SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2000; **30**:1–17

PARMON: a portable and scalable monitoring system for clusters

Rajkumar Buyya*,†

School of Computer Science and Software Engineering, Monash University, Room 130, Bld 63, Clayton Campus, Melbourne, Vic 3168, Australia

SUMMARY

Workstation/PC clusters have become a cost-effective solution for high performance computing. C-DAC's PARAM 10000 (or OpenFrame, internal code name) is a large cluster of high-performance workstations interconnected through low-latency and high bandwidth networks. The management and control of such a huge system is a tedious and challenging task since workstations/PCs are typically designed to work as a standalone system rather than part of a cluster. We have designed and developed a tool called PARMON that allows effective monitoring and control of large clusters. It supports the monitoring of critical system resource activities and their utilization at three different levels: entire system, node and component level. It also allows the monitoring of multiple instances of the same component; for instance, multiple processors in SMP type cluster nodes. PARMON is a portable, flexible, interactive, scalable, location-transparent, and comprehensive environment based on client–server technology. The major components of PARMON are *parmon-server*—system resource activities and utilization information provider and *parmon-client*—a GUI based client responsible for interacting with parmon-server and users for data gathering in real-time and presenting information graphically for visualization. The client is developed as a Java application and the server is developed as a multithreaded server using C and POSIX/Solaris threads since Java does not support interfaces to access system internals. PARMON is regularly used to monitor PARAM 10000 supercomputer, a cluster of 48+ Ultra-4 workstations powered by the Solaris operating system. The recent popularity of Beowulf-class clusters (dedicated Linux clusters) in terms of price–performance ratio has motivated us to port PARMON to Linux (accomplished by porting system dependent portions of parmon-server). This enables management/monitoring of both Solaris and Linux-based clusters (federated clusters) through a single user interface. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: client–server; clusters; cluster monitoring; high-performance computing; Java

INTRODUCTION

During the 1980s, computer scientists believed that the performance of the computer could be improved by creating faster, more efficient processors. But in the 1990s, this idea is being challenged by the concept of clustering, which essentially means interconnecting two or more computers to perform

*Correspondence to: Rajkumar Buyya, School of Computer Science and Software Engineering, Monash University, Room 130, Bld 63, Clayton Campus, Melbourne, Vic 3168, Australia.
†E-mail: rajkumar@csse.monash.edu.au

Received 1 December 1998
Revised 21 December 1999
Accepted 22 December 1999

shared functions as a single system. The rapid advances in the areas of computing and communication technology (availability of commodity high-performance microprocessors and high-speed networks) facilitate this transition. These two enabling technologies are making clusters an appealing vehicle for parallel processing leading to low-cost *commodity supercomputing* [1]. Clusters aim at offering an infrastructure (hardware, software, or both) to view multiple computers as one system so that end users can use them without knowing on which computer they are actually working.

Monitoring of a cluster is a tedious and challenging task since typical workstations are designed to work as a standalone system rather than part of a workstation cluster. This can be made easier by software systems that support the monitoring of the entire system at different levels through an integrated GUI display. PARMON is one such system that allows the system administrator to monitor the entire cluster, a single node, a component of node, or activities of a single individual component. A node can have multiple instances of the same component types and PARMON supports the instrumentation of such multiple components individually.

As stated above, high-performance computing on commodity hardware is gaining wide acceptance. For this to be practicable, it is important that systems provide a single system image (SSI) at any one (or more) of the following levels.

- Hardware level.
- Operating system level.
- Message passing interface's level.
- Language/compiler level.
- Tools/application level.

The transparent access to a cluster computer is possible on a system exhibiting a single system image. The above approaches for achieving unified access to system resources have been discussed elsewhere [2]. In this paper, we focus on tools/application level SSI achieved through PARMON that offers a unified means (single window) to manage or monitor a large cluster of computers.

A large cluster consists of many computers from the same or different vendors, having different machine architectures and running different operating systems. In such a network of heterogeneous systems, the monitoring system must be portable. This is achieved by developing PARMON using the Java programming language. The motivations for using Java comes from its distinct features compared to other programming languages: 'Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded and dynamic language' [3]. Among these features, Java makes network programming easier by encapsulating connection functionality in socket classes [4]. Java is mostly used in writing distributed computing programs or GUI based programs because of its portability feature, and Java goes a lot further than most languages to obtain not just portability but identical program behavior on different platforms. Java is mainly intended for the development of object-oriented network based software for Internet applications. The networking, multithreading and GUI-based features of Java have been used extensively for developing the PARMON client.

C-DAC HPCC SOFTWARE ENVIRONMENT

C-DAC HPCC (high-performance computing and communication) software is an open and flexible parallel and distributed processing environment for a cluster of UNIX workstations [5]. The software

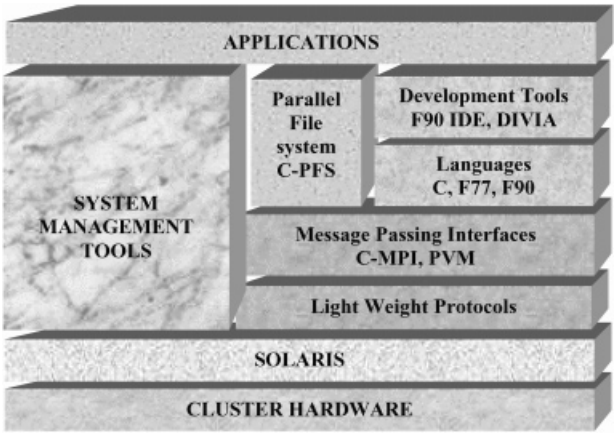

Figure 1. C-DAC HPCC software architecture.

architecture components shown in Figure 1 allows a collection of workstations to be viewed as independent workstations, a cluster of workstations, or a MPP (massively parallel processing) system connected through a scalable low latency and high bandwidth interconnection network.

