

Grid monitoring: a holistic approach.

A.P. Millar¹

1 - Dept. of Physics and Astronomy, University of Glasgow, Glasgow, G12 8QQ.

Abstract

Computational grids involve the intersection of different geographically distributed communities: the resource-users and the resource-providers. Grid monitoring is required by various super-sets of these communities as everyone wants to know something about how the Grid is performing.

Many grid monitoring systems have a top-down prescriptive approach. Grid monitoring is often considered separable from the monitoring of non-grid elements within a functioning grid site. This schism can result in grid monitoring ignoring existing technologies, often requiring some form of bridging between different systems.

In this paper, the key interested parties of grid monitoring are discussed along with what information they are interested in. The three-component model of monitoring systems is presented and illustrated with examples. From this model, we deduce the concept of a universal sensor and discuss its role in providing different information to different people.

The MonAMI project is introduced with its aim of providing an implementation of a universal sensor. Possible uses of MonAMI are discussed.

1 Introduction

A Grid provides computational resources that many end-users would like to use. Like any computational resource, Grid resources can suffer from performance problems or even complete outages. Sometimes end-users can precipitate problems by following unexpected usage patterns. These unusual usages can cause greatly reduced performance or may even cause services to fail. Under these circumstances it is important to establish what triggered a problem so it can be prevented in the future.

None of these requirements are unusual in computational services. What makes monitoring in a grid environment challenging is the interaction of the geographically distributed groups of people.

1.1 Who wants to know?

Consider a (non-Grid) service that attracts many users. These users may need to know the current status of the service. To provide this information, the service provider can monitor the service and make

current status information available through a centralised set of webpages. Often the service is distributed, but will have sufficient cohesion to enforce a single monitoring solution for the different components.

One possible translation of the central monitoring to Grid-based computing is that a single site might provide information on how their components are performing. This information would give an indication of how heavily their resources are being used. This information would be gathered (from available sensors), collected and presented perhaps on a suitable set of web pages. The gathered information would also alert the site administrator when a service is failing or performing badly, allowing a rapid response in fixing the problem. Site administrators are principally interested in whether they are providing a working service, that the resources (e.g. free disk space) are sufficient for medium term projected usage and whether they are satisfying their agreed service provision.

Another translation of this central monitoring is to a Virtual Organisation (VO). A VO represents a group of Grid users with similar aims and access to similar computational resources. They might want to monitor the Grid with particular views specific to the tasks they wish to undertake. Each VO can undertake monitoring of resources (using available sensors) and provide customised views of available data. The VO may have specific software or site-local service requirements, the availability of which dictates which sites they can use. They might also wish to conduct more detailed monitoring of the services they use to check for bottle-necks, both from their code-base and from the available sites.

A third group interested in monitoring are the software developers. With small-scale deployments, it is possible for software engineers to gain access to servers to look for the cause of performance problems. With wide-scale production deployment, this is no longer feasible. Instead, the monitoring infrastructure must allow people to gather more detailed information. This information is only needed when a bottleneck is discovered. Collecting this information routinely would be prohibitive, so the monitoring system must allow monitoring on-demand.

Yet a fourth translation of centralised monitoring is for “the grid” to monitor itself. Any grid will provide grid-level functionality: activity based on multiple sites providing similar services. Often, these services can be tested in isolation (as part of the site-level tests), but a much broader testing can only be conducted by using the multi-site functionality. Centralised grid-level monitoring tests the multi-site functionality. It is similar to the site-level monitoring but includes tests that site-administrators cannot conduct.

To illustrate introspective monitoring (the ability of a grid to monitor itself), the grid established for the forthcoming Large Hadron Collider (LHC) facility (based at CERN) will be examined briefly. The expected rates of data coming from the various LHC experiment detectors is immense: in excess of 14 PB per year (or 135 TB per day). To support processing this volume of data, the CERN partner countries have established a grid called the World-wide LHC Computation Grid (WLCG) (for further details see [1]).

