visPerf: Monitoring Tool for Grid Computing

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

† Department of Information and Communication,
Kwangju Institute of Science and Technology, South Korea
{leepro,rsr}@kjist.ac.kr

‡ Innovative Computing Laboratory,
Computer Science Department, University of Tennessee, Knoxville, USA
dongarra@cs.utk.edu

Abstract

This paper describes a monitoring tool for grid computing[5], called *visPerf*. It is difficult to see the status of a working production grid system without a customized monitoring system. The *status* includes many details of the system such as system running information, performance changes, system/software failures, security issues and so forth. Most grid middleware provide a simple monitoring tool for their systems, or provide simple tools for checking the status of the system. *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 grid middleware's status with little or no modifications to the underlying system. *visSensor* uses two methods to capture the status: indirect and direct interfaces. The indirect interface uses the *log* information generated in the run-time of the middleware. The direct one is using an interface of the grid middleware that offers the internal information of the grid system. In the *visPerf* monitoring system, users can write a log filter to refine their grid middleware's log that then plugs into the *visPerf*. The refined information via the filter is then presented to users. Also, using the information interface (profiling API) provided by a grid middleware, *visPerf* can show the internal information that is offered by the system directly. To adapt various network configurations including a firewalled system that only accepts legal network ports (e.g. web service port) from outside of the firewall, *visPerf* supports monitor proxy for its remote sensor network. Sensors of *visPerf* can make a peer-to-peer[9] network for serving information. As a case study, we present the customized monitoring system for the NetSolve[2] functional grid system using *visPerf*. Thus, *visPerf*, when connected to NetSolve, can graphically show the server resources' workload map and their parallel computing interactions.

[1]This work had been performed at Innovative Computing Laboratory as a visiting scholar, from June 2001 to June 2002

1 Introduction

From the emergence of high speed backbone network services such as vBNS[20] and the Internet2[12], *Grid Computing*[5] has become one of the most exciting new trends in high-performance computing. The concepts of ease of use and whole computing resource unification involving so many kinds of computing tools, including new network facilities and new software, have contributed to the growing interest. In a grid network, there may be large numbers of computing resources that range from personal workstations to supercomputers. These computing resources span large geographical areas ranging from inter-campus to international. Even though fault-tolerant mechanisms[8, 21, 7] 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, in a large scale system, to find the proverbial ”needle in a haystack” problem without human intervention. To maintain such large scale grid systems, we need the capability to monitor the system.. Most grid middleware[14, 11, 13, 10] have the capability to do that to some extent. But, we need a monitoring a system that is simpler and that supports a heterogeneous environment.

visPerf is a kind of monitoring system for grid computing in which multiple computing entities are involved to solve a computational problem with parallel and distributed computing tools. Originally, this work was performed as a simple monitoring facility for a specific system, NetSolve[14]. However, this work can be extended to provide a monitor for Globus[11], Condor[10], Legion[13], Ninf[4] and so forth, on a Unix system.

We describe the problems and the requirements of the monitor software for grid computing environment in section 2. In section 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 section 4, we show an example monitor, *visPerf* for NetSolve, using the *visPerf* system. Related work is presented in Section 5. Finally, we conclude the work in Section 6.

2 Support for Grid Computing

For grid computing, a monitoring system has to consider several requirements. We describe the requirements needed for such a monitor system. 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 a 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 in a large scale area, i.e. loosely coupled, it is difficult to be aware of some errors or problems that are derived from a user's application. Grid middleware supports remote machines' standard output handles(`STDOUT/STDERR`) to show the user the output of the application's execution. When the number of processes of a user application grows to a large number, it is often difficult to determine an error or a problem. The status information maintained by grid middleware is not enough to understand the system. If a user can see the interactions of the system via visualized information, 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. Even though this information is already maintained internally in the middleware, it is usually used for a resource scheduler or a decision making system not for a user's monitoring purpose. 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 to be solved and satisfied.

2.1 Problems

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

Heterogeneity In a grid computing environment, 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 the grid middleware. To monitor these computing resources related to the grid, we need a way to collect the information for the purpose of managing or investigating its working status with heterogeneity support. That is, a monitoring system has to adapt to various environments.

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 and like manner. In the case of grid middleware, it has its own mechanism for accessing remote communication with a security mechanism(e.g. Kerberos[15]). But, a monitoring system to be discussed in this paper is an independent system to the grid middleware. 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. HTTP tunneling[18], for example, can be used for addressing the access restrictions of a network behind a firewall so an application residing outside of a firewall can access a network inside of the firewalled site through a web protocol like HTTP.

