GWD-I (Informational)

B. Tierney, Lawrence Berkeley National Laboratory
R. Aydt, University of Illinois at Urbana-Champaign
D. Gunter, Lawrence Berkeley National Laboratory
W. Smith, NASA Ames Research Center
M. Swany, University of California, Santa Barbara
V. Taylor, Northwestern University
R. Wolski, University of California, Santa Barbara

GGF Performance Working Group

March 2000
Revised 27-August-2002

A Grid Monitoring Architecture

Status of this Memo

This memo provides information to the Grid community regarding the Grid Monitoring Architecture (GMA) being developed by the Global Grid Forum Performance Working Group. The goal of the architecture is to provide a minimal specification that will support required functionality and allow interoperability. Distribution is unlimited.

1 Abstract

Large distributed systems such as Computational and Data Grids require that a substantial amount of monitoring data be collected for various 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 the major components of a Grid monitoring architecture and their essential interactions. By adopting standard terminology and describing the minimal specification to support required functionality, we hope to encourage the development of interoperable high-quality performance tools for the Grid. To motivate the Grid Monitoring Architecture (GMA) design and to guide implementation, we also present the characteristics that are critical to proper functioning of a performance monitoring system for the Grid.

GWD-I (Informational)

B. Tierney, Lawrence Berkeley National Laboratory
R. Aydt, University of Illinois at Urbana-Champaign
D. Gunter, Lawrence Berkeley National Laboratory
W. Smith, NASA Ames Research Center
M. Swany, University of California, Santa Barbara
V. Taylor, Northwestern University
R. Wolski, University of California, Santa Barbara

GGF Performance Working Group

March 2000
Revised 27-August-2002

Table of Contents

2 Introduction

Performance monitoring of distributed 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. Fault detection and recovery mechanisms need monitoring data to determine whether a server is down and to decide whether to restart the server or to redirect service requests elsewhere [5][8]. A performance prediction service takes monitoring data as input to a prediction model [10], which is in turn used by a scheduler to determine which resources to assign to a job.

Several groups are developing Grid monitoring systems [4][7][8][10]. These groups, along with others in the Global Grid Forum community, recognize the need to interoperate. To facilitate this interoperation, we have developed an architecture of monitoring components that specifically addresses the characteristics of Grid platforms. A Grid monitoring system is differentiated from a general monitoring system in that it must be scalable across wide-area networks and encompass a large number of heterogeneous resources. The monitoring system's naming and security mechanisms must also be integrated with other Grid middleware. Another essential aspect of a Grid monitoring system is a set of common protocols for messaging, data exchange, and management. Other GGF working groups, such as the Open Grid Service Infrastructure Working Group [2], are addressing these issues.

In this document we describe the core Grid Monitoring Architecture (GMA) components and models for high-level communication between components of different types. This document does not address component creation or management (coordination and control), which are crucial in a production-level Grid monitoring system. We hope that these issues will be covered in future documents.

3 Design Considerations

With the potential for thousands of resources at geographically distant sites and tens-of-thousands of simultaneous Grid users, it is critical that data collection and distribution mechanisms scale. A general-purpose information management system such as an off-the-shelf database or directory service cannot efficiently meet this requirement because the characteristics of performance information are fundamentally different from the characteristics of the data these types of systems were designed to handle. In general, the following characteristics distinguish performance-monitoring information from other forms of system or program-produced data.

- **Performance information has a fixed, often short, lifetime of utility**. Most monitoring data goes stale quickly, making rapid read access important but obviating the need for long-term storage. The notable exception to this is data that gets archived for accounting or postmortem analysis.
- **Updates are frequent**. Unlike the more static forms of "metadata," dynamic performance information is typically updated more frequently than it is read. Since most extant information-base technologies are optimized for query and not for update, they are potentially unsuitable for dynamic information storage.
- **Performance information is often stochastic**. It is frequently impossible to characterize the performance of a resource or an application component by using a single value. Therefore, dynamic performance information may carry quality-of-information metrics quantifying its accuracy, distribution, lifetime, and so forth, which may need to be calculated from the raw data.

