ClusterProbe: An Open, Flexible and Scalable Cluster Monitoring Tool

Zhengyu Liang , Yundong Sun, and Cho-Li Wang
Department of Computer Science and Information Systems
The University of Hong Kong , Pokfulam Road , Hong Kong
{ zyliang , ydsun, clwang } @csis.hku.hk

Abstract

In this paper, we describe the ClusterProbe, an open, flexible, scalable, and Java-based tool for monitoring large clusters of workstations. The tool provides a open environment by developing a multiple-protocol communication interface that can be connected to various types of external accesses from the clients. The design of ClusterProbe allows it to scale up to commodious capacity with its cascading hierarchical architecture. In addition, ClusterProbe is flexible that the monitoring tool can be easily extended to adapt to the resource changes by using the Java-RMI scheme. Several useful services are implemented in ClusterProbe, including the visualization of cluster resources information in various forms and cluster fault management. These services have been used to assist the execution of a cluster-based search engine and a distributed N-body application. All experiments demonstrate high efficiency and good performance improvement.

1. Introduction

Using workstation clusters for distributed computing has become popular with the emergence of low-cost, powerful workstations and high-speed networking facilities [1,7]. Monitoring such a platform is not a trivial task since typical workstations are designed to work as an individual system rather than a part of computing cluster [6]. To maintain the integrity of system and to harness the full potential of workstation clusters, an efficient tool, which can monitor the critical system activities and cluster resource utilization, is needed. Thus, cluster administrators can observe the entire cluster with GUI visualization to manage and maintain the cluster. The retrieved resource utilization information can be used to guide the job scheduling for some parallel applications [11]. This paper proposes efficient techniques for the development of a cluster monitoring tool. Three main design issues are emphasized:

- **Open environment:** Most existing cluster monitoring tools do not provide open environments for application programs. Those tools collect and display the cluster resources information only for their integrated GUI modules. Other subsystems or applications that may need the same data are not able to share the monitoring information. For example, some monitoring tools could be accessed through a standard mechanism such as SQL, the fixed and build-in communication mode is not adequate to support the complex and rapid changed applications running on cluster [3]. We aim at building an open environment that can support multiple communication protocols. Such an environment should allow the clients to retrieve the cluster resources information provided by the monitoring tool using standard communication interfaces which are preferred and locally available.
- **Flexibility:** The resources of a cluster, including both hardware and software evolve rapidly. A good monitoring tool should be flexible and extensible to keep up with the resource changes. It should provide fast and automatic updates of its system utilities without enormous efforts in modification. A monitoring tool with fixed resources configuration needs major modification of its programs, which is difficult to adapt to new system configuration.
- **Scalability:** The monitoring tool should scale in capacity to manage a cluster composed of a large number of computers, without consuming lots of system resources. It should provide the largest degree of parallelism to facilitate the concurrent control and monitoring of the underlying cluster resources. The monitoring tool should exploit the parallelism of the execution of monitoring task, providing group control and inspection, and reducing overheads in coordinating monitoring agents and server.

We have designed and implemented an open, flexible and scalable monitoring tool, called ClusterProbe to monitor large clusters of workstations. To create an open environment, ClusterProbe provides a *multiprotocol communication interface* (MCI) to be integrated into client with their local communication environment. The MCI can support various communication protocols, such as RMI/CORBA, HTTP/HTML, TCP, UDP, and SQL. In MCI, we use multiple adaptors to provide transparent communication services for external access. ClusterProbe allows us to use more than one communicate adaptors for querying requests of different protocols from various clients. In addition, ClusterProbe adopts the *pre-formatting modules* to enable the function of offering the datum to clients with user-preferred formats including raw data, compact data, security packets and customized objects.

To make ClusterProbe flexible and extensible, we have chosen to gather and transmit data using *Java-RMI* mechanism. With RMI (Remote Method Invocation), method codes can be transmitted from the server to remote agent dynamically. This capability makes ClusterProbe naturally extensible because new resources could be monitored by automatically downloading codes from server to the remote agent without affecting the execution of existing tasks. In addition, with registration mechanism in RMI, nodes can freely join or disjoin the cluster on the fly.

ClusterProbe sets up a *cascading hierarchy* of monitoring domains to improve its scalability, which is the ability to handle large-scale monitoring task. The hierarchy allows us to retrieve and process data in parallel, restrict nodes to interesting sets of the resources, and cut down the communication to higher level domains.

In the following sections, the overview of the ClusterProbe architecture is given in Section 2. We introduce multiprotocol communication interface and pre-formatting modules in Section 3. We discuss Java-RMI mechanism in Section 4. The cascading hierarchy is discussed in Section 5. We present our implementation in Section 6 and describe our experimental results in Section 7. Related works are discussed and compared in Section 8. Finally, we summarize our conclusions and future works in Section 9.