Size of Information The speed of transferring monitored information to an appropriate location has a restriction according to the capacity of the remote system and the communication channel. So, we have to devise a way to reduce the size of the monitoring data that is changed according to its run-time system and configuration, i.e. the size of the system.

Interoperability Monitored information can be shared with other applications such as a grid resource scheduler and other monitoring systems. To accommodate such applications, an interoperable interface is required. This is closely related to the Access Interface previously mentioned.

Scalability In the grid network, we have to consider the scalability due to the multiple entities to be monitored. The NetSolve production grid (campus-wide grid project[3]), for example, developed by the Innovative Computing Laboratory (ICL at UTK), sometimes exceeds more than 100 hosts. This is just the case of a local grid. But, if this grid extended into other external grids, the scalability issues of a monitoring system will be more difficult than the local grid.

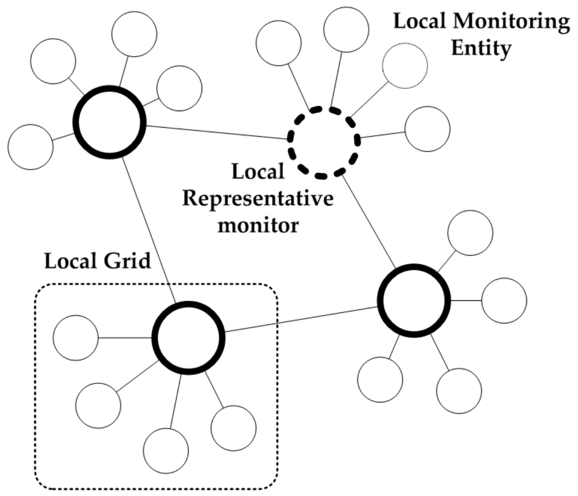


Figure 1: Centralized+Decentralized Hybrid Peer-To-Peer Network Topology for Monitoring System

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 for a monitoring system when the system is not integrated into the middleware, therefore we need a program to process the log information generated by the grid in real time. This program parses log information into a form of standard monitoring information that can be presented to a user. If a grid middleware has a monitoring facility, we can use its interface directly. It also requires a minimal effort to integrate it into the monitoring system. We believe that this can be a solution for heterogeneity problems without any changes to the underlying grid system.

Network Topology of Monitoring System To adapt the problems listed above, first a monitoring system has to construct a network for itself because it is necessary for a monitoring system to be efficient and simple. For this, we considered a peer-to-peer network topology[6] because the centralized network of typical monitoring systems does not fit the nature of grid computing. Central monitors can not monitor whole monitoring entities involved in a grid. Deploying this peer-to-peer network topology 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. By utilizing a peer-to-peer network, the monitoring system can maintain local information and then it can be shared with other outside monitoring systems. For our purposes, there are several types of peer-to-peer network topologies[6]: centralized, ring, hierarchical, decentralized, centralized+ring and centralized+decentralized. These topologies have their own advantages and disadvantages according to evaluation criteria: manageability; information coherence; extendability; fault tolerance; and scalability. We just consider the manageability, information coherence, and scalability issues. The appropriate topology for a grid monitoring system is the *centralized+decentralized hybrid peer-to-peer network topology* because we can handle the local grid system in a centralized manner and the global grid in a decentralized manner. Figure 1 illustrates the topology. Each local representative sensor captures and maintains monitored information of a *local grid* [1] in a compact form. Because there is no central data maintainer of local monitored information, there is no inconsistency between the information produced and viewed via the monitoring system.

Network Access Interface(Protocol) To solve the interface access problem as mentioned before, we have to use some tolerable mechanism. We considered two network access interfaces (protocol): general TCP data communication (using monitor's own communication protocol) and XML-RPC[22](using legal web protocol). Depending on the condition of a site to be accessed, we can use these two methods adaptively. Usually, we do use a monitor's own communication protocol for accessing a remote monitoring sensor. If there is a restriction on accessing it due to a network security policy, we can use XML-RPC through HTTP channel(e.g. legal web port: 80).

