

White Paper: A Grid Monitoring Service Architecture (DRAFT)

Brian L. Tierney, Brian Crowley, Dan Gunter, Mason Holding, Jason Lee, Mary Thompson
Lawrence Berkeley National Laboratory

Abstract

Large distributed systems such as Computational Grids require a large amount of monitoring data for a variety of tasks such as fault detection, performance analysis, performance tuning, performance prediction, and scheduling. Ensuring that all necessary monitoring is turned on and the data is being collected can be a very tedious and error-prone task. In this paper we propose an architecture for a Grid Monitoring Service to automate the execution of monitoring sensors and the collection event data.

1.0 Introduction

The ability to monitor and manage distributed computing components is critical for enabling high-performance distributed computing. Monitoring data is needed to determine the source of performance problems and to tune the system for better performance. Fault detection and recovery mechanisms need monitoring data to determine if a server is down, and whether to restart the server or redirect service requests elsewhere. A performance prediction service might use monitoring data as inputs for its prediction model [12], which would in turn be used by a scheduler to determine which resources to use.

As distributed systems such as Computational Grids [5] become bigger, more complex, and more widely distributed, it becomes important that this monitoring and management be automated. Our approach to this problem is to use a grid monitoring service which is designed for use in a Grid environment.

This monitoring architecture is sufficiently general that it could be adapted for use in distributed environments other than the Grid. For example, it could be used in large compute farms or clusters that require a great deal of system monitoring to ensure all nodes are up and healthy.

2.0 Monitoring Service Architecture

The architecture uses a producer/consumer model, similar to several existing Event Service systems, such as the CORBA Event Service [1]. Sensors (producers) publish their existence in a directory service. Clients (consumers) look up the sensors in the directory service, and then subscribe (i.e.: connect) directly to the sensor or sensor manager. Event data is not sent anywhere unless it is requested by a consumer. Many of the current event service systems, including CORBA, send all event data to a central component, which consumers then contact. In this model, only sensor location and summary data is sent to a central directory server. Event data goes directly from producer to consumer. We believe this model will scale much better in a Grid environment.

The system consists of the following components, shown in Figure 1:

- sensors
- sensor managers
- event consumers
- directory service
- event suppliers
- event archives

We describe each of these components in detail below.

2.1 Components

sensors

The system is designed to control the execution of *sensors*. A sensor is any program which generates a time-stamped performance monitoring event. For example, we have sensors that monitor CPU usage, memory

usage, and network usage. Sensors are also used to monitor “error” conditions, such as a server process crashing, or CRC errors on a router.

We envision the following types of sensors:

- **host sensors:** These sensors perform any host monitoring task, such as monitoring CPU load, available memory, or TCP retransmissions. Host sensors may be SNMP-based, and therefore run remotely from the host being monitored.
- **network sensors:** These sensors perform SNMP queries to any network device, typically a router or switch. Information on what SNMP data is being collected is published in the directory service.
- **process sensors:** Process sensors generate events when there is a change in process status (for example, when it starts, dies normally, or dies abnormally). They might also generate an event if some dynamic threshold is reached (for example, the average number of user over a certain time period exceeds a given threshold).
- **application sensors:** Sensors can also be embedded inside of applications. These sensors might generate events if a static threshold is reached (for example, the number of locks taken exceeds threshold), user connect/disconnect, change of user password, a UNIX signal received, or any other user-defined event. Application sensors can also be used to collect detailed performance monitoring data about the application to be used for performance analysis.

sensor manager

The sensor manager is responsible for starting and stopping the sensors. The sensor manager determines which sensors to run from a configuration file or from requests from a sensor manager GUI.

event consumers

An event consumer is any program that requests data from a sensor. Consumers may run on the same host as the sensors if they are only interested in data from that host’s sensors, or they may be remote, collecting data from several sensors on several hosts. There are many possible types of consumers, including the following:

- **real-time monitor:** This consumer is used to collect monitoring data in real time for use by real-time analysis tools. It checks the directory service to see what data is available, and then “subscribes” to all the sensors it is interested in. The sensors then send the event data to the consumer as it is generated. Data from many sensors is then merged into a file for use by programs such as *nlv*, the NetLogger real-time event visualization tool [10].
- **archiver:** This consumer is used as to collect data for the archive service. It subscribes to the event suppliers, collects the event data, and places it in the archive. It also creates an archive directory service entry indicating the contents of the archive.
- **process monitor:** This consumer can be used to trigger an action based on an event from a server process. For example, it might run a script to restart the processes, send email to a system administrator, call a pager, etc.

