A Decentralized, Scalable, and Autonomous Grid Monitoring System

Laurent Baduel [1] and Satoshi Matsuoka [1,2]

[1] Tokyo Institute of Technology,
2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan
[2] National Institut of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
baduel@smg.is.titech.ac.jp matsu@is.titech.ac.jp

Abstract. Grid monitoring systems collect a substantial amount of information on the infrastructure's status in order to perform various tasks, more commonly to provide a better use of the grid's entities. Modern computational and data grids have become very complex by their size, their heterogeneity, their interconnection. Monitoring systems as any other grid's tools have to adapt to this evolution.
In this paper we present a decentralized, scalable, and autonomous grid monitoring system able to tackle the growths of scale and complexity. System's components communications are hierarchically organized on a peer-to-peer overlay network. Fresh information is efficiently propagated thanks to an *directed* gossip protocol that limits the number of message. Automation of key management operations eases system administration and maintenance. This approach provides scalability and adaptability. The main properties of our application are presented and discussed. Performance measurements confirm the efficiency of our system.

1 Introduction

Grid platforms with their ever-growing communication infrastructures and computing applications become larger and larger, which results in an exponential complexity in their engineering and maintenance operations. To efficiently handle such large and complex systems monitoring is necessary. By providing a global view of the system monitoring tools allow identifying performance problems and assisting in resources scheduling. Modern large scale systems do not allow anymore centralized organizations, with hand deployment, configuration, and administration. Automation of key operations must be introduced in such systems to free the administrators and programmers of many tiresome tasks.

After presenting related work, we propose a grid monitoring system built to address these challenges. It provides a scalable and portable monitoring of a wide range of entities connected in distributed systems. This decentralized tool achieves its communications thanks to a peer-to-peer overlay network. Peer-to-peer has become a popular way to communicate on grid thanks to its scalability,

its decentralization, and its resistance to faults. It has already proved their efficiency in many aspects of distributed applications such as embarrassingly parallel computing, persistent and scalable storage, and especially in file sharing and dissemination. We use a gossip protocol to quickly spread information in the entire system. Components of the system are organized as a directed acyclic graph to guide information from their source to the bases storing the entire system state. This, combined to over-aged information filtering, reduces the amount of exchanged messages. Finally we have deployed a first implementation on a real grid and evaluated performance of our system.

In summary the contributions of this article are *(1)* the details of the conception of a decentralized and scalable grid monitoring system in which components are hierarchically organized through a directed acyclic graph on a peer-to-peer overlay network that provides a valuable communication layer thanks to a gossip multicast protocol, assures good performance on grids, and allows self-management of the system; *(2)* a performance evaluation of this system driven in a real large-sized grid. Speed of information dissemination, age of recorded information, impact of messages limitation, and dynamic adaptability are examined and discussed. This paper rather focuses on the fast and decentralized dissemination of information than on other aspects of monitoring such as sensors implementation or database organization.

The rest of this article is organized as follow: Section 2 describes the general properties of a monitoring system and insists on the specific requirements for grid environments. Then it presents related work. Section 3 introduces the architecture for monitoring system on which we based our implementation. Section 4 presents our original communication scheme based on peer-to-peer and what we name a *directed* gossip protocol. Then Section 5 details self-management mechanisms introduced in the system to help scalability and maintenance. Section 6 details the implementation and presents performance evaluations of the application. Finally Section 7 concludes the article and presents expected future works.

2 Grid Monitoring Systems

The activity of measuring significant resources parameters allows analyzing the usage, the behavior, and the performance of a cluster or a grid. It also provides Grid monitoring systems to collect a substantial amount of information on the infrastructure's status in order to perform various tasks, more commonly to provide a better use of the grid's entities.

Monitoring a system consists of observing events and communicating them to who are interested in that information. There are commonly two systems on a grid. According to [1], a *grid monitoring system* manages rapidly changing status information, such as the load of a CPU or the throughput of a network link. The high dynamicity of data that a grid monitoring system must handle makes it different from a *grid information system* which handles more static data, for instance the hardware configuration of a node. Although grid infrastructures can benefit a lot from a unified system that handles both roles. The

global knowledge provided by a unified grid monitoring system helps resource scheduling, allocation and usage: reservation tools can be plugged in order to distribute resources and guarantee quality of service.

