# GARDMON: A Java-based Monitoring Tool for Gardens Non-dedicated Cluster Computing System

**Rajkumar Buyya**
School of Computer Science and Software Engg.
Monash University
Clayton Campus, Melbourne, Australia
http://www.dgs.monash.edu.au/~rajkumar/

**Binu Thomas Koshy and Rajesh Mudlapur**
School of Computing Science
Queensland University of Technology (QUT)
GP Campus, Brisbane, Australia

**Abstract:**

*The QUT's Gardens project aims to create a virtual parallel machine out of a network of non-dedicated computers (workstations/PCs). These systems are interconnected through low latency and high bandwidth communication links such as Myrinet. Gardens is an integrated programming language and system designed to utilize the idle workstation's CPU cycles to support adaptive parallel computing.*

*A Gardens computation consists of a network of communicating tasks, dynamically mapped onto a network of processors. Tasks are created dynamically and each task consists of a stack and a collection of heap segments in which dynamic data structures are stored.*

*We designed and developed a Gardens cluster monitoring system called Gardmon. It is a portable, flexible, interactive, scalable, location-transparent, and comprehensive environment for monitoring of Gardens runtime activities. It follows client-server methodology and provides transparent access to all nodes to be monitored from a monitoring machine. The features of Gardmon in monitoring Gardens adaptive parallel computing system seem satisfactory.*

## 1. Introduction

We have seen a dramatic increase in workstation performance in the last few years, with workstations doubling in performance every 18-24 months. This is likely to continue for several years, with faster processors and multiprocessor machines.

The recent advances in high speed networks and improved microprocessor performance are making clusters or networks of workstations an appealing vehicle for cost effective parallel computing. Clusters built using commodity hardware and software components are playing a major role in redefining the concept of supercomputing [1].

Studies have shown that over half of the workstations' CPU cycles are unused even during peak hours. One possible use of these idle cycles is to run parallel applications on a network of computers.

Idle workstations represent a considerable computational resource as yet untapped. Gardens is an integrated programming language and system designed to utilize such resources; it supports parallel computation across networks of otherwise idle workstations. Gardens is targeted at workstation networks which utilize state-of-the-art communication networks such as Myrinet. Such systems have the potential to provide supercomputer levels of performance. Thus Gardens enables a virtual supercomputer to be dynamically constructed from idle workstations [2].

A Gardens computation consists of a network of communicating tasks (unit of work), dynamically mapped onto a network of processors. Tasks are created dynamically and each task consists of a stack and a collection of heap segments in which dynamic data structures are stored [3]. Heap segments are not shared between tasks; they are partitions of a global virtual address space, which is partitioned across processors. Thus tasks occupy disjoint regions of an address space partitioned across processors (see Figure 1).

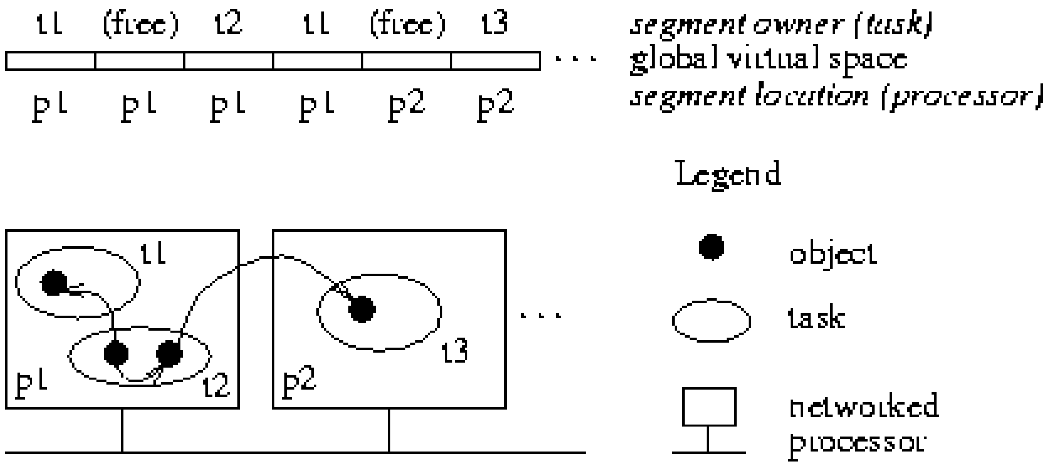


**Figure 1** Heap Segments, Tasks, and Processors in Gardens System. (*Source [2].*)

Tasks of a parallel application adapt themselves to a changing set of workstations under the control of a GATE (Gardens Tasking Environment) and this task migration machinery is described in detail in [3].

