeneous view of resources.

Native monitoring agents can produce data in many different formats. Data could be:

- Encoded in binary, ASCII, Unicode, or other formats.
- Represented using comma-delimited fields, name-value pairs, or a metadata language like XML or SGML.
- Described in a context specific way, requiring the client to have knowledge of the underlying system in order for the data to be meaningful.
- Measured using different units.

For example, one agent might represent memory using bytes, while others may use Kbytes or Mbytes. In addition, the range of data provided by an agent may vary, depending on the underlying host type.

Finally, the granularity of data may vary. Granularity refers to the size of the units of data released by an agent. Fine-granularity means individual data units are small and specialised to specific data fields, for example describing the value of a register, or representing the one-minute system load. Coarse-granularity refers to

the release of records, or blocks of related data; at the most extreme an agent may release many Kilobytes of data. Granularity can also be affected by the number of resources an agent is monitoring, such as in the case of a cluster.

In order for normalisation to occur the following conditions must hold true.

1. Standard semantics must be applied to resource data. For example, common entity and attribute names must be defined and used, regardless of the native terminology employed by agents. Here various agents may refer to processor load as CPUload, load, or by an Object Identifier (OID) such as 1.3.6.1.4.1.2021.4.5.0.
2. Resources and attributes should be logically organised into standard groups, so that resource values can be uniquely referenced and retrieved from monitored entities. For example, the relationship between processors within a host and the nodes with in a cluster.
3. A standard query language is required to retrieve information regardless of the underlying agent's native protocol and query syntax. This applies when synchronously querying a resource for data and also when subscribing to receive asynchronous events.
4. Data units must be represented using standard units. If raw data returned by an agent uses a non-standard metric, the data value should be converted, to use the standard unit.
5. If an agent does not natively provide a requested data field, then the normalisation process should attempt to construct the field by combining or filtering other available and suitable fields.

Ideally the client should unaware of the type of native agent that provided resource data. This requirement applies to the process of selecting and retrieving specific data as well as the format of the returned results.

2.1.1 XML Naming Schemas

XML naming schemas provide a way to enforce our normalisation requirements. A naming schema is an abstract description of the name, unit, meaning, organisation and relation of attributes that compose a given entity within a domain of reference. For example, the GLUE [4] naming schema provides a common conceptual model used for grid resource monitoring and discovery. GLUE provides common terms of reference and semantics that

define how computing, storage and network resources, and their attributes, should be labelled, described and related.

2.1.2 Translation Schemas

In GridRM we use the term translation schema to define an agent-specific instantiation of a naming schema. While a naming schema provides a conceptual semantic description of a class of resource, the translation schema is an implementation of a naming schema for a specific agent. For example the GLUE-CE Host translation schema for Ganglia, determines which Ganglia XML elements are needed to complete a GLUE-CE Host query. The translation schema may summarise or convert native agent (e.g. Ganglia) data units as appropriate to complete the query. For example memory values reported in Kbytes by Ganglia are converted to Mbytes to meet GLUE requirements.

2.2 Local Layer

The gateway's Local Layer consists of modules for querying local resources, performing normalisation, event handling, storage, and retrieval of historical data, and security.

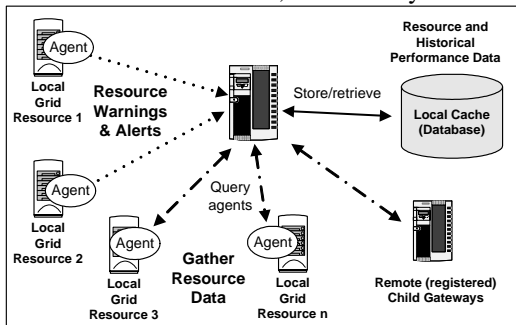


Figure 2: The Local Layer Overview

A key element of the Local Layer is the driver framework that is used for interacting with a variety of local agents. Framework drivers hide details of an agent's underlying communication protocol, query language and data formats.

Clients can query an arbitrary resource in a standard way and retrieve information described in a standard format. Clients do not require prior knowledge of resources; drivers can be queried to return details of naming schemas that are appropriate to the selected resource.