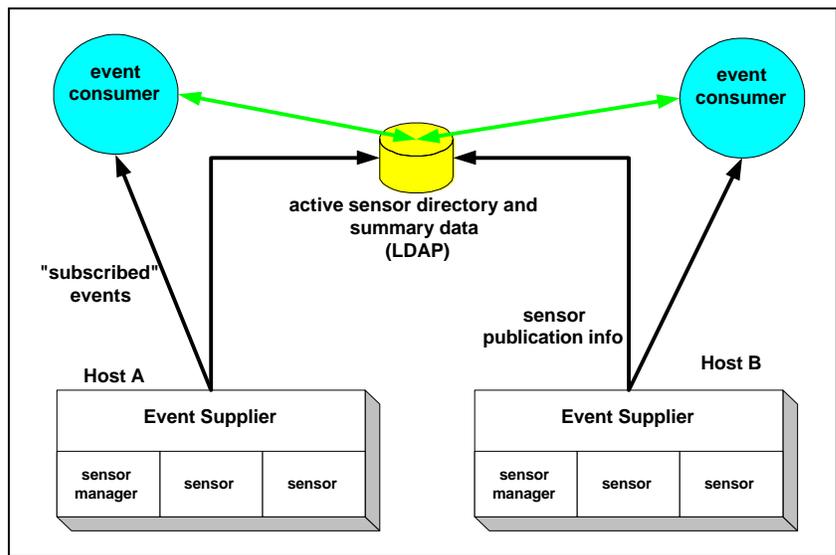


Figure 1: Architecture Components

- **overview monitor:** This consumer collects information from sensors on several hosts, and uses the combined information to make some decision that could not be made on the basis of data from only one host. For example, one may want to trigger a page to a system administrator at 2 A.M. only if both the primary and backup servers are down.

directory service

The directory service is used to publish the location of all event suppliers and sensors. This allows consumers to discover which monitoring data is currently available for subscription. Query-optimized directory services such as LDAP, Globus MDS [3], the Legion Information Base, and the Novell NDS, all provide the necessary base functionality for this. We suggest using LDAP, because it is a simple, standard solution. LDAP servers can be hierarchical, with referrals to other LDAP servers which contain the directory service information for each site. We are also interested in exploring the “event notification” service of LDAPv3 [11] as soon as it is available. This service lets a client register interest in an entry (i.e.: sensor running) with the LDAP server, and LDAP will notify the client when that entry becomes available or is updated.

Current implementations of LDAP servers are optimized for read access, and do not work well in an environment with many updates. However, it is possible to use the LDAP protocol only for naming, and not use the LDAP database. Thus we can explore the use of other storage mechanisms that might handle data updates better if this update rate becomes a problem.

event suppliers

Event Suppliers are responsible for keeping the sensor directory up to date, and listening for data requests from event consumers. The event supplier also can be configured to compute summary data. For example, the event supplier can compute 1, 10, and 60 minute averages of CPU usage, and publish this data into the directory service along with the entry for the CPU sensor.

Event suppliers can accept several types of requests from consumers. The consumer may request all event data, or only to be notified of certain types of events. For example the *netstat* sensor checks for TCP retransmissions every second, but most consumers only want to be notified when a retransmit occurs, and not get an event every second. A consumer can also request that an event be sent only if its value crosses a certain threshold. Examples of such a threshold would be if CPU load becomes greater than 50%, or if load changes by more than 20%.

Event suppliers can also service “query” requests from consumers. In query mode the consumer does not open an event channel, but only requests the most recent event.

event archives

It is important to archive event data in order to provide the ability to do historical analysis of system performance, and determine when/where changes occurred. While it may not be desirable to archive all monitoring data, it is necessary to archive a good sampling of both “normal” and “abnormal” system operation, so when problems arise it is possible to compare the current system to a previously working system. Archives might also be used by performance prediction systems, such as the Network Weather Service (NWS) [12].