Within the WLCG, sites are not truly autonomous but must go through a vetting process before becoming part of the grid[2]. Within the UK, contribution to this grid effort has been coordinated through the GridPP project[3]. Membership of these groups engenders a sense of community between the collaborating sites, moreover membership of WLCG is subject to signing a Memorandum of Understanding (WLCG MoU[4]) stating the expected level of service provision. Under these circumstances centralised Grid-wide monitoring is strongly required. Various centralised monitoring projects exist within WLCG[5] and GridPP[6].

In summary, a Grid service running on a particular site might be monitored by the local site-administrator, each of the different VOs that use the site, by the different software developers and centrally across the grid. The monitoring requirements are not static. Over time, other groups of users might require additional monitoring if, for example, new VOs are formed.

Each group interested in monitoring grid services might use different systems for monitoring as there is currently no single well-accepted, universally deployed monitoring system. This lack of consensus presents a problem in gathering the information needed to providing the monitoring information.

One possible solution to this problem is presented in this paper. The work is open-ended and is suitable for collaboration.

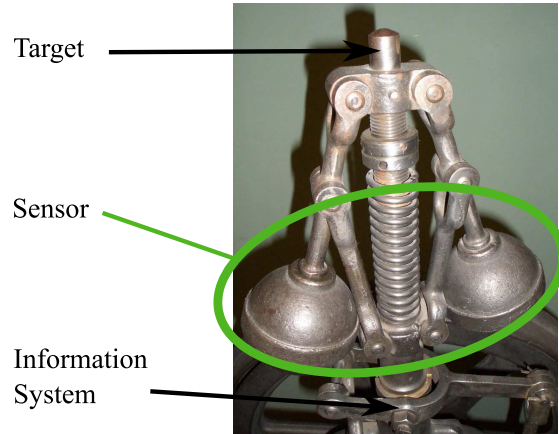


Figure 1: An example of the three-component model implemented in hardware.

2 Terminology

In discussing monitoring systems, it is useful to define the terminology that will be used throughout this paper.

Different monitoring systems exist with varying complexity. These systems may have many different components with which they provide the monitoring service. However, at the most abstract level, all monitoring systems can be understood in terms of the three-component model.

2.1 The three-component model

The three-component model places components of a monitoring system into one of three categories: target, sensor or information system.

A target is the object of the monitoring, something one wishes to ascertain its current state. It is a common feature amongst targets that, although they might provide some facility by which their current state can be monitored, they do not actively undertake any monitoring themselves. Examples of targets are file-systems, databases, or grid services.

The information system provides some method of storing the target’s current state. An information system has no knowledge of how to get information from the target; it only provides a medium to store the captured information. Examples of information systems include a webpage, some portion of memory, a database, or an email delivered to the site administrator.

A sensor is a component that is sensitive to the target’s current state. It uses this sensitivity to gather the target’s current state and store this data within some information system. A sensor translates information from the target to the information.

A simple mechanical example of the three-

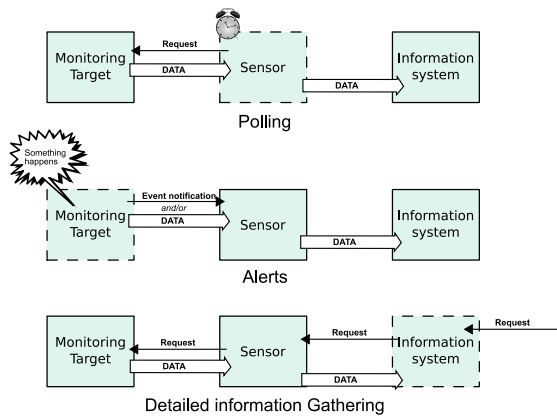


Figure 2: The three monitoring dataflow models: polling, alerts and detailed information gathering (DIG). The dashed box indicates which component initiated the request for information.

component model (the Watt governor) is shown in Figure 1. Here, the monitoring target is the engine, the sensor is the pendulum (which is sensitive to the rotational speed) and the information system is the height of the collar.