Figure 3 shows the steps that occur when a client queries a resource (step numbers refer to numbers highlighted in the figure):

Step 1: The client uses a standard query language (SQL) to retrieve CPU load from named resource. The client selects a naming schema, such as GLUE CE Host, to query the resource.

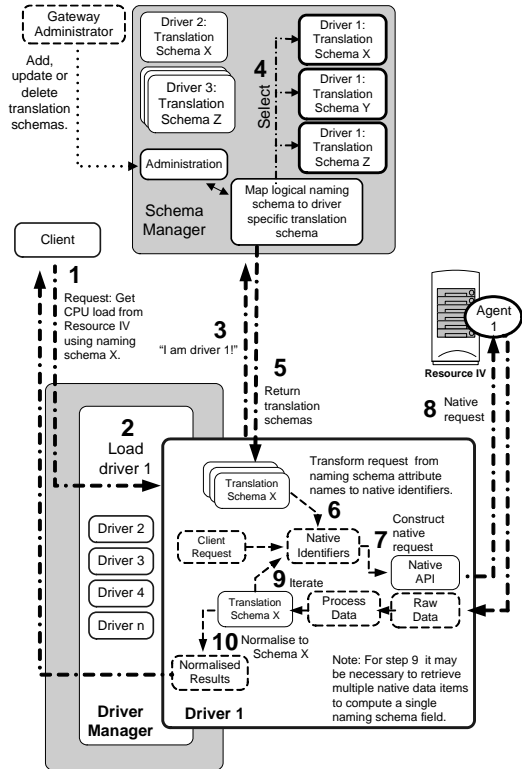


Figure 3: Driver operation

Step 2: The Driver Manager selects an appropriate driver for the agent being queried, such as the Ganglia driver.

Step 3: The driver identifies itself to the Schema Manager and requests all translation schemas that apply, such as Ganglia to GLUE.

Step 4: The Schema Manager locates the appropriate translation schemas and returns them to the driver.

Step 5: Translation schemas are dynamically loaded into the driver. If appropriate translation schemas are not available then the request terminates and the driver returns an error to the client. In Figure 3, translation schema X is available and returned to the driver; the client request can proceed.

Steps 6-8: The client query is composed using GLUE CE field names. The translation schema maps these names into native identifiers appropriate for the specific agent. The native agent API is then used to submit the request and raw data is returned to the driver.

Steps 9-10: The translation schema provides the rules for converting raw results into GLUE CE. If the agent does not provide the data needed to

construct a given attribute then null values are used to indicate the lack of data. The translation schema combines raw data values and performs conversions or parsing as necessary. The results are returned to the client as normalised and marked up information.

Drivers internally use, for example, GLUE, to logically organise resource data into groups. To a GridRM client these groups appear as relational databases, with tables and fields. In fact, regardless of the heterogeneity of the underlying data sources being queried, at the lowest level a client is only ever aware of interacting with (apparent) relational databases that provide a standard description and presentation of data.

The driver interface supports a query language (SQL) allowing clients to specify exactly which information should be filtered or fused and returned from a native agent. SQL is used throughout GridRM, from clients, through gateway layers and down to resource drivers. The separation between driver implementation and translation schema ensures that:

1. New naming and translation schemas can be added to the gateway at runtime and be made available to drivers dynamically without disrupting gateway operation,
2. A driver implementation does not require modification when a new naming schema is added to the gateway or an existing naming schema is updated,
3. A driver can support one or many translation schemas; the driver implementation is not affected,
4. Multiple clients can query the same driver concurrently using different naming schemas, thereby retrieving information in a format that meets their particular requirements,
5. Administrators can restrict client access to certain naming schemas at runtime.

Different drivers may implement the same naming schemas, but their native agents may not contain adequate data to compose certain data fields (see Figure 4). In these situations, drivers normally return null values, however the administrator could potentially specify which fields should be queried from each driver in order to populate a single result. The driver manager consults the Schema Manager for field definitions, constructs separate requests for each driver, submits them, and combines the results. Higher-level components are unaware that multiple drivers have been queried.

