Design and Implementation of a Monitoring and Scheduling System for Multiple Linux PC Clusters*

Chao-Tung Yang†, Chun-Sheng Liao‡, and Ping-I Chen

High-Performance Computing Laboratory
Department of Computer Science and Information Engineering
Tunghai University, Taichung, 40704, Taiwan R.O.C.
email: {ctyang, g932834}@thu.edu.tw

Abstract

Managing and monitoring a cluster is both a tedious and challenging task, since each computing node is designed as a stand-alone system rather than a part of a parallel architecture. Beowulf systems will need a richer set of software tools to improve usability and re-configurability. In this paper, a software system that allows the centralized administration of a generic Beowulf cluster is proposed. This system also provides web services and applications to monitor large-scale clusters with task scheduling.

Keywords. *Cluster computing, Monitoring System, Scheduling System, Multiple Linux PC Clusters.*

1. Introduction

As the performance of commodity computer and network hardware increases, and their prices decrease, it becomes more and more practical to build parallel computational systems from off-the-shelf components, rather than buy CPU time on very expensive supercomputers. In fact, the cost performance ratio of a Beowulf cluster platform is between three to ten times better than that for traditional supercomputers [2, 3, 4]. The Beowulf architecture scales well, and it is easy to construct. In addition, one only pays for the hardware, since most of softwares are free. Beowulf is a multi-computer architecture which can be used for parallel computations. It is a system, which usually consists of one server node and one or more client nodes connected together via Ethernet or some other type of network, such as SCI, Myrinet and Infiniband. It is a system built using commodity hardware components, like any PC capable of running Linux, standard Ethernet adapters, and switches [1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12].

Monitoring the status of a Beowulf-style cluster platform can be a daunting task for any system administrator, especially if the cluster system consists of more than a dozen computing nodes. Since Linux is not absolutely stable, hardware problems can cause nodes to crash or become inaccessible, and chasing down problem computing nodes in a 500-node cluster is painful. Managing and monitoring a cluster is both a tedious and challenging task, since each node is designed as a stand alone system rather than a part of a parallel architecture. Beowulf systems will need a richer set of software tools to improve usability and re-configurability [5, 13].

As the PC cluster becomes a popular low cost high-performance computing platform, it is hard to manage, mainly due to the lack of powerful migration and monitoring tools. This research paper presents our efforts to resolve this problem by developing a PC cluster monitoring system. Still in this system, it also provides web services and applications to monitor large-scale clusters with task scheduling.

In the second generation Beowulf cluster, the master node can perform the migration of uncompleted tasks during runtime and this is the point where we start. In addition, we will also present how we can migrate tasks dynamically during runtime. We introduce an original algorithm to arbitrate threads between job executive and job listener, as it used signal programming technique to notify the main job executive when the job is submitted and our scheduler could be modified to keep thread-safe for synchronicity. We also examined the special properties of thread in Linux operating system for our implementation.

The remaining of this paper is organized as follows. In section 2, we make a background review. In section 3, we discuss the system and software architecture of our design. In section 4, we

* This work is supported in part by National Science Council Taiwan, under grant no. NSC94-2622-E-029-002-CC3.
† The corresponding author
‡ email: csliao@nchc.org.tw

demonstrate some examples of system utilities, and finally in section 5, conclusions are presented.

2. Background

A Beowulf class cluster consists of PCs, based on AMD and Intel x86, Compaq Alpha, Power PC processor architectures. Other components are Pure Mass-Market COTS. Typically they use a UNIX-like operating system; such as Linux, BSD, or Solaris. Message passing is normally used for communications between nodes; typically using MPI, PVM, or BSP. The clusters are set up and run as single user environments, which are optimized for the applications being executed.

Several tools have been developed to monitor a large number of machines as stand-alone hosts as well as hosts in a cluster. These tools can be useful because they monitor the availability of services on a host and detect if a host is overloaded, but they do not generally provide performance monitoring information at the level of detail needed to tune the performance of a Beowulf cluster. In contrast to existing systems, which usually display information only graphically, our project integrates performance monitoring with scheduling systems. In the following sections, we discuss open-source cluster-monitoring tools.

Ganglia is an Open Source project (available on SourceForge at http://ganglia.sourceforge.net) with a BSD license. It grew out from the University of California, Berkeley, Millennium Cluster Project (see http://www.millennium.berkeley.edu) in collaboration with the National Partnership for Advanced Computational Infrastructure (NPACI) Rocks Cluster Group. Ganglia provide a complete, real-time monitoring and execution environment based on a hierarchical design. It uses a multicast listen/announce protocol to monitor node status, and uses a tree of point-to-point connections to coordinate clusters of clusters and aggregate their state information. Ganglia uses the eXtensible Markup Language (XML) to represent data, eXternal Data Representation (XDR) for compact binary data transfers, and an open source package called RRDTool for data storage (in Round Robin databases) and for graphical visualization.