Monitoring dedicated to grids is subject to a growing interest. The recent large grids do not support anymore efficiently the existing monitoring tools. Most of existing solutions are adaptations of cluster oriented monitoring tools, and then lay to scalability and robustness issues. As detailed further the Network Weather Service is built around a centralized controller, and the Globus Monitoring and Discovery System suffers performance and scalability issues due to its LDAP architecture. On the contrary, the grid monitoring system we propose is adapted to the grid thanks to its scalability and fault tolerance ability, and also thank to its autonomous management.

The rest of this section presents a survey about grid monitoring systems and their potential autonomous mechanisms for configuration and adaptation.

2.1 Network Weather Service (NWS)

The Network Weather Service [2] is a distributed system that periodically monitors and dynamically forecasts the performance that various network and computational resources can deliver over a given time interval. The components of a NWS resource monitoring system are: a *name server*: a centralized controller that keeps a registry of all components and monitoring activities; *sensors* that produce resource observations; *memories* that store resource observations; and *forecasters* that process resource observations. Limits of the NWS architecture come from the presence of the name server. It introduces bottleneck, single point of failure, and does not embed security mechanism. Moreover the connections between sensors and the name server are manually managed by the administrator before starting the system. NWS is hardly applicable to a grid scale, mostly because of the absence of a real database.

2.2 Globus Monitoring and Discovery System (MDS)

The Monitoring and Discovery System [3] is the information services component of the Globus Toolkit and provides information about the available resources on the grid and their status. MDS is based on the Lightweight Directory Access Protocol (LDAP). The distributed nature of the LDAP architecture is appealing: information is organized hierarchically, and the resulting tree might be distributed over different servers. In a grid perspective, the hierarchy often reflects the organization of the grid: from continental networks to national networks to local networks. Leaves are single resource, like cluster of computers, single computer or storage element.

However, the LDAP architecture is appropriate to store static information, like the number of processors in a cluster, or the size of a disk partition. When data are more frequently changing, the LDAP architecture is a wrong choice: it is unsuitable to support frequent write operations. Indeed the LDAP Client Update Protocol is based on the assumption that “data changes, renames, and

deletions of large subtrees are very infrequent" [4]. Under that condition LDAP architecture suffers serious performance and scalability problems.

2.3 Relational Grid Monitoring Architecture (R-GMA)

The Structured Query Language (SQL) allows manipulation of a relational database. Several implementations of this database are designed for extremely demanding applications. They offer a good compromise between the distributiveness, and the cost of query operations. In particular the database can be replicated in order to improve scalability and fault tolerance. The R-GMA [5] was developed to address those problems. The scalability of the architecture is improved by introducing components that combine data from the database and cache the results. R-GMA is an implementation of the GGF's GMA model (see Section 3) and is now developed as part of the *Enabling Grids for E-science in Europe* (EGEE) project. A strength of R-GMA is its ability to support queries which combine information across objects of different class (a *join* operation).

R-GMA focuses on the way data are stored, i.e. the relational database that is actually implemented by a virtual database distributed and accessible by hidden component named *mediator*. Concerns of easy deployment and adaptability are not the main issues of R-GMA.

2.4 Ganglia

Ganglia [6] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It is based on a hierarchical design targeted at federations of clusters. It widely uses technologies such as XML for data representation, XDR[3] [7] for compact and portable data transport, and RRDtool[4] [8] for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve low per-node overheads and high concurrency. The implementation has been ported to a large 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.

Ganglia's implementation consists in the collaboration of two daemons. `gmond` running on every node provides local network monitoring by recording and communicating with multicast primitives. `gmeta` offers local networks federation by taking in charge the communication with other remote `gmeta` daemon. Ganglia is efficient and easy to deploy on small or medium sized clusters. However the copy on each node of the entire local network status may lead to scalability problem. Moreover some equipment with limited resources that may be present in a grid (such as personal digital assistant or various scientific equipments) may not be able to host a `gmond` daemon and its database. Finally the deployment and maintenance of double system `gmond`/`gmeta` require expertise from the administrators.

[3] XDR stands for eXternal Data Representation.
[4] RRD stands for Round Robin Database.

