Document: GWD-Perf-16-1
Category: Recommendation
Obsoletes: N/A
Expires: January, 2002

Ruth Aydt
University of Illinois, Urbana-Champaign
Dan Gunter
Lawrence Berkeley National Laboratory
Warren Smith
NASA
Martin Swany
University of Tennessee, Knoxville
Valerie Taylor
North Western University
Brian Tierney
Lawrence Berkeley National Laboratory
Rich Wolski
University of Tennessee, Knoxville

July, 2001

A Grid Monitoring Architecture

Abstract

Large distributed systems such as Computational and Data Grids require a substantial amount of monitoring data be collected for a variety of tasks such as fault detection, performance analysis, performance tuning, performance prediction, and scheduling. Some tools are currently available and others are being developed for collecting and forwarding this data. The goal of this paper is to describe a common grid monitoring architecture with all the major components and their essential interactions. To aid implementation, we also discuss the performance characteristics of a Grid Monitoring system and identify areas that are critical to proper functioning of the system.

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 and application 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 [13][9]. A performance prediction service might use monitoring data as inputs for a prediction model [15], which would in turn be used by a scheduler to determine which resources to use.

Several groups are developing Grid monitoring systems to address this problem [10] [15][8][13] and these groups have seen a need to interoperate. To facilitate inter operability, we have developed an architecture to address the specific requirements of a Grid monitoring system. These requirements are discussed in more detail in [GMA-REQ], but briefly a Grid monitoring system is differentiated from a general monitoring system in that it must be scalable across wide-area networks, include a large number of heterogeneous resources, and be integrated with other Grid middleware in terms of naming and security issues. We believe the Grid Monitoring Architecture (GMA) described here addresses these concerns and is sufficiently general that it could be adapted for use in distributed environments other than the Grid. For example, it could be used with large compute farms or clusters that require constant monitoring to ensure that all nodes are running correctly.

2.0 Architecture and Terminology

The Grid Monitoring Architecture consists of three types of components, shown in Figure 1:

- Directory Service: supports information publication and discovery
- Producer: makes performance data available (performance event source)
- Consumer: receives performance data (performance event sink)

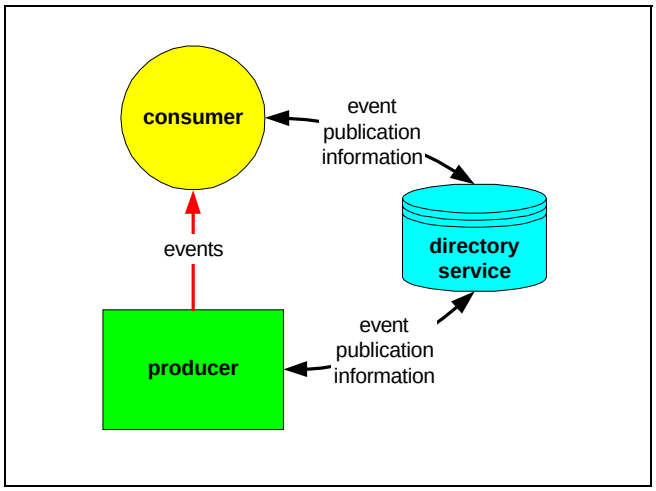

Figure 1: Grid Monitoring Architecture Components

The GMA is designed to handle performance data transmitted as timestamped (*performance) events*. An event is a typed collection of data with a specific structure that is defined by an *event schema*. Performance event data is always sent directly from a producer to a consumer.

The GMA supports both a streaming publish/subscribe model, similar to several existing Event Service systems such as the CORBA Event Service [1], and a query/response model. For both models, producers or consumers that accept connections publish their existence in the directory service. Consumers can use the directory service to discover producers of interest, and producers can use the directory service to discover consumers of interest. Either a producer or a consumer may initiate the connection to a discovered component. Communication of control messages and transfer of performance data occurs directly between each consumer/producer pair without further involvement of the directory service.

