CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Wide-Area Resource Monitoring using GridRM and jGMA

M. A. Baker[*,†], G. M. Smith and M. Grove

The Distributed Systems Group, University of Portsmouth, Mercantile House, Hampshire Terrace, Portsmouth, PO1 2EG, U.K.

SUMMARY

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, to network components, and specialised data sources. This paper describes, GridRM, a standards-based framework for gathering data and information from resources distributed across the Grid that is independent of any given middleware and that can utilise legacy and emerging monitoring technologies.

The first part of the paper introduces GridRM and discusses why such a system is needed. In the second part of the paper we describe GridRM's architecture; here we focus on the so called Local and Global layers, and also detail jGMA, our wide-area event-based messaging system. We then move on to briefly describe GridRM's Web portal, which provides a basic demonstration of the system's ability. In the final part of the paper we summarise and conclude by highlighting our experiences and briefly outline future work.

KEY WORDS: *Grid, Monitoring, GMA, jGMA, Messaging, Standards-based*

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

[*]Correspondence to: The Distributed Systems Group, University of Portsmouth, Mercantile House, Hampshire Terrace, Portsmouth, PO1 2EG, U.K.
[†]E-mail: mark.baker@computer.org

Received 10 January 2004
Revised TBD

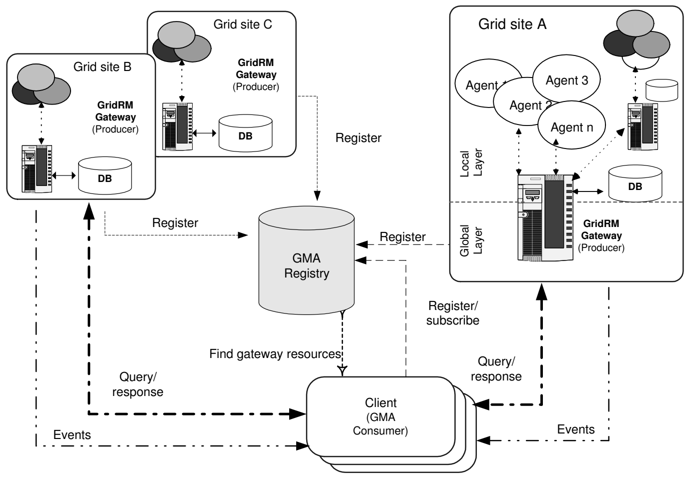


Figure 1. The GridRM Architecture

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 GridRM project is to produce ubiquitous and seamless mechanisms to communicate and interact with heterogeneous data sources using a standardised and extensible architecture.

In Section 1 we introduce GridRM and discuss why such a system is needed by the Grid community. In Section 2 we provide a detailed overview of GridRM's architecture, in particular we hightlight the capabilities of the Local and Global layers, along with our event-based messaging system, known as jGMA. In Section 3 we describe a Web portal that we have developed to demonstrate GridRM's capabilities. Finally, in Section 4, we summarise and conclude the paper and then outline the work we intend undertake in the near future.

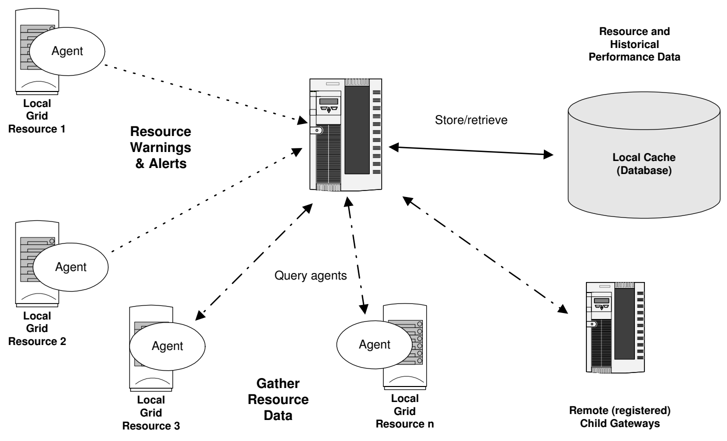


Figure 2. The Local Layer Overview

2. THE ARCHITECTURE OF GRIDRM

GridRM provides an extensible architecture that operates over the wide area and supports an individual site's 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 (an architectural sketch is shown in Figure 1).