## 2. GARDMON Overview

The two major components of Gardmon are: ***gardmon-server***—system resource activities and utilization information provider and ***gardmon-client***—GUI based client responsible for interacting with gardmon-server and users for data gathering in real-time and presenting information graphically for visualization.

We have designed two types of servers: one for monitoring node operating system (called UnixMon) and another for monitoring Gardens system. *The capabilities/features of UnixMon are subset of PARMON monitoring system discussed in [4]*. The parameters that can be monitored through UnixMon are listed in Table 1.

| Service Name | Service Description | Type of Display |
|---|---|---|
| ***Users*** | This service provides the information about the users, the time since they have logged, user name, process id. | Textual |
| ***Processes*** | This function acts like a `top` command in Unix. It displays all the processes in the system. | Textual |
| ***Mem_free_perc*** | This service provides information about the physical memory that is free (%). | Graphical |
| ***Mem_used_perc*** | This service provides information about the physical memory that is used (%). | Graphical |
| ***Idle_perc*** | This service provides the information about the idle percentage of the CPU. | Graphical |
| ***Wait_perc*** | This service provides the information about the wait percentage of the CPU. | Graphical |
| ***Kernel_perc*** | This service provides the information about the kernel usage percentage of the CPU. | Graphical |
| ***User_perc*** | This service provides the information about percentage of the CPU utilized by user. | Graphical |

**Table 1** Cluster Node Parameters Monitored through GARDMON.

| Service Name | Service Description | Type of Display |
|---|---|---|
| ***Heap Count*** | Provides the information about the number of heaps that are available on a particular node. | Graphical |
| ***Heap Details*** | For all live tasks, provide information about the heap such as heap size, initial free block, next free block and their memory locations. | Textual |
| ***Heap Usage*** | Provides the information about the total amount of heap that is allocated to a particular task | Graphical |
| ***AllLiveTasks*** | For all the live tasks, provides the information about heap size, location of the heap, initial free block that is available, size of that block and its memory location. | Textual |
| ***NumOfTasks*** | Provides the total number of tasks that are running on each node. | Graphical |
| ***StackUsage*** | Provides the information about the amount of stack that is allocated to a particular task. | Graphical |
| ***StackDetails*** | Provides the information about the stack, such as, stack size, stack memory location and the parent object. This information is provided for an individual task. | Textual |
| ***Status*** | Provides the information about the status of individual task at any particular instance. | Textual |
| ***Memory Usage*** | Provides the information about the heap fragmentation. | Graphical |
| ***GetReadyQ*** | From the ReadyQ, gets the information about all the threads that are waiting for the execution. It also provides the information about the amount of memory that is allocated to individual thread. | Graphical |

**Table 2** Gardens System Parameters Monitored through GARDMON.

The Gardmon client is designed, developed, and implemented using the state-of-the-art object-oriented, client-server, and Java computing technologies. The Gardmon-server is developed as a multithreaded server using Gardens language (an extension of Oberon language to support parallel computing). The cluster node monitoring-server has been developed in C language. The gardmon-server supports monitoring of Gardens system's related parameters listed in Table 2.