3 Monitoring System's Design Approach

Because of the demands of monitoring systems for grid computing, we have identified several problems and requirements in the previous section. Based on those problems and requirements, we have designed a monitoring system architecture. Figure 2 presents the overview of the *visPerf* monitoring system. The figure illustrates the NetSolve grid middleware as a representative grid system with *visPerf*. Most grid middleware[11, 14, 13, 10] consists of three parts as illustrated in the figure; client, management, and resource (server). So, we will explain 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 at 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* is a viewer/controller for controlling the remote sensor as well as analyzing and visualizing the collected information interactively for the user. As a remote sensor, our monitoring system has *visSensor* that works for monitoring a local machine and a viewer/controller, *visPerf*, through a dedicated communication port.

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

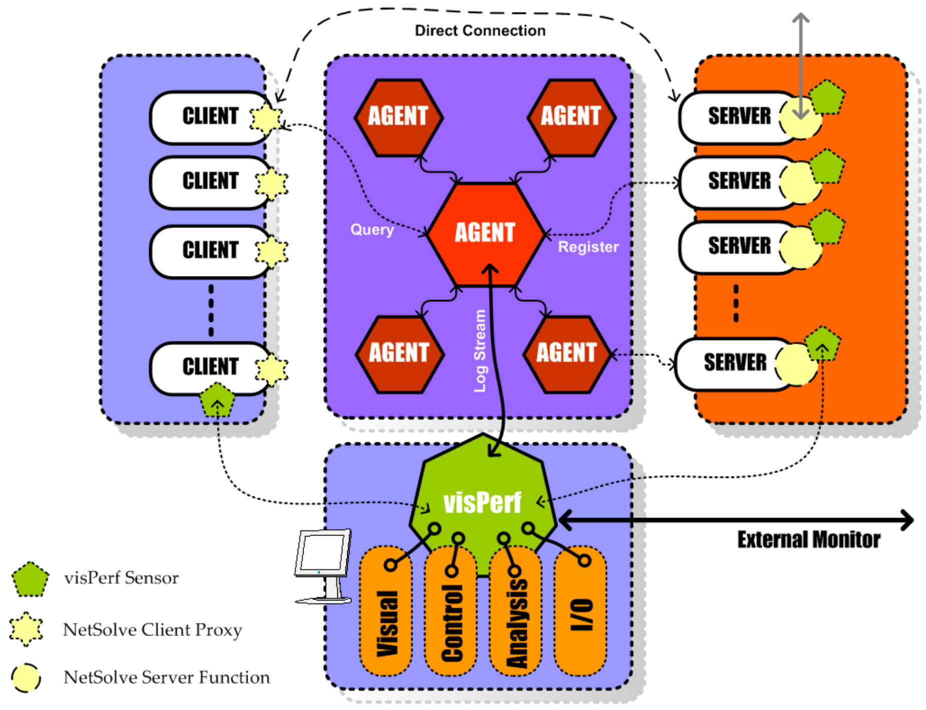


Figure 2: Overview of *visPerf* Monitoring System

3.1 Local Monitor: *visSensor*

visSensor is a monitor residing on a local machine. It is responsible for sensoring a local machine's status by gathering that machine's performance information (CPU workload, memory status, disk I/O, and so forth) and special purpose monitoring information that is tailored for a specific system like grid middleware (i.e. monitoring application interactions).

A local machine's general performance information is gathered by a local machine's system performance measuring tools such as `vmstat`, `iostat`, `top` of UNIX-based system performance tools. *visSensor* maintains the local machine's general performance information by periodically parsing values produced by `vmstat` and `iostat`. This performance information is stored in *visSensor* itself. Based on a request of a remote monitor/viewer, this information is transferred through a network channel to the destination. One advantage of maintaining the local machine's performance information is the richness of the information. Any extra performance measuring facility can be easily integrated into *visSensor* as a module. *visSensor* can keep track of the long-term performance trends that reflect the performance fluctuation of the machine. It includes the statistical information on the machine's performance. This can be used for grid middleware as a clue for the resource scheduling. A similar tool, NWS (Network Weather Service)[17], has already been developed for this purpose. But, NWS is focused on forecasting network performances (bandwidth and latency) of end-to-end communication between two machines using lightweight network probes and adaptive prediction algorithms.