3 The Grid Monitoring Architecture

The Global Grid Forum (now known as Open Grid Forum) has introduced the Grid Monitoring Architecture (GMA) [9] which offers scalability and flexibility required by a grid monitoring system. This architecture simply identifies three kinds of component:

- the *producers* retrieve various information of a device and make them available to other GMA components; for instance a sensor reporting the CPU load of a computing node.
- the *consumers* request monitored information for which they have interest; for instance a resource broker that wants to locate suitable computing nodes.
- the *directory service* supports information publication and discovery of components as well as monitored information. A directory service is a place where producers advertise their data, and consumers advertise their needs.

In addition to those three basic components, the GMA lets room for *intermediate components*. Those components consist of both a consumer and a producer. They allow aggregation, filtering, forwarding, or broadcasting of the information received by other producers. Often monitoring tools use *aggregator* components. Those components help at the scalability of the system by avoiding communication bottlenecks between the producers and the directory service: the number of exchanged messages is reduced.

Scalability is assured by the separation of the publication, discovery, and query tasks. The GMA does not specify the way the components communicate among each others (message content or network protocol) or the format used by the directory service to store the information. Because of its flexibility and scalability we based our autonomic grid monitoring system on the GMA.

4 Decentralized and Scalable Communication Scheme

The priorities when building a grid tool is to ensure scalability and fault tolerance. Peer-to-peer networks and Gossip protocols scale very well thanks to their very decentralized organization. The decentralization of those systems also provides a good tolerance to faults by providing alternate paths of communication and replication of data.

4.1 A peer-to-peer architecture

Grids and peer-to-peer have both an identical approach to the accomplishment of their goal: the use of overlay structures. However we can make an important distinction between the approaches of grids and peer-to-peer:

- *Grids* provide a large amount of services to moderated-sized communities with a generally high quality of service. Because of their hierarchical and static organizations grids are vulnerable to faults.

– In contrast *peer-to-peer systems* provide limited and specialized services to a very large amount of users. Peer-to-peer systems are strongly resistant to failure as a whole, but they do not provide a high quality of service. This limitation results from the fact the services are of nature as being provided in mass, and thus lead to various problems such as the node volatility, the best effort performance provided by the Internet, etc.

According to [10] the complementary nature of the strengths and weakness of the two approaches suggests that the interests of the two communities are likely to grow closer to each other. The main goals of current grids architecture are to increase their scalability and to provide a better handling of failures. Symmetrically peer-to-peer systems aim at improving the range of their services.

Current grids' component communications are fragile mainly because of static organizations and the absence of alternate paths of communication. They need to be sustained by suitable overlays that are scalable and fault resilient. Peer-to-peer libraries are the first stone to answers those concerns. As mentioned in Section 2 the main issue of current monitoring systems is their centralized and static organization. Centralization introduces weakness in a system because of the single point of failure problem and often leads to scalability issues by introducing bottleneck. A static organization makes the maintenance of a large system specially challenging for the administrators.

As the GMA model does not specify communication mechanism for data transfer, we decided to use peer-to-peer oriented communications. Figure 1 presents the structure of a monitoring system using peer-to-peer interconnections. The producers are in contact with aggregators. With regard to fast spreading of the information and fault-tolerance, every producer keeps a contact with several aggregators (more details below). Similarly, the aggregators communicate with the directory service. The directory service is made of components that store information about the producers and the values they have produced. We name those components *global storages*. For the same reasons as one producer keeps contact with several aggregators, one aggregator keeps contact with several global storages. Finally the consumers contact the directory service (i.e. the global storages) to get the information it is interested in. A consumer may ask for the location of the producer of a particular kind of event, and then contact the producer in order to be directly notified of the information produced. A consumer may also ask for the values stored in the global storage.

By introducing distribution, and indirect communication through a peer-to-peer network, we have to be careful regarding performance. The major challenge of our grid monitoring system is to disseminate the information coming from the producers as fast as possible to the global storages. Outdated information is useless since it is no more relevant of the current node's status that may have radically changed. The monitoring system has to be low resources consuming. Monitoring activities must not impact the execution of other applications in the grid. CPU and bandwidth consumptions have to be significantly low.