The Global Layer, which provides inter-grid site, or Virtual Organisation (VO) interaction, is interconnected with an implementation of the Grid Monitoring Architecture (GMA), known as jGMA [3] (described in sub-section 2.2.1). 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 resource information that can be tailored to a client's needs.

2.1. The Local Layer

The Local Layer consists of modules for querying local resources, performing normalisation, event handling, storage, retrieval of historical data, and security (see Figure 2). A key element of the Local Layer is the driver framework that is used for interacting with a variety of local agents. The drivers hide details of an agents 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 translations (see sub-section 2.1.3) that are appropriate to the selected resource.

2.1.1. 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. In order for normalisation to occur, standard semantics must be applied to resource data, this is discussed in detail in [4].

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.

2.1.2. 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 [5] 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.

2.1.3. 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 [6] , 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. The Global Layer: Use of jGMA

The Global Layer provides mechanisms to support interaction between gateways and clients. The Global Layer uses jGMA [3] which is a pure Java reference implementation of the GGF's GMA [7], which represents the characteristics required for a scalable grid-based monitoring

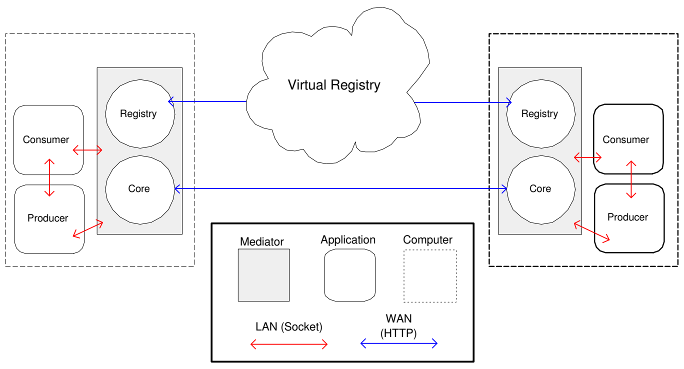


Figure 3. The jGMA Architecture

infrastructure. In GMA producers or consumers publish their exitence in the directory service (registry), which other producers or consumers may search for, connect to, and interact directly with.

Our motivation for developing jGMA, is described elsewhere [8][9], but can be summarised by saying existing systems were either embedded into larger software packages, such as the Monitoring and Discovery System (MDS) of Globus [10], and the Network Weather Service [11], and deemed difficult to breakout into a standalone version (desired for our purpose). Alternatively, the standalone versions presented other issues that were not easily overcome, including:

- Calling Python (pyGMA) [12] from Java, which is a requirement of GridRM, is not straightforward. While the Jython project [13] allows the use of Java from within Python, there is no simple mechanism for invoking Python from within Java without creating a customised and potentially complex JNI bridge.
- R-GMA [14] does provide a Java API, and initially it was thought that R-GMA would be a suitable implementation for GridRM. However, further investigation found that R-GMA was less than ideal for our purpose; the drawbacks are discussed in [9].

2.2.1. The jGMA Architecture

In attempting to create a reference implementation of GMA we designed jGMA with a number idealised features, including complying with the GMA specification, capable of scaling from LAN to WAN proportions, and across thousands of sites with millions of producers/consumers, supporting both non-blocking and blocking events, and capable of taking advantage of TLS [15] and/or the GSI [16]. In addition, we wanted the actual implementation to have a small well defined API, a minimal number of other installation dependencies, be easy to install and configure, be fast, have a minimal impact on its hosts, and be capable of working through firewalls.

The key component of GMA is its registry. We are developing a distributed registry service capable of plugging-in a number underlying technologies, ranging from a simple text files, to more sophisticated offerings based on Xindice [17], or MySQL [18].

The jGMA Architecture is shown in Figure 3 and consists of three main entities:

- Mediators to permit producers and consumers discover each other and allow remote communications,
- Consumers,
- Producers.

jGMA supports both blocking and non-blocking I/O; this provides the flexibility and functionality that will be required in most circumstances. jGMA has two modes of event passing; local, where communications are within one administration domain, i.e. behind a firewall, and global, when traversing more than one administrative domain, i.e. via one or more firewall(s). The former users sockets to pass events, and the latter HTTP, here the Mediator controls inter-site communications.

