A Survey of Grid Monitoring Systems

Yifan Pan
Indiana University
panyif@cs.indiana.edu

ABSTRACT
Monitoring is important in grid system in that it can effectively monitor resources and collect important information, such as status and performance, regarding particular resources of interest. This survey paper aims to give an overview on some of the existing grid monitoring techniques and systems to provide a general idea of this area.

1. INTRODUCTION
The grid can be viewed as a distributed system that combine different scattered communities to form Virtual Organization, in order to make fully use of the computational resources and collaborate for the common goals for the communities [6]. However grid resources tend to be unstable in that they may come and go dynamically so that we need to monitor the resources to make sure their status. Furthermore, grid monitoring systems provide a way for other system administration tasks such as scheduling or performance analysis. Thus monitoring systems are crucial in grids.

According to [6], grid monitoring has some unique characteristics that are different from traditional monitoring, including scalable support for data delivery models across distributed organizations, and tradeoff between the extensibility and compactness of the data format, which can be seen an issue of interoperability.

There are many existing grid monitoring systems out there [1, 3, 4]. Because of the different nature of grid systems they aim at, they have very different architectures, as well as the terms and techniques they used. Global Grid Forum proposes a Grid Monitoring Architecture (GMA) [5] which provides and characterizes different components in a grid monitoring system. Based on different characteristics of the components, [6] provides a taxonomical classification of many existing grid monitoring systems, providing a good overview of this area.

This paper will only survey a few existing systems, whose characteristics are largely different. Before getting into the details of the systems, we will briefly review the monitoring processes and the GMA architecture [5]. Then we will review some systems, namely RGMA [1], NaradaBrokering [3] and INCA [4], to characterize the similarities and key differences in these systems.

2. BACKGROUND
In this section, we briefly review the monitoring process, as well as the Grid Monitoring Architecture.

2.1 Monitoring Process
According to [6], the monitoring of distributed systems typically involves 4 stages: a) The generation of events (time-stamped performance data), in which the monitoring process enquire entities (any network resources) to generate the data needed for monitoring. b) The processing of events, which is application-specific determined by how these events should be used. c) The distribution of events, which determines how the processed events should be distributed. d) The presentation of events, generally involving a GUI, in which the events will be presented nicely and meaningfully to the users.

Typically a monitoring system should handle these 4 stages of monitoring process well. Moreover, there are other requirements that are needed for monitoring systems, including scalability, extensibility, data delivery models, portability, security and also data freshness. A more detailed description can be found in [6].

The Grid Monitoring Architecture [5] proposed by Global Grid Forum aims at providing a basic standard architecture for grid monitoring systems. The following section will discuss about that.

2.2 Grid Monitoring Architecture (GMA)
The Grid Monitoring Architecture consists of three types of components:

- Producer: any process that produces performance data (the event source)
- Consumer: any process that receives performance data (the event sink)
- Directory Service: a service supports information publication (for the producers) and discovery (for the consumers).

For the interactions between producers and consumers, 2 different models are supported by GMA, namely the publish/subscribe model and the query/response model. In publish/subscribe model, specific event types can be subscribed
Figure 1: GMA architecture

so that the desired events can be transferred from the producers to the consumers. Subscribers will then only receive events that they are interested in. Both producers and consumers can initiate the stream of events. In query/response model, consumer will initiate the interaction, and then the sending and receiving of events can happen.

Meanwhile, the GMA allows a compound producer/consumer (referred as republisher in [6]). It implements both consumer and producer interfaces so that tasks such as filtering or summarizing can be supported.

GMA is an architecture and does not define implementation details. Different systems, in fact, are based on the ideas in GMA and implemented. Some systems before GMA actually had similar ideas as well, and in some sense the deficiencies of those then existing systems can be seen as a motivation of the proposing of GMA.

In the following sections, we will go over some existing grid monitoring systems.

3. RELATIONAL GRID MONITORING ARCHITECTURE (R-GMA)

Relational Grid Monitoring Architecture (R-GMA)[1] is a relational implementation of GMA. It is developed within the European DataGrid Project as a grid information and monitoring system.

By relational, it means that producers announce what they have to publish with an SQL CREATE TABLE statement, and publish events through an SQL INSERT statement. Consumers can collect information they want, which are specific events having some particular characteristics or having certain values on some database fields, using an SQL SELECT statement. This implementation gives the Virtual Organizations an impression that a relational database exist for each of them, and this can bring the flexibility of the relational model. All the producers are quite independent in R-GMA. As can be seen here, R-GMA implementation can support both publish/subscribe model and query/response model in GMA.

