visPerf: Monitoring Tool for Grid Computing

DongWoo Lee[1], Jack J. Dongarra[2], and R.S. Ramakrishna[1]

[1] Department of Information and Communication
Kwangju Institute of Science and Technology, Republic of Korea
{leepro,rsr}@kjist.ac.kr
[2] Innovative Computing Laboratory
Computer Science Department, University of Tennessee, Knoxville, USA
dongarra@cs.utk.edu

Abstract. It is difficult to see the status of a working production grid system without a customized monitoring system. Most grid middleware provide simple system monitoring tools, or simple tools for checking system status. *visPerf* is a general purpose grid monitoring tool for visualizing, investigating, and controlling the system in a distributed manner. *visPerf* is a system based on a distributed monitoring sensor, *visSensor*, in which the sensor uses methods to monitor the status of grid middleware with little or no modifications to the underlying system.

1 Introduction

Primarily due to the emergence of high speed backbone network services such as vBNS and the Internet2, Grid Computing [6] has become one of the most exciting new trends in high-performance computing. Ease of use and total computing resource unification involving many kinds of computing tools, including new network facilities and new software, have contributed to the growing interest. The computing resources span large geographical areas ranging from inter-campus to international dimensions. As the number of resources increases, so does proneness to faults. Even though fault-tolerant mechanisms [9,12,10] exist for grid middleware, human intervention is still required for recognizing certain problems of the system. Because huge amounts of data are produced by many components of the system in the form of logs and trace events, it is often very difficult, especially in a large scale system, to find the proverbial "needle in a haystack" without human intervention. To maintain such large scale grid systems, we need the capability to monitor the system. Most grid middleware [2,4,1,7] have this capability to some extent. A monitoring system that is simple and that supports a heterogeneous environment is the need of the hour. *visPerf* is a kind of monitoring system for grid computing in which multiple computing entities are involved in solving a computational problem with parallel and distributed computing tools.

We describe the problems and the requirements of the monitor software for grid computing environment in Sect. 2. In Sect. 3, the system architecture of *visPerf* is presented with design concepts in detail, including its sensor architecture, monitor viewer, and monitor peer-to-peer network. In Sect. 4, we show

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 233–243, 2003.

an example monitor, *visPerf* for NetSolve. Related work is presented in Sect. 5. Finally, we conclude the work in Sect. 6.

2 Support for Grid Computing

For grid computing, a monitoring system has to consider several requirements. It is also a design goal that the monitoring system be for grid computing. Most grid middleware consists of three parts: *client* (e.g. user application), *management* (e.g. resource scheduler), and *resource* (e.g. server, storage and etc). For the client part, users develop their application with grid middleware's programming interface. In the working phase, the user's grid application contacts the resource scheduler to get a resource for its computation. Then, the application can use the assigned resource. Because resources on a grid are located over a wide area (i.e. loosely coupled), it is difficult to be aware of errors and/or problems that result from a user's application. Grid middleware supports remote machines' standard output handles[1] to show the user the output of the application's execution. When the number of processes of a user application becomes large, it is often very difficult to track the errors/problems. If the interactions of the system can be visualized, then the user can better understand and maintain the system. Besides capturing the interactions of a grid application, it is useful to gather information such as the performance of a local machine's processor workload, disk usage and so on, which are related to remote machines in the grid. Users demand a tool for monitoring their system with little effort. To accommodate demands of a monitoring system for grid computing, there are several problems and requirements that must be met.

