An XML-Based Protocol for Distributed Event Services

Warren Smith
NASA Ames Research Center
Moffett Field, CA 94035

Dan Gunter
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

Darcy Quesnel
Argonne National Laboratory
Argonne, IL 60439

Abstract

A recent trend in distributed computing is the construction of high-performance distributed systems called computational grids. One difficulty we have encountered is that there is no standard format for the representation of performance information and no standard protocol for transmitting this information. This limits the types of performance analysis that can be undertaken in complex distributed systems. To address this problem, we present an XML-based protocol for transmitting performance events in distributed systems and evaluate the performance of this protocol.

Keywords: event service, XML, distributed computing, computational grids.

1 Introduction

There are many different projects from government, academia, and industry that provide services for delivering events in distributed environments. The problem with these event services is that they are not general enough to support all uses and they speak different protocols so that they cannot interoperate. We require such interoperability when we, for example, wish to analyze the performance of an application in a distributed environment. Such an analysis might require performance information from the application, computer systems, networks, and scientific instruments. In this work we propose and evaluate an XML-based protocol for the transmission of events in distributed systems.

One recent trend in government and academic research is the development and deployment of computational grids [14]. Computational grids are large-scale distributed systems that typically consist of high-performance compute, storage, and networking resources. Examples of such computational grids are the DOE Science Grid [3], the NASA Information Power Grid [8, 18], and the NSF Partnerships for Advanced Computing Infrastructure [9, 10]. The major effort to deploy these grids is in the area of developing the software services to allow users to execute applications on these large and diverse sets of resources. These services include security, execution of remote applications, managing remote data, access to information about resources and services, and so on. There are several toolkits for providing these services such as Globus [4, 13], Legion [7, 15], and Condor [1, 19].

As part of these efforts to develop computational grids, the Global Grid Forum [5] is working to specify general protocols and APIs to be used by various grid services. These specifications will allow interoperability between the client and server software of the toolkits that are providing the grid services. The goal of the Performance Working Group [6] of the Grid Forum is to codify best practices and promote interoperability for the storage and distribution of performance data. The resulting specifications must support tasks such as profiling parallel applications, monitoring the status of computers and networks, and monitoring the performance of services provided by a computational grid.

This paper provides an overview of a proposed protocol and data representation for the exchange of events in a distributed system. The protocol exchanges messages formatted in

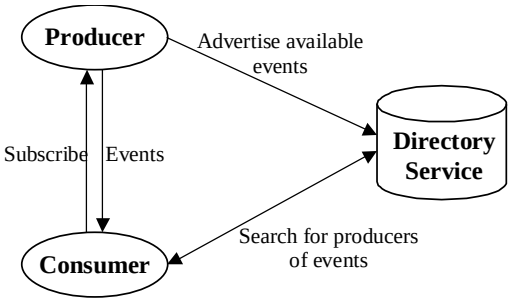

Figure 1. Grid Monitoring Architecture.

XML and it can be layered atop any low-level communication protocol such as TCP or UDP. Further, we discuss Java and C++ implementations of this protocol and their performance.

The next section will provide some further background information. Section 3 describes how we represent events and related information using XML. Section 4 describes our protocol and Section 5 discusses the performance of two implementations of the protocol.

2 Background

The Grid Forum Performance Working Group has defined the basic architecture shown in Figure 1. This architecture consists of three components: a producer, a consumer, and a directory service. A *producer* is something that is producing performance data, each unit of which is called an *event*. This producer can be an application profiler, a host monitor, or anything else. A *consumer* is something that consumes or receives events. A consumer might be a tool to calculate how much time is spent in each function of an application or a graphical interface showing the status of a set of hosts. A *directory service* is a database that is used to store and retrieve information about producers and consumers. It is accessed using a protocol such as the Lightweight Directory Access Protocol (LDAP) [17]. A host monitor may advertise itself in the directory service so that a consumer can search the directory service and find the monitor for a certain host. The consumer can than contact that producer in order to receive events about that host.

