

# MDS4 and Project Deployments

July 28, 2005

Contact: [mds-internal@mcs.anl.gov](mailto:mds-internal@mcs.anl.gov)

## Abstract

We describe the Monitoring and Discovery System shipped with Globus Toolkit version 4 (MDS4) and some ways in which it can be used within a wide area deployment. After a brief overview of MDS4, this document discusses what types of data are appropriate to use with this system, and then walks through a deployment for gathering metascheduling data and several options for collecting software deployment information.

## 1 MDS4 Overview

Many Grid projects are in the process of evaluating their currently deployed monitoring systems, most of which grew out of combining disparate local approaches. Often the overall monitoring system is found lacking in terms of flexibility, support for standard query interfaces, scalability, and support for Web Services interfaces.

The Globus Toolkit's Monitoring and Discovery System (MDS) implements a standard Web Services interface to a variety of local monitoring tools and other information sources. As shown in Figure 1, MDS4 can be understood as a "protocol hourglass," defining standard protocols for information access and delivery and standard schemas for information representation. Below the neck of the hourglass, MDS4 interfaces to different local information sources, translating their diverse schemas into appropriate XML schema (based on standards such as the GLUE schema whenever possible). Above the neck of the hourglass, various tools and applications can be constructed that take advantage of the uniform Web Services query, subscription, and notification interfaces to those information source that MDS4 implements.

Grid computing resources and services can advertise a large amount of data for many different use cases. MDS4 was specifically designed to address the needs of a Grid monitoring system – one that publishes data that is of use by multiple people across multiple administrative domains. As such, it is not an event handling system, as is NetLogger, or a cluster monitor in its own right, as is Ganglia, but acts as a unifying layer above these information sources. Needless to say, one needs to be careful when designing a monitoring system to strike the right balance between quantity and timeliness of information, on the one hand, and associated costs, on the other. MDS4 allows the user to manage such tradeoffs by controlling just what monitoring data is published.

MDS4 builds on query, subscription, and notification protocols and interfaces defined by the WS Resource Framework (WSRF) and WS-Notification families of specifications and implemented by the GT4 Web Services Core.

Building on this base, we have implemented a range of *information providers* used to collect information from specific sources. These components often interface to other tools and systems, such as the Ganglia cluster monitor and the PBS and Condor schedulers (see Table 1 for a current list).

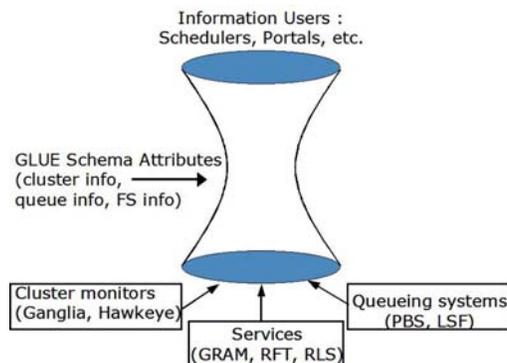
**Table 1: MDS4 information providers, existing and in progress**

| Source                                      | Information  |
|---|--|
| Ganglia cluster monitor                     | Basic cluster data   |
| Hawkeye cluster monitor                     | Basic cluster data   |
| PBS scheduler                               | Basic queuing data   |
| LSF scheduler                               | Basic queuing data   |
| GT4 WS GRAM                                 | Job status information                                     |
| GT4 Reliable File Transfer (RFT) service    | Data transfer status information                           |
| GT4 Replica Location Service (RLS)          | Replica update types, catalog data , general server status |
| GT4 Community Authorization Service (CAS)   | ServerDN, VODescription                                    |
| <i>Clumon cluster monitor (in progress)</i> | <i>Basic cluster data</i>                                  |
| <i>Torque scheduler (in progress)</i>       | <i>Basic queuing data</i>                                  |
| <i>Nagios cluster monitor (in progress)</i> | <i>Basic cluster data</i>                                  |

MDS4 also provides two higher-level services: an *Index* service, which collects and publishes aggregated information about information sources, and a *Trigger* service, which collects resource information and performs actions when certain conditions are triggered. These services are built upon a common *Aggregation Framework* infrastructure that provides common interfaces and mechanisms for working with data sources. This framework can work with arbitrarily detailed resource data and information sources of a variety of types: mechanisms are provided to (a) query arbitrary WSRF services (including, but not limited to, the Information Providers just listed, for resource property information, and (b) execute a program to acquire data.

Finally, a web-based user interface called *WebMDS* provides a simple XSLT-transform based visual interface to the data.

Additional information on MDS4 can be found at <http://www.globus.org/toolkit/mds>. See also the tech report <http://www-unix.mcs.anl.gov/~schopf/Pubs/mds4.sc.pdf>.