2.2 The three types of dataflow models

Monitoring involves the flow of information from the target to the information system. Using the three-component model, we can classify these interactions as one of three types, based on which component initiated the flow of information.

The three types of monitoring interaction are polling, alerts and detailed information gathering (DIG). These correspond to the dataflow being triggered by the sensor, target and information system respectively, as shown in Figure 2.

Support for polling dataflow is perhaps the most commonly implemented system. The sensor, acting on an interrupt from an internal timer, reads the current state of the sensor. It stores this information within the information system. This information is typically displayed as a graph or subject to further analysis.

Support for alerts dataflow is available from some targets. Typically, this dataflow is triggered by some device or software service and will alert the sensor of some asynchronous event. Examples of asynchronous events include user activity or running out of some resource. The sensor can translated this information into a suitable form before sending it to the information system. This can be used for accounting, alerts when some component is failing or logging activity.

The third dataflow model is DIG. Some external agent (e.g. the end-user) requests additional infor-

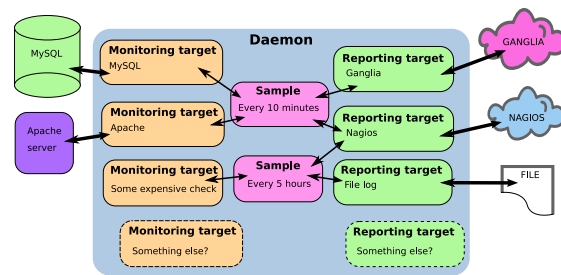


Figure 3: Components of MonAMI, illustrated with a typical configuration for site-local monitoring.

mation. The request for detailed information is processed and the results are sent back to the information system. One possible use for DIG is when a developer or end-user is investigating a problem with a production service. Another example of DIG is gathering configuration parameters to check that services have been configured correctly.

2.3 Hybrid dataflow systems

A three-component system, although useful in describing the basic monitoring interactions, is often not found in real-world implementations; instead, interactions between different components give rise to more complex behaviour. In general, complex systems can be broken down into component parts, each of which follow the three-component model.

For example, a monitoring system might watch performance metrics and send an email when the values pass beyond acceptable limits. This system can be broken down into two subsystems: one that periodically monitors the current performance, the other that sends an email when there is a problem. The first subsystem operates under the polling dataflow model, periodically updating the current measured values. The second subsystem operates under the alert dataflow model, sending an email when there is a problem. The two subsystems are linked because the information system in the polling subsystem is the target for the alert subsystem.

3 MonAMI

If all interested parties involved with providing a Grid service were to use the same monitoring infrastructure then a suite of sensors could be developed that would satisfy the monitoring demands. In practise different people are using different monitoring infrastructures. This leads to the awkward situation where there is no one clear monitoring system for which grid middleware developers should provide sensors, no clear monitoring system that people developing monitoring front-ends should look to.

One could view the job of a sensor as having two parts: to gather information about a target and to report this information to some information system. If these two parts were separated, then the collected information could be sent to any number of information systems, based on the sensor's configuration. This would allow the sensor to provide information to any number of interested parties, from local site-administrators through to VO- or Grid-specific monitoring.

We can define a *universal sensor* as being a sensor that can send gathered information to many different (ideally, to all necessary) information systems. To be useful, the sensor should be extensible, so support for additional information systems can be added.

Further, if the sensor configuration is separated into different parts (based on the different interested parties) then providing monitoring information for different people via different information systems becomes tractable. Each group interested in monitoring some grid services can specify their interests independent of others and the universal sensor will capture sufficient information and deliver it according to the different delivery requirements.

The MonAMI project[7] provides a framework for developing a universal sensor. It uses a plugin infrastructure to support different information systems, support which can be extended to include additional information systems as needed.

It is important to state that MonAMI does not aim to be a complete monitoring solution. Data within any information system has value only if it is then further analysed, for example presented as a graph on a webpage, triggering an email, or used for trend-analysis to predict when computing resources need updating.