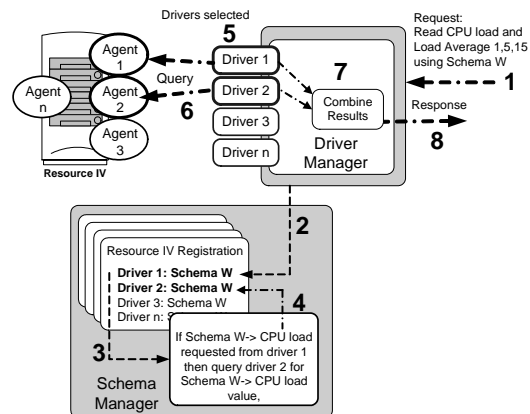


Figure 4: Using multiple agents to fulfil a single resource request

2.3 Resource Management

As well as querying agents for monitoring information, it is desirable to interact with agents to perform remote management.

GridRM supports remote client control over the operation of native agents, where appropriate. In fact the mechanisms previously described for monitoring resources can be applied directly to achieve uniform resource management and control. For example, a management naming schema can be used to interact with an agent.

Given a query language (SQL) and appropriate naming schemas, a client is able to read and write information into an agent. On receiving a read request (e.g. SQL SELECT), the driver queries the agent to return monitoring information as normal. On receiving a write request (e.g. SQL UPDATE) the driver performs a management function on the agent. For example, a client might query an agent to determine the current free disk space threshold for sending warning events, and reconfigure that threshold by updating the value of a field.

2.4 The Event System

Events generated by native agents must be captured, presented to clients in a standard format and logged as necessary for historical analysis. Figure 5 gives an overview of the Local Layer event system.

The event system is comprised of an Event Manager (EM) and multiple Native Event Listeners (NEL) modules. NELs provide agent specific event handling mechanisms; they listen on network ports and buffer incoming events. On receiving an event, each NEL is responsible

for transforming the native agent event into a standard format. This process is analogous to the normalisation performed by drivers. Event translation schemas that describe event conversion operations are dynamically loaded into each NEL. Normalised events are passed to the EM.

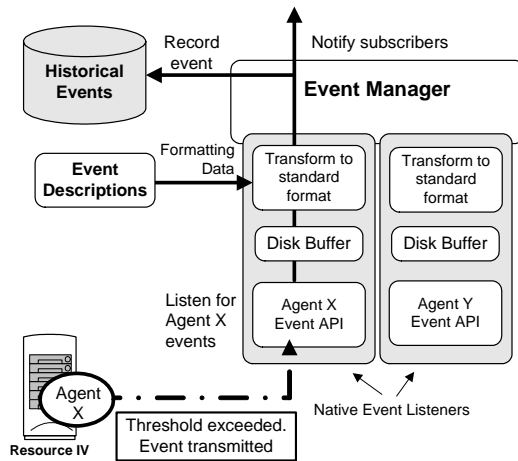


Figure 5: Local Layer Event System

The EM logs events to the gateway database for historical analysis, as required. The EM determines which normalised events should be transmitted to subscribed clients and forwards the events appropriately. Events intended for GMA clients are transmitted via the gateway's Global Layer. Administrators can also subscribe for events to be delivered by email or potentially other mechanisms like SMS.

The event system is designed to be independent of the driver architecture for a number of reasons:

1. Not all agents provide event mechanisms, for example Ganglia gmond uses synchronous request/response interaction; NetLogger only outputs events and SNMP provides both.
2. To receive events, a permanent server socket must be listening; this is not the case with driver instances, which are transient.
3. The event system is agent driven; predefined events are pushed to the gateway in response to changes in the agent's environment. Drivers, on the other hand are client controlled; data is pulled from agents in direct response to a client's needs.
4. The separation of event and driver operation implies that administrators can potentially control the operational priority of each. For example event processes may

be assigned a higher priority than drivers, so that events are propagated with minimal delay on a heavily queried gateway.

2.4 Global Layer

The Global Layer provides mechanisms to support interaction between gateways and clients. It can also enforce a site's coarse-grained security policies.

The Global Layer is based on the GGF GMA[5] and implemented using jGMA. The jGMA registry binds remote GridRM components together; gateways register their contact details, the types of event they produce and the categories of resource they manage. Clients locate gateways in the jGMA registry using keyword searching. Valid keywords include event names and resource categories. Thereafter, clients directly query gateways to interact with registered resources, and subscribe for events in order to be notified when conditions within the monitored environment change.