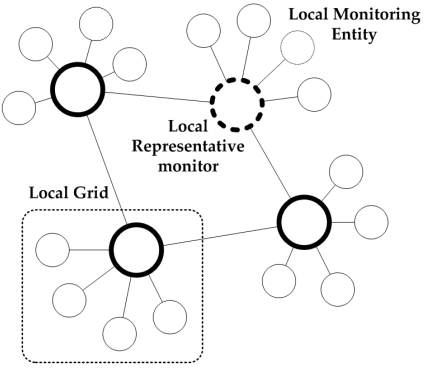


Fig. 1. Centralized+decentralized hybrid peer-to-peer network topology

[1] `STDOUT/STDERR`

2.1 Problems

While building a monitoring system for grid computing, we face several problems such as the following:

Heterogeneity. Hardware and software are configured to be used by grid middleware. In the case of a local system that is in the form of a cluster-like server system, the middleware does not need to consider heterogeneity support because it usually uses the same types of machines and software. But in the grid scale computing environment, the components of the system are located across a wide area and have various types of components, including grid middleware. We need a way to collect the information for the purpose of managing or investigating its working status with heterogeneity support.

Access Network. Due to the various network access policies including firewall, NAT, and so forth, sometimes it is difficult to monitor remote machines in a simple manner. In the case of grid middleware, it has its own mechanism for remote communication with a security mechanism (e.g. Kerberos). To get consistent access to a remote monitoring object, we have to use a tolerant protocol or interface to a network site having restricted access protocols.

Size of Information. The speed of transferring monitored information to an appropriate location is restricted by the capacity of the remote system and the communication channel. We have to devise a way to reduce the size of the monitoring data.

Interoperability. Monitored information can be shared with other applications such as grid resource scheduler and other monitoring systems. To accommodate such applications, an interoperable interface is required.

Scalability. In the grid network, we have to consider the scalability due to the multiple entities to be monitored. The NetSolve production grid [5], for example, sometimes exceeds 100 hosts. This is the case with just a local grid. But, if this grid extends into other external grids, the scalability issues of the monitoring system become more complicated.

2.2 Requirements

Support Various Systems. To monitor various grid systems, we have to have a way to get information from a target running system. We considered two ways for accomplishing this: *indirect* and *direct* interfaces. If grid middleware has no monitoring facility offering its internal status to an external program, we have to use indirect information, that is, *log* information as a form of an accessible data object. This information can be used by a monitoring system when the system is not integrated into the middleware, and hence the need for a program to process the log information generated by the grid in real time. If the grid middleware has a monitoring facility, we can use its interface directly. It also requires minimal effort to integrate it into the monitoring system.

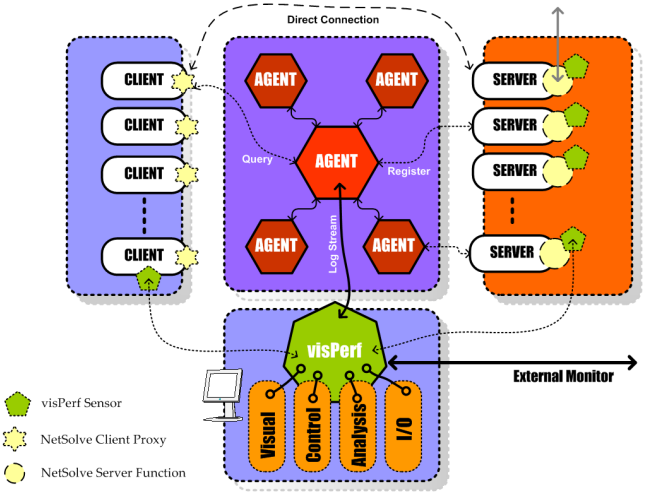


Fig. 2. Overview of *visPerf* monitoring system

Network Topology of Monitor. To deal with the problems listed above, a monitoring system should build a network for itself because it is necessary for a monitoring system to be efficient and simple. Because the centralized network of typical monitoring systems does not fit well with the nature of grid computing, we considered a peer-to-peer network topology [8]. Deploying this peer-to-peer network can cope with large amounts of data to be transferred to one point and can address the scalability problem due to the bottleneck effect when accessing the monitoring system. The monitoring system can maintain local information and then it can be shared with other outside monitoring systems. The appropriate P2P topology is the "centralized+decentralized hybrid P2P network topology" (shown in Fig. 1) because with this we can handle the local grid system in a centralized manner and the global grid in a decentralized manner. Each local representative sensor captures and maintains monitored information of a local grid [2].

