A Peer-to-Peer Infrastructure for Autonomous Grid Monitoring

Laurent Baduel and Satoshi Matsuoka

TOKYO INSTITUTE OF TECHNOLOGY, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan

Email: baduel@smg.is.titech.ac.jp and matsu@is.titech.ac.jp

Abstract

Modern grids have become very complex by their size and their heterogeneity. It makes the deployment and maintenance of systems a difficult task requiring lots of efforts from administrators and programmers. Our goal is to investigate the concepts that underlie autonomic computing systems, especially for grid environment. We believe that peer-to-peer overlay networks are a valuable basis to support some of the main issues of autonomic computing in the particular case of grids.

This article presents the construction of an autonomous, decentralized, scalable, and efficient grid monitoring system. The components of this application negotiate through a peer-to-peer network in order to provide autonomic behaviors and exchange data. We present a solution based on a gossip broadcast protocol upon a hierarchical, directed, and acyclic graph to rapidly diffuse information in the system while limiting the number of messages. The software architecture is detailed, and then the first results of its performance are presented and analyzed.

1 Introduction

Today's IT systems with its ever-growing communication infrastructures and computing applications are becoming more and more large in scale, which results in exponential complexity in their engineering, operation, and maintenance. 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 in order to free the administrators and programmers of many tiresome tasks.

We propose to address those challenges using an autonomic computing architecture where communications are achieved using a peer-to-peer overlay network. Peer-to-peer systems have already proved their efficiency in many aspects of distributed applications: embarrassingly parallel computing, SETI@Home [4]; persistent and scalable storage, OceanStore [14]; and especially in file sharing and dissemination, Napster [2], eMule [3], BitTorrent [1], etc. We believe peer-to-peer networks also offer a valuable communication mechanism in grid environment thanks to scalability, its resistance to faults, and its ability to cover different administrative domains.

The contributions of this article are *(1)* the description of the usage of hierarchically organized peer-to-peer overlay networks to provide an adapted communication layer for the self-management of autonomic system in grids, and *(2)* the conception of a decentralized and scalable grid monitoring tool according to our principles. This application demonstrates the viability of our approach. To overcome the problem of fast information dissemination in a peer-to-peer network, a gossip multicast protocol spreads the sensors' information into the system. Efficiency of the protocol is improved by the organization of the monitor's components in a directed acyclic graph. Time and number of exchanged messages is reduced. The performance of this monitoring tool is presented and discussed.

The rest of this article is organized as follow: Section 2 presents the autonomic systems and peer-to-peer overlay networks to provide a valuable architecture for autonomous operations in a grid. Section 3 describes the conception of a grid monitoring system built following our principles, and present early results. In Section 4 we discuss related works. Finally, Section 5 concludes and presents our perspectives.

2 Context

Autonomic computing and peer-to-peer overlay network can be combined to offer an efficient framework to build large grid applications.

2.1 Autonomic systems

Autonomic computing designates large and complex self-managed systems in which elements (themselves self-managed) interact with each others in order to organize themselves and obtain a satisfactory general behavior. Modern large-scale computing systems widely distributed across

1-4244-0910-1/07/$20.00

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.

As presented in [12] self-management falls in four categories: *(1)* the *self-deployment* configures automatically the system; *(2)* the *self-optimization* tunes the system to obtain the best performance; *(3)* the *self-healing* detects, diagnostics, and fixes malfunctions; *(4)* the *self-protection* defends the system against malicious attacks and cascading failures.

Each element of an autonomic system is responsible for managing its own internal state and behavior and for managing its interactions with an environment that consists largely of signals and messages from other elements and the external world. In the first step of its life cycle, a component has to enter into the system by registering itself into a directory service and request connections to services providers. Then it becomes operational and enters into its main activity. In addition to a regular functional behavior, autonomic management requires a component to provide monitoring in order to observe and so be able to optimize its activity.