For specific system tailored monitoring, we support two methods: *log-based monitoring* and *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 using 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. Let us assume that the user wants to see the running sequence of a client application. At first, a client application contacts the grid resource scheduler for requesting a resource for its computing. Then, the scheduler returns the information of an available resource to the client application. After that, the client sends a request to a selected resource for solving the client's problem. Finally, if the client has no problem to be solved with the computing resource, the connection between the client and the resource is closed. From the beginning of this scenario to the end, the grid middleware logs the interaction to its 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
neo14.sinrg.cs.utk.edu
neo9.sinrg.cs.utk.edu
neo10.sinrg.cs.utk.edu
neo8.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 anakin27.sinrg.cs.utk.edu: workload = 100
Server neo13.sinrg.cs.utk.edu: workload = 100
Server cypher05.sinrg.cs.utk.edu: workload = 100
Server anakin31.sinrg.cs.utk.edu: workload = 0
Server cypher01.sinrg.cs.utk.edu: workload = 0
Server neo6.sinrg.cs.utk.edu: workload = 100
Server cypher13.sinrg.cs.utk.edu: workload = 1596
..<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 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 one of 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 machine. By tracking the sequence of a call by logging, we can get the internal performance information like "`workload = 100`" as in the sample log above. Because all the internal performance information maintained in the grid middleware is collected into the resource scheduler, we can also get extra performance information provided by the grid middleware itself, in addition to the general performance information measured by a sensor.

In addition to *log-based monitoring*, some systems have *API monitoring* (monitoring facility) or profiling support for internal or external monitoring. For example, NetSolve has "*Transactional Logging Facility*" that is used by components of NetSolve to send its activities to an information database server of NetSolve. The database is a simple *key* and *value* pair store. Finally, monitoring software can get the collected data to show its activities in a timely manner.

After *visSensor* is set up as a stand-alone network daemon, *visSensor* starts to perform the sensory functions. There are two methods for delivering the monitored information to a subscriber (i.e. viewer/controller); *push* and *pull* mechanisms. If *visSensor* is configured to transfer its data to a subscriber, *visSensor* sends its monitored data to a pre-specified host (using *push*). If a subscriber wants to get the monitored data on demand, it sends a request to the *visSensor* (using *pull*).

In the case of NetSolve, the agent is the center for logging its trace. In other words, all components of NetSolve transfer their logs into 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. So, *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. Filtering the raw logs of a specific system into a standard, compact semantic format not only reduces the size of data to be sent but also provides a standard format to be processed in a viewer/controller. If a log filter module for a specific system exists, it can be utilized without any extra effort by the monitor viewer (visualizer). The right side of Figure 3 shows components *visSensor* in which there are several layers. The "`Info/Log Collector`" works at collecting a local machine's general performance information (e.g, `I/O and Kernel` by using `/proc`, which is a special in-memory file system to present the kernel's status) and to filter a specific system's log (e.g. `NetSolve` log filter in the figure). Additional log filters can be plugged into it. *visSensor* serves its monitoring data to an external subscriber through two network protocols: raw monitor protocol and XML-RPC protocol. A sensor has two modes according to its local function: *slave mode* and *representative mode*. In slave mode, it obeys instructions of its pre-configured representative sensor. In representative mode, it subscribes its slaves monitored data and acts as a monitor proxy. This will be explained in Section 3.4.

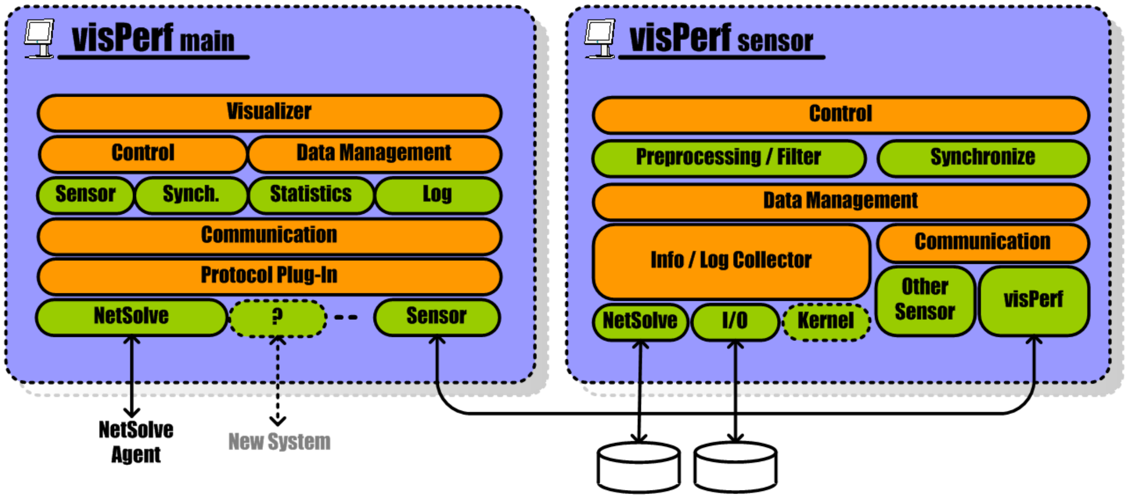


Figure 3: Components of *visPerf*: Sensor and Main Controller (visualizer)