Logically the Global Layer appears as Figure 6, where a range of clients interacts with gateways to request resource information.

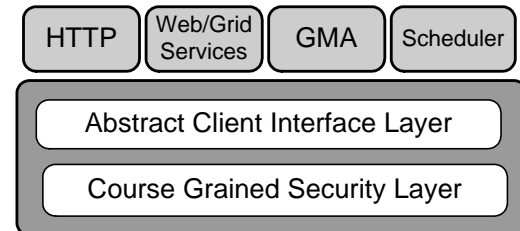


Figure 6: Logical representation of the Global Layer

GridRM security is layered: site-level policies are implemented in each gateway's Global Layer while resource security is provided at the Local Layer. Global Layer security can be configured with high-level policies to restrict client access to a site, based on VO or client IP address. For example, a site policy may decree that all requests from clients that do not belong to VO A are blocked, or that non-VO A clients are only permitted to retrieve cached information for a subset of resources. If permitted through the Global Layer, client requests face additional checks at the Local Layer.

2.4.1 Searching for Resources

Clients may wish to search the GMA registry for gateways and resources based on a number of metadata values such as:

1. Gateway name, VO membership, physical organisation name, site name (e.g. site X of organisation Y), geographical location (e.g. by gateway longitude and latitude),
2. By the categories of resource a gateway manages.

Typically metadata held by the registry is likely to be small in volume, and static or infrequently changing in nature.

The metadata that gateways provide as part of their GMA registration conforms to a standard GridRM template. Records detailing each gateways' local resources are intentionally coarse-grained, and indicate the categories of resource and types of event that can be produced. Clients can locate a resource and then query the gateway directly to retrieve real-time information.

We avoid registering all resource attributes directly in the GMA registry, as the overhead of maintaining potentially thousands of resource entries is likely to degrade the registry's performance.

3. Related Work

Although many systems are available in the field for resource and service monitoring, we found [6] that no single system adequately meets our requirements for Grid monitoring. For example, the majority of systems require custom agents to be installed on monitored hosts. GridRM does not suffer from this restriction; instead it aims to combine data from legacy agents and modern monitoring services. Given the existing investment in time and money for administrating resources across an organisation, we feel it is important to utilise the existing infrastructure as much as possible. Other systems that do not require the installation of additional custom agents include MapCenter [7], MDS3 [8], and R-GMA [9].

However, MapCenter is focused on monitoring the availability of services and provides limited scope for combining a unified interface with general resource monitoring. Although MDS3 provides customised provider mechanisms for inserting resource information from remote native agents, the main focus of the architecture is on interacting with OGSA-based grid

services. R-GMA provides an extensible mechanism to publish resource information from remote native agents. However, R-GMA producer instances are restricted to providing information for a single naming schema.

The majority of related projects require that agents submit information in a common format, prior to submission into the monitoring system. The few projects that actively support any degree of data normalisation include CODE [10], Mercury [11], and R-GMA. CODE agents currently use non-standard event naming schemas to describe their output. Mercury provides mechanisms to automatically convert native resource data to a specified naming schema. R-GMA provides mechanisms for user codes to register a naming schema, which is published into the system and for clients to query resources based on registered naming schemas.

GridRM advances the use of naming schemas by giving drivers the ability to support multiple naming schemas at runtime. This allows the same driver code to interact with an agent, while multiple clients can concurrently query for information formatted in different ways. Clients are free to select, for each query, whichever naming schema best suits their requirements. Furthermore, while GLUE is currently GridRM's default naming schema, our approach allows administrators to extend the description of resources by adding support for new naming schemas.

Projects can be categorised by the monitoring paradigms they support. For example, the majority of related projects [6] provide periodic polling capabilities; resources are repeatedly queried at predefined intervals and results are cached in a manager component for later retrieval. Other projects provide event-based monitoring where agents push resource information to a manager's cache, as local conditions change. Only a few projects, including CODE, MapCenter, Mercury, and R-GMA, provide an interface where clients can request a resource to be queried on demand. For the most part, where a query interface exists, we have found the articulation of queries to be limited. R-GMA is an exception and provides an expressive SQL syntax. GridRM does not restrict the approach used to gather resource information; instead each of the aforementioned monitoring paradigms can be used.