The communication between components in a self-managed system becomes extremely important, not only regarding the messages exchanged to accomplish the functional task of the component but also regarding the messages exchanged for self-management purposes. The autonomic components continuously interact together from the time of their configuration and deployment to the end of their activity. Such a behavior induces a concern about the communication mechanisms between components.

2.2 A peer-to-peer architecture

The concept of autonomic computing has appeared in order to help administrator to manage large and sophisticated system. Grids fall precisely in this category. Grids impose specific constraints and requirements especially about communication. One of the latest and popular ways to achieve communication in grid is to use peer-to-peer networks.

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.

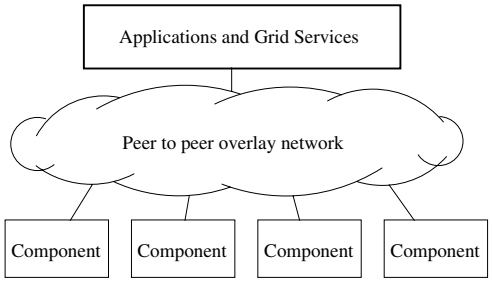

Figure 1. System architecture

According to [8] 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 services they propose.

The communication mechanisms required by the self-management of an autonomic component have to fit with the environment in which it evolves. Current grids' intern system component communication infrastructures are fragile mainly because of their static organization 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 of communication in grid environment. With addition of specific behaviors such as autonomic management, peer-to-peer networks become an effective solution. Figure 1 presents the way we propose to build application for grids: a peer-to-peer layer achieves the communication between autonomic components, and so allows them to interact efficiently on a grid.

3 A grid monitoring system

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 [20], 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 in Section 4, 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.

3.1 The Grid Monitoring Architecture

The Global Grid Forum has introduced the Grid Monitoring Architecture (GMA) [16] which offers scalability and flexibility required by a grid monitoring system. This architecture simply identifies three kinds of component as shown in Figure 2:

- 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 of the GMA 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.

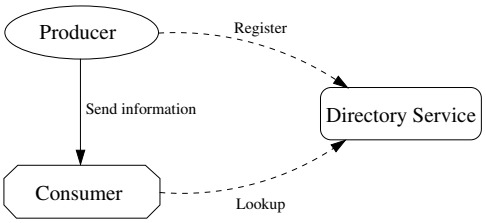

Figure 2. Grid Monitoring Architecture components

3.2 Model's description

The main issue of current monitoring systems is their centralized and static organization (see Section 4). Centralization introduces weakness in a system because of the single point of failure problem. Moreover centralization 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 choose to use peer-to-peer oriented communications, for the same reason we chose peer-to-peer communications for self-management purposes. Figure 3 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 storage*. 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.

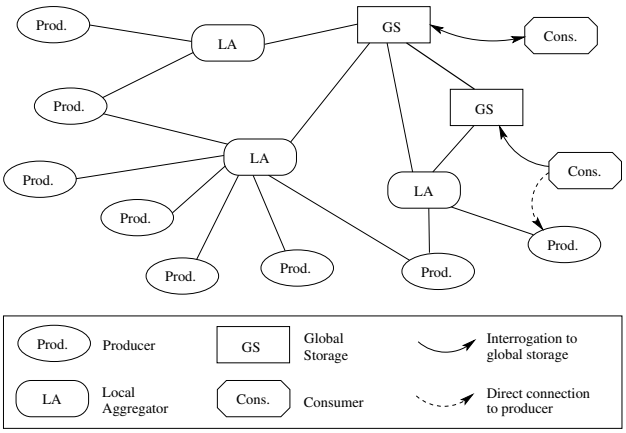

Figure 3. Infrastructure

Following the main recommendations of [18], the four properties of a self-managed system presented in Section 2.1 are handled as follow:

Self-configuration When a new component wants to enter the system it has to follow some steps. First the component must enter into the peer-to-peer network. Then the component may download some codes it wants to run from other components already in activity in the system. Finally the component tries to locate the more suitable and efficient components with which it needs to communicate.

