Grid monitoring system survey

by Tian Xu txu@indiana.edu

Abstract

The process of monitoring refers to systematically collect information regarding to current or past status of all resources of interest. Due to the dynamic nature of grid systems, the monitoring components are becoming essential in order to utilize different types of distributed resources and provide consistent and unified services to users. This paper gives a general survey on grid monitoring systems by identifying key components and popular interaction models in monitoring process, introducing important requirements and inspecting different architectures from various existing distributed monitoring systems.

Introduction

Grid systems provide universal and consistent access to distributed resources (computing, data storage and analysis, instruments). However, these resources will dynamically register, leave, change message types and even shut down with various reasons from time to time. In order to coordinate between multiple and potentially heterogeneous recourses and services, grid systems need to systematically collect information regarding the current and past status of grid recourses; a process known as monitoring [1].

In a variety of cases, such as scheduling, data replication, accounting, performances analysis and optimization of distributed systems, monitoring is a crucial component. Along with the development of various grid system, their large scalability and real-time services requirement from users have made monitoring essential for effective management, especially in detecting fault and bottlenecks. Also, monitoring system can provide user level pattern identification, such as inca2 [2], for long-term resource planning and service optimization of grid systems.

The difference between traditional monitoring system, which has been studied for decades, and grid monitoring system lies in the fact that grid monitoring is characterized by significant requirements such as scalable support for both pull and push data delivery models applied over vast amounts of current and past monitoring data that may be distributed across organizations [1]. Also, the data format of a monitoring system has to balance between extensibility, self-description and compactness. The former is required to accommodate the ever expanding types of monitored resources, whereas the latter is a prerequisite for non-intrusive and scalable behavior. The problem is further complicated by the continuous evolution of grid middleware
and the lack of consensus regarding data representation, protocols and semantics, leading to ad
hoc solutions of limited interoperability. Existing proprietary network and host monitoring
applications lack the openness required for interoperability and customization, while they also
impose significant financial costs. Few of the equivalent open source projects have a potential
and in fact some of them are actually employed for grid monitoring. Therefore, on one hand,
managing the large amount of resources in grid systems has made monitoring essential; on the
other hand, it also provide challenges for grid monitoring systems.

This paper aims to provide an detailed introduction of typical structure of grid monitoring
systems and provide an overview of existing monitoring architectures. In section 2, some
background of monitoring processes will be introduced (general stages and requirements),
including an overview of rid Monitoring Architecture [3]. In section 3, an approach of taxonomy
is described. In section 4, an overview of several existing popular grid monitoring systems will be
stated and discussed. And, conclusions and summations will be given in section 5.

Background

This section will introduces some basic backgrounds of grid monitoring systems. In grid
monitoring systems, the process that monitoring the activities of all resources and events are
usually referred as sensors. Also, an event schema refers to the schema that defines the type of
structure and semantics of all types of events.

The monitoring process

Monitoring distributed systems, according to literature [8,9], usually includes four stages: 1) 
generation of events, sensors enquiring entities and encoding the measurements according to a
given schema; (ii) processing of generated events is application-specific and may take place
during any stage of the monitoring process, typical examples include filtering according to some
predefined criteria, or summarizing a group of events (i.e., computing the average); (iii)
distribution refers to the transmission of the events from the source to any interested parties;
(iv) finally, presentation typically involves some further processing so that the overwhelming
number of received events will be provided in a series of abstractions in order to enable an end-
user to draw conclusions about the operation of the monitored system. A presentation, typically
provided by a GUI application making use of visualization techniques, may either use a real-time
stream of events or a recorded trace usually retrieved from an archive. However, in the context
of grids, we generalize the last stage as consumption since the users of the monitoring
information are not necessarily humans and therefore visualization may not be involved.

General requirements

[1] listed a set of general requirements for monitoring systems, which may vary according to
specific cases and supporting underlying systems:
Scalability: due to the nature of grids, monitoring systems have to deal with increasing number of different types of resources, communications and end users.

Extensibility: allowing dynamic extension with respect to the resources and types of events generated by these resources, including event schema and producer/consumer protocols.