The *SMILE Cluster Management System* (SCMS) is an extensible management tool for Beowulf clusters. SCMS provides a set of tools that help users monitor, submit commands, and query system status; maintain system configuration, among others. System monitoring is limited to heartbeat-type measurements.

The *Network Weather Service*, although not targeted at Beowulf clusters, is a distributed system that periodically monitors and dynamically forecasts the performance various network and computational resources can deliver over a given time interval. The service operates a distributed set of performance sensors (network monitors, CPU monitors, etc.) from which it gathers system condition information. It then uses numerical models to generate forecasts of what the conditions will be for a given time frame. NWS is used for various meta-computing systems such as Globus.

3. System Design

The concept of our system is to improve the availability of monitor system in the distributed computing environment. Nowadays, the monitor system is not well developed on user requirements. Therefore, we started on the user interaction and the manner of application executing, and developed the applications by the portability of the Java Virtual Machine. Our system can be divided into three applications of the Cluster architecture.

- **Observe server:** The role of the Observe server is to run the collect daemon that gets the information of each cluster’s total information observed from the master node and replicate the data to their local file based database for the usage of the web interface.
- **Master node:** The master node executes the master daemon that could collect the information of their slave nodes to their local file based database and response the Observe server.
- **Slave node:** All other nodes of our cluster must run the slave daemon. The slave program must get information in user specific metrics like CPU speed, available size of memory, load of this node and other information user interested in.

On the other side, we have a separate role to display and control our cluster in another way, there are two types of this role:

- **Web portal:** We use two tools that generate the web service for controlling and presenting the information of our system. The former tool used is a drawing tool named Round Robin Database Tool (RRDtool), which draws the state chart from the collected information in the Observe server. The latter is the web front-end portal created by PHP. When user is connecting to the portal, he can retrieve the information by the state chart and control the system by the web interface.
- **PDA application:** The mobile devices are not suitable for displaying detail information and remote controlling. We need to simply our information and design the appropriate interface for this usage. The Java application framework is suitable for this type of application and we choose it to develop our simple application. Implementation of this work is to connect the Observe server, get all

metrics of our information and directly present these to a classified format.

Our system has been implemented for nodes within a private network; it resides on one node, which controls all the others with remote commands. This choice allows easy installation and upgrade, and it needs to have daemons running on computing and service nodes. On the other hand, this choice can scale if the number of nodes is huge. The software has been implemented for managing a cluster of clusters, on public networks.

The flow of this system is shown in Figure 1, where the master nodes can collect the system information form its slave nodes in the multiple Linux PC clusters. The Observe server will gather all information from master nodes, and send to Web server for displaying form remote users and applications. The system architecture and software architecture are shown in Figure 2. Also, the Observe server will be called to provide services and information form the Web server.

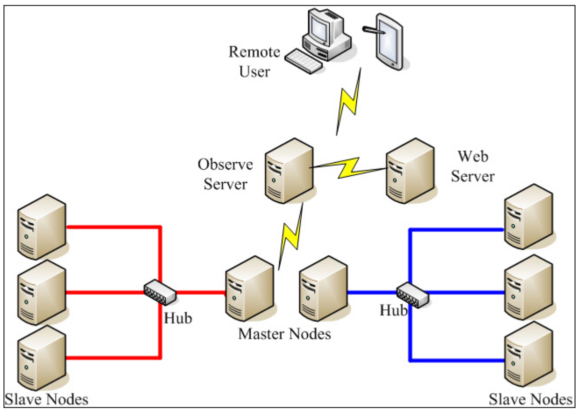
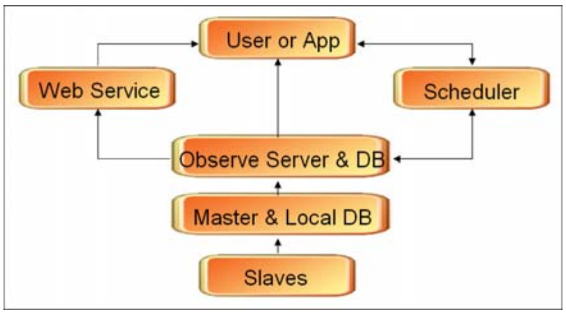


Fig. 1. System overview.