Let's illustrate this procedure with the example of a new producer entering into the monitoring system. First the producer looks for joining the peer-to-peer overlay network. The system offers two ways to discover this network: *(1)* by broadcasting on the local network (often provided by the peer-to-peer library), or *(2)* assuming the existence of a *bootstrap group*, composed of nodes expected to be always online. Their addresses are registered under a single name, using DNS. This second mechanism is necessary for a new peer belonging to a private network in which no node is currently involved in the peer-to-peer network to reach the peer-to-peer network. The bootstrapping issue occurs in all non-centralized peer-to-peer systems.

Any device to be observed need not necessarily to implement its own sensors but can retrieve it from somewhere else in the grid. For instance, the sensors of computational nodes are quite the same in all the different administrative areas of the grid: it provides information about CPU load, network throughput, system, etc. Thanks to the ability of some peer-to-peer systems to discover and download published code in the network, the producer can automatically recover a sensor code from another producer.

Finally, the new producer looks for the components which with it has to communicate: the aggregators (through which it communicates to the global storage). The producer asks the peer-to-peer network to find an arbitrary amount of aggregators defined by the administrator. Then the producers automatically check the *round trip time* (RTT) with those aggregators. We use RTT as metric to express proximity of two peers: our supposition is that a low RTT means the two belong to a same cluster, a high RTT means the two peers are distant, probably on two different domains of the grid. The producer keeps contact only with a subset of those aggregators: *(1)* with the "close" aggregators (smallest RTT values) in order to communicate efficiently, but also *(2)* with some of the aggregators of higher RTT values in order to best spread the information of the producer to "far" areas. Thus the producer component is automatically configured to efficiently disseminate its information in the all grid.

Self-optimization A basic policy of self-optimization is the dynamic dimensioning of the "service's size" depending on the variation of its load. It means the increase or decrease of the elements involved in the grid service in accordance with the increase or decrease of the service's load. When a component becomes overloaded it must redirect some of its work to another already running and underloaded component, or take the initiative to create a new component to answer the need of additional processing.

The aggregator and global storage are an example of such mechanism. An aggregator (or global storage) becomes overloaded when it receives more messages than it can process. The component detects this overload when its message queue reaches a threshold and continues to grow. At this time, the component looks for an underloaded similar component by a search in the peer-to-peer network. If its search is a success, it notifies some of the producers communicating with it to reference this new component. If the component fails to find an underloaded component, it takes the decision to create a new component. This new component is instantiated on a host found by a search in the peer-to-peer network and satisfying some requirements such as enough memory, CPU, and bandwidth free. Half of the producers connected to the overloaded component, which have the smaller RTT with the new component, are chosen to be redirected to the new component.

Another aspect of the self-optimization consists of the regular observation of the system's performance. If this performance does not obey to constraints set by the administrator, the system may decide to take some measure. For instance, if the communication time between an aggregator and a global storage exceed a specified time, the aggregator may be migrated in a location where this constraint can be respected. Another example can be the momentary reduction of a component communication frequency in order to eliminate network congestions. If network sensors warn congestion in a part of the network, the components of this network part should double the period of time between their communications until the congestion disappear.

Self-healing In a similar way, a basic policy of 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. We can distinguish two cases: the producers, and, the aggregators and global storages. When a producer fails to execute or terminates unexpectedly, the system tries to re-execute the component. If the component fails again after repeated trial, the host node is considered as a failed node.

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. If 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.

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 communication model of our monitoring has to be scalable and low resource consuming. It is unacceptable that a producer sends several times its information into the system; however this information has to be disseminated as fast as possible in the entire system. 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. There are a number of variations on how this is done. In "push" gossip the process that initiates the gossip "infects" the selected target with information. It has been shown that in a group of n machines if one machine starts out with a novel piece of information it takes $O(log\ n)$ rounds for every machine to become aware with that information [13].