2. Overview of the ClusterProbe Architecture

ClusterProbe system consists of three main components: (1) The *monitoring server* (2) The *monitoring proxy* (3) The *agent*. Figure 1 shows the overview of the ClusterProble archtecture built in a cluster.

The *monitoring server*, which resides on one powerful node of the cluster, is responsible for handling the requests from clients and forwarding the monitoring results to the clients that are of interest. A client application could be a parallel task that requests the cluster resource utilization data from the Clusterprobe, so as to apply its load balancing strategies. Or it could be just the cluster administrator who wants to view the cluster status through the Internet. All clients communicate with monitoring server through a communication adaptor. Communication adaptors that support various types of communication protocols are available in ClusterProbe.

A *monitoring proxy* is responsible for managing a subset of the cluster nodes in the same domain, containing a set of nodes based on resources type or job allocation policy. For examples, in Figure 1, a batch job domain can be formed, if dedicated execution environment of parallel applications is needed. A *domain* may also be constructed by nodes with different hardware configuration or software installation, as long as they are requested by the applications. The design of the monitoring proxy is also for the scalability of the monitoring tool itself. It is simply a partitioning mechanism to

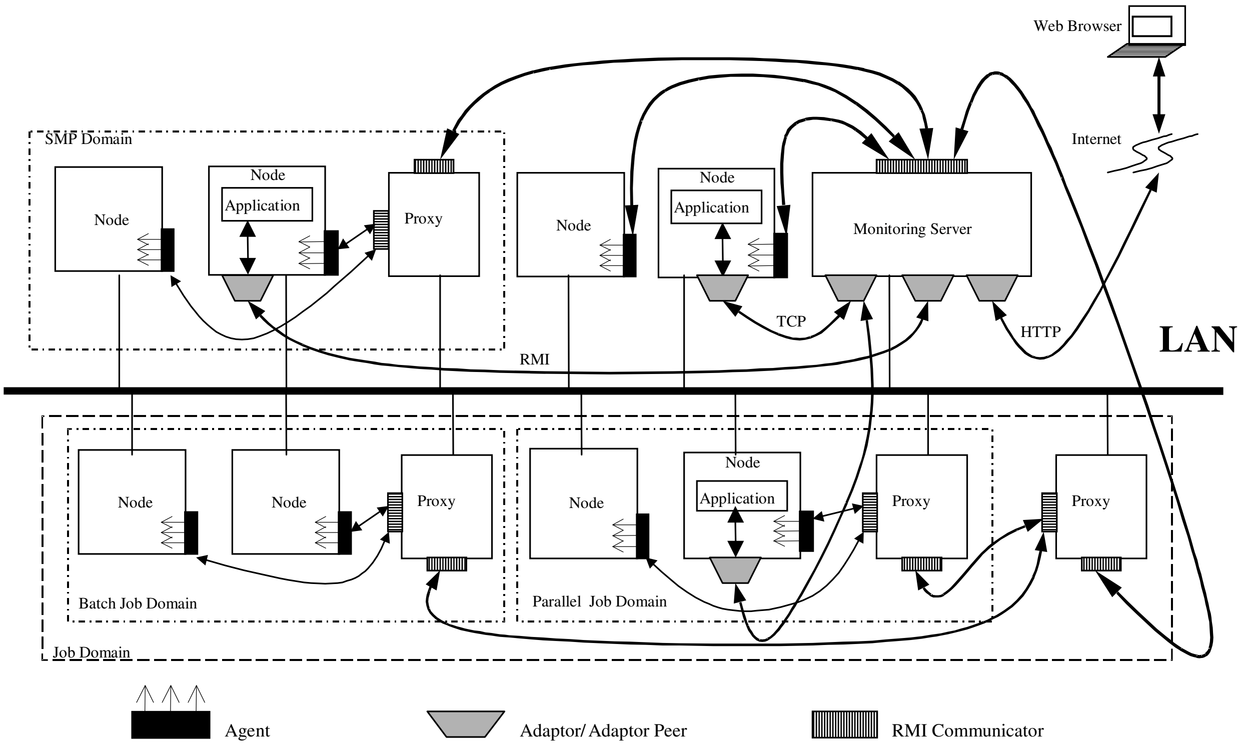


Figure 1: Overview of ClusterProbe

ease the management of monitoring tool. Thus, the monitoring tool can be used to monitor a large cluster. A *monitoring proxy* accepts the requests from upper domain, processes and forwards the data to upper proxy.

The *agent* executes as a daemon on all the nodes that comprise the cluster, downloads the monitoring sessions from the server or proxy of its domain, collects and reports local resources status. All agents communicate with their *monitoring proxy* using RMI.