Figure 4 shows the layers that make up jGMA. The jGMA APIs have abstract interfaces between each component, which allow them to be altered or added to without necessitating changes within other layers. This makes the API easy to extend, for example adding HTTPS or SOAP would just require adding by a new handler capable of receiving and sending their messages.

The Mediator API is only used internally by jGMA, not by developers of producers and consumers. The registry component provides functionality for naming new clients, discovering other mediators and searching for information about other producers and consumers. The Ping Service is used as a heartbeat to monitor the health of all active jGMA components.

Producers and consumers are developed using the Client API that contains fourteen method calls, which are provided by the `GmaConnector` class. Using more than one basic producer or consumer enables more complex clients to be created.

2.2.2. API Changes

The underlying design and infrastructure of jGMA has evolved since its inception. The changes were driven by the need to improve performance, functionality, and robustness. For example, early versions used Apache Tomcat [19] to provide the mediator with an HTTP server, now we use Jetty's [20] embedded HTTP server to reduce the start-up costs as well as simplifying various jGMA APIs. The latest version of jGMA uses many new features of Java 1.5, including the concurrency and collection framework to improve performance of asynchronous message handling, and new library calls, such as blocking queues and URL connect timeouts.

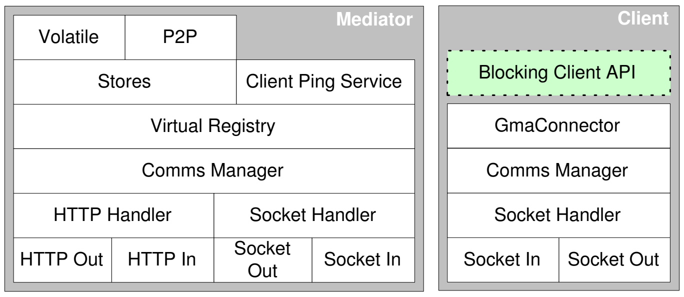


Figure 4. A Layered View of the Mediator and Client.

	Latency (Micro-seconds)		
Message Size (Bytes)	**Localhost Socket (a)**	**LAN Socket (b)**	**LAN HTTP (c)**
2	1015 (18)	1018 (62)	2627
8192	1166 (31)	1347 (1250)	4092
1048576	10748 (3646)	51614 (45347)	239383

Table I. jGMA Latency benchmark results (simple Java socket tests in brackets).

2.2.3. Initial jGMA Evaluation

A Java implementation of the traditional Ping-Pong test was used to measure point-to-point performance. Latency and bandwidth were measured using three different configurations of producers, consumers, and mediators:

a) Direct communication between a producer and consumer on a single node,
b) Direct communication between a producer and consumer over Fast Ethernet,
c) Communication between a producer and consumer over Fast Ethernet via two mediators communicating with HTTP.

Latency (**Table I**): For messages <8 Kbytes there is a fixed 1 millisecond additional latency over the baseline performance of Java sockets for both cases (a) and (b). This is the internal latency of jGMA due to the overhead of marshalling the message and the cost of storing and retrieving the messages from the send and receive buffers. For messages of 2 bytes, HTTP has an extra 1-millisecond latency, which is due to the additional, internal messaging between clients and the mediators, and the overheads of processing the inter-mediator HTTP encapsulated message. Beyond 8 Kbytes, the extra socket sends and HTTP encapsulation have a greater impact.

	Bandwidth (Mbytes/Second)		
Message Size (Bytes)	**Localhost Socket (a)**	**LAN Socket (b)**	**LAN HTTP (c)**
2	0.0021 (0.1057)	0.0018 (0.0305)	0.0012
8192	7.5618 (246.4440)	6.5996 (6.7499)	5.5120
1048576	82.5000 (274.2650)	11.0417 (11.1523)	10.8333

Table II. jGMA Bandwidth benchmark results (simple Java socket tests in brackets).