Network Access Interface (Protocol). To solve the interface access problem mentioned before, we have to use some form of a flexible mechanism. We considered two network access interfaces (protocol): general TCP data communication (using monitor's own communication protocol) and XML-RPC [13] through a web server. Depending on the condition of a site to be accessed, we can use these two methods adaptively.

3 Monitoring System's Design Approach

With these problems and requirements in mind, we have designed a monitoring system architecture. Figure 2 presents the overview of the *visPerf* monitoring

[2] a.k.a local cluster or one unit of local machines using a same grid middleware

system[3]. Most grid middleware [4,2,1,7] consists of three parts as illustrated in the figure: client, management, and resource (server). This will be explained below, with the NetSolve system in mind. In the client portion, there is a client application equipped with a grid middleware-aware software library. This client application (grid middleware library) uses a resource scheduler of its own in order to get a resource to be used. For this grid configuration, we positioned a monitoring system in each part. The *sensor* is a service daemon for collecting a local host's information and propagating that information to a subscriber monitor viewer. The *main controller*, *visPerf*, is used to control the remote sensor as well as to analyze and visualize the collected information interactively by the user. As a remote sensor, *visSensor* works for monitoring a local machine.

3.1 Local Monitor: *visSensor*

visSensor is a monitor residing on a local machine. It is responsible for sensing a local machine's status by gathering that machine's performance information[4] and special purpose information that is tailored for a specific system like grid middleware (i.e. system interactions). A local machine's general performance information is gathered by performance measuring tools such as `vmstat`, `iostat`, `top` of UNIX-based system performance tools.

For specific system tailored monitoring, we support two methods: log-based monitoring and profiling API-based monitoring. As mentioned previously, there are various run-time systems of grid middleware. The simplest way to monitor a run-time system without any modification to the system is to use its log file because most run-time systems log their status into the log file for the purpose of debugging and monitoring. Although this method is simple, the log file of a grid middleware application has to have a formal form to depict its status in a consistent manner, that is, with a rule (or grammar) of the log file. To capture the running sequence in the case of NetSolve, logs have to have a meaningful format such as:

```
..<omitted>...
NS_PROT_PROBLEM_SUBMIT: Time 1017336647 (Thu Mar 28 12:30:47 2002),
Nickname inttest, Input size 12, Output Size 12, Problem Size 1,
ID leepro@anaka.cs.utk.edu

Server List for problem inttest:
neo15.sinrg.cs.utk.edu
neo9.sinrg.cs.utk.edu
..<omitted>...
NS_PROT_JOB_COMPLETED_FROM_SERVER: neo15.cs.utk.edu inttest 0
..<omitted>...
Server cypher12.sinrg.cs.utk.edu: latency: 1012   bandwidth: 870885
Server neo13.sinrg.cs.utk.edu: workload = 100
..<omitted>...
```

The log presented above, for example, is a part of the NetSolve log file that is produced by its central agent (the resource scheduler of NetSolve). We can

[3] The figure illustrates the NetSolve grid middleware as a representative grid system with *visPerf*

[4] CPU workload, memory status, disk I/O, and so forth.

figure out the sequence of the call from a client ("`ID leepro@anaka.cs.utk.edu`") to a server "`neo15.sinrg.cs.utk.edu`". The client sends a request ("`NS_PROT_PROBLEM_SUBMIT`") to the resource scheduler. Then, the scheduler presents the available resource list. The server ("`neo15.sinrg.cs.utk.edu`") is selected for the client. In this case, the first on the resource list is the selected resource because NetSolve's agent calculates collected performance information for each resource, applies a scheduling algorithm internally, and then sorts the lists in the order of the "most idle" (least loaded) machine. In addition to log-based monitoring, some systems have a profiling API-based interface for internal or external monitoring. For example, NetSolve has "Transactional Logging Facility" that is used by components of NetSolve to notify its activities to an information database server of NetSolve. In the case of NetSolve, all the components transfer their logs onto the agent. So, the rate of logging into a log file is very high when there are many computing resources. Consequently, the *visSensor*'s consumption of the log file in real time requires that large amounts of data be transferred to a subscriber. To alleviate this effect, *visPerf* has a preprocessing filtering function. This has two advantages: lightweight data size to be transferred to a subscriber and a standard format of a log to support various types of grid middleware. Figure 3 (right) shows components of *visSensor* in which there are several layers. The `Info/Log Collector` collects a local machine's general performance information (e.g, I/O and Kernel by using `/proc`) and filters a specific system's log (e.g. `NetSolve` log filter in the figure).