3.0 Components and Interfaces

In this section we further define the functionality and interfaces of the consumer, producer, and directory service components.

3.1 Directory Service

To describe and discover performance data on the Grid, a distributed directory service for publishing and searching must be available. The GMA directory service contains information about producers and consumers that accept requests. When producers and consumers publish their existence in the directory service they typically specify the event types they produce or consume. In addition, they may publish static values for some event data elements, further restricting the range of event data they will produce or consume. This publication information allows other producers and consumers to discover the types of event data that are currently available, the characteristics of that data, and the sources or sinks that will produce or accept each type of data. The directory service is not responsible for the storage of event data -- only per-publication information about which event instances can be provided. The event schema may also, optionally, be stored in the directory service.

The functions supported by a directory service are:

1. Authorize modify – Establish the identity of a client for adding, updating, or removing entries.
2. Authorize search – Establish identity of a client for searching.
3. Add – Add an entry to the directory.
   - Preconditions: The client is authorized to add the entry. The entry conforms to the directory's schema. The entry is not a duplicate.
   - Postconditions: The entry is in the directory.

4. Update – Change the state of an entry in the directory.
    - Preconditions: The client is authorized to update the entry.
    - Postconditions: The entry now has the new values.
5. Remove – Remove an entry from the directory.
    - Preconditions: The client is authorized to remove the entry.
    - Postconditions: The entry is not in the directory.
6. Search – Perform a search for a producer or consumer of a particular type and possibly with fixed values for some of the event elements. The client should indicate whether only one result, or more than one result, if available, should be returned. An optional extension would allow the client to get multiple results one element at time using a "get next" query in subsequent searches.
    - Preconditions: The client is authorized to perform the search.
    - Postconditions: The result(s) of the search are returned, which includes a well-defined null value for searches which did not match any entries in the directory.
7. Version request – A client may request the current version of the interface. The version numbering system is TBD.

3.2 Producer

A producer is any component which sends events to a consumer. The functions supported by a producer are listed below. The first function should be supported by producers that allow wish to handle new event types dynamically. Functions 2-6 should be supported by producers which initiate the flow of events; functions 7-9 should be supported by producers which allow consumers to initiate the flow of events.

1. Locate Event: Search the schema repository for the event schema. The schema repository may be the directory service.
    - Preconditions: The producer is authorized to search the schema repository.
    - Postconditions: The repository is unchanged (read-only operation).
2. Locate Consumer: Search the directory service for a consumer.Corresponds to Directory Service *Search.*
    - Preconditions: The producer is authorized to search the directory service.
    - Postconditions: The directory service is unchanged (read-only operation).
3. Register: Add/remove/update entry(ies) in the directory service describing events that the producer will make available to consumers. Corresponds to Directory Service *Add*, *Remove* and *Update*.
    - Preconditions: The producer is authorized for add/remove/update on the directory service.
    - Postconditions: The directory service entries are added or removed, or updated with the new data.
4. Accept Query: Accept a query request from a consumer.
    - Preconditions: The consumer is authorized to query this producer.
    - Postconditions: One or more event(s) are returned in the reply.
5. Accept Subscribe: Accept a subscribe request from a consumer.
    - Preconditions: The consumer is authorized to subscribe to this producer.
    - Postconditions: Further details about the event stream are returned in the reply.
6. Accept Unsubscribe: Accept an unsubscribe request from the consumer.
    - Preconditions: The subscription exists. The consumer is authorized to terminate it.
    - Postconditions: The subscription is terminated. No more events will be sent for this subscription.
7. Initiate Query: Send a single set of event(s) to a consumer as part of a query "request".
    - Preconditions: The producer is authorized to send these event(s) to this consumer.
    - Postconditions: One or more event(s) are delivered.
8. Initiate Subscribe: Request to send event(s) to a consumer, which are delivered in a stream.
    - Preconditions: The producer is authorized to connect to the consumer and send these event(s).
    - Postconditions: Further details about the event stream are returned in the reply.
9. Initiate Unsubscribe: Terminate a subscription to a consumer.
    - Preconditions: The subscription exists and the producer is authorized to terminate it.
    - Postconditions: The subscription is terminated. No more data will be accepted for this subscription.