The HPCC software allows users to develop and execute sequential, message passing and data parallel programs. An efficient MPI (message passing interface) implementation provides the necessary support for developing message-passing applications, while support for data parallel programs is provided through HPF (high-performance Fortran). High-performance communication protocols (such as active messages) and a rich set of program development, system management and software engineering tools are aimed at offering high performance services and usability.

C-DAC HPCC software is a complete solution for enterprises that need to create and execute parallel programs on UNIX clusters. The performance for both parallel and distributed applications is guaranteed through efficient implementation of lightweight communication protocols such as active messages. The effective system administration and management is supported through a tool called PARMON. It allows an effective monitoring and administration of the multiple resources of a large UNIX cluster.

A detailed discussion on the C-DAC HPCC environment can be found at the C-DAC web site (http://www.cdac.org.in/). The rest of the paper focuses on PARMON's architecture, monitoring of system resources and activities, implementation, related works and future directions.

PARMON OVERVIEW AND SYSTEM ARCHITECTURE

PARMON allows the user to monitor system activities and resource utilization of various components of workstation clusters. It monitors the machine at various levels: component, node and the entire

system level exhibiting a single system image. PARMON allows the system administrator to monitor the following.

- Aggregation of system resources utilization.
- Process activities.
- System log activities.
- Kernel activities.
- Multiple instances of the same resource.

PARMON allows the user to define events that trigger automatically whenever an event condition is satisfied. It also provides physical and logical views of the components of the system.

The important features of PARMON include the following.

- Exploits developments of latest technologies, hardware and software features for communication and imaging.
- Allows the user to build system database (nodes and groups) comprising node name, communication interfaces, and specific area of a disk to be monitored.
- Supports listing of system information and machine configuration of all nodes of a cluster.
- Allows instrumentation of system resources such as CPU, disk, memory and network, and their parameters, both at macro and micro level.
- Supports monitoring of cluster at node level, group level or entire system level, and thus exhibits a single system image.
- Allows the parallel execution of selected operations on a single or group of workstations, real-time and interactive resource monitoring (e.g. processor, memory, disk and network utilization), and normal maintenance tasks such as node or cluster shutdown.
- PARMON client is portable across all platforms supporting the Java runtime system, JVM (Java Virtual Machine) and PARMON server is portable across machines running Solaris or Linux.

PARMON architecture

PARMON consists of the parmon-client and parmon-server (parmond). The system model shown in Figure 2 follows the client–server paradigm with nodes to be monitored acting as servers and the monitoring systems or user-stations acting as clients. The cluster nodes can be monitored from any workstation, PC, or a node of the cluster itself.

The PARMON server is loaded on all the nodes that need to be monitored. A client requests the values of parameters through message passing and gets the server's response in the form of messages. The client interprets the messages and converts them into appropriate format for graphical visualization.

A client can either monitor all the nodes or selectively monitor a few nodes of the cluster. For effective monitoring, the concept of group is supported. A set of nodes forms a group and nodes are selected based on the allocation of resources to various user groups. Such a grouping mechanism helps in monitoring and gathering usability statistics, with which the system administrator can change the resource allocation strategy.

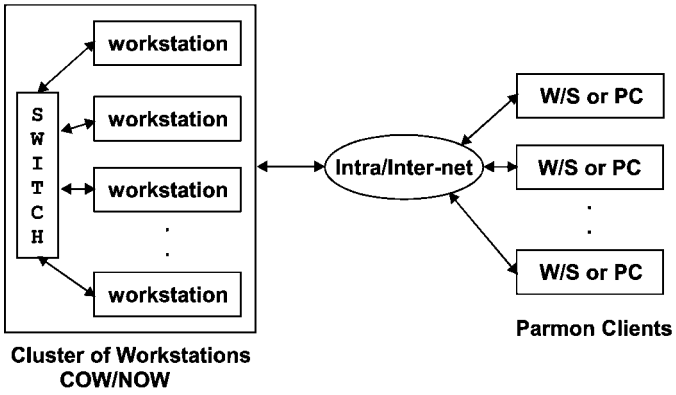

Figure 2. PARMON system model.

AN EYE INTO KEY PARMON FEATURES

In this section we discuss PARMON features for monitoring cluster system activities and resource utilization as this knowledge helps in understanding PARMON design and implementation model. The user interacts with PARMON through PARMON launcher shown in Figure 3 to monitor utilization and status of system resources such as memory and disk and system activities such as execution of a program [6]. It also allows software instrumentation of resource activity parameters through kernel-data-catalog.