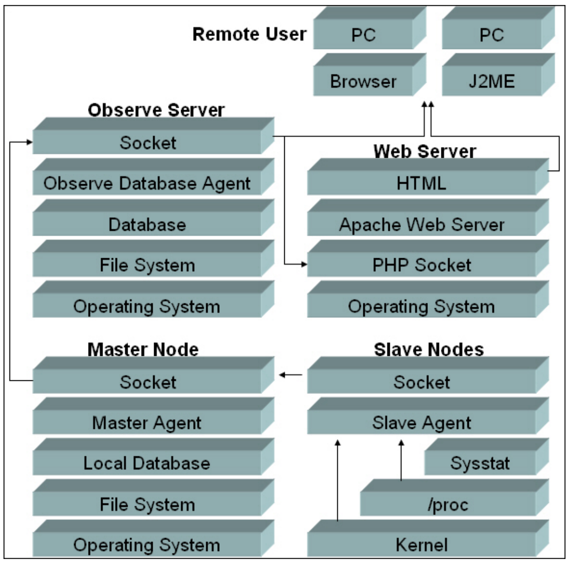
The functions of three daemons in our system are listed as:

1. Slave Daemon: The Slave daemon can obtain the related system information of each slave node from Kernel, and provide the services to its Master node in the PC cluster.
2. Master Daemon: The Master Daemon is responsible to collect the system information from all slave nodes into cluster, and put the related information into Local Database. Local Database is used for the purpose that will not allow the loading too high of master node for an instant. The function of Local Database can be view as a buffer.
3. Collect Daemon: This daemon is running on the Observe Server. It is used for collecting the system information of each master node of multiple Linux PC clusters. It can provide services by using database to applications or the remote users.

The system architecture of scheduler is shown in Figure 3. We implement the multiple priority queue of the scheduler and a user can specify the priority value for his job when submission. Our scheduler is based on the multi-thread architecture and specifically, this type of scheduler can improve the performance of job scheduling and the usage of the memory. The Receive Thread is the service of job submission, when job submitted; it was send to different queue by user specification. The scheduling algorithm is based on the weighted Round-Robin algorithm. The scheduling of each queue is Round-Robin, and the job chosen is through the user specific priority value. For the information of job execution, we could find it out thought the email notification service or through our web portal. In our system, each cluster has one scheduler and it could help them control their own jobs by classification.



(a)



(b)

Fig. 2. System and Software Architecture (a) System architecture (b) Software architecture.

In Figure 4, we would understand levels of the scheduler and how it works in the scheduler application. We define the Main thread as the level 1, and level 2 is the thread created by the level 1 and so on. The processing of the levels can divide to three parts:

1. The Server Thread created by the Main Thread is the level 2 thread. When it accepted the jobs after connection. It linked to the

Level 3 thread called Queue Handler and the Queue Handler stores them.

2. The Main Thread creates the Job Execution Thread. The Job Execution Thread transmits the job to the appropriate node or cluster for execution by their system state. It is the level 2 thread.
3. The Main Thread creates the level 2 thread called Status Thread for presenting the information of the specified queue. It is the level 2 thread and not shown on the Figure 4(a).

Figure 4(b) shows the queue is waiting when no tasks are in it. When the task accepts the job through the Queue Handler in the network, the scheduler is waked up by the signal, and the scheduler starts to choose a job for execution.

In the information stored by the queue, it could be processed by three threads, so we use a variable value called Mutex A for preventing the atomic execution. The purpose of the Mutex B is to prevent the other threads from transmitting the same signal to the Main Thread in the queue traversal and protect the atomic job execution. Figure 5 shows the algorithms of the main thread and queue handler.

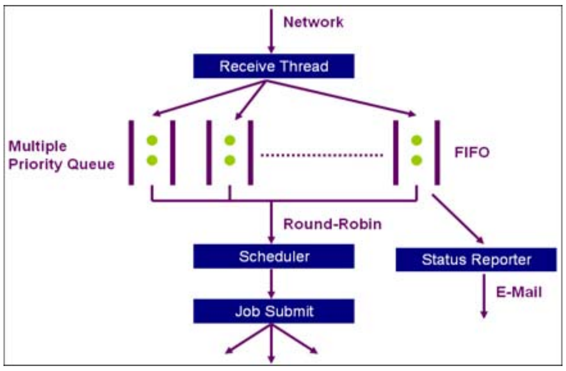


Fig. 3. The system architecture of scheduler of our system.