3.1 The MonAMI daemon

The MonAMI framework currently provides a lightweight monitoring daemon. The advantage of running a daemon process is that many targets can be monitored concurrently. The daemon will automatically aggregate information so that requests for the same monitoring information are consolidated and the impact of monitoring the target is minimised.

This daemon periodically checks the status of *monitoring targets* and reports their status to one or more *reporting targets*. This is the polling model of monitoring dataflow. At the time of writing, support for the asynchronous dataflows (alerts and DIG) is being added.

A target (whether monitoring or reporting) is a specific instance of a *plugin*. The plugin is generic concept (a MySQL server, for example), whilst the target is specific (e.g. the MySQL server running on

localhost). The advantage of this approach is that the monitoring plugins need know nothing about how the data is to be reported. Data from a monitoring target can be sent to any number of reporting targets. This also allows MonAMI to be extended, both in what information is gathered and to what information systems data is to be sent.

The list of available monitoring and reporting plugins is available on the MonAMI project webpage[7] and currently include monitoring the local filesystem, running processes, Apache HTTP[17], MySQL[18] and Tomcat[19] servers. Support exists for reporting plugins, including Nagios[15], Ganglia[16] and MonaLisa[11] in addition to simple file-based logging.

Figure 3 shows a simple configuration for MonAMI with the configuration components shown as functional blocks. In this example, routine monitoring services are reported to two information systems (Nagios[15] and Ganglia[16]). There is also a more intensive test, which is conducted far less frequently to reduce its impact. The results of this test are reported to Nagios (to alert sysadmin of undesirable results) and recorded in a log file.

3.2 MonAMI and other components

The grid community has several projects with overlapping goals and implementations. The grid monitoring workgroup of the Global Grid Forum (GGF)[8] has produced a specification for grid-level monitoring: GMA[9]. The R-GMA project[10] has produced a realisation of this work.

Separate from R-GMA, the MonaLisa project[11] has a monitoring infrastructure that is widely used, including a number monitoring targets and end-client applications.

Within WLCG, effort is underway in developing site-level and grid-level functionality tests through the SAM[12] project. This information is aimed towards providing information for site-administrators. For end-users, the Dashboard[13] project aims to provide monitoring information that is easy to navigate.

As previously stated, MonAMI does not aim to be a complete solution, but rather to provide useful data to other existing projects. It aims to be part of these larger systems by providing information on key grid services. As a universal sensor, it aims to report to whichever combination of monitoring information systems is currently in use.

Support for additional systems can be added by writing an additional reporting plugin for MonAMI. The configuration can be updated to include the extra reporting, allowing data to be reported without affecting any existing monitoring.

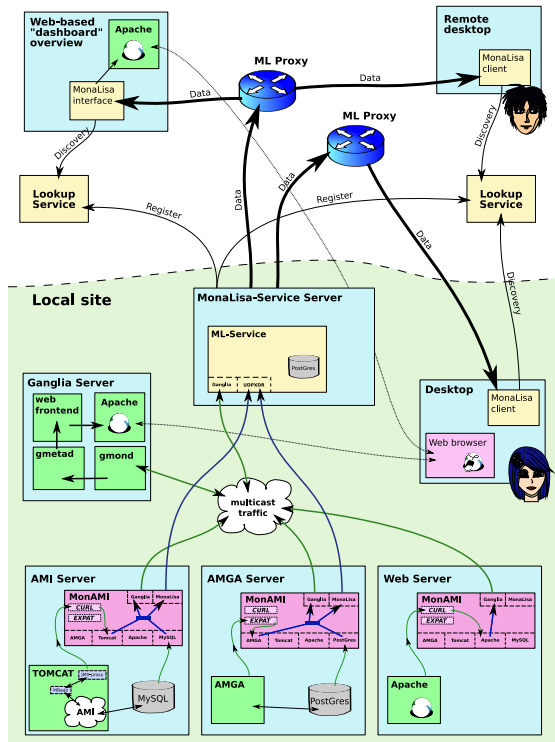


Figure 4: An example of MonAMI deployment.