Data delivery models: monitoring information and consumer patterns may vary, with this regard, the monitoring system should be able to support various pull and push data delivery models.

Portability: in order to monitor specific types of resources and support their visibility to grids, monitoring systems should encompass great level of portability.

Security: monitoring systems should also be able to provide security services as access control, secure transport of monitoring information.

Freshness: monitoring information generally needs to be updated in real time and consumers also needs to estimate the freshness of events, thus, a high accuracy global notion of time is needed in grid monitoring systems.

**A Grid Monitoring Architecture**

In 2002, the Global Grid Forum (GFF) proposed the Grid Monitoring Architecture (GMA) [3] as a solution to facilitate the development of interoperable and high performance monitoring middleware. The main components of GMA are as follows in Figure 1:

![Figure 1: The GGF Grid Monitoring Architecture [3]](image)

A producer is a process implementing at least one producer Application Programming Interface (API) for providing events. A consumer is any process that receives events by using an implementation of at least one consumer API. A registry is a lookup service that allows producers to publish the event types they generate, and consumers to find out the events they are interested in. Additionally, a registry holds the details required for establishing
communication with registered parties (e.g., address, supported protocol bindings, security requirements). Even for systems with no notion of events, registries can be useful for producers and consumers discovering each other.

GMA defines three types of interactions between producers and consumers. Publish/subscribe refers to a three-phase interaction consisting of a subscription for a specific event type, a stream of events from a producer to a consumer, and a termination of the subscription. Both the establishment and the termination of a subscription can be initiated by any of the two parties. A query/response is an one-off interaction initiated by a consumer and followed by a single producer response containing one or more events. Finally, a notification can be sent by a producer to a consumer without any further interactions.

In addition to the three core components, the GMA defines a republisher (referred as compound component or intermediary) and a schema repository: A republisher is any single component implementing both producer and consumer interfaces for reasons such as filtering, aggregating, summarizing, broadcasting, and caching; A schema repository holds the event schema, that is, the collection of defined event types. If a system is to support an extensible event schema, such a repository must have an interface for dynamic and controlled addition, modification and removal of any custom event types.

The GMA, being an architecture, does not define implementation details such as employed data model, event schema, protocol bindings, registry engine and so on. Probably the most important feature of the GMA is the separation of the discovery and retrieval operations.

**Taxonomy**

in [1], a scope-oriented taxonomy of monitoring system is proposed. The categorization is mainly constructed based on a system's provision of GMA components: characteristics of a system's producers and republishers (Figure 2). For more detailed information, please refer to [1].

![Figure 2: The categories of the proposed taxonomy of monitoring systems in [1]. S means sensors (monitoring processes); C means customers; P means producers; R means republishers; H means hierarchical republishers. For more information, please refer to [1].](image-url)
The goal of this taxonomy is to provide a simple means to describe a monitoring system’s features with respect to: 1) its compliance with core GMA components; 2) main target of monitored entities; 3) whether a system can or has to operate on top of another system [1].

The monitoring systems discussed in this paper mainly belong to Level 2 or Level 3 according to this taxonomy. This type of systems have at least a second layer of republishers and are characterized in the fact that different types of functionalities are distributed over different hosts, may have an arbitrary structured hierarchical republishers and have the potential for scalability.

**Overview of several existing monitoring systems**

**Hawkeye**

Hawkeye [4] is a monitoring and management tool developed by the Condor group and designed to automate problem detection and software maintenance within a distributed system for clusters of computers. It utilizes two central ideas: using Condor classad language to identify resources (XML-encoded Condor’s classified advertisements, attribute-value pairs with optional use of expressions); and classad matchmaking to execute jobs based on attribute values of resources to identify problems in a pool.

![Figure 3: The architecture of Hawkeye [5]](image)

The architecture of Hawkeye includes four main parts: 1) pool, a set of computers, one of which serves as the centralized manager and others as monitoring agents; 2) manager (republisher), the manager is the leading computer in the pool, it collects and records monitoring information from other agents and receives queries about the status of any members in the pool; 3)
monitoring agents (producer), a set of distributed information service components that collects classads from each of its modules and integrate them into a single startd classad and they can also answer some queries about a particular module; 4) module, a sensor that shows resource information in a classess format.