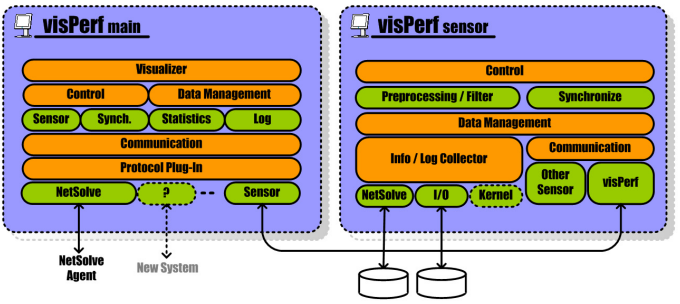
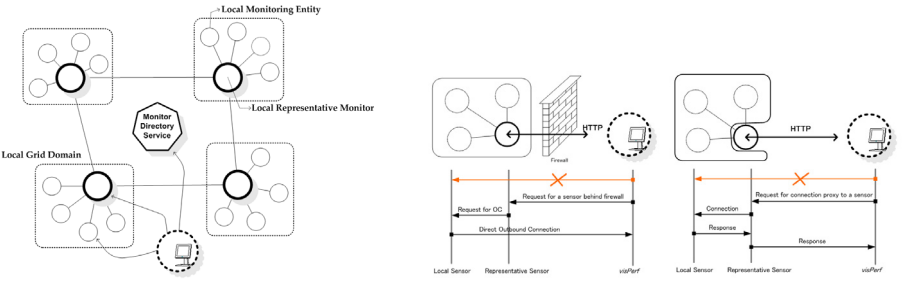


Fig. 3. Components of *visPerf*: sensor and main controller (visualizer)

3.2 Managing and Presenting Monitored Information: *visPerf*

visPerf is used for browsing and controlling the status of a local machine via a local sensor (*visSensor*). Figure 3 shows the relationship between the *visPerf* main controller and the local *visSensor* and components of each side. *visPerf* employs several protocols as a communication subsystem. For example, the `NetSolve` module at the bottom of the figure is used for communicating with the NetSolve agent to query its resource availability. The `Sensor` protocol module is the core communication module that provides two kinds of communication protocols: raw

monitor protocol and XML-RPC protocol. The access protocol can be changed in accordance with the user environment. Also, a user can make its own visualizer through an XML-RPC interface of any programming language. Using the `Control` module, *visPerf* can control local sensors registered in the main controller or directly control a local sensor by specifying its network address. This module periodically sends a synchronization message to its local sensors. If a sensor is out of order or having a problem, a user can discover it. The monitored information is presented via multiple graphical presentations: the *performance fluctuation graph* and the *interaction map*. The interactions of the middleware are displayed in an animated resource map. The sequence of an interaction animation is based on the logs defined in the appropriate sensor filter in which the log runs from the beginning of a call (from a client application) to the end of the call.



(a) (b)

Fig. 4. (a) Monitor directory service: *visPerf* can connect to a local sensor through MDS, (b) monitor proxy: behind firewall (left), private IP network with an access point (right)

3.3 Peer-To-Peer Monitor Network

As mentioned in Sect. 2.2, a network topology for a monitoring system is needed to manage the system effectively. Sensors are located across multiple grid domains to monitor local systems. Through the peer-to-peer network of a monitoring system, it is possible to see the representative sensor's summarized information as well as the specific local machine's status with a direct connection. Locally, the representative sensor maintains the status of its domain machines in which sensors also exist for each target machine in slave mode. To find out the appropriate sensor to be contacted, *visPerf* sends a query to the MDS (Monitor Directory Service)[5] (Fig. 4a) or to a local sensor with a unique domain name (e.g. UT-ICL[6]). When a local representative sensor is about to start service, the monitor registers itself with the specified MDS. When the user's *visPerf* uses the local monitor, the local sensor connected by the user forwards the query to the

[5] It's not the MDS of Globus.