3. Multiprotocol Communication Interface and Pre-formatting Modules

The *monitoring server* is the only entrance for clients to access the monitoring tool. The server handles requests from all clients, distributes monitoring sessions to corresponding nodes of cluster and supplies the monitoring data to clients that are of interest. The building components of *monitoring server* are shown in Figure 2.

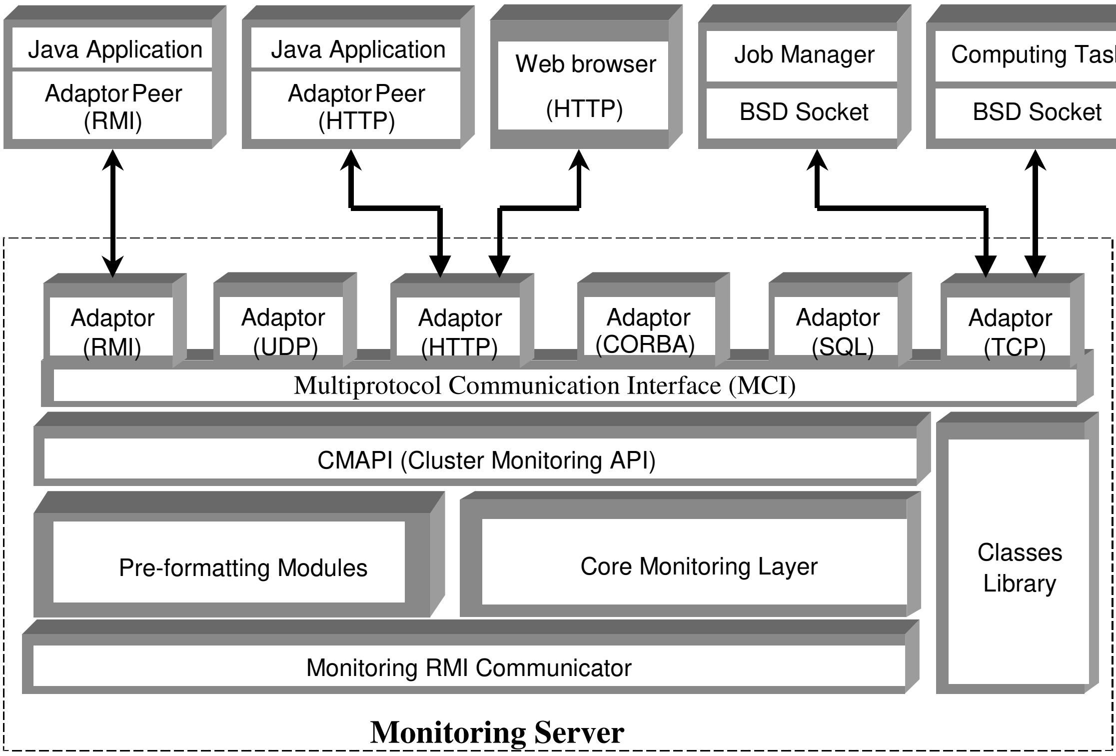


Figure 2: Architecture of Cluster Monitoring Server

The first issue we address is to provide an open environment for sharing monitoring information with various types of clients in the cluster. To solve this problem, we develop a *multiprotocol communication interface* that supports various communication protocols. In addition, we have designed the *pre-formatting modules* to offer the resources data to clients with user-preferred formats.

In MCI, the function of multiple protocols is achieved by using multiple adaptors. An *adaptor* provides access to monitoring server through a particular communication protocol. It opens an external access channel for clients that use this particular protocol to retrieve resources information from monitoring server. MCI can include any number of adaptors, allowing it to be accessed through different protocols. The adaptors convert the querying requests to standard monitoring instructions and wrap the monitoring data with associated protocol.

The client decides the type of access to an adaptor. There are two types of access, namely *direct access* and *peer access*. For examples, Web browser using the HTTP protocol or an application using BSD socket protocol can access an adaptor directly. Both cases are direct accesses. On the other hand, most Java clients access an adaptor through an adaptor peer. For examples, Java clients may use RMI or HTTP protocol to communicate with each other's. In this case, it is a peer access.

We have defined a set of application program interface called *Cluster Monitoring API* (CMAPI), which standardize the functions of adaptors. Thus, an adaptor is a class that implements the interfaces defined in CMAPI. With *classes loading* technology in Java, an adaptor can be plugged in without restarting the monitoring server.

Clients may access monitoring tool for obtaining all kinds of cluster information. To enhance the usability of data and to reduce the loading of clients, we have developed the *pre-formatting modules* to pack the raw data with user-preferred formats. The data can be sent with raw data format or formatted packages that is processed by two or more modules.

In ClusterProbe, the pre-formatting modules include a number of reusable and generic function components for common formats, including filters for compact format, cryptographers for security format, chart/graph generators for chart/graph format and etc.