This architecture provides a flexible method for selecting what gets archived, because the archive is just another consumer. In some environments very little will be monitored, and in others, it maybe desirable to archive everything. With this approach the archive need not get in the way of any real-time monitoring.

2.2 Use of Multicast

The publish/subscribe model used in the architecture maps well to an IP multicast model [2], where the sensors could multicast event data to all subscribers. In the case where there are many consumers that wish to subscribe to events from the same sensor, multicast is essential to obtain scalability. Multicast is also useful in a cluster environment where a large number of hosts on the same subnet are being monitored.

3.0 Security Issues

A distributed monitoring system such as this creates a number of security vulnerabilities which must be analyzed and addressed before such a system can be safely deployed on a production Grid. Remote users may need to start sensor programs, connect to event suppliers to receive detailed information about the Grid system, and publish event information. Finding event suppliers and publishing event data is done through an LDAP server. Starting sensors is done through the Sensor Manager.

Current LDAP servers provide user/password style protection to regions of the LDAP tree which can be used to protect the lookup and publishing functions. Normally these passwords are sent in clear text, but there are SSL-enabled versions of LDAP, e.g. Netscape™, where the LDAP server has a key which can be used to encrypt the connection to the client. The client application needs to provide a user name and password. Since the event publishing is performed by the event suppliers, these components also need to have credentials representing an entity that is allowed write permission by the LDAP server.

There are several possible approaches to an integrated security solution. To the extent that we expect monitoring services to be deployed within a Grid environment, it seems desirable to use the Globus Security Interface API (GSI [5]). However, this interface relies on X.509 certificates for user identification and would require that consumers coordinate with the Globus client interface to send the Grid user credential. Since the current LDAP security is based on a user/password mechanism, a server-side LDAP wrapper that maps credentials to user names and passwords similar to the GSI map file would be required. However, closely integrating the monitoring service with GSI may limit their usefulness in non-Globus environments.

4.0 Sample Use

An example use of this service is shown in Figure 2. Monitoring data is collected at both the client and server host, and at all network routers between them. All event data is sent to a real-time monitor consumer for real-time visualization and analysis. Network sensors publish summary throughput and latency data in the directory service, which is used by a “network-aware” client [9] to optimally set its TCP buffer size. Server and router data is also sent to the archive.