Bandwidth (**Table II**): The maximum bandwidth attained in case (a), is approximately 30% of the maximum. Whereas, in case (b) it is approximately 99%. This disparity is thought to be due to jGMA using a single thread to send messages. This means that when the available bandwidth is high, such as in case (a) and the message size is small the maximum number of messages that can be sent per second is bound by the speed of the CPU. This is why the bandwidth for messages <8 Kbytes for case (a) is lower than that of the plain Java sockets and why the bandwidth is better for 1 Mbyte messages.

Benchmarks of previous versions of jGMA (see [9]), which used Tomcat and Java 1.4 showed HTTP bandwidth of about 2 Mbytes/s. The new results show a maximum attainable bandwidth of 10.8 Mbytes/s, which is 98% of that for Java sockets. This is an improvement of more than four-fold over the old implementation.

In order to make more efficient use of bandwidth with small messages; multiple consumers and producers should be used, as a single client cannot saturate the network bandwidth. However by dividing the bandwidth by the size of message we can see that when sending these small messages a single client still allows about 960 messages per second to be sent using both HTTP and socket communications.

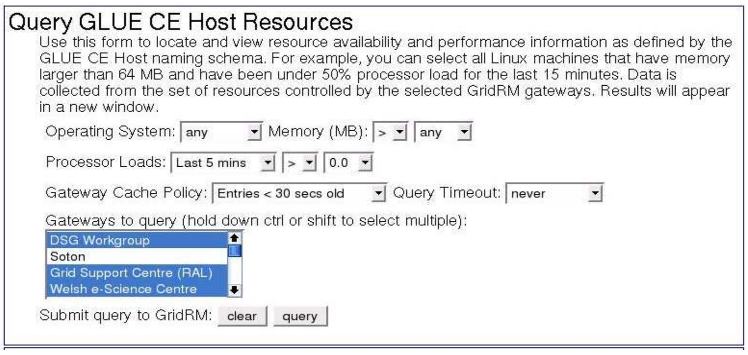
2.2.4. jGMA's use in GridRM

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 interact directly gateways to query registered resources, and subscribe for events in order to be notified when conditions within the monitored environment change.

2.2.5. Searching for Resources

Clients may wish to search the jGMA 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.


Query GLUE CE Host Resources

Use this form to locate and view resource availability and performance information as defined by the GLUE CE Host naming schema. For example, you can select all Linux machines that have memory larger than 64 MB and have been under 50% processor load for the last 15 minutes. Data is collected from the set of resources controlled by the selected GridRM gateways. Results will appear in a new window.

Operating System: any Memory (MB): > any

Processor Loads: Last 5 mins > 0.0

Gateway Cache Policy: Entries < 30 secs old Query Timeout: never

Gateways to query (hold down ctrl or shift to select multiple):

DSG Workgroup
Soton
Grid Support Centre (RAL)
Welsh e-Science Centre

Submit query to GridRM: clear query

Figure 5. The GridRM Query Interface

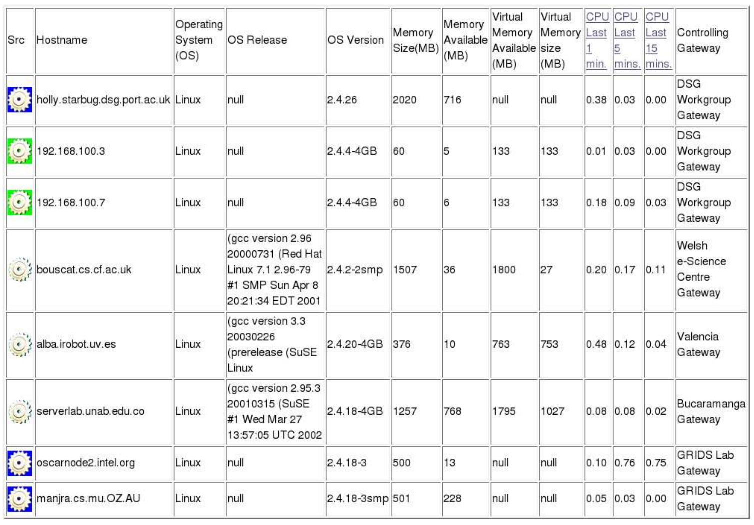
Typically metadata held by the registry is likely to be small in volume, and static or infrequently changing in nature. The metadata that gateways provide as part of their jGMA registration conforms to a standard GridRM template. Records detailing each gateways' local resources are intentionally coarse-grained, 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 jGMA registry, as the overhead of maintaining potentially thousands of resource entries is likely to degrade the registry's performance.