4. Java-RMI Mechanism

Java Remote Method Invocation (RMI) [9] is Java's remote procedure call (RPC) mechanism. RMI is a distributed object technology that lets you add Java functionality throughout the system in an incremental, yet seamless way. Because of this feature, RMI can move *methods* (class implementations) from server to agent dynamically.

In ClusterProbe, an *agent* is a daemon that running on a monitored node to collect and report the resource configuration and system-related statistics, according to the monitoring instructions specified by the server. The agents can process the monitoring data and react to resource changes, rather than just sending information to server or proxy. For example, one agent allows low-level events to be handled locally without alarm reporting.

We define an interface that describes how to retrieve the resources data from underlying operation system. The agents can download the implementation of the interface from the server when new resources are created. They can monitor new resources as soon as the implementations became available. Updated implementation can also be delivered to the agents through the same mechanism. This allows the monitoring of resources to be implemented and started at any time. The *monitoring session*, which is a thread that executes the codes defined in the downloaded implementation of the interface, will therefore be executed on the agents. This approach provides faster upgrade without installing any new software on agents. It gives us the maximal flexibility and extensibility, since adding and changing resources require us to write only one new Java class and install it once on the monitoring server.

The building components of an agent include *RMI communicator* and *Monitoring Bus.* The *RMI communicator* handles communication between agent and server. The *Monitoring Bus* controls the monitor sessions and calls the appropriate session to perform the requested operations.

Because monitoring codes can be downloaded from server and inserted to agent dynamically, the agent is simple and light-weighted. Furthermore, starting the monitoring agent is the only operation for adding a new node to the cluster. The agent will lookup the server and register to server by itself.

5. Cascading Hierarchy of Monitoring Domain

A cluster system should be able to scale up or scale down for affordability and market volume. Thus, scalability should also be considered while designing the cluster monitoring tool. That is, the monitoring tool should be able to scale to monitor a large cluster without scarifying its efficiency.

We have designed a *cascading hierarchy* of monitoring domains to improve the scalability. The cluster nodes are partitioned dynamically into disjoined groups named as domain, according to monitoring services. Figure 3 shows the design of the *hierarchy* of monitoring domains. One *monitor proxy* is set up in each domain (except the top domain) and provides following functions:

1) Register or unregister to the proxy of upper domain.
2) Receive the monitoring instructions from upper proxy and distribute the monitoring session to agents or proxies under this domain.
3) Merge/sort the monitoring data from this domain and forward it to upper proxy.
4) Handle the fault events or forward them to upper proxy.

A *monitoring proxy* is responsible for managing the nodes or proxies located in the domain where the proxy is residing. The upper proxy can access this domain through the proxy without knowing the details of nodes in this domain.

Owing to these functions, the hierarchy provides many benefits. With this hierarchical structure, the agents can download the monitoring sessions from the proxy in their own domain, instead of the single monitor server. This hierarchy also allows specialization of nodes. We can group the nodes equipped with interesting resources into the same domain, so that the monitoring operation can be simpler. Unnecessary monitoring data won't be collected to save the execution time and system resources. In addition, the monitoring operation codes implemented by the client applications could be more portable. For example, all SMP nodes can be assigned in the same domain, allowing clients needing the configuration and status of SMP nodes to use the same monitoring operation codes when new SMP nodes join the cluster since the proxy of the SMP domain can delegate all the nodes below it.

Furthermore, this hierarchy can reduce the communication to high level domain. By merging and filtering the data, the proxy can decrease the size of the packets. As the amount of data gathered increases and the size of the cluster grows, the reduction of communication becomes more significant.