Producers can deliver events in a stream or as a single response per request. In streaming mode a virtual connection is established between the producer and consumer and events can be delivered along this connection until an explicit action is taken to terminate it. In query mode the event is delivered as part of the reply to a consumer-initiated query, or as part of the request in a producer-initiated query.

Producers are also used to provide access control to the event data, allowing different access to different classes of users. Since Grids typically have multiple organizations controlling the resources being monitored, there may be different access policies (firewalls possibly), different frequencies of measurement, and different performance details for consumers "inside" or "outside" of the organization running the resource. Some sites may allow internal access to real-time event streams, while providing only summary data off-site. The producers would enforce these types of policy decisions. This mechanism is especially important for monitoring clusters or computer farms, where there may be a large amount of internal monitoring, but only a limited amount of monitoring data accessible to the Grid

3.3 Consumer

A consumer is any component that receives event data from a producer. The functions supported by consumers are listed below. All consumers that handle new event types dynamically should support the first function. Functions 2-5 should be supported by consumers that initiate the flow of events; functions 6-8 should be supported by consumers that allow a producer to initiate the flow of events.

1. Locate Event – Search the schema repository for the event schema. The schema repository may be the GMA directory service.
    - Preconditions: The consumer is authorized to search the schema repository.
    - Postconditions: The schema repository is unchanged (read-only operation).
2. Locate Producer – Search the directory service for a producer. Corresponds to the Directory Service *Search*.
    - Preconditions: The consumer is authorized to search the directory service.
    - Postconditions: The directory service is unchanged (read-only operation).
3. Initiate Query – Request event(s) from a producer, which are delivered as part of the reply.
    - Preconditions: The consumer is authorized to receive these event(s).
    - Postconditions: Zero or more event(s) are returned in the reply.
4. Initiate Subscribe – Requests event(s) from a producer, which are delivered in a stream.
    - Preconditions: The consumer is authorized to connect to the producer and receive these event(s).
    - Postconditions: Further details about the event stream are returned in the reply.
5. Initiate Unsubscribe – Terminate a subscription.
    - Preconditions: The subscription exists and the consumer is authorized to terminate it.
    - Postconditions: The subscription is terminated. No more data should be sent, or will be accepted, for this subscription.
6. Register – Add/remove/update entry(ies) in the directory service describing events that the consumer accepts from a producer. Corresponds to Directory Service *Add*, *Remove*, and *Update.*
    - Preconditions: The consumer is authorized for add/remove/update on the directory service.
    - Postconditions: The directory service entries are added or removed, or updated with the new data.
7. Accept Query – Accept a query request from a producer. The "query" will also contain the events.
    - Preconditions: The producer is authorized to query this consumer.
    - Postconditions: One or more event(s) are received as part of the query request.
8. Accept Subscribe – Accept a subscribe request from a producer.
    - Preconditions: The producer is authorized to subscribe to this consumer.
    - Postconditions: Further details about the event stream are sent to the producer in the reply to the request.

9. Accept Unsubscribe – Accept an unsubscribe request from the producer.
    - Preconditions: The subscription exists. The producer is authorized to terminate it.
    - Postconditions: The subscription is terminated. No more events will be accepted for this subscription.

3.4 Compound Producer/Consumer

There may also be components that are both consumers and producers. For example, a consumer might collect event data from several producers, and then use that data to generate derived performance events made available in a different event type, as shown in Figure2.