Systems that collect and distribute performance information should satisfy certain requirements:

- **Low latency**. As previously stated, performance data is typically relevant for only a short period of time. Therefore, it must be transmitted from where it is measured to where it is needed with low latency.
- **High data rate**. Performance data can be generated at high data rates. The performance monitoring system should be able to handle such operating conditions. In addition, performance data may be very bursty, so average and burst data rates should be specified in advance so as to not overwhelm the consumer.
- **Minimal measurement overhead**. If measurements are taken often, the measurement itself must not be intrusive. Further, there must be a way for monitoring facilities to limit their intrusiveness to an acceptable fraction of the available resources. If no mechanism for managing performance monitors is provided, performance measurements may simply measure the load introduced by other performance monitors.
- **Secure.** Typical user actions will include queries to the directory service concerning event data availability, subscriptions for event data, and requests to instantiate new event monitors or to adjust collection parameters on existing monitors. The data gathered by the system may itself have access restrictions placed upon it by the owners of the monitors. The monitoring system must be able to ensure its own integrity and to preserve the access control policies imposed by the ultimate owners of the data.
- **Scalable**. Because there are potentially thousands of resources, services, and applications to monitor and thousands of potential entities that would like to receive this information, it is important that a performance monitoring system provide scalable measurement, transmission of information, and security.

In order to meet these requirements, a monitoring system must have precise local control of the overhead and latency associated with gathering and delivering the data. We believe that data discovery needs to be separated from data transfer if this level of control is to be achieved.

In the Grid, the amount of available performance information will be very large, and searches of this space will have unpredictable latencies. These potentially large latencies must not impact every request for performance information. Instead, searches should be used only to locate an appropriate information source or sink, whereas operations with a more predictable latency should be used to transfer the actual performance information. In this way, individual producer/consumer pairs can do “impedance matching” based on negotiated requirements, and the amount of data flowing through the system can be controlled in a precise and distributed fashion based on current local load considerations.

In order to separate data discovery from data transfer, metadata must be abstracted and placed in a universally accessible location, called here a “directory service,” along with enough information to bootstrap the communication between the data source and sink. Scalability results from restricting and organizing the metadata so that the directory service itself may be distributed and so that the rate of communication between distributed nodes increases slowly relative to the total amount of data transferred. This model differs from the “event channel” model of the CORBA Event Service [1], which combines the mechanism for finding the data that should be transferred with the actual transfer into a single “searchable” channel of information. In contrast, in our design performance event data, which makes up the majority of the communication traffic, travels directly from the producers of the data to the consumers of the data.

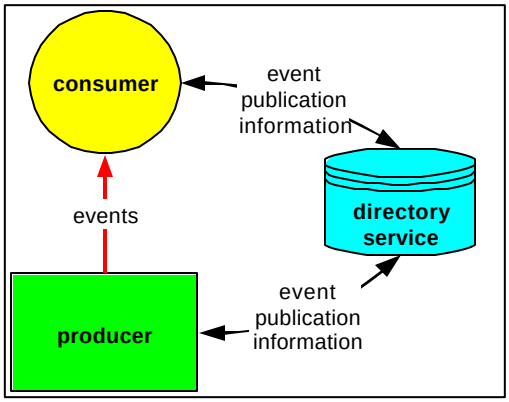

Figure 1: Grid Monitoring Architecture Components

4 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)

The GMA is designed to handle performance data transmitted as time-stamped (*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.

4.1 Directory Service Interactions

Producers (and consumers that accept control messages) publish their existence in directory service *entries*. The directory service, or registry, is used to locate producers and consumers. Note that the term "directory service" is not meant to imply a hierarchical service such as LDAP [9]; any lookup service could be used. The directory service serves to bootstrap communication between consumers and producers, as entries are populated with information about understood wire protocols and security mechanisms.

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 interaction with a discovered peer. In either case, communication of control messages and transfer of performance data occur directly between each consumer/producer pair without further involvement of the directory service.

4.2 Producer/Consumer Interactions

The GMA architecture supports three interactions for transferring data between producers and consumers: publish/subscribe, query/response, and notification.

The GMA publish/subscribe interaction has three stages. In the first stage, the initiator of the interaction (this may be either a producer or consumer) contacts the "server" (if the initiator is a consumer, the server is a producer, and vice versa) indicating interest in some set of events. The mechanism for specifying events of interest is not addressed in this document. Additional parameters needed to control the data transfer are also negotiated in this stage. These may include where to send the performance events, how to encode or encrypt the performance events, and how often to send the performance events, buffer sizes, and timeouts. The initial contact and other communication during this stage are done via an exchange of control messages between the initiator and the server. At this point, there is state in both the producer and consumer, called a *subscription*. In the next stage of the interaction, the producer (which may have been the initiator or the server for this interaction) sends one or more performance events to the consumer. In the final stage, either the producer or consumer terminates the subscription, possibly with additional control messages.