[6] It is not the Internet domain name, but one managed by the monitor system.

MDS. The MDS returns the host name, its network port number, and the type of the target remote sensor. Because the sensor can be a representative sensor of a local domain, the type of the sensor is needed to determine its functionality. With the response from the query, *visPerf* can directly contact the local sensor to get its status.

3.4 Multiple Access Points: Monitor Proxy

For security, most organizations introduce a network firewall system as their front-end. In this case, it is impossible to connect to a local sensor directly from outside the network except for a secure inbound network port[7]. In addition, most local cluster systems use an internal network address scheme (e.g. private IP address or NAT) due to the lack of IP addresses for their working nodes except an exposed node connected to the Internet. This also makes it difficult for a monitoring system to connect to a local sensor from outside directly. Due to these reasons, we introduced a "*Monitor Proxy*" to support a secure connection from outside a network. The monitor proxy has two functions: (i) Delegate a connection request to a local sensor through a representative sensor that uses HTTP tunnelling, and (ii) Provide a connection proxy through a representative sensor. Both methods (i) and (ii) are applied to connect from outside the firewall. In case of a private IP network (or NAT), method (i) is applied to connect from outside the network. If a local sensor is not able to connect to an outside network, it has to use a kind of "*gateway host*" of its cluster or grid. If the representative sensor is located in the exposed node, it can create a connection on behalf of the inside sensor. The right side of Fig. 4b illustrates this situation. For security against a malicious connection, we use the md5 authentication method.

4 Case Study: visPerf for NetSolve

As an example of *visPerf*, we present the customized monitor for the NetSolve grid middleware[8] It provides remote access to computational resources including hardware and software and it supports different architectures and operating systems. NetSolve provides blocking and non-blocking calls to a problem on an available server. It also offers simultaneous multiple calls such as *farming* calls[9]. We can view the parallelism of NetSolve using our monitoring system. To view the interactions of a system, we have to prepare a log filter that is used for refining a specific system's log into a form of *visPerf* as mentioned in Sect. 3.1. To track the sequence of a NetSolve call, we added new logs to the NetSolve system, which are small modifications to the source code of NetSolve. This is just for completing visualization of *visPerf*. The parallelism of non-blocking calls can not

[7] e.g. SSH, HTTP port and so forth.

[8] The NetSolve project [2] is being developed at the Innovative Computing Laboratory of the University of Tennessee Computer Science Department.

[9] A user's data is divided into independent parallel NetSolve calls to use computing resources simultaneously

be shown because there is no log type to indicate the end of a NetSolve call. In addition to this log indicating the sequence of a NetSolve call, this log filter parses performance notification logs. Figure 5 (left) shows a snapshot of a user's problem being submitted between client application and server through one central NetSolve agent of a production grid, which occured during the middle of a NetSolve testing program (`Test` of current release of NetSolve 1.4). In addition to this interaction map, users can investigate a specific host using *visPerf* as in Fig. 5 (right). Figure 6 illustrates the performance fluctuation of a machine's CPU and disk I/O.