The Grid Forum Performance Working Group is defining the protocols and data representations required by this architecture. This includes:

- A definition of events and information related to events,
- The protocol for communicating between producers and consumers of events, and
- A definition of the structure and organization of the data in the directory service.

In this paper we describe a proposed producer-consumer communication protocol and the event information that is required by this protocol. Our protocol consists of a XML encoding of messages and the state machines that describe when these messages are sent. We do not specify the transport protocol on top of which our protocol will be layered. The transport protocol could be UDP, TCP, HTTP, SSL, or any number of other protocols. We choose to use XML to represent our data for several reasons. First, XML provides a textual representation of data that is readable and therefore easier to debug. We could have selected a binary representation of our data for improved performance, but a textual approach seems more appropriate at our current experimental stage. Second, XML is self-describing and hierarchical, which makes it easy to represent structured event data. Third, XML was selected instead of any other textual representation because of the large and growing number of XML tools available and the growing number of people familiar with XML.

Another approach we could have taken was to use SOAP [12] or XML-RPC [11] and thus avoid explicit representation of the XML for each message. While this approach is a valid one, it has several drawbacks. First, neither SOAP nor XML-RPC has low-level transport bindings: SOAP has HTTP and SNMP bindings, and XML-RPC has only an HTTP binding. These bindings are not suitable for all of the situations we wish to address. Second, for SOAP, there is a lack of fully-featured implementations in the languages we are interested in and/or licensing restrictions on the available implementations. We will continue to track these XML-based protocols and may adopt them in the future.

Another approach would have been to use CORBA [20], for instance the CORBA Event Service. This approach, while also valid, would

impose a significant administrative and development overhead if the target community did not already use CORBA, as is the case with the academic and scientific communities. SNMP [21] was also considered, but was not used due to its inability to handle streaming data efficiently.

3 Events and Event Parameters

Before we describe our protocol, we first describe how we use XML to represent events and event parameters. In this section and the following sections we provide example XML representations of the data we are representing.

As mentioned before, events are the basic unit of information in our architecture. An event is a named set of <name, value> pairs where the values are typed and there is always a pair that contains the time the event was generated. We represent this time using a time stamp that is a string formatted according to the proposed Grid Forum standard format [16] This format is an extension of the ISO 8601 time format [2]. Each element will also have two optional attributes: units and accuracy. The units attribute indicates the units associated with the element's value (e.g.: 'degrees', or 'bytes') and the accuracy attribute indicates what range of likely "real" values are represented by the element's value (e.g. '+/-5.0').

Associated with each event is a set of parameters that describe the information that can be passed to a producer of events as part of a subscription or query. The event parameters consist of a set of <name, value> pairs. Each element can have a units attribute associated with it. An examples of an event and its parameters are shown in Section 3.1.

3.1 CPU Load

The CPU load event is a simple event for containing the load information returned by the Unix uptime command. We therefore use the event name "UptimeCPULoadEvent" for this event to differentiate it from other means of measuring CPU load. This event must contain the following elements:

- TimeStamp. The time at which the CPU load event was generated.
- Load1. The 1 minute CPU load reported by uptime.
- Load5. The 5 minute CPU load reported by uptime.
- Load15. The 15 minute CPU load reported by uptime.
- HostName. The name of the host the load measurement is made on.

Here is an example of such an event in our XML encoding:

```
<UptimeCPULoad
xmlns="http://www.gridforum.org/Performance/Events">
  <Load1>1.5</Load1>
  <Load5>1.6</Load5>
  <Load15>1.3</Load15>
  <HostName>foo.gov</HostName>
  <TimeStamp>2000-11-09T21:51:45Z</TimeStamp>
</UptimeCPULoad>
```

When asking for a CPU load event, the following input parameters can be specified:

- Period. The amount of time between each uptime event generation. This parameter is only used when a subscription is performed. If this parameter is specified for a query, it is ignored.

An example of how to specify this parameter is:

```
<UptimeCPULoad
xmlns="http://www.gridforum.org/Performance/EventParameters">
    <Period units="min">10</Period>
</UptimeCPULoad>
```

4 Protocol

This section describes the XML protocol we use for communication between producers and consumers. Due to space limitations, we do not provide the XML schema or state machines for our protocol. Our protocol supports three major classes of interactions between producers and consumers.