3.1 Query and Producer Types

Three types of queries are supported by R-GMA: history, latest and continuous. In order to support these 3 query types, 5 different types of producers are provided. DatabaseProducer supports latest queries by holding only the latest records. Two types of producers are used to support continuous queries. The StreamProducer writes information to a memory structure so that events are not lost when system crash happens. The last type of producer is CanonicalProducer. It has no user interface to publish data via an SQL INSERT so that it triggers user code to answer SQL. Cleanup mechanisms are supported so that events may be deleted. It can be imagined that by treating every event as a database tuple, and treating the publishing of events as SQL statement, these mechanisms can be very user-friendly.

The compound producer/consumer, which is called archiver in R-GMA, is also an important component in R-GMA. It can collect events for the consumer by first collect all the information from different producers, do the filtering and then distribute the data to the consumer.

3.2 Architecture

The architecture of R-GMA is highly similar to the one described in GMA. It has a registry (directory service) to hold the information about the producers. It also has a mediator that uses information in the registry and helps user to construct query plans dynamically.

R-GMA is based on servlet technology and multiple APIs are available, including Java, C++, or C.

4. NARADABROKERING

Above we have described R-GMA, an implementation based on GMA standard. In this section we will review NaradaBrokering [3], which is a framework enabling Peer-to-Peer (P2P) grid monitoring. A P2P grid can comprise services originally available on traditional grids and P2P networks, and can naturally support the situation when features from both cases are desirable. P2P grid thus integrates ideas of computational grids, web services, P2P networks as well as message-oriented middleware.

Being a message infrastructure instead of merely a monitoring system, it is fully compliant with Java Message Service (JMS) and supports SOAP message, JMS message and complicated events. It can support either publish/subscribe mode, or P2P mode. It has an efficient algorithm to find a shortest route to send the events to the destination. NaradaBrokering supports many underlying data transport protocols including TCP, UDP, multicast, SSL, HTTP, etc.

4.1 Basics

In the GMA architecture, while it has a directory service that supports the publication and discovery of the events, actual event transfer happens directly from producer to consumer. This is not the only way of transferring data. We know that message-oriented middleware can serve as a storage of messages and deliver messages to destination whenever the destination entity is available. We can use that as a broker for the message transferring. Publisher post
messages to an intermediate broker, and subscribers register subscriptions with that broker and the broker can then do the filtering.

NaradaBrokering is based on brokering network, consisting of multiple brokers. A hierarchical broker network structure is imposed, where a broker is a part of a cluster which is a part of a larger cluster. Brokers within a cluster are strongly connected so that it can be robust even facing network failures.

4.2 Events Dissemination

Using a brokering network for system monitoring, we require that brokering network be efficient so that events can be disseminated from the publishers to the subscribers in timely fashion. In NaradaBrokering infrastructure, every event has a destination list (comprising clients) and the brokering system is responsible for computing the broker destinations (targets) and ensuring the efficient delivery of events to the targeted brokers and the clients. As events pass through the network, they will get updated so that the dissemination is snapshot. The routing task is crucial here. A near optimal routing is deployed here because the brokers in the optimal route may not be associated with the event, and sometimes due to failure, the actual shortest path from every broker to the targeted broker may not be computed.

4.3 Performance Comparison with R-GMA

Huang et al. gave an experimental study comparing NaradaBrokering and R-GMA for Real-Time Grid Monitoring [2]. They compared NaradaBrokering and R-GMA on following aspects: real-time performance, concurrent connections and throughput, and scalability.

The results of NaradaBrokering tests show that NaradaBrokering has a very good real-time performance and high throughput. Among different transport protocols, they found that TCP is the best to reach high performance. The message size and publishing rate will both affect the performance. And the scalability of NaradaBrokering, is not very impressive in their experiments. On the version they were using (tested in 2007) the concurrent support is limited. R-GMA, on the other hand, has lower performance and throughput, as it takes longer time to process data. Producers may need to wait for a short time for avoiding data delay or lost. Concurrent connection support is a bit limited as well. A distributed R-GMA network has better support on the concurrent connection and has better performance. It does have good scalability.

5. INCA

The systems mentioned earlier in this paper, including R-GMA or NaradaBrokering, provide system-level monitoring in that they provide system-level information on the utilization of grid resources. Inca system [4], on the other hand, provides user-level grid monitoring. Periodic and automated user-level testing of the software and services can be supported in Inca so that users can better utilize grid resources and support grid operation.

On system-level monitoring systems, low-level host statistics and queue information can be collected. While this type of monitoring information is sufficient for showing the utilization of grid resources, as pointed out in [4], it does not provide the way for detecting high-level user problems such as incompatible software components within grid.