Aggregation in visualization of resource utilization

The use of aggregation in visualization allows one to scale the visualization to a whole cluster. The administrator can create user-groups containing a set of nodes based on a resource allocation policy and monitor them using PARMON as shown in Figure 4. The same statistics for different nodes are combined to obtain a single statistic. This technique is called group/machine utilization. The results shown in Figure 4 can also be presented using pie charts or bar graphs in PARMON.

Process activities

The utilization of a CPU resource can be measured by monitoring process activities, which helps in identifying CPU/memory-intensive processes. The parameters that can be monitored using PARMON are: pid, command, uname, uid, nice, status, user (%), sys (%), total (%), total CPU (%), and start up time. PARMON continuously updates the process and system data at a user specified sampling interval. A snapshot of process activities is shown in Figure 5, and this information can be sorted based on selected parameters such as process-id, user name, etc. The information displayed by this appears similar to the `top` utility.

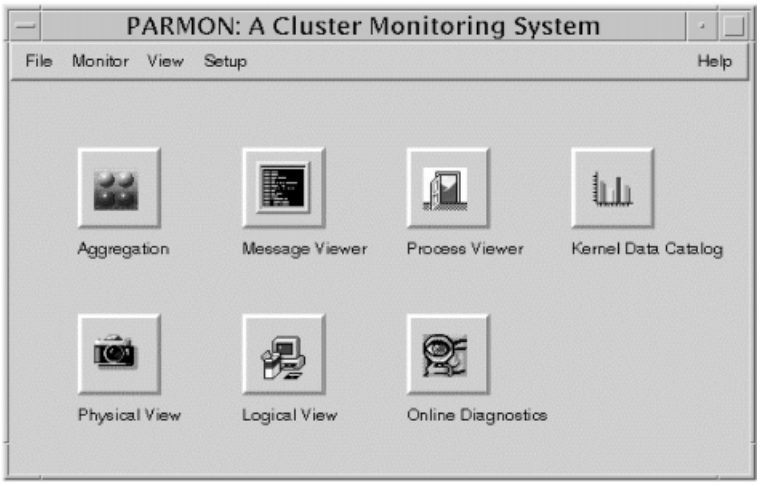

Figure 3. PARMON launcher.

Group Usage Matrix

File TimeSet

Group Name	Num Of Nodes	CPU Usage	Disk Usage	Mem Usage
Krish_Group	4	14%	98%	84%
Hardware	3	12%	98%	84%
HPCC_Group	4	12%	98%	84%
SSDG_Group	5	11%	98%	84%
Networking_Group	3	12%	98%	84%
RealTimeSystem	4	12%	98%	84%

Figure 4. Resource utilization by groups in a cluster.

System logs

PARMON helps to effectively monitor the system logs maintained by the operating system. It allows one to process system messages and syslog files for entries that occur at a specific time, or for entries that contain a specific keyword or word pattern.

Kernel activities

PARMON supports software instrumentation of system resources (such as CPU, memory, disk and network) and their activities. When a particular resource has more than one instance, PARMON allows

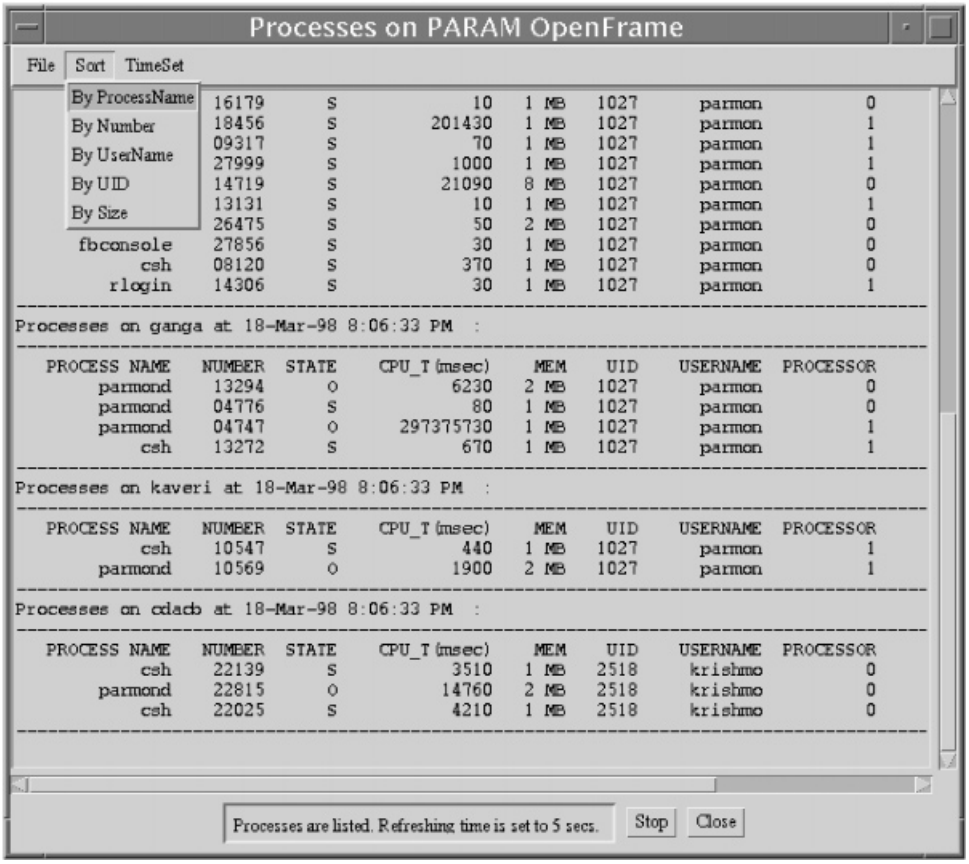

Figure 5. Information on resources consumed by processes.

monitoring of each resource instance individually. The invocation of the kernel-data-catalog option allows instrumentation of kernel activities related to resources such as CPU, memory, disk and network, as discussed below.