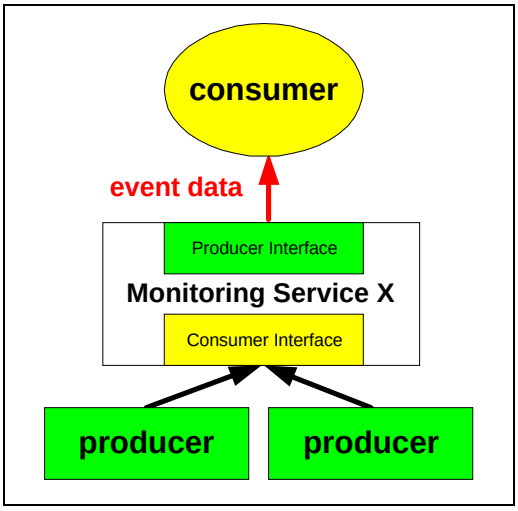

Figure 2: Joint Consumer/Producer

4.0 Security Issues

A distributed 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. The users of such a system are likely to be remote from the machines being monitored and to belong to different organizations.

Typical user actions will include queries to the directory service concerning event data availability, subscriptions to producers to receive event data, and requests to instantiate new event monitors or to adjust collection parameters on existing monitors. In each case, the domain that is being monitored is likely to want to control which users may perform which actions.

Public key based X.509 identity certificates [5] are a recognized solution for cross-realm identification of users. When the certificate is presented through a secure protocol such as SSL (Secure Socket Layer), the server side can be assured that the connection is indeed to the legitimate user named in the certificate.

User (consumer) access at each of the points mentioned above (directory lookup and requests to a producer), would require an identity certificate passed though a secure protocol, e.g. SSL. A wrapper to the directory server and the producer could both call the same authorization interface with the user's identity and the name of the resource the user wants to access. This authorization interface could return a list of allowed actions, or simply deny access if the user is unauthorized. Communication between the producer and the sensors may also need to be controlled, so that a malicious user can not communicate directly with the monitoring process.

5.0 Sample Use

An example use of the GMA is shown in Figure 3. Event data is collected on each host and at all network routers between the hosts, and aggregated at a producer. The producer registers the availability of the data in the directory service. A real-time monitoring consumer subscribes to all available event data for real-time visualization and performance analysis. A "network-aware" client [10] uses throughput and latency data to optimally set its TCP buffer size. The producer also sends the data relating to the "server" and "router" nodes to an archive.

6.0 Related Work

There are numerous existing systems with an event model similar to the one described here. CORBA includes an "event service" [1] that 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