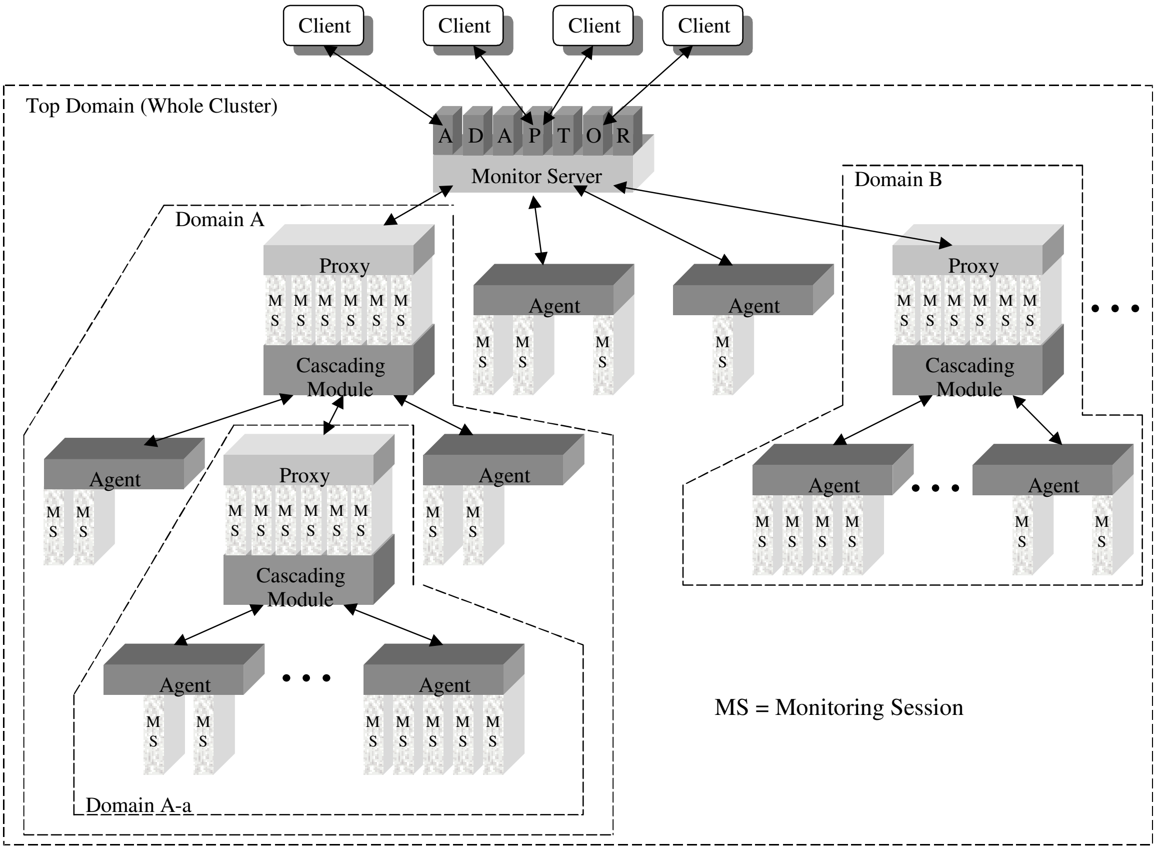


Figure 3: The Hierarchy of Monitoring Domain

6. Implementation

ClusterProbe has been implemented on a cluster of PCs running Linux 2.1.90 or 2.2.1, including 22 WinChip PCs, 4 Dell PowerEdge 4-way SMP servers, 8 Dualon (Dual Celeron) SMP PCs and 10 Pentium II PCs, interconnected by ATM, Gigabit Ethernet and Fast Ethernet switches.

The *monitoring server* is the core of the ClusterProbe. It is located in one of the PowerEdge SMP server with 1 GB memory, running Linux 2.2.1. The *monitoring server* adopts multiprotocol communication interface composed of multiple adaptors. Multiple adaptors provide multiple channels to the tool through various communication protocols. The adaptor will transform the request from original protocol to monitoring instruction defined in CMAPI and convert the monitoring data with associated protocol. All adaptors should be defined as identical structure so that they can be accessed without knowing the details. So far, we have implemented the adaptors for RMI, HTTP and BSD Socket (TCP). The following is the brief definition of the universal interface for adaptor:

```
import java.lang.*;
public interface Adaptor implements Runnable {
    protected Message readMsg ();
    public Instruction transfer (Message req);
    public Message buildMessage(MonData data);
    protected void sendMsg(Message msg);
    void run(); //keep listening the requests
}
```

Before sending the data to clients, the monitoring server can wrap the data with the *pre-formatting modules*. Three kinds of modules are supported in ClusterProbe:

- **Filter module:** The module can be used to filter the reduplicated or stale data and batch the related data.
- **Security module**: Using Java Cryptography Extension (JCE) package, the module can authenticate data by encryption.

- **Chart/Graph module**: The output of this module is a customized object that can be displayed as a chart or a graph, such as sheet, strip chart, pie chart, LED sign etc.

The *monitoring proxies* could be considered as special agents that can manage the agents or other proxies below them in its domain. Four major *monitor proxies* are configured at this moment to monitor our benchmark tests on Directed Point communication subsystem [12] in the WinChip cluster, Cluster-based search engine built in the 4 Dell PowerEdge servers, the Java Thread Migration project (JESSICA) [13] on the 8 Dualon Cluster, and the 10 Pentium II PCs for developing parallel N-Body algorithm [15], respectively. Selected experimental results will be discussed in Section 7.

The *agent* resides on each nodes of the cluster. It will collect resource configuration information and system-related statistics. All agents are communicated through RMI. Because RMI allows us to download methods from server automatically, the agent code can be written as simple as below:

```
import java.rmi.*;
public interface Agent extends Remote {
    void monitor (Session session)
     throws RemoteException;
}
public interface Session extends Seriablizable {
   void run();
}
```

Note that the resources consumed by the agent is light. The size of agent code can be as small as 300 Kbytes, depending on the number of sessions they support. Some resources parameters that can be collected by the agent are listed in Table 1.