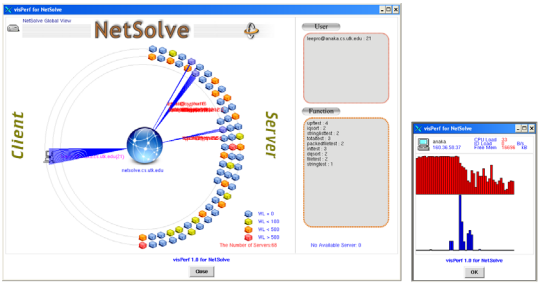


Fig. 5. Visual presentation: interaction map of NetSolve (left), performance graph (right): CPU workload (red bar) and I/O workload(blue bar)

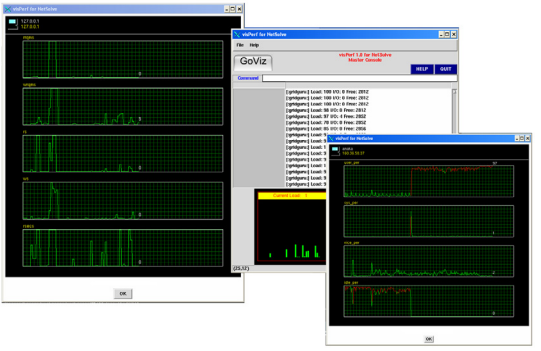


Fig. 6. Sensing local machine's CPU and I/O performance

5 Related Work

Globus HBM [10] is a monitor facility to detect faults of a computing resource involved in Globus. It checks the status of a target machine and reports it to a higher-level collector machine. GridMonitor Java Applet [3] is a kind of monitor for Globus system. It works by displaying the grid information and server status for all sites including Globus MDS. JAMM [11] is an agent-based monitoring system for grid environments that can automate the execution of monitoring

sensors and the collection of event data. It supports not only the system's general performance including network and CPU workload, but also application sensors that are embedded inside applications to notify an overload by threshold variables. It is a type of a system that collects performance information of the grid environments. The difference from our work is that *visPerf* is designed for monitoring activities using grid middleware dependent information via direct and indirect interfaces. Our system works by visualizing interactions of the working system and showing useful information on the system including performance information in a simple, practical manner.

6 Conclusion and Future Work

This paper presented *visPerf* as a monitoring tool for grid middleware to show the running activities and performance information. This monitoring tool will be improved to serve as a more general monitoring system to support different types of grid middleware by adding new sensor functions and log filters. At the same time, using this underlying networked sensing system, we will attempt to make this monitor tool a useful information provider.

Acknowledgments. This work has been supported by the BK21 program in K-JIST, South Korea and the Innovative Computing Laboratory (ICL) and in part by the National Science Foundation grant NSF ACI-9876895.

References

1. James C. French Alfred C. Weaver Paul F. Reynolds Jr. Andrew S. Grimshaw, William A. Wulf. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-20, Department of Computer Science, University of Virginia, Charlottesville, Virginia, USA, June 1994.
2. D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001.
3. Mark Baker and Garry Smith. Gridrm: A resource monitoring architecture for the grid. In *Springer-Verlag, LNCS (2536)*, page 268 ff, 2002.
4. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
5. SInRG (Scalable Intracampus Research Grid). `http://icl.cs.utk.edu/sinrg/`.
6. Carl Carl Kesselman (ed) Ian Foster (ed). *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
7. M. J. Litzkow, M. Livny, and M. W. Mutka. Condor : A hunter of idle workstations. In *8th International Conference on Distributed Computing Systems (IEEE)*, pages 104–111, Washington D.C, June 1988.
8. Nelson Minar. Distributed systems topologies. `http://www.openp2p.com`.
9. Anh Nguyen-Tuong. Integrating fault-tolerant techniques in grid application. Computer Science Dept. Dissertation, University of Virginia, Virginia, Auguest 2000.

10. P. Stelling, I. Foster, C. Kesselman, C.Lee, and Gregor von Laszewski. A Fault Detection Service for Wide Area Distributed Computations. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, pages 268–278, Chicago, IL, 28-31 July 1998.
11. B. Tierney, B. Crowley, D. Gunter, J. Lee, and M. Andrew Thompson. A monitoring sensor management system for grid environments. *Cluster Computing Journal*, 4, 2001.
12. Job B. Weissman. Fault tolerant wide-area parallel computing. 2000. Proceedings of the International Parallel and Distributed Computing Symposium.
13. XMLRPC. `http://www.xmlrpc.com`.