3. GridRM Demonstration Portal

A Web-based portal has been constructed to demonstrate GridRM's data harvesting capabilities. The portal provides an example of a higher-level service built on top of the Global Layer (GL) that hides details such as the SQL query syntax and gateway interaction, from the user.

The portal is constructed using Java Servlets, XHTML, Cascading Style Sheets (CSS) and jGMA. Users interact with the portal through Web browsers; the portal interacts with GridRM by exchanging XML documents over jGMA. The query interface shown in Figure 5, allows users to search for computer-based resources by operating system, physical memory, and processor load, across a range of gateways.

The attributes in Figure 5 are set to query multiple gateways for all computer-based resources that have a five-minute CPU load that is greater than zero. The query interface raises the user's perception of resource monitoring by hiding SQL query language details behind graphical widgets. Figure 6 shows a set of results from the query, rendered in XHTML. Columns three (OS) to twelve (CPU Last 15 mins) contain query results described using the GLUECE_host



Src	Hostname	Operating System (OS)	OS Release	OS Version	Memory Size(MB)	Memory Available (MB)	Virtual Memory Available (MB)	Virtual Memory size (MB)	CPU Last 1 min.	CPU Last 5 mins.	CPU Last 15 mins.	Controlling Gateway
🟦	holly.starbug.dsg.port.ac.uk	Linux	null	2.4.26	2020	716	null	null	0.38	0.03	0.00	DSG Workgroup Gateway
🟩	192.168.100.3	Linux	null	2.4.4-4GB	60	5	133	133	0.01	0.03	0.00	DSG Workgroup Gateway
🟩	192.168.100.7	Linux	null	2.4.4-4GB	60	6	133	133	0.18	0.09	0.03	DSG Workgroup Gateway
⚙	bouscat.cs.cf.ac.uk	Linux	(gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-79 #1 SMP Sun Apr 8 20:21:34 EDT 2001	2.4.2-2smp	1507	36	1800	27	0.20	0.17	0.11	Welsh e-Science Centre Gateway
⚙	alba.irobot.uv.es	Linux	(gcc version 3.3 20030226 (prerelease (SuSE Linux	2.4.20-4GB	376	10	763	753	0.48	0.12	0.04	Valencia Gateway
⚙	serverlab.unab.edu.co	Linux	(gcc version 2.95.3 20010315 (SuSE #1 Wed Mar 27 13:57:05 UTC 2002	2.4.18-4GB	1257	768	1795	1027	0.08	0.08	0.02	Bucaramanga Gateway
🟦	oscarnode2.intel.org	Linux	null	2.4.18-3	500	13	null	null	0.10	0.76	0.75	GRIDS Lab Gateway
🟦	manjra.cs.mu.OZ.AU	Linux	null	2.4.18-3smp	501	228	null	null	0.05	0.03	0.00	GRIDS Lab Gateway

Figure 6. Portal Tabular Results: Computer Hosts with Last 5-minute Load >0

naming schema, Columns one (Src), two (Hostname), and thirteen (Controlling Gateway) provide metadata that is used to highlight aspects of GridRM operation.

It is worth noting that null values indicate that a resources native agent was unable to provide adequate data to meet the GLUE requirements for the corresponding field, and therefore the translation schema was unable to construct a result. The metadata fields included in the portal's output highlight the use of different GridRM gateways and drivers. For example, the portal acted as a proxy by submitting the user query to multiple gateways and combining the results. In this instance, five controlling gateways were found to provide resources that met the query constraints. Of the eight resources returned, three different types of native agent were used to provide information. The Src (column one), highlights the drivers that were used, in this case Ganglia, SNMP v1 and Linux /proc.

4. SUMMARY AND CONCLUSIONS

4.1. GridRM

The GridRM framework data harvesting capabilities 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.