For the GMA query/response interaction, the initiator must be a consumer. The interaction consists of two stages. The first stage sets up the transfer, similar to the first stage of publish/subscribe. Then, instead of performance event transfer followed at some later time by a terminating unsubscribe, the producer transfers all the performance events to the consumer in a single *response*. This interaction maps particularly well to request/reply protocols such as HTTP.

The GMA notification interaction is a one-stage interaction, and the initiator must be a producer. In this type of producer/consumer interaction, the producer transfers all the performance events to a consumer in a single *notification*.

Protocols for control and event data channels are not specified by the GMA. Moreover, the wire protocol used to communicate control information between the producer and consumer and the wire protocol used to transfer performance events (data) may be completely different. System implementers may support one or more wire protocols, for example, SOAP/HTTP, LDAP, or XML/BXXP, choosing those best suited to their own requirements. Common wire protocols are clearly essential, and will be defined in future GGF documents.

Another important issue not addressed in this document is an event delivery model. Higher-level "service protocols" which use wire protocols such as SOAP, UDP, or BXXP must be defined to control event delivery semantics. Possible event delivery modes include at-most-once, at-least-once, or exactly-once delivery. Transactional protocols might be used to implement these delivery semantics. Future work on the GMA must define a framework for defining and negotiating the event delivery model. This is particularly important if one wishes to use GMA as a more general event handling system.

5 Components and Interfaces

In this section we further define the functionality and interfaces of the directory service, producer, and consumer components. We also introduce the notion of "compound components" and discuss potential sources of measurement data.

5.1 Directory Service

In order to describe and discover performance data on the Grid, a distributed directory service for publishing and searching must be available. The GMA directory service stores information about producers and consumers that accept requests. When producers and consumers publish their existence in the directory service, they typically also publish information regarding the types of events they produce or consume, along with the meta-information about accepted protocols, security mechanisms, and so forth, as described in Section 4. This publication information, or *registration*, allows other producers and consumers to discover the types of event data that are currently available or accepted, the characteristics of that data, and ways to gain access to that data. The directory service is not responsible for the storage of event data itself – it contains only per-publication information about which event instances can be provided or accepted. The event schema may, optionally, be available through the directory service.

Four functions are supported by the directory service.

1. Add: add an entry to the directory.
2. Update: change an entry in the directory.
3. Remove: remove an entry from the directory.
4. Search: perform a search for a producer or consumer based on some selection criteria. The client should indicate whether a single match, or multiple matches, if available, should be returned. An optional extension would allow the client to get multiple matches one at time using a "get next" query in subsequent searches.

5.2 Producer

A producer is any component that uses the producer interface to send events to a consumer. A given component may have multiple producer interfaces, each acting independently and sending events. The term *producer* is used interchangeably, and inexactly, to refer both to a single producer interface and to a component that contains at least one producer interface.

The core interaction functions that may be supported by a producer are listed below.

1. Maintain Registration: add/update/remove directory service entry or entries describing events that the producer will send to a consumer. Corresponds to Directory Service *Add, Update,* and *Remove*.
2. Accept Query: accept a query request from a consumer. One or more events are sent to the consumer in response to the query. Corresponds to Consumer *Initiate Query*.
3. Accept Subscribe: accept a subscribe request from a consumer. Further details about the subscription are returned in the reply. If the subscription is successfully established, the producer sends events to the consumer until the subscription is terminated. Corresponds to Consumer *Initiate Subscribe*.
4. Accept Unsubscribe: accept an unsubscribe request from the consumer. If this succeeds, the corresponding subscription will be closed and no more events will be sent for this subscription. Corresponds to Consumer *Initiate Unsubscribe*.
5. Locate Consumer: search the directory service for a consumer. Corresponds to Directory Service *Search*.
6. Notify: send a single set of event(s) to a consumer. Corresponds to Consumer *Accept Notification*.
7. Initiate Subscribe: request to consumer to send it events. Further details about the subscription are returned in the reply. If the subscription is successfully established, the producer sends events to the consumer until the subscription is terminated. Corresponds to Consumer *Accept Subscribe*.
8. Initiate Unsubscribe: terminate a subscription with a consumer. If this succeeds, the subscription will be closed, and no more events will be sent for this subscription. Corresponds to Consumer *Accept Unsubscribe*.