Resource Name	Description	Resource Parameters
CPU	Configuration of processors	Number, type, mode, vendor, fpu, bogomips and etc.
CPU Usage	Statistics of processors	Utilizations of user, system, nice and idle modes, temperatues of each processor
Load	Statistics of system load	System load average for past 5, 10,15 minutes
Memory	Statistics of memory	Total mem, used mem, free mem, shared mem and buffered mem
File System	Configuration of file system	Mounted point, mode, total size, used and free size
Process	Process execution information	PID, user, cpu%, mem%, size, start time, start command and etc.
Disk	Statistics of hard disks	Device name, total size, used size, I/O speeds and etc.
Network Interface	Configuration of NIC	Interface name, hardware addr., speed, IP addr., mask, state, MTU and etc.
Network Usage	Statistics of networking	amounts of received, sent, forwarded and fail packets for IP, ICMP, TCP and UDP

Table 1: Resources monitored by ClusterProbe

7. Experimental Results

ClusterProbe is an open monitoring tool for cluster. It can serve for a multitude of parallel applications or subsystems. In this section, we briefly discuss four typical examples.

7.1. Web-based Cluster Management

We have designed and implemented a Web-based cluster management tool to monitor and manage the cluster resources. The management tool interacts with ClusterProbe to obtain various kinds of cluster resource information.

With ClusterProbe, the cluster can be managed with minimum human intervention. New resources can be easily and immediately integrated and software upgrades become trivial. With HTTP adatpor and pre-formatting modules, ClusterProbe can offer customized chart/graph applets to the Web browser. Administrators could select the monitored resource, range, refresh time and display type. The security module is used to authenticate all users when they access the monitoring tool through the Internet. Figure 4 shows two snapshots of the visualization of our web-based management system by using StripChart and PieChart pre-formatting modules, respectively.

Figure 4: Snapshots of Web-based Management

7.2. Scalable WWW Search Engine (SWSE)

SWSE is a simple parallel full-text WWW searching engine on a cluster of PCs. The goal of SWSE is to achieve high throughput and short response time for serving queries, by using the collective computing power and storage capacity of the cluster. The implementation of this engine follows a multiple-query-server-multiple-data-server model. We have installed the SWSE system on a domain composed of 4 Dell PowerEdge 4-way SMP PCs, interconnected by an IBM 8275-416 Fast Ethernet Switch, and running a background application that generates intensive computation but non-uniform load distribution on the cluster nodes. ClusterProbe plays a significant role to improve performance in SWSE system.

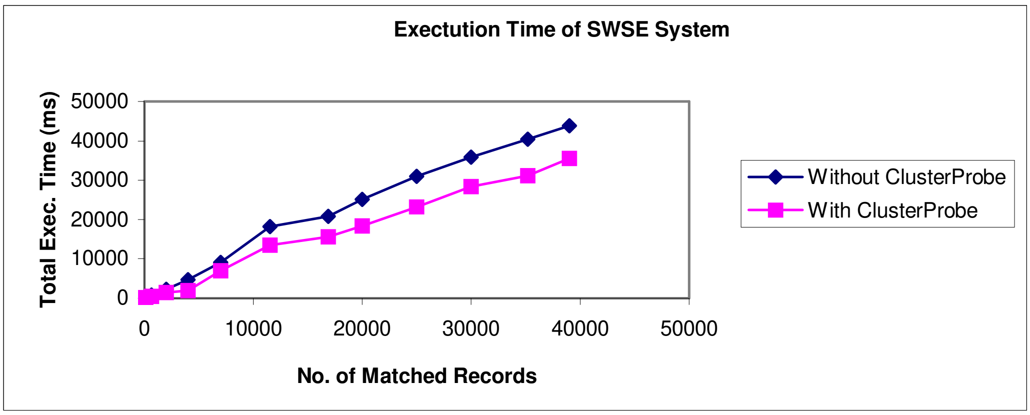
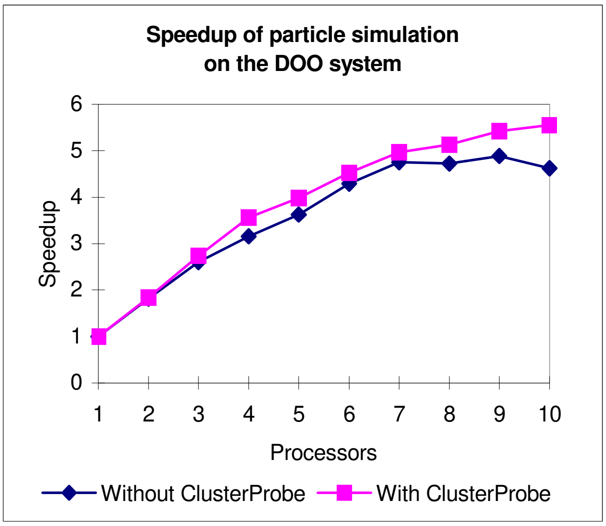