Figure 4 presents the dissemination of the information from one node to an entire population (20 experiments for each size of population). We observe that even in very large scale (1,000,000 nodes) the entire population is notified of the new information in less than 30 rounds. Moreover, near 90% of the population is already informed at the 22nd round. Gossip-based multicast is an effective tool for providing highly reliable and scalable message dissemination.

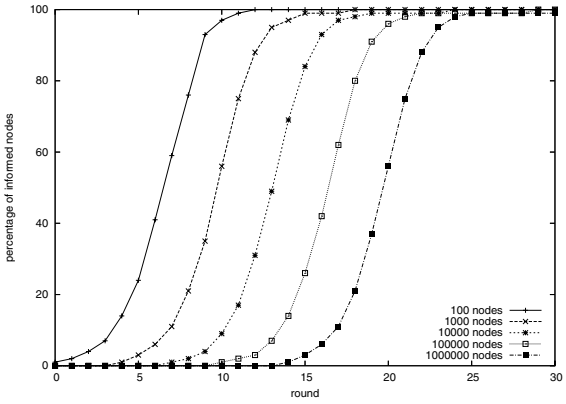

Figure 4. Gossip broadcast

In our system, the gossip protocol is modified. 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, 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, one of the challenges of our system is to spread information from one producer to all the global storages in the shortest time.

3.3 Model's performance

We built a simulator in order to evaluate the capabilities of our monitoring system in very large environments. This simulator allows having a global view of the entire system, and so observing the dissemination of information.

In our first experiment, we want to observe the adaptation of the number of aggregator when the amount of producers increases. From a 100 producers system, we add 10 new producers each iteration, until the system reaches a size of 1,000 producers. For the simulation, an aggregator is declared overloaded when it receives 100 or more information during one iteration. In that case a new aggregator is created and inherits half of the producers communicating with the overloaded aggregator.

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 84.78 producers).

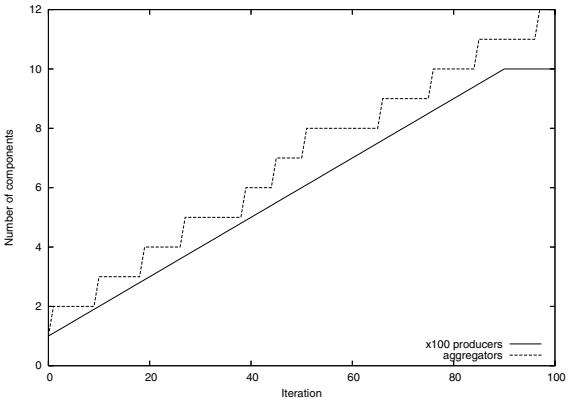

Figure 5. Adaptation to the system size

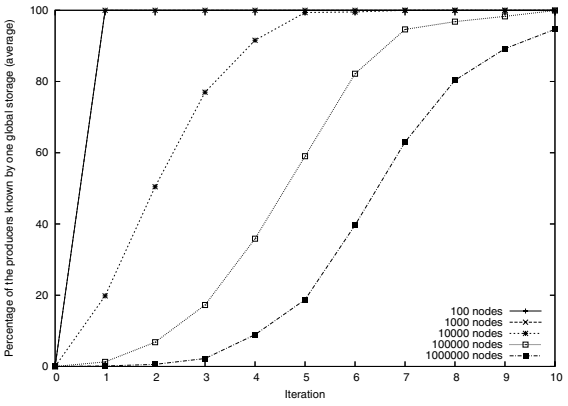

Figure 6. Knowledge of the system in the global storages

The second experiment consists of observing the time required by the global storages to be informed of the presence of the producer nodes. It means the time required by the global storages to receive at least one information from each producer in the system. The time is expressed by *iterations*. During one iteration, each producer gossips to one aggregator, each aggregator gossips to one global storage, each global storage gossips to another global storage. 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 5 aggregators. Each producer is linked to 10 randomly chosen aggregators using a uniform distribution, each local aggregator is linked to 10 global storages (randomly chosen), and each global storage is linked to 5 other global storages (randomly chosen). There is no assumption about what could be a real heterogeneous connectivity. Figure 6 presents the results. The curves plot the average percentage of system knowledge in all the global storages of the system.