## 3. Related Work

There are many projects investigating system administration of clusters that support parallel computing, including:

- PARMON—from Centre for Development of Advanced Computing, India—is a comprehensive environment for monitoring large cluster [4]. It uses client-server technology to provide transparent access to all nodes to be monitored. Its two major components: *parmon-server*—system resource activities and utilisation information provider and *parmon-client*—a Java application capable of gathering and visualizing real-time cluster information.
- The Berkeley NOW system administrations tool gathers and stores data in a relational database. It uses a Java applet to allow users to monitor a system from their browser [5].
- The SMILE (Scalable Multicomputer Implementation using Low-cost Equipment) administration tool is called *K-CAP* [6]. Its environment consists of *compute nodes*—these execute the compute-intensive tasks, a *management node*—a file server and cluster manager as well as a *management console*—a client that can control and monitor the cluster. K-CAP uses a Java applet to connect to the management node for monitoring.

- Solstice *SyMon*—from Sun Microsystems—allows standalone workstations to be monitored.
- The Node Status Reporter (NSR) provides a standard mechanism for measurement and access to status information of cluster [8]. Parallel applications/tools can access NSR through the NSR Interface.

## 4. Architecture and Implementation

GARDMON follows client-server architecture. The gardmon-server resides on all the nodes that comprise a cluster and client can reside on any cluster-node or non-node (not part of the cluster). The gardmon-server, which is responsible for monitoring Gardens system becomes part of the Gardens Tasking System. The architecture and interaction between various components of gardmon is shown in Figure 2. For simplicity, gardmon-server is shown as running on only one node of a cluster. In reality, it should be loaded on all the nodes of a cluster to be monitored.