Figure 5: Performance results of SWSE System

We compare two sets of execution time data for serving various numbers of queries in a batch. One set of data is collected by using the round robin algorithm without accessing ClusterProbe. Another one is collected by using the CPU utilization and Disk IO speed information gained via HTTP adaptor of ClusterProbe to choose node with the lightest workload to conduct search. With the assist of ClusterProbe, we can achieve 5-20% performance improvement.

7.3. DOO N-Body

The DOO N-Body test was performed in a Distributed Object-Oriented (DOO) system for solving N-Body problem on heterogeneous cluster [15]. The system allows a group of distributed objects on multiple hosts to work cooperatively in computation. By using the RMI adaptor peer, the system can inquire ClusterProbe the information about host configuration and current status such as workload and CPU utilization. Referring to the data provided by ClusterProbe, the DOO system could dynamically configure the computing environment so as to be adaptive to the computing requirements of an application, as well as the available resources in the cluster.



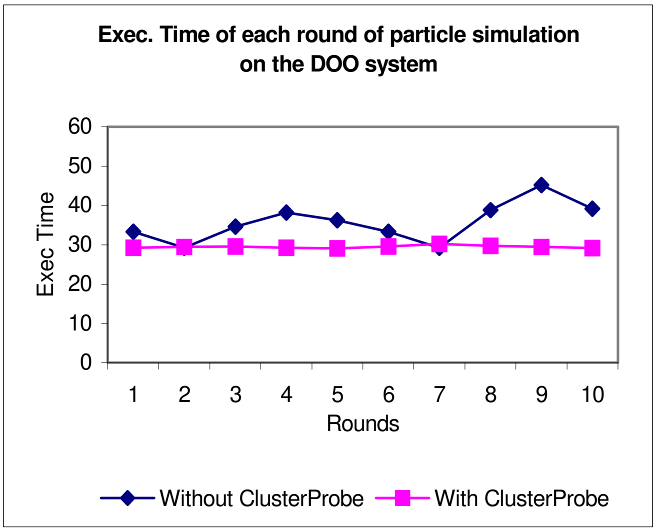


Figure 6: Performance results of particle simulation on the DOO

Fig. 6 compares the performance of the two algorithms for solving N-body problem of particle simulations in the DOO system running in a dedicated environment with 10 Pentium II PCs. One algorithm assigns tasks to fixed machines. The second algorithm dynamically selects light-loaded machines according to the node information provided by ClusterProbe. Overall, the ClusterProbe can improve the execution time around 5-20%.

7.4. Fault Management

To keep high availability, cluster system requires facilities to detect, locate, isolate, and recover from failure. We use ClusterProbe to detect node failure by matching the state of resources with the abnormal conditions, identify when and where failure occurs, and notify the interested event handlers to isolate or recover from failure.

We have implemented the *global event* facility in ClusterProbe to assist in locating failure, defining and handling events. Java provides *local events,* in which an AWT component informs other components that something interesting has happened. In our case, we want to send and receive event objects over the cluster between distributed agents. For each node in our cluster, it receives fault messages in the form of event objects from one or many remote agents, and it sends fault messages to other agents.

Global event structuring mechanism is identical to the local event model of Java, except that instead of source reference within a single Java Virtual Machine, a global name for the event is used. Furthermore, because the nodes involved of one global event are distributed, multicast can be used for efficient RMI communication, instead of Java's local point-to-point casting.

There are several advantages to use global event. When an event happens, it not only affects the local resources that cause the event, but also affects the remote resources, concurrently or orderly. So the first advantage of using global event is the possibility of integrating the distributed event handlers running on each node. Furthermore, we can use global events to diagnose the real problem. For instance, a proxy receives several events in the same time and all the events report networking problems, then the proxy checks the event sources and finds that all of the sources of the events are connected to a same switch, so the networking problem may be caused by the switch.

8. Related Works

Java Dynamic Management Kit (JDMK) is the first Java-based solution for building and distributing intelligence into network device [10]. It is a multiple agent development tool on the market to integrated web-based, push/pull technologies as well as support for multiple management protocol such as SNMP, HTTP and RMI. Both JDMK and ClusterProbe follow similar approaches by implementing multiple adaptors in the distributed system. However, in JDMK each node acts as an individual information source, the access to the node must be explicit, while ClusterProbe aggregates the information through the monitoring server. Besides, for a universal management toolkit, JDMK does not specifically address the issues such as pre-formatting modules and global event management.