CPU parameters

The monitoring of CPU parameters helps the user to understand how the CPU is being utilized. PARMON allows the monitoring of number of mutexs, interrupts, context switches, system calls (Figure 6), forks, execs, page in, page out, swap-in and swap-out operations performed per second, and to visualize these activities using graphically. It also allows the monitoring of the process run queue, I/O queue and swap queue.

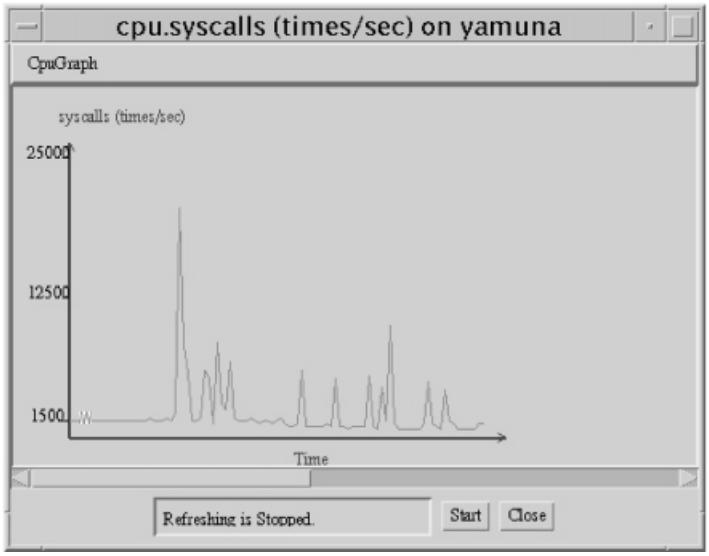

Figure 6. Systems calls generated by processes running on a node in a cluster.

Memory parameters

PARMON allows continuous instrumentation of memory availability, memory in-use, free memory, percentage of memory in-use, reserved swap space, allocated swap space and available swap space.

Disk parameters

PARMON allows the monitoring of disk operations such as reads, writes, number of jobs waiting in the queue for disk service, and disk request run-time and wait-time.

Network parameters

The software instrumentation of network parameters such as input packets, output packets, errors in packet transmission helps to detect network bottlenecks. PARMON supports the display of percentage of incoming and outgoing data packets containing packet format errors.

Component view: physical and logical

PARMON displays the entire system physical picture and a few important components, which helps the user to quickly understand the machine's physical look and feel. It also displays different views of

the machine. The picture files (static in nature) used in displaying physical appearance of the system are scanned image files of photographs taken using cameras.

The Logical View displays system components in a hierarchical manner dynamically. System components include processing elements, file system and network components, which are probed dynamically. Each item in the hierarchy diagram can be further probed for more detailed information. For instance, when the file system (fs) is probed, it shows all logical partitions of the disk and other details such as partition name, allocated disk space and disk usage, both in terms of bytes and percentage.

Device control

PARMON supports the control of multiple instances of the same resource, for instance, it allows CPUs in a SMP node to be set to on-line or off-line mode.

Events generation

PARMON allows the administrator to define events such as sending e-mail when the user crosses resource utilization limits. This helps the administrator to have effective control over system resource utilization and, therefore, change resource allocation policies.

Diagnostics

Under Solaris, PARMON integrates SunVTS to do validation, testing, and stress testing on external devices such as disk, tape drives and network connectivity. SunVTS helps to locate problems under Solaris with components that have managed to pass the configuration test (POST), as well as problems with external devices that POST knows nothing about.

Data representation

PARMON uses pie charts, bar charts and line graphs for representing resource usage such as disk and memory utilization as well as graphs for representing various kernel activity parameters of the CPU, disk, network, etc.

On-line help

PARMON offers a comprehensive online help facility. The user can choose a service-type option in the main help menu and then navigate to specific details of how each service can be accessed and interpreted. The online help also has a glossary of technical terms and parameters that are often used in PARMON or among the cluster computing community.

Miscellaneous features

The additional features supported by PARMON includes the following:

- A user specific command can be issued to selected nodes or all nodes in a cluster.
- The listing of users working on selected nodes or entire cluster.
- It supports the broadcasting of messages to a selected or all nodes in a cluster.
- It retrieves system information, configuration, and packages installed on nodes.

PARMON IMPLEMENTATION

The two major components of PARMON are *parmon-server*—system resource activities and utilization information provider and *parmon-client*—a GUI based client responsible for interacting with the parmon-server and users for data gathering in real-time and presenting information graphically for visualization. The PARMON client is designed, developed, and implemented using the state-of-the-art, object-oriented client–server and Java computing technologies. The PARMON-server is developed as a multithreaded server using POSIX/Solaris threads and C as Java does not support direct functionality for accessing system/kernel internals. The PARMON-client also maintains information related to cluster setup, groups and node location. This information needs to be created by a user so that PARMON knows about all those nodes that form a cluster or a group.