3.2 Managing and Presenting Monitored Information: *visPerf*

Locally, *visSensor* works by monitoring the general system performance and a specific system's activities using aforementioned approaches while managing its collected data and serving it to a external viewer/controller that is called, *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* uses 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 (server) availability. Usually, this function is implemented in *visSensor* via the Java applet version. 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. Through the communication protocol, *visPerf* manages some usage statistics based on log data. It maintains the number of users and resources to be used by the middleware. 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. To check sensors registered in a viewer/controller, this module also 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 information monitored from local sensors is delivered to a viewer/controller, then it is presented via multiple graphical presentations. In our system, there are two types of presentation: the *performance fluctuation graph* and the *interaction map*. The general performance information including CPU workload and disk I/O workload are shown in a line graph. The interactions of the middleware are displayed in a resource map with an animation. 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. The screen-shots are presented in the case study section of this paper.

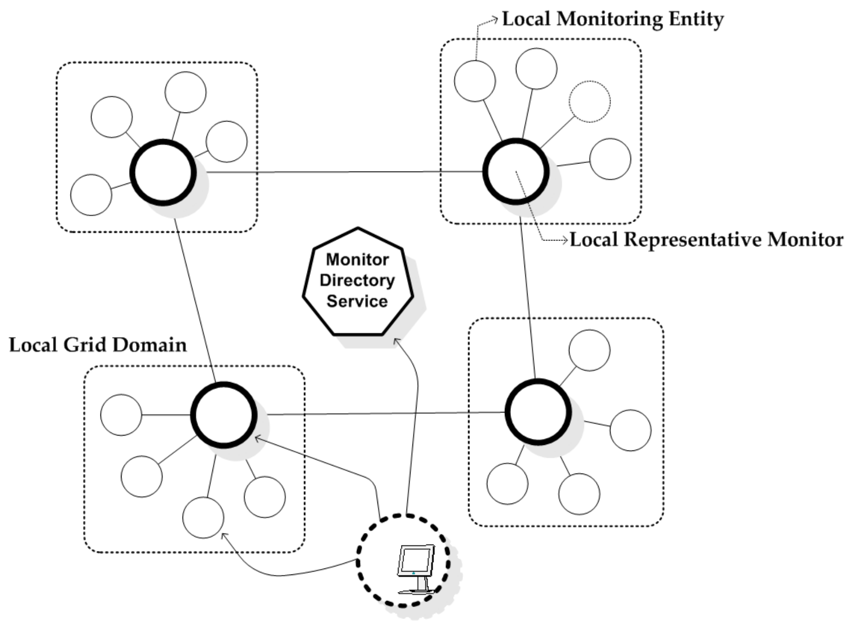


Figure 4: Monitor Directory Service: *visPerf* can connect to a local sensor through MDS.

3.3 Peer-To-Peer Monitor Network

As mentioned in Section 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 the 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) or to a local sensor with a unique domain name (e.g. UT-ICL, it is not the Internet domain name). Figure 4 illustrates MDS. When a local representative sensor is about to start service, the monitor registers itself to the specified MDS. When the user's *visPerf* uses the local monitor, the local sensor connected by the user forwards the query to the 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.

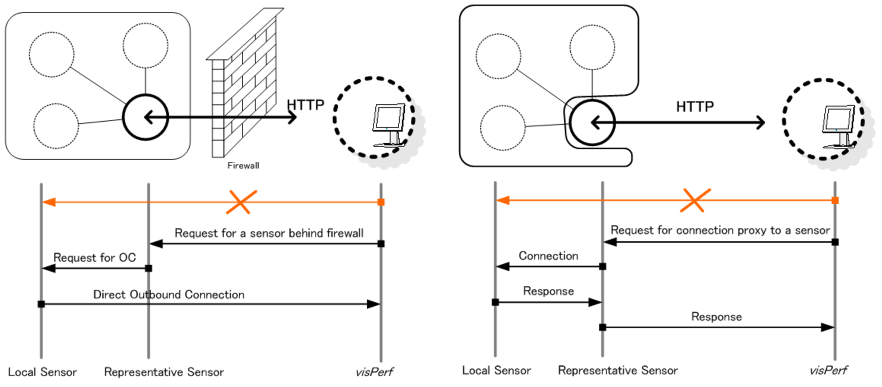


Figure 5: Monitor Proxy: Behind Firewall (left), Private IP Network with an access point(right)