In the first interaction, a consumer subscribes to specific events from the producer and the producer sends these events to the consumer. These events are sent out over a period of time until the producer or consumer ends the subscription. We call this interaction a *consumer-initiated subscription*.

The second type of interaction is the *producer-initiated subscription.* First, the

producer contacts a consumer to request a subscription. Then events are sent from the producer to the consumer until the subscription is terminated. This type of interaction is useful, for example, when a producer sends events to an archive. In this case, the archive is the consumer.

The third type of interaction is a simple *request/reply*. In this case, a consumer requests information from a producer and the producer replies with the information. Our two previous interactions include request/reply interactions but our protocol includes two instances of this interaction that stand on their own. First, there is a *query* interaction. In this interaction the consumer queries a producer for a single event and the producer replies with the event. Second, there is an *available events* interaction where a consumer requests a list of the events available from a producer and the producer replies with the list.

4.1 General Message Format

In general, each message consists of:

1. The number of bytes in the message. For our TCP binding, this is a 32-bit integer in network byte order.
2. The XML tags that indicate the message type.
3. Request messages always have a requester-unique request ID chosen by the requestor. This request ID is an attribute of the message tag
4. Reply messages always have a request ID, which matches the request ID of the request that is being replied to.
5. Reply messages always have a return code and may have a detailed return message. The Return element indicates if an operation was successful (Success) or a failure (Failure). These return codes will most likely be expanded later to contain more detailed error codes. The ReturnDetail element contains a text message that contains detailed user-readable information about the status of a request.
6. The message-specific data inside the XML tags that identify the message.

We define three XML name spaces for use in our protocol. The name space *http://www.gridforum.org/Performance/Events* contains the events defined by the Grid Forum Performance Working Group, the name space *http://www.gridforum.org/Performance/EventParameters* contains the parameters defined by the working group that can be specified when asking for an event or events, and the name space *http://www.gridforum.org/Performance/Protocol* contains the elements which make up the messages of our protocol. Further, we allow any group to define events and event parameters in their own name spaces for use with our protocol.

4.2 Consumer-Initiated Subscription

When a consumer wants to receive a stream of events from a producer, it subscribes to the producer for the events. After a subscription successfully takes place, events are sent from the producer to the consumer until either the consumer or producer unsubscribes. There are five messages in this process, described in the following sections.

4.2.1. Subscribe Request

The subscribe request message initiates a subscription and consists of:

- A consumer-unique request ID (required).
- A consumer-unique subscription ID (required).
- Event parameters element (required).
- Any input parameters needed to generate events (optional).

Here is an example of a subscribe request message:

```
<SubscribeRequest
xmlns="http://www.gridforum.org/Perfor
mance/Protocol" requestID="1">
  <SubscriptionID>12</SubscriptionID>
  <UptimeCPULoad
xmlns="http://www.gridforum.org/Perfor
mance/EventParameters">
    <Period units="sec">600</Period>
  </UptimeCPULoad>
</SubscribeRequest>
```

In the future, we will add an optional event filter to subscription request messages. The filter specifies which events in the stream of events should be sent on to the consumer. For example, a filter may indicate that only CPU load events with a 5-minute load average greater than or equal to 5.0 should be sent to the consumer.

4.2.2. Subscribe Reply

The subscribe reply message is sent in response to a subscribe request and consists of:

- The requestID (required) of the request that this message is in reply to.
- Return (required). Success means the request was successfully completed, Failure means the request failed. Other return codes to represent more detailed failures will most likely be added in the future.
- ReturnDetail (optional). Text giving further information about the successful or unsuccessful subscribe.
- An optional producer-unique SubscriptionID that identifies the subscription that was successfully made by the consumer (if one was). The subscription ID should be present if the subscription was successful and should not be present if the subscription was not successful.