**Figure 1: The MDS4 hourglass provides a uniform query, subscription and notification interface to a wide variety of information sources, web services, and other monitoring tools.**

## **2 Guidelines for Information Inclusion**

The keys to developing a useful monitoring and discovery system—and to using a tool such as MDS effectively—are first to identify the information that is required and second to determine how best to make that information available.

In considering what information should be accessed using what kind of tool, we suggest the following guidelines. If the data is a well-known set of attributes and schema will be known before the query is made, then it makes sense to expose it using an MDS4 information provider. The data becomes a set of resource properties and can have a common, unified interface that provides both push and pull mode access to the data, regardless of whether its source is a service, database, file, or something else. This includes streaming data as well.

For example, if a project uses several biology databases, information to answer the following queries might be considered: what is the contact point for the database, what is the load on the server for this database, what query language does it support, etc. As a second example, some information might be kept in a database and accessed through the MDS. For example, a monitoring system might write a well known set of attributes to a database and a project might wish to access that data through MDS. An information provider could be written to access the database for that data since both the database type and the set of queries would be defined at the time of deployment.

If the need is for generic database queries, then we recommend use of OGSA-DAI, distributed as part of GT4, as a Grid-enabled interface to relational and XML databases. This tool enables easy interactions with heterogeneous databases over a Grid.

In general, we suggest groups discuss what data they need (what is the use case), and then how they want to represent it (the schema), before they decide whether the data should come from a file, database, or specialized tool.

## **3 MDS4 and TeraGrid: Monitoring Resource Availability**

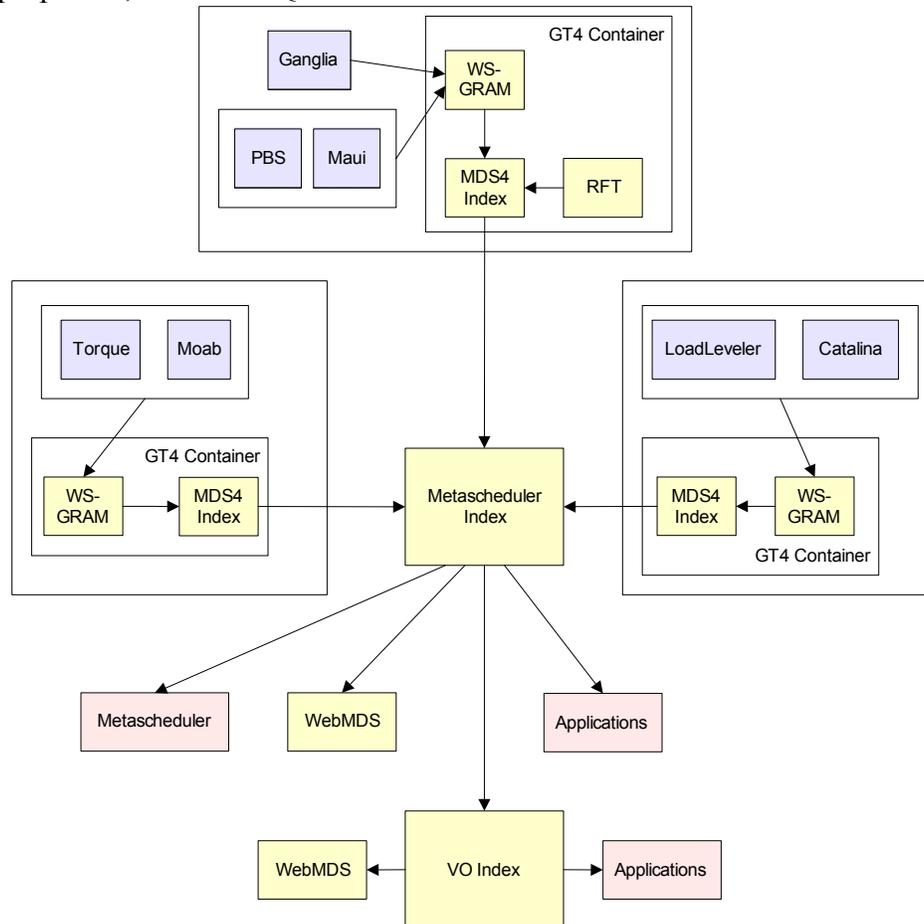
As an example of an MDS4 deployment, this section details our work with the TeraGrid project to provide both “static” and “dynamic” data (e.g., queue lengths, architecture types) relevant to selecting the “best” resource(s) to use for a particular job. End-users (via a web interface), metascheduling systems, and other applications can use this deployment to find the resource(s) that best meet their needs. A picture of the planned TeraGrid deployment is in Figure 2.