3.4 Multiple Access Points: Monitor Proxy

With the peer-to-peer network of the monitoring system, users can browse the information of a remote machine with location transparency. 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 of the network. Except for a secure inbound network port (e.g. SSH, HTTP port and so on) and all outbound network connections, most inbound connections are not permitted by the firewall system. So, the outside monitor does not contact a sensor host located behind a firewall. In addition to this problem, 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 a master node (i.e. gateway node) connected to the Internet. This also makes it difficult for a monitoring system to connect to a local sensor from outside directly. To solve this environmental problem, we introduced a ”*Monitor Proxy*” function to support a connection from outside of a network. The monitor proxy performed by *visSensor* has two functions:

(i) Deliver a connection request to a local sensor for an outside monitor through a representative sensor that uses HTTP port for its network connection, and (ii) provide a connection proxy through a representative sensor.

In case of a firewall, both methods (i) and (ii) are applied to connect from outside of the firewall. If an outbound connection of a local sensor is permitted by a firewall, and if the local representative sensor notifies the local sensor of a request from an outside monitor, the desired inside sensor can contact the outside monitor in the reverse direction. Then these two monitors can communicate with each other. The left side of Figure 5 illustrates this situation. In case of a private IP network, method (i) is applied to connect from outside of 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. Most local clusters have an exposed node (e.g. master host) to the Internet. If the representative sensor is located in the node, it can create a connection on behalf of the inside sensor. So, a monitor that wants to connect to a local sensor behind a private network with a connection point can contact the local sensor through its representative sensor. The right side of Figure 5 illustrates this situation. For the security of a malicious connection, we use the md5 authentication method[16]. The method used to do authentication is very simple. Each sensor has a file containing a set of keys that are used to salt a md5 hash. The information being transferred for authentication before actual communication between two sensors has its md5 checksum calculated using this salt, and is then transferred to the destination, along with the md5 hash result. At the destination, the sensor will get a key, obtain the salt value from a key file, and then calculate the md5 hash value. The key selected from a key file is determined by an index value from a connection with the originating sensor. If the two are in agreement, authentication is successful.

4 Case Study: visPerf for NetSolve

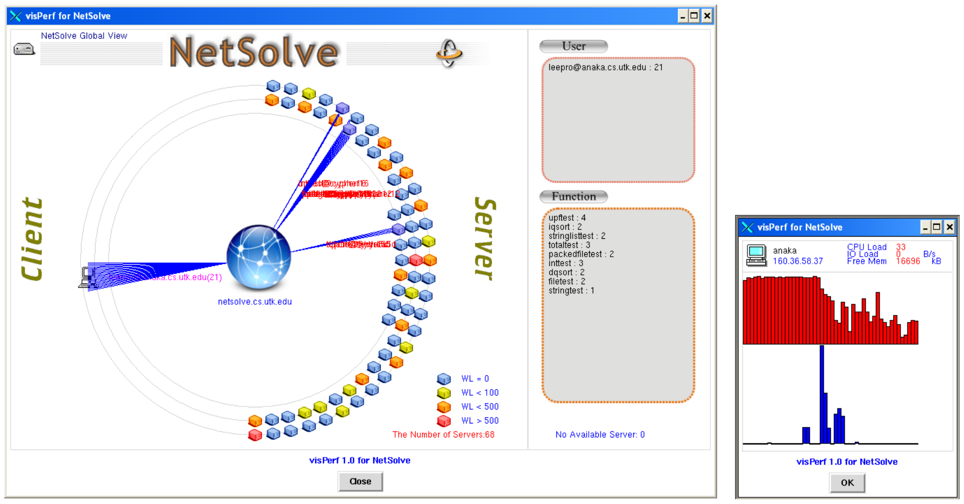
As an example for *visPerf*, we present the customized monitor for the NetSolve grid middleware. The NetSolve project[14] is being developed at the Innovative Computing Laboratory of the University of Tennessee Computer Science Department. It provides remote access to computational resources including hardware and software and it supports different architectures and operating systems including Unix and Windows platforms. Because of its deployable software architecture, NetSolve has been used with other grid middleware such as Globus[11] and Condor[10]. In this section, we show the interactions of the NetSolve production grid using *visPerf*. NetSolve provides blocking and non-blocking calls to a problem on an available server. It also offers simultaneous multiple calls such as ”*farming*” calls in which a user’s data is divided into independent parallel NetSolve calls to use computing resources simultaneously. Farming is a new way of calling NetSolve to manage large numbers of requests for a single NetSolve problem. We can view the parallelism of NetSolve using our monitoring system.