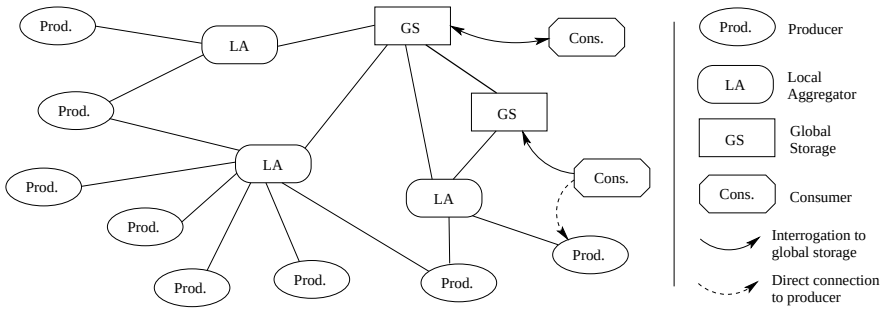

Fig. 1. Infrastructure

4.2 A Directed Gossip Protocol

The growth of large scale distributed applications is driving the need for scalable and reliable communications. Many network-level reliable multicast protocols are based on IP Multicast that is not widely deployed, resulting in a need for application-level broadcast protocol.

The communication model of our monitoring has to be scalable and low resource consuming. The system must spread the data as fast as possible in the entire system to ensure reactivity. On the other hand it is necessary to save bandwidth resource, for instance by avoiding multiple sending of the same data. We can not accept any component of the system emits several times the same info. Our solution is to use a *gossip protocol* [11] in order to broadcast the information. A pure gossip protocol takes place in rounds where in each round a participating process selects randomly another process and share information with it. In "push" gossip the process that initiates the gossip communicate its information to the selected target. It has been shown that in a group of n machines if one machine starts out with a new piece of information it takes $O(log\ n)$ rounds for every machine to become aware with that information [12].

The interest for using Gossip protocols on top peer-to-peer networks comes from two main points. *(1)* As mentioned before gossip protocols are very efficient to propagate information in large systems, like peer-to-peer systems. *(2)* Similarly to peer-to-peer systems, gossip protocols are totally decentralized and thus perfectly scalable.

In our system, we slightly modified the gossip protocol. All the nodes need not to get the information, but only the global storages. Thereby, the gossiping is performed only from producers to aggregators, from aggregators to global storages, and from global storages to global storages. This hierarchical structure, that is a directed acyclic graph from producers to global storages, provides a double benefit: by reducing the number of nodes to be informed it increases the scalability and reduces time and amount of messages required for all those nodes to be informed. As explained before, a challenge of our system is to spread information from one producer to all the global storages in the shortest time.

5 An Autonomous Monitoring System

Autonomic computing designates complex self-managed systems in which elements interact with each others in order to organize themselves and obtain a satisfactory general behavior. Modern large-scale computing systems widely distributed across multiple administrative domains have reached an unbelievable complexity. Autonomic computing offers to solve this problem through a smart and increased automation, freeing system administrators of many burdensome activities. Indeed, in modern large scale and distributed environments, and particularly grids, main difficulties has moved from application programming to configuration and maintenance.

As presented in [13] self-management falls in four categories: the *self-deployment* configures automatically the system; the *self-optimization* tunes the system to obtain the best performance; the *self-healing* detects, diagnostics, and fixes malfunctions; the *self-protection* defends the system against malicious attacks and cascading failures. In [14] we have presented the interests of using autonomic computing and how automatic behaviors can be implemented thanks to a peer-to-peer overlay network. The implementation of the four aspects of self-managed systems is based on the properties of peer-to-peer libraries, particularly the ability to dynamically find resources.

Self-configuration When a new component enters into the system it has firstly to connect to the peer-to-peer network, then it may automatically retrieve some codes from other components in activity in the system, and finally it locates the components with which it will communicate. We will illustrate this procedure thanks to the producer component. A new producer firstly joins the peer-to-peer overlay network by searching for peers on the local network (broadcast) or by contacting a *bootstrap group* composed of nodes expected to be always online. Then as sensor's implementation for computational nodes may be quite similar, the producer may download the sensor's code from another producer already in activity. Finally the producer establishes contacts with a number of aggregators defined by the administrator. Those aggregators are selected regarding to the *round trip time* (RTT) that is the metric we use to express distance. A panel of aggregators with all RTTs is kept in order to uniformly spread information to "close" and "far" area of the network.