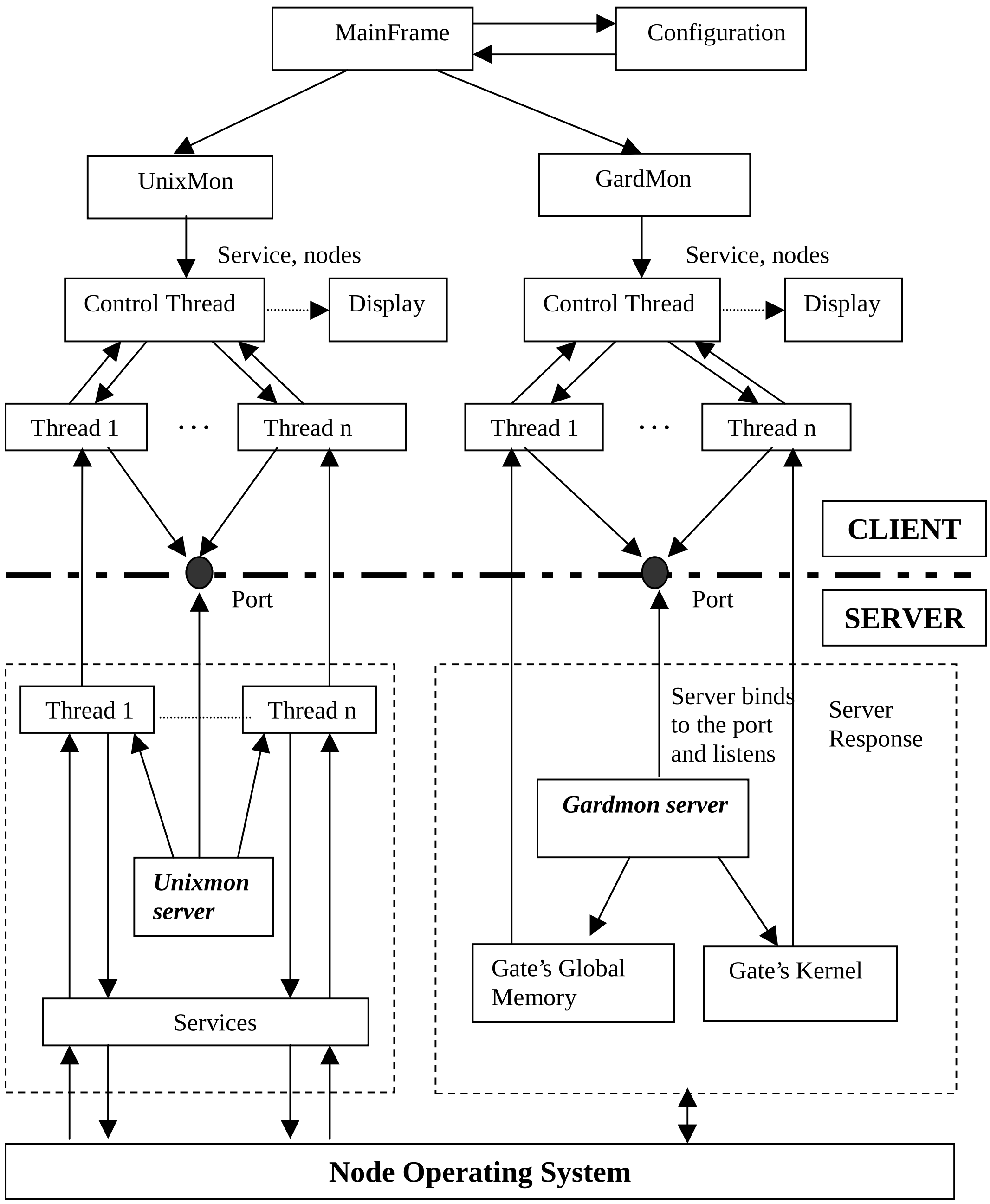


**Figure 2** GARDMON Components and their Interactions.

## Gardmon Components Interaction

The knowledge of the mechanism of interactions between various components and actions that takes place between them (see Figure 2) will help in visualization of the architecture and implementation. Hence, we discuss the same in the following paragraphs assuming that *gardmon-server is running on all the nodes of a cluster to be monitored*.

When gardmon is started, it creates a *Main-Frame* displaying three options related to:

1. Open the configuration screen
2. Open the GARDENS monitoring screen
3. Open the Unix OS monitoring screen

While starting the client, we need to enter the port numbers (it should be the same as the port number entered while starting the gardmon-server) for each of the servers over which client communicates to them. From the main screen we have the option of starting either of the monitoring screens, or the configuration screen. When we run the application for the first time, we have to start up the configuration screen where we enter the names of the nodes to be monitored. These selected nodes are then stored in a file, which can be accessed in future cases, thus the user need not enter the contact details of nodes to be monitored each time the application is executed. Once the nodes have been selected, user can choose either of the monitoring screens and perform the monitoring. The client has been designed in such a manner that the user is allowed to open multiple monitoring displays simultaneously. From each monitoring screen, the user is presented with an array of services, which can be invoked to monitor activities of the selected or all the nodes.

Once the service has been selected, the client creates a thread called the control-thread and passes the details of the service to be monitored. It then fetches contact details (node IP address, etc.) of nodes to be monitored stored in node configuration-file. The user can monitor multiple parameters simultaneously. For each service, the client creates a separate instance of the control thread responsible for communicating with the concerned nodes.

Within the control thread, the number of nodes to be monitored are known, the control thread creates a proportional number of request threads to communicate with the servers running on each of the nodes, i.e., for each node to be monitored a request thread is created. This request thread connects to the server using sockets at the specified port number and then sends the service request, the server on receiving the request, processes it and then sends the values back to the request thread and then closes the connection. The request thread takes in the value and sends it back to the control thread, then it closes all the connections with the server and kills itself.

Thus for every request, a new request thread is created which stays alive for only one set of transaction with the server. It should be noted that the server is asleep all the time, it only becomes active when it receives a request from the client, which it processes and after sending back the values, it kills the connection and becomes inactive till the next request.

Once the control thread receives the values from the servers of the nodes being monitored, it manipulates the data according to a predefined set of calculations and then presents the results (either in graphical or textual form).

## A Brief Discussion on Implementation

Gardmon server is implemented as multi-threaded-server, which allows responding multiple clients' requests simultaneously with improved response time and throughput. As indicated earlier, Unix monitoring server is developed using 'C' language and POSIX threads. Whereas, Gardens monitoring server is developed using Oberon with extended parallel programming features.