Producers can 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" the organization running the resource. For example, some sites may allow internal access to real-time event streams, while providing only summary data off-site. The producers can enforce these policy decisions.

In addition to the core GMA producer functionality described above, producers could provide many other services. Examples of these include event filtering, caching, and intermediate processing of the raw data as requested by a consumer. For example, a scheduling consumer might request that a prediction model be applied to the CPU load measurement history from a particular compute resource, and be notified only if the predicted load falls below a specified threshold, indicating that the resource is ready to accept new tasks. A "smart" producer could apply the model supplied by the consumer with the subscription request, and send events only when the resulting load predictions are below the threshold value.

Information on the services supported by a given producer would be published in the directory service, along with the event information.

5.3 Consumer

A consumer is any component that uses the consumer interface to receive event data from a producer. A given component may have multiple consumer interfaces, each acting independently

and receiving events. The term *consumer* is used interchangeably, and inexactly, to refer both to a single consumer interface and to a component that contains at least one consumer interface.

The core interaction functions that may be supported by a consumer are listed below.

1. Locate Producer: search the directory service for a producer. Corresponds to Directory Service *Search*.
2. Initiate Query: request one or more events from a producer, which are delivered as part of the reply. Corresponds to Producer *Accept Query*.
3. Initiate Subscribe: request establishment of a subscription with a producer. Further details about the subscription are returned in the reply. If the subscription is successfully established, the producer sends events until the subscription is terminated. Corresponds to Producer *Accept Subscribe*.
4. Initiate Unsubscribe: terminate a subscription. If this succeeds, the corresponding subscription will be closed, and no more events will be accepted for this subscription. Corresponds to Producer *Accept Unsubscribe.*
5. Maintain Registration: add/update/remove directory service entry or entries describing events that the consumer will accept from the producer. Corresponds to Directory Service *Add, Update,* and *Remove*.
6. Accept Notification: accept a single set of event(s) from a producer. Corresponds to Producer *Notify.*
7. Accept Subscribe: accept a subscribe request from a producer. Further details about the subscription are returned in the reply. If the subscription is successfully established, the producer sends events until the subscription is terminated. Corresponds to Producer *Initiate Subscribe*.
8. Accept Unsubscribe: accept an unsubscribe request from the producer. If this succeeds, the subscription will be closed, and no more events will be accepted for this subscription. Corresponds to Producer *Initiate Unsubscribe*.
9. Locate Event Schema: search request to the schema repository for a given event type. The schema repository may be the GMA directory service.

Many types of consumers are possible. A few are listed here as illustrative examples.

- Archiver: aggregate and store event data in long-term storage for later retrieval or analysis. An archiver may also act as a GMA producer when the data is retrieved from storage.
- Real-time monitor: collect monitoring data in real time for use by online analysis tools. An example is a tool that graphs cpu_load information in real-time.
- Overview monitor: collect events from several sources and use the combined information to make a decision that could not be made on the basis of data from only one producer. For example, one might trigger a call to the system administrator's pager at 2:00 am if both the primary and backup servers are down.

5.4 Intermediaries

A consequence of the separation of data discovery from data transfer is that the protocols used to perform the publish/subscribe, query/response, and notification interactions described in Section 4.2 can be used to construct intermediaries that forward, broadcast, filter, or cache the performance events.

The building block for these advanced services is the compound producer/consumer, which is a single component that implements both producer and consumer interfaces. For example, a consumer interface might collect event data from several producers, use that data to generate a new derived event data type, and make that available to other consumers through a producer interface, as shown in Figure 2. Many Grid services may in fact be both consumers and producers of monitoring events. For example, event archives would likely implement both producer and consumer interfaces.

Use of these intermediate components can lessen the load on producers of event data that is of interest to many consumers, with subsequent reductions in the network traffic, as the intermediaries can be placed “near” the data consumers.