An example of a subscribe reply message is:

```
<SubscribeReply
xmlns=http://www.gridforum.org/Perform
ance/Protocol" requestID="1">
  <Return>Success</Return>
  <SubscriptionID>99</SubscriptionID>
</SubscribeReply>
```

4.2.3. Unsubscribe Request

Unsubscribe requests can originate at either the producer or consumer. In either case, the message has the same format. The unsubscribe request message consists of:

- A sender-unique requestID (required).
- The SubscriptionID (required) generated by the message target (i.e. producer if the sender is the consumer, consumer if the sender is the producer) that identifies the subscription that is being terminated.

An example of an unsubscribe request message is:

```
<UnsubscribeRequest
xmlns="http://www.gridforum.org/Perfor
mance/Protocol" requestID="9">
 <SubscriptionID>1234</SubscriptionID>
</UnsubscribeRequest>
```

4.2.4. Unsubscribe Reply

The unsubscribe reply message is sent in response to an unsubscribe request consists of:

- The requestID (required) of the request that this message is in reply to.
- Return (required).
- ReturnDetail (optional).

An examples of a unsubscribe reply message is:

```
<UnsubscribeReply
xmlns="http://www.gridforum.org/Perfor
mance/Protocol" requestID="9">
  <Return>Success</Return>
</UnsubscribeReply>
```

4.2.5. Event

An event message is sent from the producer to the consumer after a subscription is initiated. An event message consists of:

- The subscription ID (required) that was generated by the consumer.
- The event (optional) in the format described in Section 3. The event should be present if an error is not reported.
- Error (optional), indicating that an error occurred while generating the event.
- ErrorDetail (optional) which provides further information about the error that occurred while generating the event. This element should only occur in conjunction with the Error element.

Example event messages are shown below.

```
<Event
xmlns="http://www.gridforum.org/Perfor
mance/Protocol" subscriptionID="1234">
  <UptimeCPULoad
xmlns="http://www.gridforum.org/Perfor
mance/Protocol">
    <Load1>1.5</Load1>
    <Load5>1.6</Load5>
    <Load15>1.3</Load15>
    <TimeStamp>2000-11-
09T21:51:45Z</TimeStamp>
  </UptimeCPULoad>
</Event>
```

4.3 Producer-Initiated Subscription

There are cases where a producer of events may want to initiate a subscription. A common case is when a producer wants to archive the events it is generating. The request and reply messages used during a producer-initiated subscription are identical to those used for a consumer-initiated subscription. The only difference is that the producer requests the subscription instead of the consumer.

4.4 Querying for an Event

Often a consumer will want just one event from a producer. Instead of having a consumer subscribe, receive 1 event, and then unsubscribe, we allow a consumer to query a producer for an event. A query consists of a query request message that a consumer sends to the producer and a query reply message that the producer sends to the consumer in response to the query request message. The query reply includes the event that was requested.

4.4.1. Query Request

The query request message is very similar to the consumer subscribe request message and consists of:

- A request ID (required).
- Event parameters element (required).
- Any input parameters needed to generate events (optional).

Here is an example QueryRequest message:

```
<QueryRequest
xmlns="http://www.gridforum.org/Perfor
mance/Protocol" requestID="15">
  <UptimeCPULoad
xmlns="http://www.gridforum.org/Perfor
mance/Events"/>
</QueryRequest>
```

4.4.2. Query Reply

The query reply messages are similar to the event messages and consist of:

- A request ID (required) to identify which QueryRequest this reply is for
- Return (required).
- ReturnDetail (optional).
- The event data in the format described in Section 3.

An example query reply message is:

```
  <QueryReply
xmlns="http://www.gridforum.org/Perform
ance/Protocol" requestID="15">
  <Return>Success</Return>
  <UptimeCPULoadEvent
xmlns="http://www.gridforum.org/Perform
ance/Events">
    <Load1>1.5</Load1>
    <Load5>1.6</Load5>
    <Load15>1.3</Load15>
    <TimeStamp>2000-11-
09T21:51:45Z</TimeStamp>
  </UptimeCPULoadEvent>
</QueryReply>
```

4.5 Requesting Available Events

Even though our architecture in Figure 1 shows a directory service that will be used to contain information on the events that are available from a producer, it is also convenient to be able to obtain this information from producers directly.