The various components of gardmon-server and its interactions with Gardens tasking system are shown in Figure 3. GO2Memory (maintained by GATE) is the global memory area, which stores all the parameters that are required for monitoring the task activities during run time. When the Gardmon server receives a request from the client, a service thread is created. Service thread is of the type *'kernel thread'*, since request from the client has to be processed immediately. Any task that is to be processed will be placed at the end of the ReadyQ tail, hence the probability of providing the service to the requested client at a particular instance will be reduced. Thus, by using the thread of type *'kernel thread'*, that task will be placed in the front of the ReadyQ and processes the request immediately.

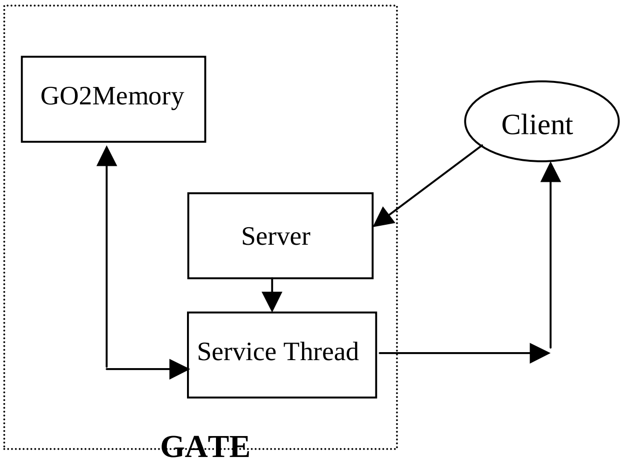


**Figure 3** GATE and Gardmon Server.

### GATE Data Structures

GATE maintains various data structure related to Gardens task, threads, heap, and thread queue. The data structures accessed include:

```
KernelTypes.Task
KernelTypes.Thread
HeapList
freelist.GetFreeBlkList()
ReadyQ
task.threadList
```

The instances of above data structures are accessed by gardmon-server to provide information related to:

- Tasks
- Threads
- Heap
- Stack
- Task Queue
- Thread Queue

The Gardens related parameters that Gardmon allows to monitor are shown earlier in Table 2. The `Task` and `heapList` data structures can be accessed to find out the total number of live tasks that are running on a Gardens system. The gardens-sever on each node, gets a pointer to the last task in task-list using: `task:=S.VAL(Task,heapList[hpidx])` where `task` is a variable of type `KernelTypes.Task`; `hpidx` is used to find a particular task in which a counter is maintained to count the total number of live tasks. The client periodically makes a request for this information (based on the user choice) and presents the number of Gardens tasks running on each node as shown in Figure 4.

In the implementation of Gardmon client Java's AWT features and sockets have been used extensively. The main class skeleton is shown in Figure 5, but due to space limitation the implementation details are not discussed.