Protocols for interaction between client and server

Since PARMON is designed to run independent of other monitoring systems, it uses a simple mechanism (not any standard protocol) for transporting data between its clients and servers. The transport layer consists of connected socket (TCP/IP) streams and the application layer uses messages—character strings. The client retrieves required monitoring information from the server running on the cluster nodes by exchanging messages. The client request message normally contains a code indicating what it is looking for. The server reply message contains a series of strings, whose length varies from one request type to another.

The following interaction takes place between the PARMON user, client and server repeatedly for every action of the user:

(i) The user interacts with a PARMON client GUI for monitoring and selects the appropriate option.
(ii) The PARMON client interprets the user's request and invokes the appropriate event handler.
(iii) The event-handler performs the appropriate function. In most cases, it creates an object of user-defined service class—an extended `Frame` class—having the capability to interact with the server and the user.
(iv) The user selects a set of nodes to be monitored.
(v) The PARMON client maps the node name to IP address by accessing the node-database, and establishes a connection with the server by creating an object of the `Socket` class by supplying the IP address and parmon-socket number (default or user supplied). It also creates input and output streams for this socket object.
(vi) The PARMON client converts the user request into messages and communicates them to the server.
(vii) The PARMON server interprets the client messages, processes its requests, and communicates the results to the client.

(viii) The PARMON client interprets the messages and converts them into appropriate format; that is, it presents in the form of a graph, pie chart, bar graph, table, or in a text box.
(ix) The PARMON client then closes all connections made to the servers.

The implementations of the above steps with finer details are discussed in the following subsections.

PARMON client implementation in Java

The PARMON client is a GUI based tool responsible for interacting with the users and the PARMON servers. The communication between the PARMON client and server takes place through the sending or receiving of messages over TCP/IP sockets, and follows a simple PARMON-specific protocol.

A core of PARMON client class hierarchy is `ParmonSocket` that handles the mapping of node names to their physical addresses (IP) by accessing the node database, establishes a socket connection to the server, and creates input and output streams to the socket. It takes a list of strings representing node names and port number over which the client and server communicates as arguments. It also has the methods that allow sending commands and receiving node responses. A cluster node response for the PARMON client request/command for node status contains the following two items.

| **Request_Status** | **Data/Error** |

The *Request_Status* can be SUCC, STOP, or CONT and the client needs to interpret it as follows.

> **SUCC**: The server honors the client's request and the next item, *Data*, contains the result.
> **STOP**: The server could not serve the request from the client, so stop further requests through the current connection. The second item, *Error*, contains the error message and the reason for this error.
> **CONT**: The current request cannot be served and the remaining requests from the client can continue. The second item, *Error*, contains the error message and the reason for this error.

The parmon-server response message is handled by the user-defined class `ParmonSocket` methods `read()` and `response()`. After sending a request, the client invokes the method `read(int i)` that fetches the `node[i]` response message and returns the request status (*Request_Status*). Based on the value *Request_Status* (success or failure), the client interprets the contents of the message buffer (which can be extracted through `response()`—separates status condition from the response portion. The string returned by `response()` contains the actual result of the client request. The client then converts the string into multiple tokens and each token contains the data whose meaning is interpreted and processed by the client as per the agreed 'request-response' protocol between the parmon-client and the parmon-server.

Kernel Data Catalog window in Java

The Kernel Data Catalog window shown in Figure 7 allows one to select the node and type of parameters to be instrumented. This class extends the class `java.awt.Frame` and its layout is set to null by `setLayout(null)`, so that the components' positions can be controlled by the programmer. This window's resizable property is set to false by `setResizable(false)`.

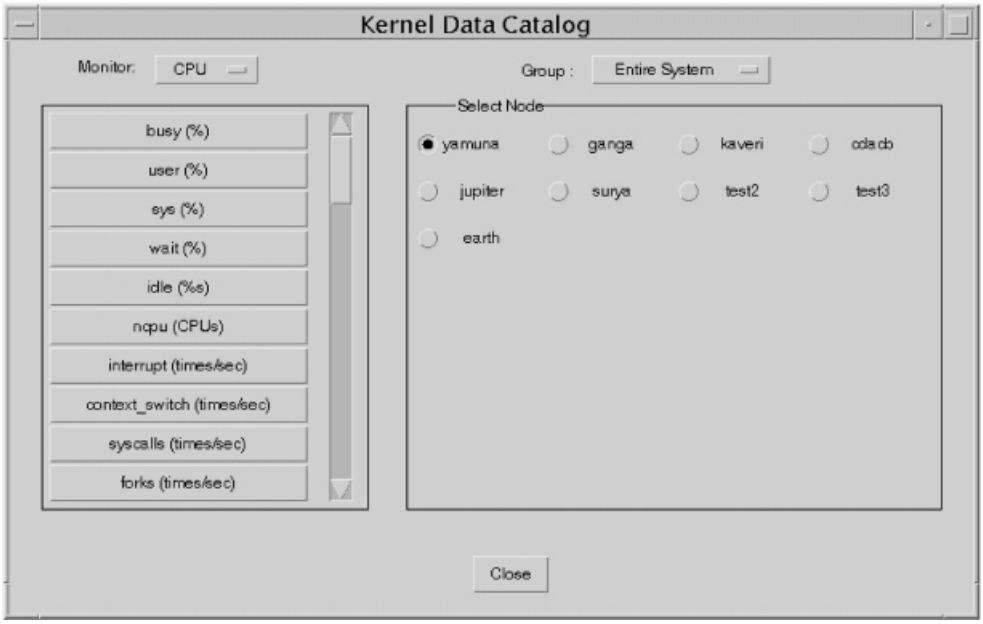

Figure 7. Kernel Data Catalog window.