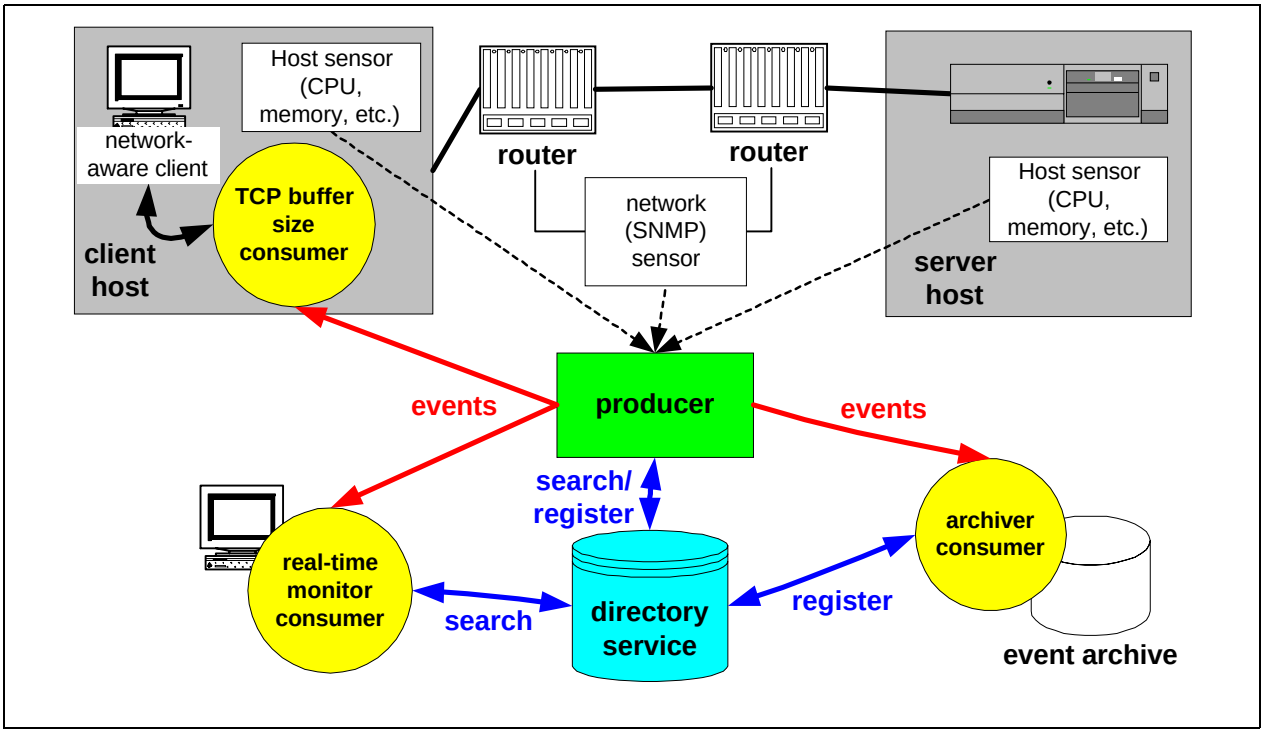

Figure 3: Sample Use of Monitoring System

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 [7]. 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. Another related system is Autopilot [8], which implements a similar publish/lookup/subscribe architecture. This list of systems is not intended to be exhaustive, but only illustrative of the usefulness of the proposed architecture.

7.0 Acknowledgements

Input from many people went into this document, including almost all attendees of the various Grid Forum meetings.

The LBNL portion of this paper was supported by the U. S. Dept. of Energy, Office of Science, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division, under contract DE-AC03-76SF00098 with the University of California.

The UIUC portion of this paper is based upon work supported by the National Science Foundation under Grant No. 9975020. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8.0 References

[1] CORBA, “Systems Management: Event Management Service”, X/Open Document Number: P437, http://www.opengroup.org/onlinepubs/008356299/

[2] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, “A Directory Service for Configuring High-Performance Distributed Computations”. In *Proceedings 6th IEEE Symposium on High Performance Distributed Computing, August 1997.*

[3] The Globus project: See http://www.globus.org

[4] 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.

[5] Housely, R., W. Ford, W. Polk, D. Solo, “Internet X.509 Public Key Infrastructure”, IETF RFC 2459. Jan. 1999

[6] Jini Distributed Event Specification”, http://www.sun.com/jini/specs/

[7] Peng, X, “Survey on Event Service”, http://www-unix.mcs.anl.gov/~peng/survey.html

[8] Randy L. Ribler, Jeffrey S. Vetter, Huseyin Simitci, and Daniel A. Reed, “Autopilot: Adaptive Control of Distributed Applications,” Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.

[9] W. Smith, “Monitoring and Fault Management,” http://www.nas.nasa.gov/~wwsmith/mon_fm

[10] Tierney, B., B. Crowley, D. Gunter, M. Holding, J. Lee, M. Thompson A Monitoring Sensor Management System for Grid Environments Proceedings of the IEEE High Performance Distributed Computing conference (HPDC-9), August 2000, LBNL-45260.

[11] 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/

[12] 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/

[13] A. Waheed, W. Smith, J. George, J. Yan. “An Infrastructure for Monitoring and Management in Computational Grids.” In *Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems*.

[14] Wahl M., Howes, T., Kille S., “Lightweight Directory Access Protocol (v3)”, Available from ftp://ftp.isi.edu/in-notes/rfc2251.txt

[15] 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/