Self-optimization Our policy for self-optimization is the dynamic dimensioning of the system size (number of components) according on the variation of its load. The size should be increase when the system gets overloaded and decrease in case of underload. In our system, the aggregators (and the global storages) have to adapt to the number of producer. Those components becomes overloaded when they receive more messages than they can handle or when they use more resources than they are supposed to do. An overloaded component takes the initiative to redirect some its producers to another component or, if no acceptable component is found, to create a new component. We consider an aggregator underloaded if during three successive evaluations the aggregator has received less

than a fixed number of messages. The three evaluations allows smoothing the adaptation of the system and avoiding over-reactivity. We may also consider to observe the throughput, memory, or CPU usage of the component. Overtaking a certain threshold actions may be taken. Another aspect of the self-optimization may consist in regular observations of the system's performance. If the performance does not obey to constraints set by the administrator, the system may decide to take some measure. An example can be the momentary reduction of a communication frequency in order to eliminate network congestions.

Self-healing A basic policy for self-healing can consist of the automatic detection of component's failure and the dynamic replacement of the failing component with the guaranty the system remains globally coherent. When an aggregator or a global storage fails, the failure is detected by the producers. If a producer does not succeed to communicate with a component, a failure is suspected. The producer broadcasts a message in the peer-to-peer network indicating the suspicion of a component failure, with the identity of the component. If one or more other producers confirm the failure of this component, an election mechanism decides where to restart the failed component. The election mechanism basically looks by a peer-to-peer search for the free host providing the more bandwidth, memory, and CPU power. In the case a failed component is not detected, the self-optimization mechanism may observe the overload of one component and decide the creation of a new one (c.f. self-optimization mechanism).

Self-protection To make the system resistant to cascade failure we utilize the robust communication mechanism of some peer-to-peer networks that are able to re-route messages if a communication link falls and address peers that may have been disconnected during a while. We also rely on the replication of all aggregator and global storage components. The information aged x in one of those components is also present in another component with an age y, x and y beeing very similar. De facto there is no need to actively maintain consistency between those components.

6 Experimentations

This section presents some details of the implementation such as the peer-to-peer library we used and its particular features. Then it introduced the grid platform on which we performed experimentations. Relevant performance measurements are presented and discussed, proving the qualities of our monitoring in term of efficiency, scalability, and adaptability.

6.1 Implementation Details

JXTA [15] is a set of open and generalized peer-to-peer protocols that allow any connected device on the network to communicate and collaborate as peers. The JXTA protocols are independent of any programming language, and multiple

implementations (called bindings) exist for different environments thus it is well adapted to the heterogeneous environments that compose a grid. JXTA has its own independent naming and addressing mechanism: a peer can move around the network, changes its transport protocol and network addresses, even being temporarily disconnected, and still addressable by other peers. This capability allows being resistant to the volatility of nodes in a grid. Moreover JXTA provides secure communication and access to resources following a role-based trust model. It provides also the possibility to cross firewall under the condition peers support HTTP. *Rendezvous peers* maintains a cache of advertissements and forward discovery requests to help other peers discover resources. Each rendezvous peer is like any other peer but keeps a list of other known rendezvous peers and a list of the peers that are using it as a rendezvous. *Relay peers* maintains information about the paths to other peers and routes messages to peers. They also forward messages on the behalf of peers that cannot directly address another peer (NAT environments). We configured those peers in order to connect peers from different sub-networks that compose a grid.

JXTA provides three levels of communication: *(1)* the *endpoint service* providing asynchronous, unidirectional, and unreliable static point-to-point communications; *(2)* the *pipes* providing asynchronous, unidirectional, and unreliable dynamic communications; and *(3)* the *JXTA sockets* adding reliability. We base our system on *BiDiPipes* that are pipes enhanced with bidirectional and (optinal) reliable communication. As exposed in [16] pipes provides better performance than sockets, moreover C/C++ implementation of JXTA does not provide the socket API. Indeed sockets are not part of the core specification of JXTA. In one direction bidipipes transport information from producer to aggregator, then to global storage; in the other direction the pipe is used to relay control signals for autonomic purposes. Control signals are for instance *RTT Request* and *Reply*, *Ping*, *Component failure suspicion*, *New component available* (for acquaintenances redirection), etc.