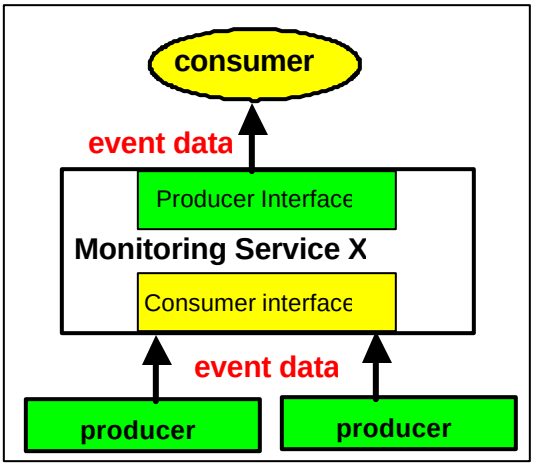

Figure 2: Compound Producer/Consumer

5.5 Sources of Event Data

The data used to construct events can be gathered from many sources. Hardware or software sensors that sample performance metrics in real time constitute one type of data source. Another is a database with a query interface, which can provide historical data. Entire monitoring systems, such as the Network Weather Service [10] can serve as a source of events. Additionally, application timings from tools such as Autopilot [4] or NetLogger [6] can provide events related to a specific application.

A producer may be associated with a single source, all sources on a given host, all sources on a given subnet, or an arbitrary group of sources. Figure 3 shows one possible configuration, but the architecture allows the performance system implementers to choose the configuration that best suits their scalability and reliability needs. The GMA specifies neither the relationship nor the interface between the measurement data sources and the GMA producer.

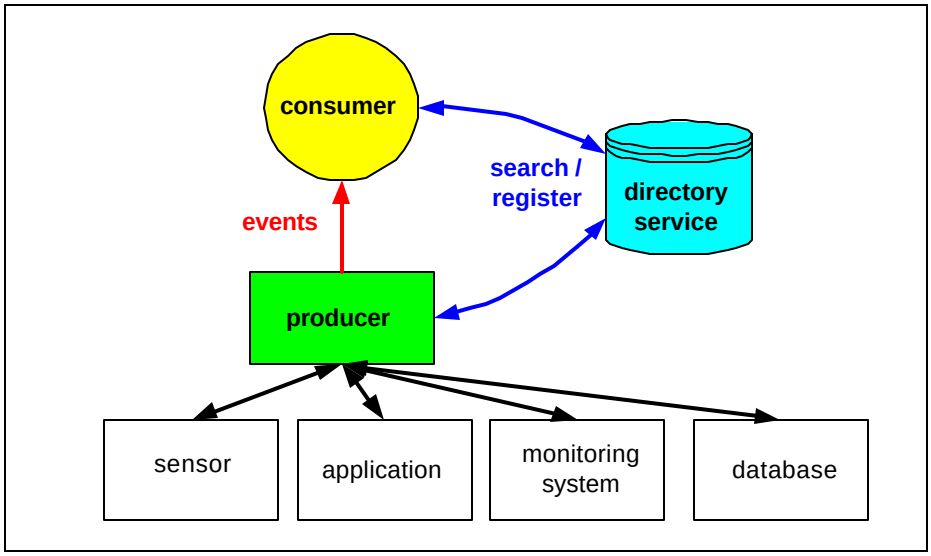

Figure 3: Sources of Event Data

6 Sample Use

A sample use of the GMA is shown in Figure 4. Event data is collected on the two hosts and at the network routers between them. The host and network sensors are the sources of the measurement data, which is managed by a producer. The producer registers the availability of the host and network events in the directory service. A real-time monitoring consumer subscribes to all available event data for real-time visualization and performance analysis. The producer is capable of computing summaries of network throughput and latency data based on parameters provided by a "network-aware" client. This client uses the summarized network information to optimally set its TCP buffer size. The producer's event data is also sent to an archive.