There are two panels created and added to the kernel data catalog window. The left panel shows the parameters related to the resources that the user could instrument. The panel has a choice component whose items are CPU, disk, network and memory, and another panel contains buttons indicating parameters related to the resource selected in the choice component.

The right panel shows the group lists and the respective nodes in them. At any time, one of these nodes can be selected for monitoring resource activities. The required resource activity will be chosen from the left panel. This panel has a choice component and another panel with radio-buttons. The choice component has group names as its items. The group information will be obtained from the group database. The panel having radio-buttons represents nodes. Therefore, only one node can be selected for monitoring at any time. When the group is selected from the choice component, the nodes belonging to that group are displayed in the radio-buttons panel.

When the user clicks on any parameter, the event handler will be invoked by supplying event details. The event handler will determine the type of parameter to be monitored and the node name, and invokes/creates the appropriate method/object that performs instrumentation of the selected parameter by retrieving required information from the parmon-server running on the selected node.

Node and group database

The PARMON client maintains nodes and group information in `Nodes.db` and `Groups.db` files. The file `Nodes.db` maintains the information about nodes in the system for monitoring. The user

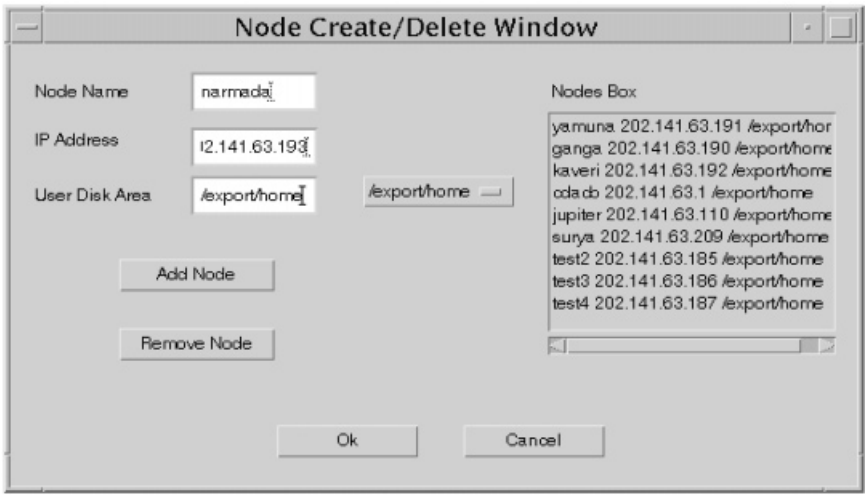

Figure 8. Node Setup window.

interface for creating a node database is shown in Figure 8. It requires the name of the server and its IP address. On the right there is a list of nodes. If the node has to be removed from this list, one selects the respective node name, clicks on the Remove Node button and finally confirms the changes to be incorporated into the `Nodes.db` file.

PARMON allows creation of user groups containing a set of nodes based on resource allocation and monitoring. The Group Create window allows the user to select a set of nodes that forms the user group. The group's details (group name and its nodes) are maintained in the `Groups.db` file. The Group Modify window offers the flexibility to change the configuration of the group by selecting new nodes or deselecting nodes from the list of nodes.

PARMON server implementation

The multithreaded implementation of servers allows them to respond to multiple client requests simultaneously with improved response time and throughput. The server creates a new thread for every request it receives and assigns the responsibility of serving the client requests to this new thread; it then proceeds to handle other requests waiting in the queue or to wait for new requests.

The PARMON server is a multithreaded server and it communicates with clients over the parmon-port (one of the free sockets reserved for parmon) and provides defined services as discussed below.

PARMON server socket

The PARMON server opens a connection endpoint by invoking `socket()` and binding it to a specified port or parmon-port through `bind()`. It blocks and waits via a client request for a connection

by invoking `listen()`. When the connect request arrives at the server machine, the server calls `accept()` to accept the connection over which the server and clients communicate. Once the connection is established, the server creates a service-thread responsible for serving the client requests and then continues with serving other requests.

Multithreading PARMON server

The PARMON server is multithreaded using the POSIX/Solaris threads interface to enhance its portability. However, only one of them is used for creating an executable file through conditional compilation. The PARMON server and client communicate with each other by exchanging messages. The client requests parameters through messages, the server creates a thread called a service-thread for each request by calling `thr_create()` for Solaris threads and `pthread_create()` for POSIX threads. The service-thread interprets the meaning of requests, processes them on the local node and communicates the results to the client. Both service-thread and client may communicate with each other more than once in order to complete the request. After serving the client request, the service-thread dies by releasing resources (such as buffers, sockets) occupied by it during its lifetime.