4.1 Customized Log Filter

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 Section 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*. Without this modification, interactions of NetSolve can not be shown. The parallelism of non-blocking calls can not 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. This information presents the current CPU workload and server’s network performance.

4.2 Visual Presentation

Figure 6 (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 Figure 6 (right). From this visual information, users can see the whole picture of the current running system. For browsing a sensor’s status including underlying interactions of the grid middleware, a Java applet version of *visPerf* is available.

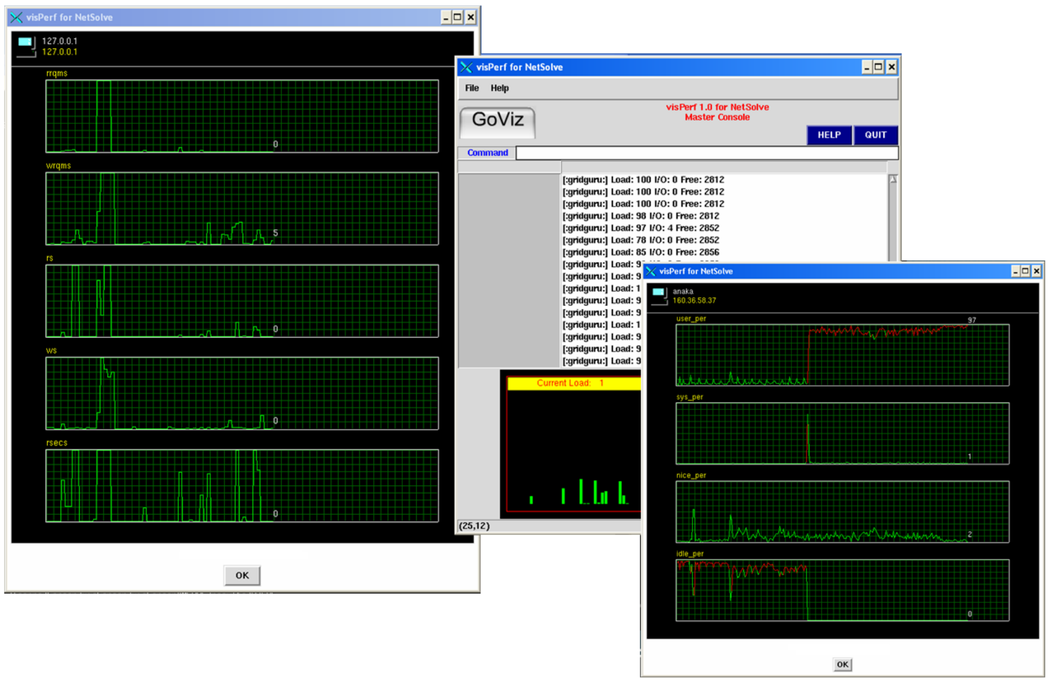


leepro@anaka.cs.utk.edu : 21

upftest : 4
iqsort : 2
stringlisttest : 2
totaltest : 3
packedfiletest : 2
inttest : 3
dqsort : 2
filetest : 2
stringtest : 1

Figure 6: Visual Presentation: Interaction Map of NetSolve(Left), Performance Graph(Right): CPU Workload(red bar) and I/O workload(blue bar)

Using *visSensor*, we can obtain useful information for a grid middleware's scheduler. Figure 7 illustrates the performance fluctuation of a machine's CPU and disk I/O. We are currently working on a performance forecasting project (SWS: Storage Weather Service) on a storage system. It is similar to NWS (Network Weather Service)[17], which deals with network performance. Based on *visSensor*, SWS collects CPU and disk I/O performance. Using various mathematical estimation methods, SWS will offer performance trend information to a subscriber including a grid middleware resource scheduler using this *visPerf* framework.



```
[:gridguru:] Load: 100 I/O: 0 Free: 2812
[:gridguru:] Load: 100 I/O: 0 Free: 2812
[:gridguru:] Load: 100 I/O: 0 Free: 2812
[:gridguru:] Load: 98 I/O: 0 Free: 2812
[:gridguru:] Load: 97 I/O: 4 Free: 2852
[:gridguru:] Load: 78 I/O: 0 Free: 2852
[:gridguru:] Load: 85 I/O: 0 Free: 2856
[:gridguru:] Load: 9
[:gridguru:] Load: 9
[:gridguru:] Load: 1
[:gridguru:] Load: 9
[:gridguru:] Load: 9
[:gridguru:] Load: 1
[:gridguru:] Load: 9
[:gridguru:] Load: 9
[:gridguru:] Load: 9
```