5.1 Goals and Features

According to [4], here are some features that user-level grid monitoring should include:

- Runs from user account to reflect user experience
- Executes a standard user GSI credential when authentication is required
- Emulates a user by configuration based on user documentation
- Centrally manages the configuration of user-level tests
- Easily updates and maintains user-level tests
- Provides a indication of grid status
- Automates the periodic execution
- Executes locally on grid resources

And here are some of the features provided in the Inca 2 system (the then latest version of Inca):

- Collects variety of monitoring results
- Captures the context of a test or benchmark as it executes
- Eases the process of writing tests and benchmarks
- Provides means for sharing tests and benchmarks
- Easily adapts to new resources and monitoring requirements
- Stores and archives monitoring results
- Securely manages short-term proxies
- Measures the system impact of tests and benchmarks on the monitored resources

5.2 Inca 2 Architecture

There are mainly three core components in the Inca 2 architecture, namely the agent, the depot, and reporter manager. The agent and reporter manager coordinate the execution of tests and stores the results to the depot. Reporter repositories containing user-level tests and benchmarks, called reporters, are the inputs to Inca 2. A configuration file describing how to execute the tests, which can be created by the GUI tool incat, is also provided in the input. Data consumers can then query the collected results and displayed them to the users.

Here is a brief summary of components in Inca 2 systems:

- Reporters: An inca reporter is an executable program which tests on some aspect of a system or software. Multiple types of data can be accommodated using Inca reporter schema.
- Reporter Repositories: Reporter Repositories are collection of reporters, packages and libraries, and a catalog file.
- Inca administration tool (incat): Incat is a Java Swing GUI that Inca administrator use to generate configuration file.
• Agent: An agent is a server that implements the configuration.
• Reporter Manager: The Inca reporter manager is a process for scheduling and managing the execution of Inca reporters on a single resource.
• Depot: The depot is used for storing configuration information and also the data produced by the reporter. It can be queried by the data consumers.
• Data Consumer: The data consumer will query the Inca depot and display results in a user-friendly way.

As can be seen from the descriptions above, as Inca 2 focuses on user-level monitoring, the architecture focuses more on how to effectively collect user inputs, how to enhance the tests and measurements sharing among users, how to provide a way for generating user-specific configuration, how to manage and schedule the execution of reporters from different users, and how to display results in a user-friendly way. Such issues may not occur in previous discussed system-level monitoring systems. Inca 2 aims to provide maximum flexibility for users to generate specific tests and benchmarks. So in the paper they focus more on describing the deployment of user-level monitoring tasks, instead of the underlying implementation details of the system.

6. CLASSIFICATION TAXONOMY

As mentioned earlier, [6] provides a taxonomy to classify grid monitoring systems. The categories in the taxonomy are named from 0 to 3 depending on the provision and characteristics of a system’s producers and republishers:

Level 0 Events flow directly from the processes monitoring certain entities (called sensors) to consumers. There is no producer API in this level and will prevent distribution of events to remotely located components in a programmable fashion.

Level 1 In first-level systems, producers provide generic API so that events are remotely accessible.

Level 2 Second-level monitoring systems feature at least one type of republisher with a fixed functionality, which may be different on different producers but is predefined.

Level 3 Monitoring systems of this level are highly flexible, in that the republishers are configurable. Also, they have a potential for scalability and may form a standalone Grid Information Services. Many systems fall into level 2 or level 3. For example, R-GMA falls into level 3.

In addition to the level information, [6] uses other qualifiers for classification. Type of entities is one of the qualifiers. It describes the type of entities that are primarily monitored by a system, and can have the value of H (for hosts), N (for networks), A (for applications), V (for availability) and G (for generic, composed of at least H and N). Another qualifier is stackable, denoting (using letter S) whether the monitoring system can operate on top of another monitoring system or not. According to [6], for example, R-GMA has the taxonomy label L3.G.S, which is of the highest flexibility in the taxonomy.

This taxonomy provides a very good view of studying existing monitoring systems. We can see from this paper the pros and cons of some current monitoring systems.

7. CONCLUSIONS

In this survey paper we review some of the architectures and systems for grid monitoring. Not many comparisons are done among these systems, as they are different in nature and have different aims. GMA and its implementation, R-GMA, provides a straightforward architecture and implementation of grid monitoring system. NaradaBrokering, being a message infrastructure for P2P grids, has a broader goal, yet fits perfectly for the grid monitoring task. Inca 2, instead of supporting system-level monitoring like most of existing systems do, provides a user-level grid monitoring. A nice taxonomy was provided in [6] which can categorize and better study existing monitoring systems.

8. REFERENCES