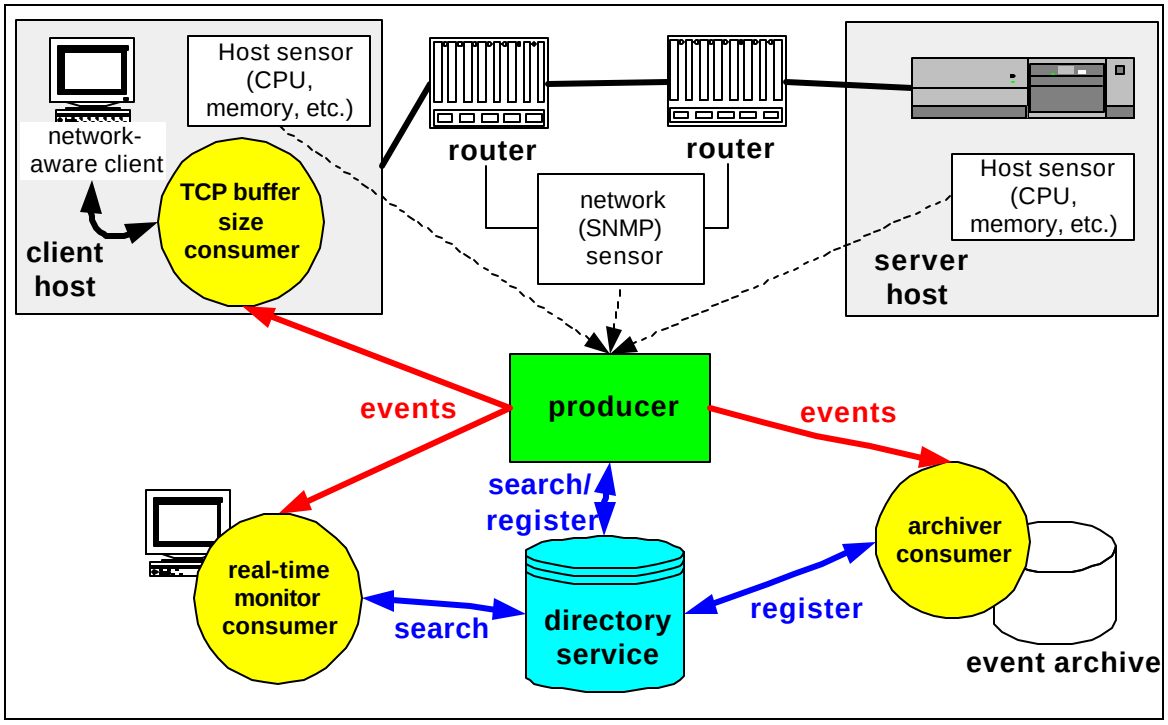

Figure 4: Sample GMA Usage

7 Implementation Issues

The purpose of a monitoring system is to reliably deliver timely and accurate information without perturbing the system. We believe that the architecture described can be implemented with acceptable levels of performance, scalability, and security. Unsatisfactory implementations are, however, also possible.

In this section we present implementation characteristics that have emerged from development experiences as being important to the success of monitoring and dynamic performance analysis systems. These characteristics are presented as a guide to developers producing or intending to produce implementations of the GMA. We recommend that the following strategies be considered and incorporated in implementations that are intended to serve as more than proof-of-concept investigations.

- **System components must be fault tolerant**. In large-scale distributed systems, failures will occur. For example, computer systems with monitoring servers will go down, and these monitoring servers should be restarted automatically from check-pointed internal

data. Directory servers will go down, and the data in these servers should be replicated in other servers. Networks can go down, and monitoring components must automatically reconnect and synchronize. The components of a monitoring system must be able to tolerate and recover from failures, and building fault tolerance into the monitoring system from the start will save effort later.