Figure 7: Sensoring Local Machine's CPU and I/O performance

5 Related Work

Globus HBM(Heart Beat Monitor)[7] 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. HBM has an application-level checkpoint mechanism that is also used as a fault-tolerance mechanism and can be used for other grid middleware such as NetSolve. GridMonitor Java Applet[1] is a kind of monitor for Globus system. It works by displaying the grid information and server status for all sites including Globus MDS (Metacomputing Directory Service). JAMM[19] 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 (with SNMP) and CPU workload, but also application sensors that are embedded inside of applications to notify an overload by threshold variables. It is a type of system to collect performance information of the grid environments. These systems each have a different design concept for their own purpose. The difference from our work is that *visPerf* is designed for monitoring activities using a grid middleware's 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 a grid middleware to show the running activities and various performance information. We attempting to design and implement a general monitoring tool for such middleware. For a target grid middleware, the underlying *visSensor* does the sensoring work and manages the middleware's log information and delivers it to a user in a graphical manner. To make the monitoring system more efficient, we designed a peer-to-peer network with MDS (Monitor Directory Service) to search an appropriate remote sensor by investigating its tolerance for network access restrictions. *visSensor* works as a monitor proxy, which enables the communication between two sensors in a restricted access environment such as a firewalled network or a private local IP network. This monitoring tool will be improved as a more general monitoring system to support various grid middleware by adding new sensor functions and log filters. At the same time, using this underlying networked sensoring 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. Special thanks to colleagues of the NetSolve project: Sathish Vadhiyar, Sudesh Agrawal, Keith Moore, Kiran Sagi, Zhiao Shi, Michelle Miller, Don Fike and Scott Wells.

References

[1] GridMonitor Java Applet. http://homer.csm.port.ac.uk:8088/monitorservlet/gridmonitorapplet.html.

[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] SInRG (Scalable Intracampus Research Grid). http://icl.cs.utk.edu/sinrg/.

[4] Satoshi Sekiguchi Hidemoto Nakada, Mitsuhisa Sato. Design and implementations of ninf: towards a global computing infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15:649, 1999.

[5] Carl Carl Kesselman(ed) Ian Foster(ed). *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[6] Nelson Minar. Distributed systems topologies, http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html.

[7] Globus Heartbeat Monitor. http://www.globus.org/hbm/heartbeat_spec.html.

[8] Anh Nguyen-Tuong. Integrating fault-tolerant techniques in grid application. Computer Science Dept. Dissertation, University of Virginia, Virginia, Auguest 2000.

[9] Andy Oram(ed). *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

[10] Condor Project. http://www.cs.wisc.edu/condor.

[11] Globus Project. http://www.globus.org.

[12] Internet 2 Project. http://www.internet2.edu.

[13] Legion Project. http://legion.virginia.edu.

[14] NetSolve Project. http://icl.cs.utk.edu/netsolve.

[15] Kerboros: The Network Authentication Protocol. http://web.mit.edu/kerberos.

[16] IP authentication using Keyd MD5 RFC 1828. http://www.ietf.org/rfc/rfc1828.txt.

[17] Jim Hayes Rich Wolski, Neil Spring. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15:757, 1999.

[18] Patrick Thompson. Web services - beyond http tunneling , http://www.w3.org/2001/03/wsws-popa/paper35.

[19] B. Crowley D. Gunter J. Lee Tierney, B. and M. Thompson. A monitoring sensor management system for grid environments. *Cluster Computing Journal*, 4, 2001.

[20] very high speed Backbone Network Service(vBNS). http://www.vbns.net.

[21] Job B. Weissman. Fault tolerant wide-area parallel computing. 2000. Proceedings of the International Parallel and Distributed Computing Symposium.

[22] XMLRPC. http://www.xmlrpc.com.