The GLUE schema and SQL are used extensively in conjunction with drivers to present a uniform query interface and standard description of resource data, regardless of underlying agent heterogeneity. It should be noted that a detailed comparison of GridRM with similar contempory systems is given in Chapter 3 of [4].

An international test bed [1] has been created for the purpose of studying GridRM's capabilities and performance. The test bed currently consists of twelve sites across eight 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.

4.2. jGMA

jGMA is a reference implementation of GMA. We were motivated to produce jGMA by the lack of a viable alternative to use with GridRM, and the evidence that such a low-level middleware offerring was generally needed for a range of other grid services and applications. We have presented the results of simple benchmarks that provide us with some insight into the expected performance and capabilities of jGMA.

4.3. Future Work

4.3.1. GridRM

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 test bed, 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 software, mobile agents, and introduce ontological services, so that the information gathered by GridRM can be used to police, for example SLAs and QoS agreements, and also contribute to knowledge generation for higher-level services.

4.3.2. jGMA

jGMA is at an early stage of development. Our immediate aim is to integrate components into the mediator that will provide a robust distributed Virtual Registery (VR), which will offer a fault tolerant naming, discovery, and search service across a deployment of jGMA. Currently a simple volatile registry has been implemented, this stores information about clients that are connected directly to the mediator. The VR will allow the location and query of registries managed by mediators over the wide-area. We are currently implementing a VR based on peer-to-peer technologies and algorithms.

After the registry implementation is complete we will undertake a full jGMA evaluation; here we will compare and contrast it with alternatives, such as R-GMA and the Globus MDS, such as undertaken in [21] and [22].

In the longer term we are keen to investigate the use of jGMA as part of the infrastructure for online gaming, which has recently become increasingly popular with the widespread uptake of home broadband Internet access. We believe that jGMA can provide a number of services, for example, for the distribution of user profiles, monitoring information, and user single-sign on.

REFERENCES

1. GridRM, http://gridrm.org
2. Baker, M.A. and Smith G,. GridRM: An Extensible Resource Monitoring 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/
4. Smith, G.M. GridRM: A Resource Monitoring Framework, PhD Dissertation, *University of Portsmouth, 2004*.
5. GLUE-Schema, http://www.hicb.org/glue/glue-schema/schema.htm, June 2004.
6. Ganglia, http://ganglia.sourceforge.net/
7. Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., and Wolski, R. A Grid Monitoring Architecture, *Global Grid Forum*, http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf, January 2002.
8. Baker, M.A. and Grove, M. jGMA: A lightweight implementation of the Grid Monitoring Architecture, *Proceedings of the UK e-Science All Hands Meeting (AHM 2004)*, East Midlands Conference Centre, Nottingham, from 31st August - 3rd September 2004, ISBN 1-904425-21-6
9. Grove, M, A lightweight implementation of the Grid Monitoring Architecture, *DSG Technical Report, September 2004*, http://dsg.port.ac.uk/mjeg/jGMA/jgma_report2004.pdf
10. Globus MDS, http://www-unix.globus.org/toolkit/mds/
11. Network Weather Service, http://nws.npaci.edu/NWS/
12. pyGMA, http://www-didc.lbl.gov/pyGMA/
13. Jython, http://www.jython.org/
14. R-GMA, http://www.r-gma.org/
15. TLS, http://www.ietf.org/html.charters/tls-charter.html
16. GSI working Group, https://forge.gridforum.org/projects/gsi-wg
17. http://xml.apache.org/xindice/
18. MySQL, http://www.mysql.com/
19. Tomcat, http://jakarta.apache.org/tomcat/
20. Jetty, http://jetty.mortbay.org/jetty/
21. Zhang, X., Freschl F., and Schopf J. A Performance Study of Monitoring and Information Services for Distributed Systems, Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03), *IEEE CS Press*, pp 270, 2003,ISBN:0-7695-1965-2.
22. Cooke, A. W., et al, The Relational Grid Monitoring Architecture: Mediating Information about the Grid, to appear in the Journal of Grid Computing. *Kluwer Press*, http://www.kluweronline.com/issn/1570-7873/, 2005,