### **3.1 Information Providers**

Information relevant to resource selection is available from cluster monitoring, resource management, and scheduling systems. While there is no agreement on the data needed to make resource selection decisions, based on our previous work with scheduling systems

and analysis of several common schedulers used with queued platforms such as TeraGrid, we have decided to use:

- 14 queuing attributes: LRMSType, LRMSVersion, DefaultGRAMVersion and port and host, other GRAM versions and hosts and ports, TotalCPUs, Status (up/down), TotalJobs (in the queue), RunningJobs, WaitingJobs, FreeCPUs, MaxWallClockTime, MaxCPUTime, MaxTotalJobs, MaxRunningJobs
- 10 cluster description attributes: Unique Cluster ID, Cluster Benchmark, ProcessorType, MainMemory size, OperatingSystem, StorageDevice, Architecture, Number of nodes in a cluster/subcluster, TG-specific Node properties, and Node-Queue.



**Figure 2: Planned TeraGrid deployment of MDS4 to collect metascheduler information from various sources.**

In order to make this information available for all TeraGrid computers, we are both expanding the set of data produced by the existing OpenPBS and LSF information providers and developing new information providers for the Torque local resource management system and the Nagios and Clumon monitoring systems.

By providing an hourglass, and therefore a level of indirection, we can easily accommodate additional TeraGrid sites simply by adding information providers

(extending the interface at the bottom of the hour glass). No client-level change (at the top of the hourglass) is needed to accommodate these new sites.

### **3.2 Working with Schedulers/Users**

The end “user” for this set of data are the various “metaschedulers” that may be deployed across TeraGrid. Four candidate systems are currently being considered: GRMS (integration with MDS4 discussed in appendix); Warren Smith and TACC’s queue predictor system; SDSC’s GAR system; and the Moab peer scheduling network. The data we are collecting will be appropriate for any of these that plan to interact with a queued system such as TeraGrid, and as TeraGrid’s metascheduling plans mature we will work closely with these groups to identify any additional requirements.

We also plan to develop a user-friendly web interface to this data so users can better make their own resource selection decisions and act as their own Grid-level scheduler. We are working with both the TeraGrid portal group and to-be-named application group to ensure that we meet user needs for the interface.

## **4 Software Information**

As a second example of a future MDS4 end-to-end use case, we examine the need for basic information about software. While not yet implemented in MDS4, it is in our plan to work on as soon as we find a project collaborator that has a need for the data.

We can envision three ways in which this data could be obtained.

- 1) A simple configuration file could be set up on a well-known location on the gatekeeper node for a cluster. An information provider could be written to read that file and report the data back into MDS4. The benefit of this approach is it is simple and expedient. The drawback is that there would be no way to verify the validity of the configuration file. Manually maintained files such as this inevitably become out of date.
- 2) In MDS2, we had a software information provider that would run version commands for a pre-defined list of software. This approach allows for the verification that software is actually currently installed and running on a system. However, only software for which a user command can determine the version can be verified this way. To employ this approach, we simply need to re-implement the MDS2 provider to work with the MDS4 infrastructure. One difficulty is that if the the backend node configuration is dissimilar from that of the head node (which is often the case), we must launch a job to get valid data. This approach is more complicated than the first, but would provide more accurate data.
- 3) If the project were running a piece of software such as Inca, we can develop additional reporters for Inca to check the extended software list, and then write an MDS4 information provider that would extract and publish needed data from the Inca Archiver. This approach makes the best use of a current tool for determining dynamically what software is deployed on which system, but might involve more

development time than either of the first two approaches, due to the need to implement an interface to (for example) the Inca archiver.

Note that in each of these approaches, MDS4 acts as an hourglass, hiding the implementation details of how information is obtained from the clients that use the data. In fact, different sites could each obtain their information using a different approach. To the users, they would all appear the same.

## 5 Additional Information

The primary source of MDS information is <http://www.globus.org/toolkit/mds/>. In addition, see also:

- The tech report at <http://www-unix.mcs.anl.gov/~schopf/Pubs/mds4.sc.pdf>
- WebMDS documentation: <http://www.globus.org/toolkit/docs/4.0/info/webmds/>
- Sample WebMDS server: <http://mds.globus.org:8080/webmds>
- GLUE schema: <http://www.cnaf.infn.it/~sergio/datatag/glue>
- Ganglia: <http://ganglia.sourceforge.net/>
- Hawkeye: <http://www.cs.wisc.edu/condor/hawkeye/>
- CluMon: <http://clumon.ncsa.uiuc.edu/>
- NetLogger: <http://www-didc.lbl.gov/NetLogger/>
- Inca: <http://inca.sdsc.edu>