4.5.1. Event Names Request

The available events request message is very simple and only contains a request ID. Here is an example EventNamesRequest:

```
<EventNamesRequest
xmlns="http://www.gridforum.org/Perfor
mance/Protocol" requestID="15"/>
```

4.5.2. Event Names Reply

The event names reply messages consist of:

- A request ID (required).
- Return (required).
- ReturnDetail (optional).
- One or more Event elements that do not have values. Instead, they have two attributes:
  - The name attribute specifies the name of the available event.
  - The namespace attribute specifies the namespace.

An example event names reply message is shown below.

```
<EventNamesReply
xmlns="http://www.gridforum.org/Perfor
mance/Protocol requestID="15">
  <Return>Success</Return>
  <Event name="UptimeCPULoad"
namespace="http://www.gridforum.org/Pe
rformance/Events"/>
</AvailableEventsReply>
```

5 Performance

In this section we present performance results for two independent implementations of our protocol. One implementation uses Java and the Xerces XML parser. The other implementation uses C++ and the expat XML parser. We examined the performance of these implementations using a 933 MHz Pentium III system running RedHat Linux 7.1 with JDK 1.3. We found that the C++ implementation is significantly faster. It can decode 4,300 uptime cpu load event messages a second to C++ objects and encode 28,100 event messages a

second from C++ objects. The Java implementation can decode 600 event messages a second and encode 21,900 event messages a second.

6 Conclusions

This document describes an XML-based protocol for the transmission of performance events in a distributed environment. The protocol we describe is a proposed standard in the Performance Working Group of the Grid Forum. The purpose of this protocol is to address the problem of providing performance information in a standard way so that different tools can provide and use such information. We require such interoperability in a computational grid when we wish to analyze the performance of an application that uses several different resources.

We constructed two independent implementations of this protocol that interoperate. One implementation is written using Java, and the other using C++. We found that the C++ implementation can decode messages significantly faster than the Java implementation but the encoding time is similar.

References

[1] "Condor High Throughput Computing," http://www.cs.wisc.edu/condor/.

[2] "Data elements and interchange formats - Information interchange - Representation of dates and times," International Organization for Standardization ISO 8601, 1998.

[3] "The DOE Science Grid," http://www-itg.lbl.gov/Grid.

[4] "The Globus Project," http://www.globus.org.

[5] "Grid Forum," http://www.gridforum.org.

[6] "Grid Forum Performance Working Group," http://www-didc.lbl.gov/GridPerf/.

[7] "The Legion Project," http://www.cs.virginia.edu/~legion/.

[8] "The NASA Information Power Grid," http://www.ipg.nasa.gov.

[9] "The National Computational Science Alliance," http://www.ncsa.uiuc.edu/access/index.alliance.html.

[10] "The National Partnership for Advanced Computing Infrastructure," http://www.npaci.edu/.

[11] "XML-RPC Home Page," http://www.xmlrpc.com.

[12] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," The World Wide Web Consortium 2000.

[13] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, vol. 11, pp. 115-128, 1997.

[14] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure,".: Morgan Kauffmann, 1999.

[15] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. R. Jr., "Legion: The Next Logical Step Toward A Nationwide Virtual Computer," Department of Computer Science, University of Virginia CS-94-21, June, 1994 1994.

[16] D. Gunter and B. Tierney, "A Standard Timestamp for Grid Computing." In Proceedings of the Global Grid Forum 1, 2001.

[17] T. Howes, M. Smith, and G. Good, *Understanding and Deploying LDAP Directory Services*: MacMillan Technical Publishing, 1999.

[18] W. Johnson, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid." In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, 1999.

[19] M. Litzkow and M. Livny, "Experience with the Condor Distributed Batch System." In Proceedings of the IEEE Workshop on Experimental Distributed Systems, 1990.

[20] A. Pope, *The CORBA Reference Guide*. Reading, MA: Addison-Wesley, 1998.

[21] W. Stallings, *SNMP, SNMPv2, and CMIP: The Practical Guide to Network-Management Standards*. Reading, Massachusetts: Addison-Wesley, 1993.