GARDMON is a Java-based monitoring tool for non-dedicated cluster computing system [4]. It follows client-server methodology and provides transparent access to all monitored nodes from a monitoring machine. GARDMON can monitor the entire cluster activities through a *single point of control* by using the gardmon-server. However, GARDMON could not been accessed by parallel applications or subsystems except for the gardon-client. The same situation could be found in K-CAP [16], DOGMA [8] and PARMON [5].

The Node Status Reporter (NSR) [14] and the Cluster Administration using Relational Databases (CARD) [2] both provide a standard mechanism for cluster status access, NSR interface and SQL, respectively. But the fixed communication interface is not adequate to satisfy the varied applications. Besides, with the pre-formatting modules ClusterProbe can provide more helpful, powerful, perspective and secure resources information about the cluster.

9. Conclusions and Future Works

We have presented a Java-based cluster-monitoring tool that seizes the advantages of open, flexibility and extensibility. We showed the idea of using multiprotocol communication interface for building open environment with adaptive communication protocols. Pre-formatting modules have been very useful in wrapping the resources information with appropriate formats. The RMI mechanism and cascading architecture are proven to be helpful for flexible resource access and monitoring. Several examples are also given to show the effectiveness of the tool.

We will develop more adaptors and pre-formatting modules for ClusterProbe so as to enable more applications to share the resources information. And the global event mechanism will be extended to handle more complex failure.

References

[1] T. Anderson, D. Culler, D. Patterson, 'A Case for Now', *IEEE Micro*, February 1995, pp. 54-64.

[2] Eric Anderson and Dave Patterson, 'Extensible, Scalable Monitoring for Clusters of *Computers', The Proceedings of the 11th Systems Administration Conference (LISA'97)*', Oct 1997.

[3] Joel Apisdorf, Kevin Thompson, and Rick Wilder, 'OC3MON: Flexible, Affordable, High Performance Statistics Collection', *Proceedings of the 10th Systems Administration Conference* (LISA'1996), 1996, pp. 97-112

[4] Rajkumar Buyya, B. T. Koshy and R. Mudlapur, 'GARDMON: A Java-based Montioring Tool for Gardens Non-dedicated Cluster Computing System', *The International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'99), June 1999, pp. 2774-2780

[5] Rajkumar Buyya, K. Mohan and B. Gopal, 'PARMON: A Comprehensive Cluster Monitoring System', *The Australian Users Group for UNIX and OpenSystems Conference and Exhibition*, AUUG'98 – Open Systems: The Common Thread, 1998.

[6] Rajkumar Buyya, 'Single System Image: Need, Approaches, and Supporting HPC *systems', Proceedings of the Fourth International Conference on Parallel and Distributed Processing, Technique and Applications (PDPTA'97),* CSREA Publishers, 1997.

[7] K. Hwang and Z. Xu, 'Scalable Parallel Computing: Technology, Architecture, Programming', a graduated textbook, WCB/McGraw-Hill, New York, Feb 1998.

[8] G. Judd, M. Clement and Q. Snell, 'DOGMA: Distributed Object Group Management Architecture, System Overview', http://ccc.cs.byu.edu/DOGMA/System.html

[9] Javasoft, 'Java Remote Method Invocation – Distributd Computing for Java', http://java.sun.com/marketing/collateral/javarmi.html.

[10] Javasoft, 'Java Dynamic Management Kit: A WhitePaper',http://www.sun.com/software/java-dynamic/wp-jdmk/index.html.

[11] J. A. Kaplan, M. L. Nelson, 'A Comparison of Queueing, Cluster and Distributed Compuing Systems', Technical Report, RNS-94-006, NASA Ames Research Center, 1994.

[12] C. M. Lee, A. Tam, and C.L. Wang, 'Directed Point: An Efficient Communication Subsystem for Cluster Computing', *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems,* Las Vegas, October 1998, pp. 662-675

[13] Matchy J. M. Ma, Cho-Li Wang, Francis C. M. Lau, Zhiwei Xu, 'JESSICA: Java-Enabled Single System Image Computing Architecture', *The International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'99), June 1999, pp. 2781-2787

[14] C. Roder, T. Ludwig and A. Bode, 'Flexible Status Measurement in Heterogeneous Environment', *The International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'98), June 1998.

[15] Yudong Sun, ZhengYu Liang and Cho-Li Wang, 'A distributed Object-Oriented Method for Particle Simulations on Cluster', *High Performance Computing and Networking (HPCN) Europe 199*9, Apr 1999, pp. 245-253.

[16] Putchong Uthayopas et al., 'Interactive Management of Workstation Cluster Using WorldWide Web', *Cluster Computing Conference* (CCC'97)', 1997.