PARMON services and their implementation

The services provided by the server can be mainly classified into two categories: kernel related services, and system status and its configuration related services. The kernel related services focus on the activities of the CPU, disk, memory and network during process execution. The second type of services offered by PARMON includes information about process activities, system logs, users, physical and logical view, system information, and system configuration. The CPU parameters include busy (%), idle (%), sys (%), user (%), wait (%), context_switches, interrupts, system calls, forks, execs, pgin, pgout, run_queue_length, io_queue_length, etc. The disk parameters include operations such as reads, writes, disk wait percentage, disk active percentage, disk service percentage, etc. The memory parameters consist of available memory, free memory, used memory, swap avail, swap free, swap reserved, etc. Input packets, output packets, input error, output error, collisions, error percentage and defer percentage are some of the network parameters.

Kernel parameters

In Solaris, the `kstat` (kernel statistics) offers a general-purpose mechanism for retrieving kernel statistics. The kernel maintains a linked list of `kstat`s. Each `kstat` has a common header section and a type-specific data section and it can be characterized as belonging to some broad class of statistics; for instance, disk, net, vm, etc. This field can be used as a filter to extract related `kstat`s. Solaris offers a set of interface functions to access kernel statistics. The function `kstat_open()` initializes the `kstat` control structure, which provides access to the kernel statistics library. The invocation of `kstat_read()` fetches data from the kernel for the `kstat`, and `kstat_close()` frees all the resources that were associated with the pointer returned by the `kstat_open()`.

The physical memory parameters data can be obtained through `sysconf()`. Swap memory information is obtained through calls to kernel VM routines: The `kvm_open()` initializes a set of file descriptors to be used in subsequent calls to these routines. The `kvm_read()` transfers data from

the kernel image specified by the pointer returned by the `kvm_open()` function. The `kvm_close()` closes all the file descriptors associated with the pointer returned by the `kvm_open()` function.

System status and configuration

PARMON supports access to parameters related to system configuration, users, and processes that are implemented by using relevant system calls [7]. The `getutxent()` allows one to access information related to system users. The `readdir_r()` allows one to access information related to processes. The `sysconf()` allows one to access system configuration such as the maximum number of files that can be opened simultaneously or the maximum size of a core memory page. The `sysinfo()` allows one to access system information such as host name, release number, system architecture and vendor details.

RELATED WORKS

A number of projects are initiated to investigate the system administration of workstations and clusters. They include NOW (network of workstations) at the University of California, Berkeley; SMILE (scalable multicomputer implementation using low-cost equipments) at the Kasetsart University, Bangkok; Solstice SyMon^{TM} at the Sun Microsystems, and Alert at the University of Virginia.

The NOW system administration tool gathers and stores data in a relational database. It uses a Java applet as the primary interface, allowing users anywhere in the world to monitor the system from their browser [8].

The SMILE cluster management system (SCMS) is a Beowulf cluster administration tool [9]. Like PARMON, SCMS GUI allows the user to submit parallel command to all nodes, do system maintenance such as shutdown, and display static hardware configuration of all nodes such as CPU information. It uses shell scripts to extract the information required for cluster monitoring.

The SyMon allows users to monitor a standalone workstation and follows client–server technology for separating the node to be monitored and the monitoring station [10].

Alert consists of one or more monitoring and logging stations that function independent of each other [11]. Each monitoring station can print web pages, archive data, e-mail alerts and keep a log of errors and their eventual correction. Both Alert client and server are written in C, with the web data archiving/accessing scripts written in Perl.

The NSR (node status reporter) provides a standard mechanism for measurement and access to status information in clusters of workstations [12]. Parallel applications/tools can access NSR through its tool/NSR interface.

Unlike Berkeley NOW and Smile cluster monitoring systems, PARMON's user interface is developed as a Java application rather than an Applet. PARMON also allows the user to monitor the cluster system from anywhere in the world if the nodes of the cluster are accessible through the Internet. PARMON can also be used to monitor Beowulf-class Linux-based clusters. The depth of instrumentation of system parameters/activities supported by PARMON is rather extensive compared to other related tools.

While both appear to serve the same need—to simplify the job of the system administrator—Alert and PARMON differ slightly in both their purpose and implementation. PARMON appears to be an

easy-to-use tool for administrators to detect and correct problems through the same interface. Alert, on the other hand, takes a more passive role. The flow of information is one-way. While the administrator can use Alert to detect problems and (more recently) examine a history of problems, it does not provide a facility to administer machines.

CONCLUSIONS AND FUTURE DIRECTIONS

A good management system such as PARMON is crucial in exploiting clusters as a low-cost platform for high performance computing. PARMON aims to offer a single window to manage and control large homogeneous and heterogeneous clusters. It is being used to monitor PARAM OpenFrame supercomputer, which is a cluster of 48+ Ultra-4 nodes running the SUN-Solaris operating system. It has also been used in monitoring Beowulf-class clusters running Linux. Theoretically, PARMON is scalable across a cluster of hundreds of computers including an Internet-based global parallel cluster.

PARMON introduces some overhead on the network as its clients and servers communicate with each other by exchanging messages. The message size and the amount of time required to serve client requests depend on the request-type. There is no way to avoid this network overhead introduced by the PARMON, unless it follows a paradigm that is different from client–server.