6.2 Performance Measures and Optimization

The *Grid'5000* project aims at building an experimental Grid platform gathering 9 sites geographically distributed in France combining up to 5000 processors. The plans are to assemble a physical platform featuring 16 clusters, each with an hundred to a thousand computers, connected by *Renater* the French Education and Research Network. Most sites are connected to Renater at 10 Gb/s (few of them still at 2.5 Gb/s). This high collaborative research effort is funded by the French ministry of education and research, INRIA, CNRS, the universities of all sites and several regional councils. Clusters composing the Grid'5000 platform are heterogeneous. To lead our experiments we used from 200 to 620 nodes (the number of nodes we used did not impact on the performance we observed cause each node was able to run the entire set of components we assigned to it). Multiple components were instanced on a same node, but for scalability reasons they were hosted in a unique Java Virtual Machine process. The collected information was limited to CPU and memory load.

The major point of interest in our system is to know how fresh the information in the global storages is, so our first experiment consists of observing the age of the information in a global storage. Each component gossips every 30 seconds. Our system being totally asynchronous we have no concern about possible time divergence between components: each component acts autonomously. At the beginning of the experiment the graph of components is already formed as follows. The ratios are arbitrarily set to 1 local aggregator for 100 producers and 1 global storage for 10 aggregators. Each producer is linked to 10 aggregators, each local aggregator is linked to 10 global storages, and each global storage is linked to 10 other global storages. The system ran one hour before being observed. Figure 2 presents the age of information in the global storages sorted by age. The instant of the observation is *age 0*.

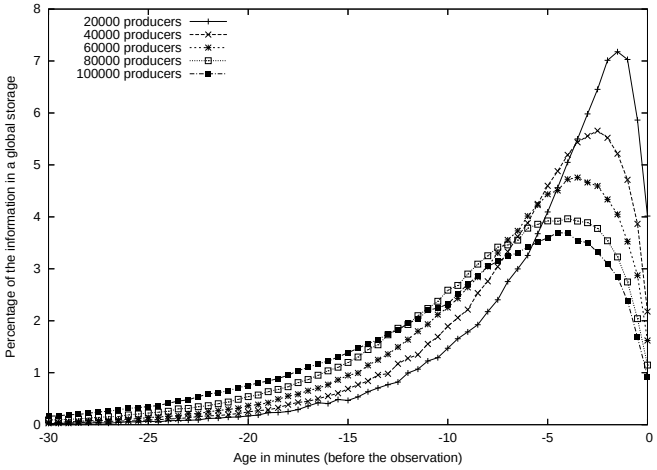

Fig. 2. Distribution of information by age

Information in global storage is recent. The curves show peaks near the time of the observation. In a 20,000 producers system the peak is at -1.5 minute with 7.179% of the total information. In a 100,000 producers system the highest amount of information is aged 4.5 minutes with 3.697%. The average age of information is given in the following table (the size of the system is exposed in producers and age is exposed in minutes).

System size	20K	40K	60K	80K	100K
Average age	5.226	6.249	7.292	8.537	9.753

From Figure 2 we observe that half of the information, the older part, may be considered as useless. Indeed it is unnecessary to relay "old" information. Moreover it allows saving 50% of aggregators and storages' messages size. Figure 3 presents the age of information in the global storages sorted by age. In

this experiment aggregators and storages only gossips the younger half of their information.

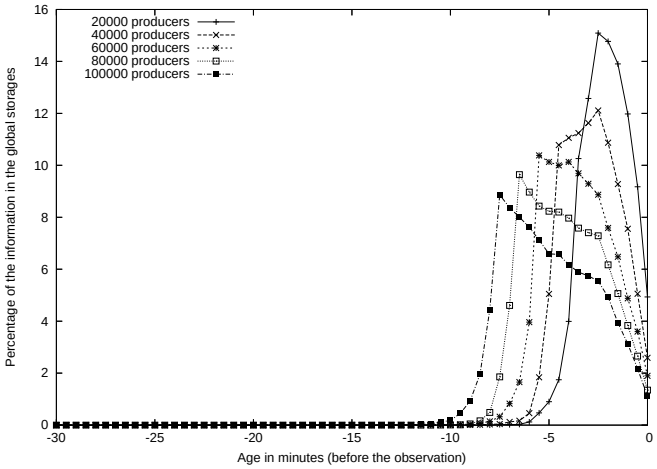

Fig. 3. Distribution of information by age with filtering of old information (50%)

We observe a reduction of the amount of older information, resulting in a higher percentage of younger information. The message reduction of course does not improve the efficiency of the system, it role is to reduce by almost half the amount of data transferred on the network.