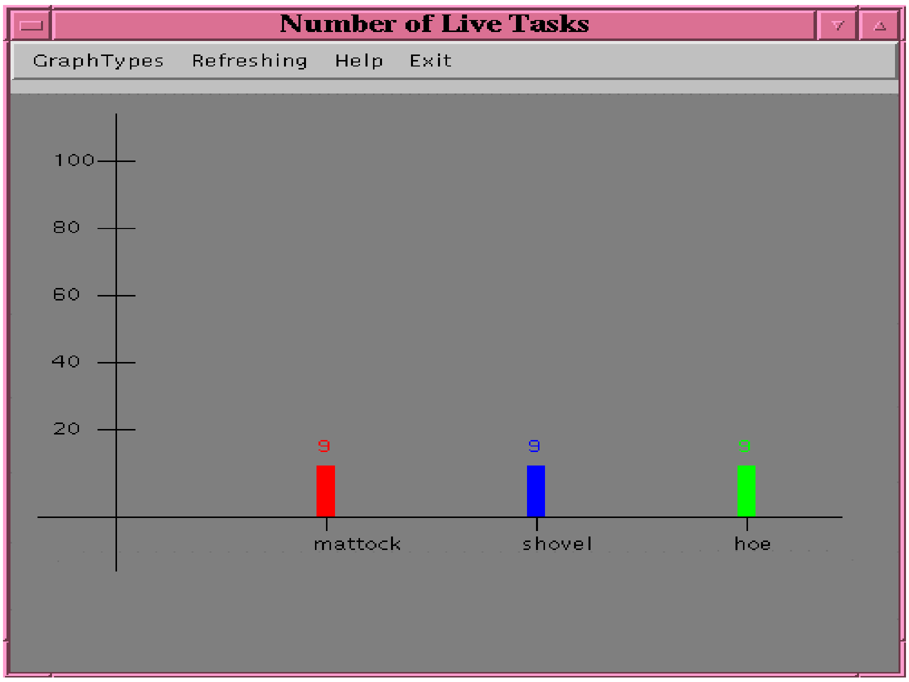


**Figure 4** A snapshot of the number of Live Tasks in Gardens Cluster.

```
:MainFrame
Data :
 private int Gport, Uport;
 private String filename = new String("/home/n2231905/node.dat");
 . . .
 Button unxmon_button, gardmon_button, config_button, close_button;
References:
 Node addnodes;
 . . .
 UnixClient unxmon; GardClient gardmon;
Methods:
 public MainFrame( int Gardensport, int Unixport)
 public void actionPerformed( ActionEvent e)
 public void setNodeDetails(String[] nodes, String[] IP, int port1, ...)
 ...
 public static void main (String args[])
```

**Figure 5** Gardmon Client Main Class Skeleton.

## Conclusions

The features of Gardmon in monitoring Gardens adaptive parallel computing system seem satisfactory and useful. It introduces a small network-overhead as its client and server communicates by exchanging messages. The services provided by Gardmon are useful for administering the activities of a large number of interconnected computers forming a cluster. That is, with just a mouse click, the entire cluster activities can be monitored through a single point of control (window). Thus, avoiding the cumbersome of monitoring each node explicitly. The information provided by Gardmon is also useful for debugging the Gardens system.

Gardmon can be easily extended to support web-based user interface for monitoring. It can be extended to provide a mechanism for load balancing.

## Acknowledgments

The work is carried out at Programming Languages and Systems (PLAS) Research Centre, School of Computing Science, Queensland University of Technology, Brisbane. We thank Prof.Clemens Szyperski, Director of PLAS Research Centre, for his advice and support during the design and development. We thank and acknowledge the enthusiastic support of Paul Roe, S.Y. Chan, and Nick Kwiatkowski during the implementation. We thank Toni Cortes, Hai Jin, and K. Mohan for their comments on the paper.

## References

1. Mark Baker and Rajkumar Buyya. *Cluster Computing: The Commodity Supercomputing.* Journal of Software - Practice & Experience, John Wiley & Sons, Inc, 1999. (to appear.)
2. Paul Roe and Clemens Szyperski. *The Gardens Approach to Adaptive Parallel Computting*, In *High Performance Cluster Computing: Systems and Architectures*, R. Buyya (ed.), 1/e, Prentice Hall, NJ, 1999.
3. A. Beitz, S-Y. Chan, and N. Kwiatkowski. *A Migration-Friendly Tasking Environment for Gardens*. In Fourth Australasian Conference on Parallel and Real-Time Systems (PART), Newcastle, Australia, Springer, 1997.
4. R. Buyya, K. Mohan and B. Gopal. *PARMON: A Comprehensive Cluster Monitoring System.*, The Australian Users Group for UNIX and Open Systems Conference and Exhibition, AUUG'98 - Open Systems: The Common Thread, Sydney, Australia, 1998
5. Eric Anderson and Dave Patterson. *Extensible, Scalable Monitoring for Clusters of Computers*. Proceedings of the 11th Systems Administration Conference (LISA '97), October 26-31, 1997, San Diego, California, USA.
6. Putchong Uthayopas et. al. *Interactive Management of Workstation Clusters Using World Wide Web*. Cluster Computing Conference (CCC'97), http://www.mathcs.emory.edu/~ccc97/
7. Sun Microsystems. *Solstice SyMON 1.1 User's Guide*, Palo Alto, CA 1996.
8. C. Roder, T. Ludwig, and A. Bode. *Flexible Status Measurement in Heterogeneous Environment*. Proceedings of the PDPTA'98 conference, Las Vegas, USA.