A number of projects allow clients to remotely manage agents. Autopilot [12], JAMM [13], NWS [14], and visPerf [15] provide remote controls to start and stop agents. In addition, Autopilot clients can modify aspects of an agent's configuration, including polling frequencies. GridRM follows this trend by potentially allowing authorised clients to interact with agents to change their configuration or set up.

4. Summary and Conclusions

GridRM provides a means of gathering data that can be used for a range of purposes, such as autonomic computing, verification of SLAs and required QOS, as well as creating higher-level knowledge. To achieve this, an open, and standards-based approach, that is independent of underlying Grid middleware, has been taken. Specifically GridRM gateways employ an extensible driver architecture to interact with existing native monitoring agents. GLUE schema and SQL are used in conjunction with drivers to present a uniform query interface and standard description of resource data, regardless of underlying agent heterogeneity.

An international test bed [1] has been created for the purpose studying GridRM's capabilities and performance. The test bed currently consists of 12 sites across 8 countries and is helping us gain real-world experience of deploying and configuring gateways at different sites, of operating behind firewalls and interfacing with existing resource agents.

4.1 Future Work

Initially we plan to develop further drivers to interface with agents from popular legacy and emerging monitoring systems. Currently GridRM is undergoing tests on our international test bed [1], based on our findings we will investigate optimisation strategies and introduce further gateway management features. These include easier overall installation and configuration, as well as the use of different database instantiations.

In the longer-term we will investigate interaction between third party scheduling, mobile agent, and ontological systems, so that the information gathered by GridRM can be used to police, for example service level and quality of service agreements, and also contribute to knowledge generation within knowledge-based systems.

References

- [1] GridRM, <http://gridrm.org>
- [2] M.A. Baker and G. Smith, GridRM: An Extensible Resource Management System, proceedings of the IEEE International Conference on Cluster Computing (Cluster 2003), Hong Kong, IEEE Computer Society Press, pp. 207-215, 2003, ISBN 0-7695-2066-9.
- [3] jGMA, <http://dsg.port.ac.uk/projects/jGMA>, June 2004.
- [4] GLUE-Schema, <http://www.hicb.org/glue/glue-schema/schema.htm>, June 2004.
- [5] B. Tierney et al, A Grid Monitoring Architecture, Global Grid Forum, <http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf>, January 2002.
- [6] G. Smith, GridRM: A Resource Monitoring Framework, PhD Dissertation, Chapter 3: State of the Art Review, University of Portsmouth, 2004.
- [7] F. Bonnassieux, R. Harakaly, and P. Primet, Automatic Services Discovery, Monitoring and Visualization of Grid Environments: The Mapcenter approach. Across Grid conference, 13-14 Feb 2003, Santiago de Compostela, Spain, February 2003.
- [8] Information Services in the Globus Toolkit 3.0, <http://www.globus.org/mds/>, August 2003.
- [9] R-GMA: Relational Grid Monitoring Architecture, <http://rgma.org>, December 2003.
- [10] Warren W. Smith. Code: A Framework for Control and Observation in Distributed Environments, <http://www.nas.nasa.gov/Research/Software/Open-Source/CODE/>, March 2004.
- [11] Mercury, <http://www.gridlab.org/WorkPackages/wp-11/>, February 2004.
- [12] Autopilot, <http://vibes.cs.uiuc.edu/Software/Autopilot/autopilot.htm>, June 2003.
- [13] Java Agents for Monitoring and Management (JAMM), <http://www-didc.lbl.gov/JAMM/>, July 2000.
- [14] Rich Wolski, Neil Spring, and Jim Hayes, The Network Weather Service: A Distributed Resource Performance

Forecasting Service for Metacomputing,
Journal of Future Generation Computing
Systems, 15(5-6), p757-768, October
1999.

- [15] Dong-Woo Lee, Jack Dongarra, and R.
S. Ramakrishna, visPerf: Monitoring
Tool for Grid Computing, International
Conference on Computational Science,
volume 2659, pages 233-243. Springer,
2003.