Efficiency of the system seems to be very slightly impacted by the reduction of messages. Thanks to that observation we can imagine a dynamic adaptation of the age of information communicated each iteration in order to obtain the best trade-off between efficiency and bandwidth consumption. The following table shows the average age of information in the storage is reduced thanks to the reduction of old information transmissions.

System size	20K	40K	60K	80K	100K
Average age	2.105	2.846	3.539	4.171	4.902

The second experiment consists of observing the time required by a new producer entering the system to be known by all the global storages. It means the propagation time required by all global storages to receive at least one information from the new producer. Figure 4 presents the results. The curves plot the average percentage of global storages in the system that already got at least one information from a new producer. The new producer is introduced at time 0 into a system running for several minutes.

In less than 4 minutes 92.0% of the global storages of a 100,001 producers system (i.e. 100 global storages according to the conditions of our experiment) have been informed of the new producers. Considering the abscence of point of

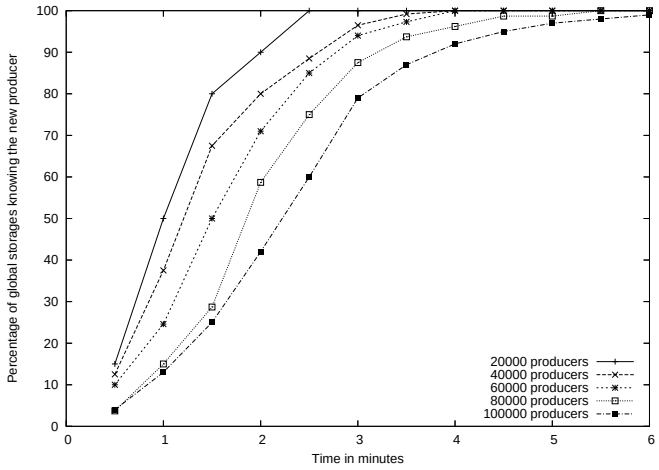

Fig. 4. Knowledge of a new producer in the entire system (i.e. all global storages)

centralization we consider the spreading of new information is very fast thanks to the gossip protocol. A first reason is that during one round the information is actually relayed three times: first from the producer to an aggregator, then in an aggregated form with other information from the aggregator to a global storage, and finally once again from the global storage to another global storage. Another reason is that our directed gossip protocol does not "broadcast" but "multicast" since only a subgroup of elements (the global storages) finally receives the information through the hierarchical architecture.

In our last experiment, we observe the adaptation of the number of aggregator regarding variation of producers in the system. From a 100 producers system, we add 10 new producers every 30 seconds, until the system reaches a size of 1,000 producers. Then number of producers remains stable for 7.5 minutes before we remove them gradually, 10 every 30 seconds. Producers cleanly exit: they notify their aggregators when they leave the system. During this experiment, we arbitrarily chose to declare an aggregator overloaded when it receives more than 100 information from the producers during three successive iterations. Similarly it is considered as underloaded if during three of its iterations it receives less than 80 information.

Figure 5 presents the number of aggregators and producers (by hundred), along the experiment. The curves remain close; it attests the quantity of aggregators increases in the same proportion as the producers. The ratio between aggregators and producers stays approximately the same over the time (average: 1 aggregator for 82.79 producers). The very fast growth of aggregators in the first minutes of the experiment results from the fact the system was already overloaded in its initial settings.