- **The data management system must adapt to changing performance conditions**. Dynamic performance data is often used to determine whether the shared Grid resources are performing well (e.g., fault diagnosis) or whether the current Grid load will admit a particular application (e.g., resource allocation and scheduling). To assess dynamic performance fluctuation, the data management system cannot itself be rendered inoperable or inaccessible by the very fluctuations it seeks to capture. As such, the data management system must use the data it gathers to control its own execution and resources in the face of dynamically changing conditions.
- **All system components must scale**. The monitoring system must be able to operate ubiquitously with respect to the resources or application components being monitored. To facilitate this scaling, one must be able to add additional producers and additional directory servers as needed, reducing the load where necessary. In the case where many consumers are requesting the same event data, the use of a producer reduces the amount of work on and the amount of network traffic from the host being monitored. Another important consideration is hierarchical control mechanisms for coordinating the resource load generated by producers; the pool of producers should not be managed as a "flat" collection. Moreover, distributed caching can be implemented by special-purpose consumer-producer nodes that are programmed to store and forward data as a way of relieving congestion and contention for particular data.
- **Monitoring data must be managed in a distributed fashion**. Having a single, centralized repository for dynamic data (however short its lifespan) causes two distinct performance problems. The first is that the centralized repository for information or control represents a single point –of failure for the entire system. If the monitoring system is to be used to detect network failure, and a network failure isolates a centralized controller from separate system components, it will be unable to fulfill its role. All components must be able to function when temporarily disconnected or unreachable because of network or host failure. The second problem with centralized data management is that it forms a performance bottleneck. For dynamic data, writes often outnumber reads with measurements taken every few seconds or minutes. Experience has shown that a centralized data repository simply cannot handle the load generated by actively monitored resources at Grid scales.
- **System components must control their intrusiveness on the resources they monitor**. Different resources experience varying amounts of sensitivity to the load introduced by monitoring. A two-megabyte disk footprint may be insignificant within a 10-terabyte storage system but extremely significant if implemented for a palm-top or RAM disk. In general, performance monitors and other system components must have tunable CPU, communication, memory, and storage requirements.
- **Efficiency/ease-of-use tradeoffs for data formats should be carefully considered**. In choosing a data format, there are tradeoffs between ease-of-use and compactness. While the easiest and most portable format may be ASCII text, including both event item descriptions and event item data in each transmission, this is also the least compact. Compressed binary representations fall at the opposite end of the ease/compactness spectrum. Another approach is transmitting only the item data values and using a data structure obtained separately to interpret the data. Implementers should carefully consider the requirements of their monitoring system when selecting data formats.
- **Security standards are useful**. Public key-based X.509 identity certificates [3] 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 cannot communicate directly with the monitoring process.

8 Security Considerations

A Grid monitoring system requires security mechanisms to ensure the integrity and privacy of both the monitoring system and the event data itself. For the most part, specifying these mechanisms is beyond the scope of this document, although some high-level discussion of security considerations and standards may apply.

9 Author Information

Brian Tierney Lawrence Berkeley National Laboratory bltierney@lbl.gov ph 1-510-486-7381 fx 1-510-486-6363	Ruth Aydt University of Illinois at Urbana-Champaign aydt@uiuc.edu	Dan Gunter Lawrence Berkeley National Laboratory dkgunter@lbl.gov	Warren Smith NASA Ames Research Center wwsmith@arc.nasa.gov
Martin Swany University of California Santa Barbara swany@cs.ucsb.edu	Valerie Taylor Northwestern University taylor@ece.nwu.edu	Rich Wolski University of California Santa Barbara rich@cs.ucsb.edu	

Glossary

GMA — Grid Monitoring Architecture, as defined by the Global Grid Forum Performance Working Group.

Acknowledgments

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

Dan Gunter and Brian Tierney are 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. Ruth Aydt is supported in part by the Department of Energy under contract DOE W-7405-ENG-36 and 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 and do not necessarily reflect the views of the funding agencies.

Intellectual Property Statement

this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

Full Copyright Notice

References

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

[2] Global Grid Forum Open Grid Service Infrastructure Working Group charter and documents. http://www.gridforum.org/ogsi-wg/.

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

[4] R. Ribler, J. Vetter, H. Simitci, and D. Reed. *Autopilot: Adaptive Control of Distributed Applications*. Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, July 1998.

[5] W. Smith. *A Framework for Control and Observation in Distributed Environments*. NASA Advanced Supercomputing Division, NASA Ames Research Center, Moffett Field, CA. NAS-01-006, June 2001.

[6] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. *The NetLogger Methodology for High Performance Distributed Systems Performance Analysis*. Proceedings of IEEE High Performance Distributed Computing Conference, July 1998. http://www-didc.lbl.gov/NetLogger/.

[7] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. *A Monitoring Sensor Management System for Grid Environments*. Proceedings of the IEEE High Performance Distributed Computing Conference (HPDC-9), August 2000.
[8] A. Waheed, W. Smith, J. George, and J. Yan. *An Infrastructure for Monitoring and Management in Computational Grids*. In Proceedings of the 2000 Conference on Languages, Compilers, and Runtime Systems, 2000.
[9] M. Wahl, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3)*. Available from ftp://ftp.isi.edu/in-notes/rfc2251.txt.
[10] R. Wolski, N. Spring, and J. Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. Future Generation Computing Systems, 1999. http://nws.npaci.edu/.