In a 100,000 producers system, it takes 10 iterations for all producers to be known by every global storage. In a 1,000,000 producers system, after 10 iterations 94.64% of the producers are known by the global storage. The dissemination of information is much faster than in the standard gossip multicast shown in Figure 4. A first obvious reason is that one iteration corresponds to three rounds because one information is actually relayed three times: from a 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 system does not "broadcast" but "multicast" since only a subgroup of elements finally receives the information through the hierarchical architecture: the global storages. Indeed the total dissemination of information is considered as over when all the global storages have received the information, not all the other components as specified in a "classical" gossip protocol. In that way we can define our gossip protocol as *directed* since it directs the information from the producers to the global storages.

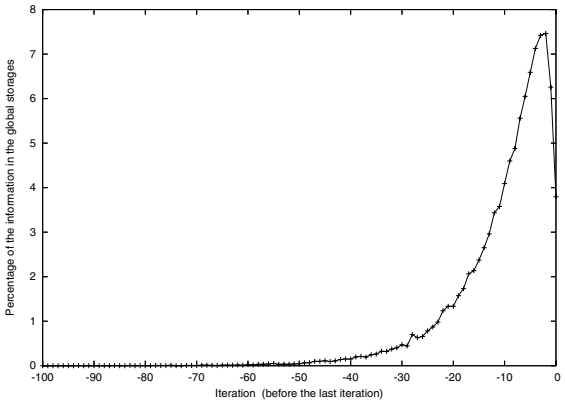

Figure 7. Age of the information in the system

The third experiment consists of evaluating the age of the information in a global storage. Because consumers may directly ask the global storages for produced information, it is important the answered information is not outdated. The conditions of this experiment are similar as the previous one. The only difference is that this system was running for more than 1,000 iterations before being observed. Figure 7 presents the average age of information in the global storages of a 100,000 producers system. The instant of the observation is noted iteration 0.

Information in global storage is recent. The curve shows a peak around iteration -2 (maximum value with 7.46%). 52.26% of the information is younger than 7 iterations, 80.97% younger than 16 iterations, and 95.79% younger than 30 iterations. If we consider that in real conditions one iteration is about 20 or 30 seconds to 1 minute, our system

is well responsive. Let's say that producers, aggregators, and global storages communicate every 30 seconds the average age of the information of a 100,000 producers system is approximately 5 minutes old (10.18 iterations old) which is very satisfactory for a such large system.

3.4 Implementation

Our implementation lets the possibility to use any peer-to-peer protocol. However we chose JXTA as default protocol. We present here the library and motivate this choice.

JXTA [10, 17] 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. JXTA 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. Such features are valuable in the context of grids.

Modules are an abstraction to represent pieces of code. Every peers, groups, services, and modules are represented by *advertisements*: XML documents representing metadata. The discovery service refers to advertisement in order to find JXTA components. In our monitoring system, the sensors are packaged as modules and can be automatically searched on the network, downloaded, and run.

Using JXTA as communication layer in a high performance computing architecture is not suitable to provide high performance. The reasons are *(1)* because of impossibility to select the computing nodes, the resulting heterogeneity of the computational power and bandwidth capacity creates unbalance in the system, and *(2)* the latency of JXTA communications is too high [5]. However, in an application with lower demands on performance the heterogeneity and the higher latencies are not real issues. Grid monitoring systems fall in this category of less demanding applications.

Our prototype application will be soon deployed on grids. In its current status only a Java version has been developed. The monitored data are limited to computational nodes' CPU, memory, and operating system information.

4 Related work

This section presents a survey about grid monitoring systems and their potential autonomous mechanisms for configuration and adaptation.

Network Weather Service (NWS) The Network Weather Service [19] 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.