In Hawkeye, every monitored node hosts a monitoring agent that periodically calculates a set of metrics, which reflect the host’s state, and communicates them to the manager. The central manager indexes the current state of nodes for fast query execution, and periodically stores it into a round robin database to maintain an archive. The monitoring information in the central manager can be accessed through command line utilities, and web and GUI frontends.

administrators can submit jobs to monitored nodes, either for unconditional execution or to be triggered as a response to specific events.

**Ganglia**

Ganglia [6] is an open source scalable distributed monitoring system for high-performance computing systems, primarily designed for computer clusters but also used in grid installations. It is based on a hierarchical design targeted at federations of clusters. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes (at the current time of writing).

![Figure 4: The architecture of Ganglia [7]](image)

At the cluster level, membership is determined with a broadcast, soft-state protocol (soft state means that membership must be periodically renewed by explicit messages or otherwise expires). All nodes have a multi-threaded daemon (Ganglia monitoring daemon) performing the following tasks: 1) Collecting and broadcasting External Data Representation (XDR) encoded
events from the local host; 2) Listening the broadcasts sent by other nodes and locally maintaining the cluster’s state; 3) Replying to consumer queries about any node in the local cluster, using XML encoded messages.

Given the above actions, a cluster’s status is replicated among all nodes, which act as producers, resulting in distribution of load and fault-tolerance, but also in high network and host intrusiveness.

An arbitrarily structured hierarchy of republishers (referred as Ganglia meta-daemons) periodically collect and aggregate events from lower level data sources, store them in round-robin databases, and provide them on demand to higher level republishers. Data sources may be either producers (on behalf of a cluster) or other republishers (on behalf of several clusters); in both cases an XML-encoding is employed.

Ganglia does not have a registry and therefore the location of producers and republishers must be known throughout-of-band means. The databases serve as archives and are also used by a web-based visualization application providing cluster- and node-level statistics. Simple command line utilities are provided for adding new event types and querying producers and republishers.

**Globus MDS**

The Monitoring and Discovery Service (MDS) [7, 8] is a grid information service, constituting the information infrastructure of the Globus toolkit. It adopts an extensible framework for managing static and dynamic information about the status of a computational grid and all its components. This service is built based on the Lightweight Directory Access Protocol (LDAP) [9].

![Architecture overview of MDS](image)

**Figure 5:** Architecture overview of MDS. Using the GRid Information protocol (GRIP), users can query aggregate directory services to discover relevant entities, and/or query information providers to obtain information about individual entities. Description services are normally hosted by a Grid entity directly, or by a front-end gateway serving the entity [7].

MDS is primarily used to address the resource selection problem [5], which is how a user can identify the host(s) that are running the applications. It mainly relies on two protocols: the Grid Information Protocol (GRIP), which allows query/response interactions and search operations;
the Grid Registration Protocol (GRRP), which is for maintaining soft-state registrations between MDS components.

The MDS framework (Figure 5) consists of information providers (sensors), Grid Resource Information Services (GRIS—producers) and Grid Index Information Services (GIIS—republishers). Producers collect events from information providers, either from a set of shell scripts or from loadable modules via an API. In addition, producers provide their events to republishers or to consumers using GRIP, and register themselves to one or more republishers using GRRP. Republishers form a hierarchy in which each node typically aggregates the information provided by lower level republishers (and producers in case of first level republishers). Republishers use GRIP and GRRP as part of the consumer and producer interfaces, though custom implementations could offer alternative producer interfaces (i.e., relational). Several roles may be served by republishers, including the provision of special purpose views (e.g. application-specific), organisation-level views and so on. Consumers may submit queries to either producers or republishers, or discover producers through republishers, in any case using GRIP.