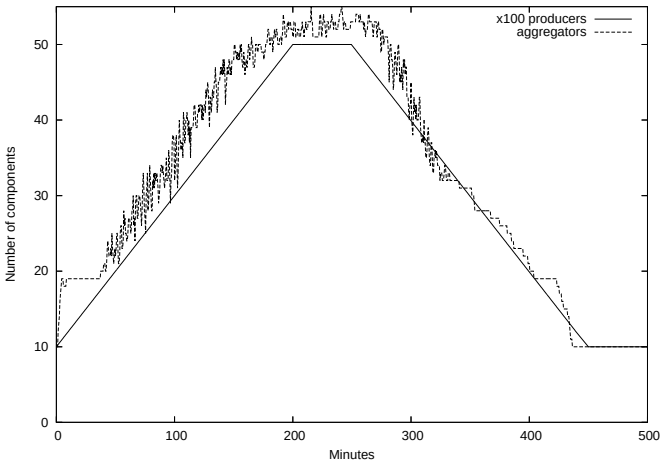

Fig. 5. Adaptation to the system size

7 Conclusion and perspectives

We presented a decentralized, scalable, and autonomous grid monitoring system able to tackle the growths of scale and complexity. The system's components are hierarchically organized on a peer-to-peer overlay network. A directed gossip protocol ensures efficient propagation of fresh information. Overproduction of messages is avoided by the organization of communication paths in DAG and by filtering on information depending on its age. Autonomic behaviors guarantee easy deployment and adaptability at runtime. The implementation was detailed as well as performance measurements that confirm the efficiency of our system. At this time the main advantages of our system are *(1)* its ability to be quickly and easily deploy; *(2)* its capacity to self-adapt; and *(3)* its fault-tolerance by information replication and alternative paths of communication. On the other hand, a drawback is to provide only "statistically" good quality of service: small probabilities remain to return a bad (too old) information.

To address this problem we will organize the global storages into a peer-group and access them all as a single distributed entity. The point is to make them communicating with each others to collaborate answering a consumer request. One of the global storages necessarily contains the freshest value of searched information, so by comparing the age of their information they can return the freshest one. This *peer-group service* guarantees to obtain the freshest information and make possible to distribute the global storages' information base leading to an improved scalability and a new reduction of messages on the network.

Our next consideration is to integrate our monitoring service with the Open Grid Software Architecture (OGSA) [17] which is more likely to become the standard way to access grid services. Also in order to solve more complex requests involving several fields of search, as R-GMA does, we think to deploy relational database in global storages.

Finally the security concerns must be taken in consideration. Our current implementation only rely on the security mechanism provided by the communication layer: the peer-to-peer library. A secure system must be aware of authentication, trust, and data integrity in order to prevent malicious attacks.

References

1. Zanikolas, S., Sakellariou, R.: A Taxonomy of Grid Monitoring Systems. Future Generation Computer Systems **21**(1) (January 2005) 163–188
2. Wolski, R., Spring, N., Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Journal of Future Generation Computing Systems **15**(5–6) (October 1999) 757–758
3. Czajkowskiy, K., Fitzgeraldz, S., Foster, I., Kesselmany, C.: Grid Information Services for Distributed Resource Sharing. In: Proceedings of the 10th international symposium on High Performance Distributed Computing, San Francisco, California, USA (August 2001) 181–194
4. Megginson, R., Smith, M., Natkovich, O., Parham, J.: Lightweight Directory Access Protocol: Client Update Protocol. RFC3928, IETF (October 2004)
5. Cooke, A.W., al.: The Relational Grid Monitoring Architecture: Mediating Information about the Grid. Journal of Grid Computing **2**(4) (December 2004) 323–339
6. Massie, M.L., Chun, B.N., Culler, D.E.: The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. Journal of Parallel Computing **30**(7) (July 2004) 817–840
7. Sun Microsystems Inc.: XDR: External Data Representation Standard. RFC1014, IETF (June 1987)
8. Oetiker, T., al.: RDDtool, logging and graphing. http://oss.oetiker.ch/rrdtool/
9. Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., Wolski, R.: A Grid Monitoring Architecture. Technical report, GGF (January 2002)
10. Foster, I., Iamnitchi, A.: On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, California, USA (February 2003)
11. Jenkins, K., Hopkinson, K., Birman, K.: A Gossip Protocol for Subgroup Multicast. In: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS), Phoenix, Arizona, USA (April 2001)
12. Pittel, B.: On Spreading a Rumor. SIAM Journal of Applied Mathematics **47**(1) (March 1987) 213–223
13. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer **36**(1) (January 2003) 41–52
14. Baduel, L., Matsuoka, S.: A Peer-to-Peer Infrastructure for Autonomous Grid Monitoring. In: Proceedings of the third International Workshop on Hot Topics in Peer-to-Peer Systems, at IPDPS, Long Beach, California, USA (March 2007)
15. Gong, L.: Project JXTA: A Technology Overview. Technical report, Sun Microsystem, Inc. (October 2002)
16. Antoniu, G., Jan, M., Noblet, D.A.: Enabling the P2P JXTA Platform for High-Performance Networking Grid Infrastructures. In: Proceedings of High Performance Computing and Communications (HPCC). Volume 3276 of LNCS., Sorrento, Italy (September 2005) 429–439
17. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: The physiology of the Grid. In: Grid Computing, Making the Global Infrastructure a Reality. John Wiley & Sons (2003) 217–249