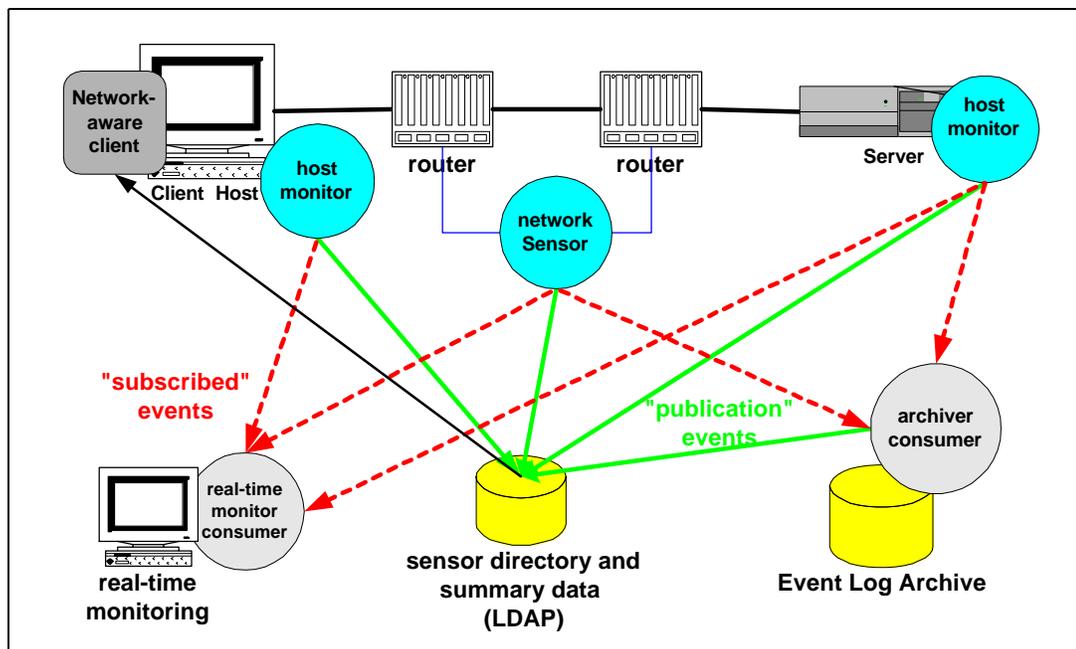


Figure 2: Sample Use

5.0 Related Work

There are many existing systems with an event model similar to the one described here. CORBA includes an “event service” [1] which has a rich set of features, including the ability to push or pull events, and the ability

for the consumer to pass a filter to the event supplier. JINI also has a “Distributed Event Specification” [6], which is a simple specification for how an object in one Java™ virtual machine (JVM) registers interest in the occurrence an event occurring in an object in some other JVM, and then receives a notification when that event occurs. There are also several other systems with alternative event models, such as the Common Component Architecture; many of which are summarized in [8]. However, we believe that none of the existing systems is a perfect match for a Grid monitoring system, therefore we have tried to combine the relevant strengths of each. Other related systems include the Pablo system [7], which has had the notion of application *sensors* for several years. Wolski et al. [13] also describe an architecture with similar characteristics as the service described here.

6.0 Acknowledgments

We are greatly indebted to many members of the Grid Forum (<http://www.gridforum.org>), from whom many of the ideas in this paper came. In particular, discussions with Rich Wolski, University of Tennessee, Ruth Ayt, University of Illinois, Ian Foster, Steve Tuecke, et. al. at Argonne National Lab, and Dennis Gannon, University of Indiana, all contributed to the architecture described here.

The work described in this paper is supported by the U. S. Dept. of Energy, Office of Science, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division (<http://www.er.doe.gov/production/octr/mics/index.html>), under contract DE-AC03-76SF00098 with the University of California. This is report no. LBNL-45260.

7.0 References

- [1] CORBA, “Systems Management: Event Management Service”, X/Open Document Number: P437, <http://www.opengroup.org/onlinepubs/008356299/>
- [2] Deering, S., “Host Extensions for IP Multicasting”, IETF RFC 1054.
- [3] Fitzgerald, S., I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, “A Directory Service for Configuring High-Performance Distributed Computations”. In Proc. 6th IEEE Symp. on High Performance Distributed Computing, August 1997.
- [4] Globus: See <http://www.globus.org>
- [5] Grid: The Grid: Blueprint for a New Computing Infrastructure, edited by Ian Foster and Carl Kesselman. Morgan Kaufmann, Pub. August 1998. ISBN 1-55860-475-8.
- [6] Jini Distributed Event Specification”, <http://www.sun.com/jini/specs/>
- [7] Pablo Scalable Performance Tools, <http://vibes.cs.uiuc.edu/>.
- [8] Peng, X, “Survey on Event Service”, <http://www-unix.mcs.anl.gov/~peng/survey.html>
- [9] Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., “A Network-Aware Distributed Storage Cache for Data Intensive Environments”, Proceeding of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896. <http://www-didc.lbl.gov/DPSS/>
- [10] Tierney, B., W. Johnston, B. Crowley, G. Hoo, C. Brooks, D. Gunter, “The NetLogger Methodology for High Performance Distributed Systems Performance Analysis”, Proceeding of IEEE High Performance Distributed Computing conference, July 1998, LBNL-42611. <http://www-didc.lbl.gov/NetLogger/>
- [11] Wahl M., Howes, T., Kille S., “Lightweight Directory Access Protocol (v3)”, Available from <ftp://ftp.isi.edu/in-notes/rfc2251.txt>
- [12] Wolski, R., Spring, N., Hayes, J., “The Network Weather Services: A Distributed Resource Performance Forecasting Service for Metacomputing,” Future Generation Computing Systems, 1999. <http://nsw.npaci.edu/>
- [13] Wolski, R and M. Swany, S. Fitzgerald, “White Paper: Developing a Dynamic Performance Information Infrastructure for Grid Systems”, available at: <http://dast.nlanr.net/GridForum/Perf-WG/white.PDF>