R-GMA

The Relational Grid Information Services Research Group of the Global Grid Forum supports the view that the only difference between information and monitoring services is that the data involved in the latter have to be time stamped. Therefore, they built RGMA [10] as part of the EU Data Grid project: a framework which combines grid monitoring and information services based on the relational model.

![Architecture of R-GMA components](image)

Figure 6: Architecture of R-GMA components [5].

In RGMA, producers are distinguished in five different classes but we limit our discussion in database and stream producers, which are indicative of the main concepts. Database producers are employed for static data stored in databases, whereas stream producers or dynamic data
stored in memory resident circular buffers. New producers announce their relations (i.e., event types) using an SQL “create table” query, offer them via an SQL “insert” statement, and “drop” their tables when they cease to exist. A consumer is defined as an SQL “select” query. In order for a component to act as either a consumer or a producer, it has to instantiate a remote object (agent) and invoke methods from the appropriate (consumer or producer) API.

The global schema includes a core set of relations, while new relations can be dynamically created and dropped by producers as previously described. Republishers are defined as one or more SQL queries that provide a relational view on data received by producers or other republishers. The registry holds the relations and views provided by database producers, stream producers and republishers.

The registry includes the global schema and is centralized, while there are efforts for a distributed implementation. A mediator uses the information available in the registry and cooperates with consumers to dynamically construct query plans for queries that cannot be satisfied by a single relation (i.e., involving “joins” from several producers).

**JMS with NaradaBrokering**

Java Message Service (JMS) is a widely accepted industry standard that aims to simplify the effort needed for applications to use Message Oriented Middleware (MOM). JMS defines a set of Java APIs (Application Programming Interfaces), with which Java programmers can send and receive messages via MOM in a uniform and vendor-neutral way regardless of what the actual underlying middleware is [11].

Data are discovered by destination. There are two kinds of destinations: queue and topic. Data are wrapped in a JMS message. JMS supports two data dissemination modes: Point-To-Point (PTP) (brokerless) and publish/subscribe (brokered). Messages are delivered via a topic. JMS supports synchronous and asynchronous data transfers. For synchronous transfer, the subscriber can either poll or wait for the next message. For asynchronous delivery, the subscriber registers itself as a listening object, and the publisher will automatically send message by invoking a method of the subscriber (callback).

NaradaBrokering [12] is an open source, distributed messaging infrastructure. It is fully compliant with JMS. NaradaBrokering supports SOAP message, JMS message and complicated events. NaradaBrokering supports PTP and pub/sub data dissemination modes, and synchronous and asynchronous data transfer modes proposed by JMS. NaradaBrokering network map number of Web Services and Grid Services specifications, such as WS-Resource Framework (WSRF), WSNotification and WS-Eventing.

Several brokers can form a Broker Network Map (BNM). A specialized node called Broker Discovery Node (BDN) can discover new brokers. NaradaBrokering has a very efficient algorithm to find a shortest route to send the events to the destination in a BNM. NaradaBrokering is a very fast message dissemination middleware and it has been successfully adopted for
audio/video conferencing. NaradaBrokering supports a number of underlying data transport protocols, including blocking and non-blocking TCP, UDP, multicast, SSL, HTTP, HTTPS and Parallel TCP streams.

In [11], NaradaBrokering has shown excellent real-time performance, high throughput and good scalability.

**Conclusion and Summary**

This paper introduced basic concepts and background of grid monitoring systems, outlined by the Grid Monitoring Architecture of the Global Grid Forum whose components are mapped to phases of different types of distributed systems monitoring. A taxonomy that was defined to allow the classification of monitoring systems with various respects, such as compliance to core GMA components, architecture extensibility and etc., was also introduced. Then, several existing popular monitoring systems were reviewed.

With detailed inspection of different monitoring systems, I've gained a profound and deep understanding of the requirements, constraints and crucial issues in monitoring systems. This largely facilitate my ability and understanding on expanding my term project.

**Acknowledgement**

I would like to thank Prof. Qiu for her instructions and guidance in the course of Distributed Systems, and on the topic of grid monitoring systems. I would also like to thank Yipan Pan and Erkang You, who are in the same team with me, for discussion of confusing issues in this topic.
Reference