[15] presents an autonomous framework for grid monitoring built on top of NWS. This work features automatic configuration of the NWS' "clique" groups by measuring the proximity of two nodes according to round trip time. In NWS, a clique group represents a set of nodes in which only one is chosen as representative for all the other to monitor bandwidth. Without clique group, the complexity to measure bandwidth grows to the square of the grid's size.

Globus Monitoring and Discovery System (MDS) The Monitoring and Discovery System (MDS) [7] 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. Under that condition LDAP architecture suffers serious performance and scalability problems.

Relational Grid Monitoring Architecture (R-GMA) The Structured Query Language (SQL) provides 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 [6] 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 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).

5 Conclusion and perspectives

We proposed a solution to achieve autonomous management in grid environment. Our approach is based on the usage of a peer-to-peer overlay network in order to provide scalable and fail-resistant communications between the components of an application. We detailed the construction of an autonomous grid monitoring system, based on the GGF's GMA model. This system uses the peer-to-peer communication for its components interaction but also for data transfer. Each aspect of self-management is discussed, and then scalability and performance are evaluated.

We may consider to integrating this monitoring service with the Open Grid Software Architecture (OGSA) [9] which is more likely to become the standard way to access grid services. Another major improvement would be to use relational database in the global storages. Like in R-GMA our system would be able to answer to more complex requests involving several fields of search.

As a future work, we must focus on the security concerns. At this time we relay 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] BitTorrent. http://www.bittorrent.com.

[2] Napster. http://www.napster.com.

[3] The eMule Project. http://www.emule-project.net.

[4] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, Nov. 2002.

[5] G. Antoniu, M. Jan, and D. A. Noblet. Enabling the P2P JXTA Platform for High-Performance Networking Grid Infrastructures. In *Proceedings of High Performance Computing and Communications (HPCC)*, volume 3276 of *LNCS*, pages 429–439, Sorrento, Italy, Sept. 2005.

[6] A. W. Cooke and al. The Relational Grid Monitoring Architecture: Mediating Information about the Grid. *Journal of Grid Computing*, 2(4):323–339, Dec. 2004.

[7] K. Czajkowskiy, S. Fitzgeraldz, I. Foster, and C. Kesselmany. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th international symposium on High Performance Distributed Computing*, pages 181–194, San Francisco, California, USA, Aug. 2001.

[8] I. Foster and A. Iamnitchi. 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, Feb. 2003.

[9] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. *Grid Computing, Making the Global Infrastructure a Reality*, chapter The physiology of the Grid, pages 217–249. John Wiley & Sons, 2003.

[10] L. Gong. Project JXTA: A Technology Overview. Technical report, Sun Microsystem, Inc., Oct. 2002.

[11] K. Jenkins, K. Hopkinson, and K. Birman. A Gossip Protocol for Subgroup Multicast. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS)*, Phoenix, Arizona, USA, Apr. 2001.

[12] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–52, Jan. 2003.

[13] B. Pittel. On Spreading a Rumor. *SIAM Journal of Applied Mathematics*, 47(1):213–223, Mar. 1987.

[14] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the OceanStore Prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, San Francisco, California, USA, Mar. 2003.

[15] K. Shirose, S. Matsuoka, H. Nakada, and H. Ogawa. Autonomous Configuration of Grid Monitoring Systems. In *Proceedings of the international Symposium on Applications and the Internet (SAINT Workshop)*, pages 651–657, Tokyo, Japan, Jan. 2004.

[16] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. Technical report, Global Grid Forum, Jan. 2002.

[17] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J. Hugly, E. Pouyoul, and B. Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Technical report, Sun Microsystem, Inc., May 2003.

[18] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart. An Architectural Approach to Autonomic Computing. In *Proceedings of the 1st International Conference on Autonomic Computing*, pages 2–9, New York, New York, USA, May 2004.

[19] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5–6):757–758, Oct. 1999.

[20] S. Zanikolas and R. Sakellariou. A Taxonomy of Grid Monitoring Systems. *Future Generation Computer Systems*, 21(1):163–188, Jan. 2005.