3.3 Example of MonAMI deployment

MonAMI can be deployed in a number of different configurations. It is possible to install a single instance of MonAMI and have this daemon monitor services remotely (for services that support remote monitoring). This deployment configuration has minimal deployment impact, but may be inappropriate for services that only allow remote monitoring with insecure password authorisation.

An alternative approach is to install MonAMI on all servers that need to be monitored. This approach is more flexible and an example of such a deployment is shown in Figure 4. In this example, monitoring information goes to both the local site administrator (who is using Ganglia[16]) and to a grid-wide information system MonaLisa[11].

Grid-level monitoring is available through the clients provided with MonaLisa, and also through centralised services that query the MonaLisa monitoring information system and provide a webpage summarising the available data (such as Dashboard[13]).

Should additional monitoring be required (such as reporting information to R-GMA or SAM), only an additional file is needed on the monitoring system; this configures MonAMI to start reporting the monitoring information to the additional monitoring.

3.4 An explicit use-case

The following use-case illustrates the benefits of using MonAMI for monitoring. The use-case assumes monitoring is taking place within WLCG, but applies more generally. The pattern here is kept abstract because currently MonAMI is being integrated into the monitoring of different services within various grid efforts and the use-case is generally applicable.

The developers of some grid software need to provide a mechanism that allows site-level monitoring of their service. This is to alert the site-administrators when the local instance of the service has failed or is in danger of failing, and provide some performance metrics to allow those components causing performance bottle-necks to be uncovered.

For providing alerts, supporting Nagios by providing suitable scripts is a choice. Nagios is commonly deployed amongst the larger sites, but its use is not uniform. Providing only Nagios monitoring scripts would support only a subset of sites within the grid. Other site-level monitoring infrastructures exist; supporting all options would require prohibitively large development and support time.

By decoupling the acquisition of data from the delivery of that data, MonAMI allows a single plugin to provide information to any supporting information system (i.e. provide data to any monitoring system MonAMI supports). Development effort can be limited to the single MonAMI plugin and documentation of this plugin's use.

If the monitoring plugin is included within the MonAMI distribution, the service monitoring is expressed as suitable MonAMI configuration files, delivered as an RPM with a dependency on an MonAMI RPM.

If the monitoring plugin is not yet included within a MonAMI release, the service monitoring RPM can include the plugin. MonAMI will include this extra plugin at start-up and provide additional monitoring capability.

The site-administrator connects the monitoring to her existing monitoring infrastructure (provided it is supported by MonAMI). This might require some site-specific adjustments, based on expecting behaviour and site-specific configuration. Nominal, expected values can be published, but the site-administrator might need to adjust these, based on observed usage patterns.

3.5 Anatomy of a plugin

A MonAMI plugin is simply a shared object (or a shared library). Under the various GNU/Linux

distributions, it is now common to include support for (and use by default) the ELF file format for executable content. ELF includes support for dynamically loading data. The core part of MonAMI (MonAMI-core) uses the OS provided linker support for loading shared objects to load plugins.

It's important to note that MonAMI-core has no internal list of what plugins are available. Instead, it scans for available plugins at run time and loads those that are available. This allows for additional plugins to be included independently of upgrading the MonAMI distribution.

Each plugin contains a brief overview of the plugin; containing the plugin name, a brief description and what operations the plugin supports.

A plugin can be mentioned within the configuration multiple times, once for each service the plugin is to monitor, or information system to which the plugin is to report. Each mention within the configuration file creates a new target based on the named plugin (as described in 3.1).