Our experience in developing PARMON shows that Java offers a flexible environment for building scalable and extensible applications. PARMON can be easily extended to support new features as it has been designed using the latest object-oriented and Internet-based Java technology. The PARMON client is fully portable as it has been developed using Java and the server needs to be ported to support new platforms.

PARMON can easily be extended to support a web-based interface for monitoring. This can be achieved by converting the PARMON-client (which is a Java application) into an applet, and building a proxy server running on web-sever that acts as a link between the PARMON server running on the cluster nodes and the PARMON web-based user interface containing the applet. Security is another important issue that PARMON needs to address when web-based monitoring support is provided, as cluster system owners wants only authorized persons to monitor their systems and its activities.

The continuous popularity of Linux-based Beowulf-class clusters has prompted us to port PARMON to Linux. This is accomplished by porting only the system (OS) dependent part of PARMON, i.e. parmon-server, to Linux. The parmon-client runs on any platform without any change due to its Java implementation.

PARMON is scalable to the extent that it can be used to monitor clusters of clusters (global parallel cluster) as long as they are accessible. Both Solaris and Linux-based clusters can be monitored individually or combine at the same time using a single PARMON instance. The user can create a group for Linux cluster and another group for Solaris cluster, or mix their nodes to create a new group using the PARMON group creation and monitoring facility. It also allows the comparison of usage statistics of different/federated clusters.

We are also planning to port PARMON to Windows NT, as NT-based clusters are becoming popular. The availability of PARMON for Unix (Solaris), Linux and NT-based clusters makes it a truly heterogeneous cluster monitoring system.

ACKNOWLEDGEMENTS

The work presented in this paper was carried out at the Centre for Development of Advanced Computing (C-DAC), Bangalore, India. The PARMON project team comprised of Rajkumar Buyya (Project Leader), Krishna Mohan, and Bindu Gopal. Krishna Mohan and Bindu Gopal contributed to the development of PARMON client and server respectively. Mohan Ram (Program Manager) and Arun Babu (Group Coordinator) have contributed during the system specification and supported throughout the project life cycle. Vikas B V has contributed to the development of the PARMON user manual.

We are grateful to Dan Hyde (Bucknell University, USA), Colin Gan (WebWorks, Singapore), and anonymous reviewers for their critical comments on improving the quality of the paper. We thank Justin Moore (University of Virginia, USA) and Putchong Uthayopas (Kasetsart University, Thailand) for providing the information related to their cluster monitoring/management systems. We appreciate Rainer Fraedrich (University of Applied Sciences, Germany) for his comments on our work and making his workstation-monitoring tool available as a free open source [13] from which we have benefited.

The author acknowledges the support of the Australian Government International Postgraduate Research Scholarship (IPRS), the Monash University Graduate Scholarship (MGS), the Distributed Systems Technology Centre (DSTC) Monash Scholarship, and the IEEE Computer Society Richard E Merwin Scholarship.

REFERENCES

1. Buyya R (ed) *High Performance Cluster Computing: Architectures and Systems*, Vol. 1, 1st edn. Prentice Hall: NJ, 1999.
2. Buyya R. Single system image: need, approaches, and supporting HPC systems. *Proceedings of the Fourth International Conference on Parallel and Distributed Processing, Techniques and Applications (PDPTA'97)*. CSREA Publishers; Las Vegas, USA, 1997.
3. Sun Microsystems. Java language—a white paper. Sun Microsystems Computer Company, Palo Alto, CA, 1996.
4. Naughton P, Schildt H. *JAVA: The Complete Reference*. McGraw Hill Inc., 1997.
5. Mohan RN, Sasi KS, Bhatkar VP, Arora RK. HPCC software: a scalable parallel programming environment for UNIX clusters. *Proceedings of the Sixth International Conference on Advanced Computing (ADCOMP'98)*, Pune, India, 1998.
6. HPCC System Software PARMON Group. *PARMON User Manual*. © C-DAC (Centre for Development of Advanced Computing), Bangalore, India, 1998.
7. Goodheart B, Cox J. *The Magic Garden Explained: The Internals of UNIX System V Release 4*. Prentice Hall, 1993.
8. Anderson E, Patterson D. Extensible, scalable monitoring for clusters of computers. *Proceedings of the 11th Systems Administration Conference (LISA '97)*, San Diego, CA, October 1997.
9. Uthayopas P, Phaisithbenchapol S, Chongbarirux K. Building a resources monitoring system for SMILE Beowulf cluster. *Proceedings of the Third International Conference/Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA'99)*, Singapore, 1998. (http://smile.cpe.ku.ac.th/).
10. Sun Microsystems. *Solstice SyMON 1.1 User's Guide*. Palo Alto, CA 1996.
11. Moore J. *Alert—Cluster Monitoring Package*. University of Virginia, Charlottesville, Virginia 22903, USA. (http://www.cs.virginia.edu/~jdm2d/alert/).
12. Roder C, Ludwig T, Bode A. Flexible status measurement in heterogeneous environment. *Proceedings of the Fifth International Conference on Parallel and Distributed Processing, Techniques and Applications (PDPTA'98)*. CSREA Publishers: Las Vegas, USA, 1998.
13. Fraedrich R. *System Monitor Xprocmon*. http://www.fh-friedberg.de/xprocmonE.html