The synchronous API requires a plugin to provide two functions (`monami_open` and `monami_close`) and at least one of a further two functions (`monami_read` and `monami_write`) depending on whether the plugin is providing support for monitoring or reporting activity respectively.

monami_open is called once for each target and allows the allocation of the resources for that target and register with remote services, if necessary. It also allows the plugin to check that the necessary parameters have been provided.

monami_close is called before MonAMI terminates. It provides a clean method of de-registering any allocated resources and removing registration with remote sites, if necessary.

monami_read is called when requesting information from a monitoring plugin. The plugin acquires the current status of the service it is monitoring and provides this information to MonAMI-core.

monami_write is called when data has been collected for the reporting plugin to store or pass on to some information system.

At time of writing, work is underway towards adding support for asynchronous monitoring. This will involve extending the above API to support the two asynchronous dataflow models (Alerts and DIG). The changes aim to be backward compatible, allowing the use of existing plugins without change.

The MonAMI source provides a framework for writing plugins; a basic plugin need do little more

that provide three of the above four functions and use a `provide` macro to create the plugin description.

Further details on how to write plugins is available within the Developer's guide, which documents MonAMI and describes the pedagogical code included in the source distribution.

4 Conclusions

MonAMI aims to provide a framework for developing a universal sensor: a sensor that can report to many different information systems. It allows the monitoring of multiple targets, with information about each target being sent to any number of information systems. MonAMI uses a plugin system so additional monitoring can be supported by adding extra plugins.

For each server, the list of what is to be monitored is separable. This allows the addition of extra monitoring requirements without affecting the existing monitoring.

MonAMI does not aim to be a complete monitoring solution, but rather a building block: a low overhead, extensible method of providing multiple groups of people with the monitoring information they need.

References

- [1] The WLCG project (until recently known as LCG). <http://lcg.web.cern.ch/LCG/>
- [2] Information about how to join WLCG, this includes testing that the WLCG software stack has been installed properly. <http://lcg.web.cern.ch/LCG/Sites/sites.html>
- [3] *GridPP: Development of the UK Computing Grid for Particle Physics* The GridPP collaboration, 2006 J Phys G: Nuclear and Particle Physics **32** N1-N20 (Technical supplement)
- [4] The WLCG "Memorandum of Understanding" document. <http://lcg.web.cern.ch/lcg/C-RRB/MoU/WLCGMoU.pdf>
- [5] Monitoring within the WLCG is organised through the Grid Operations Centre (GoC). <http://goc.grid-support.ac.uk/gridsite/monitoring/>
- [6] Within the GridPP project, centralised monitoring is available from the man website. <http://www.gridpp.ac.uk/>
- [7] The MonAMI project. Aims to be a universal sensor framework. <http://monami.sourceforge.net/>

- [8] The Global Grid Forum.
<http://www.gridforum.org/>
- [9] Grid Monitoring Architecture specification. Provides information on how compliant grid-wide monitoring should interact.
<http://www-didc.lbl.gov/GGF-PERF/GMA-WG/>
- [10] The R-GMA project: an implementation of GMA specification used within the LCG and EGEE projects.
<http://www.r-gma.org/>
- [11] The MonaLisa project. A widely used grid-wide monitoring system.
<http://monalisa.cacr.caltech.edu/monalisa.htm>
- [12] WLCG Service Availability Monitoring.
<https://lcg-sam.cern.ch:8443/sam/sam.py>
- [13] The Dashboard project; provides a web-based overview of available information.
<https://uimon.cern.ch/twiki/bin/view/LCG/ARDA-CMS-Dashboard>
- [14] The LEMON project, extensive testing for large sites.
<http://lemon.web.cern.ch/lemon/index.htm>
- [15] The Nagios project. An advanced generic tool for triggering alerts based on service availability and performance.
<http://www.nagios.org/>
- [16] The Ganglia project. A hierarchical generic tool for providing graphical monitoring of service performance.
<http://ganglia.sourceforge.net/>
- [17] The Apache HTTP server.
<http://httpd.apache.org/>
- [18] MySQL: perhaps the most popular database.
<http://www.mysql.com/>
- [19] Tomcat: the Apache foundation's Java application server.
<http://tomcat.apache.org/>