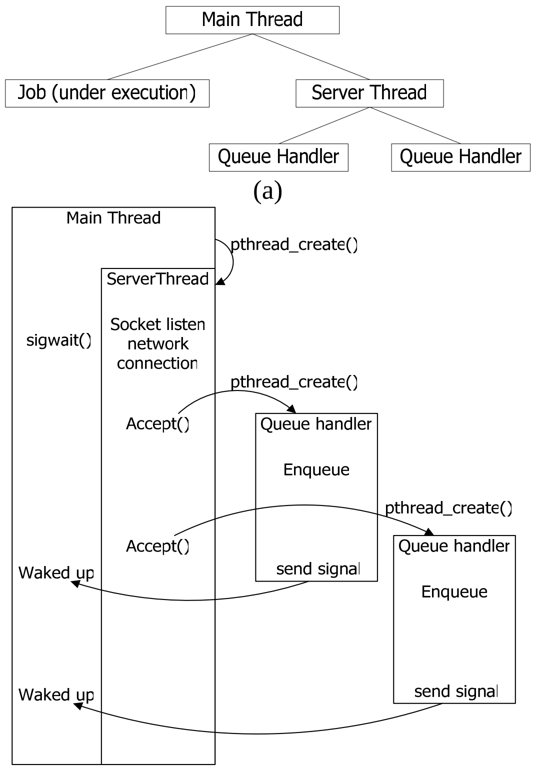
4. Experimental Results

We have developed our monitor system on three sets of four nodes Beowulf Cluster. The hardware and software specification is listed on Table 1. All daemons are implemented with C and our web portal was developed with PHP.

Figure 6 shows the snapshot of our system, from where we can obtain system information of each computing node of all cluster platforms.

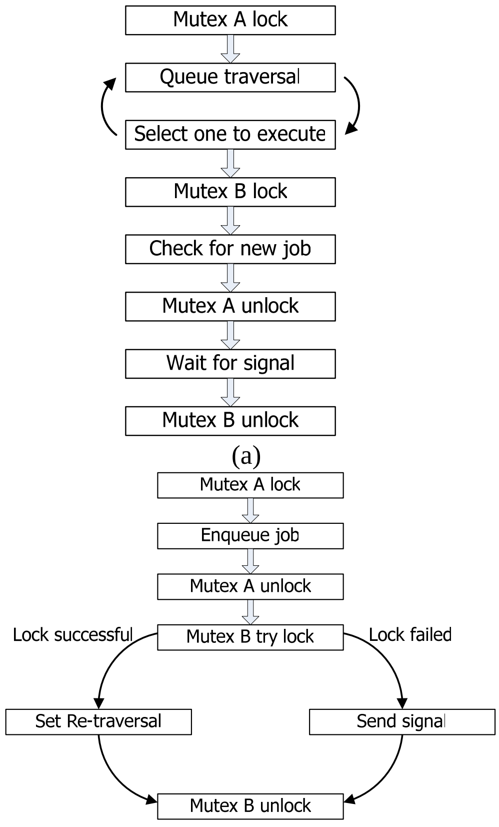
Figure 7 shows a task submission webpage, what essentially eases the user's task submission process, in any of our cluster platforms.

In Figure 8, it is shown the task status webpage, where all tasks submitted to be processed in our cluster systems are shown here, as well as the number of computing nodes used for the execution of each task.



(b)

Fig. 4. A paradigm of scheduling system. (a) The processing level of thread used in scheduler. (b) The processing flow of schedule



(b)

Fig. 5. Algorithms of the main thread and queue handler. (a) Algorithm of the main thread (b) Algorithm of queue handler

Figure 9 shows the same monitoring system we can obtain from a PC, but in a PDA screen. In this way, application developer can monitor computing nodes of cluster platforms, the physical characteristics of each computing node, the status of each task in the queue, that is, all operable using a PDA. Figure 10 shows our monitoring system that can be operated form JAVA client running on PC.

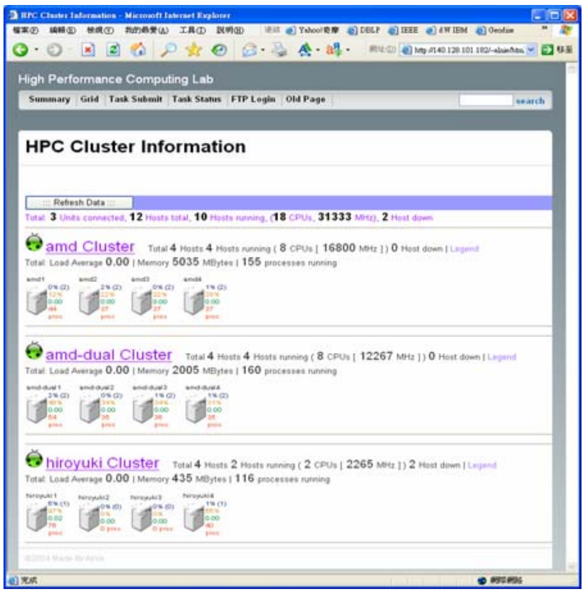
Table 1. Specifications of three Beowulf Clusters. (a) Hardware specification (b) Software configuration

(a)

	amd	**amd-dual**	**hiroyuki**
CPU	Dual AMD Athlon MP 2600+	Dual AMD Athlon MP 2000+	AMD AthlonXP 1600+
Main Memory	Master 2GB Slave 1GB	Master 512MB Slave 512MB	Master 192MB Slave 192MB
Hard Disk	Master 80G Slave 80G	Master 30GB Slave 30GB	Master 30GB Slave 30GB
Network	Gigabit Ethernet	Fast Ethernet	Fast Ethernet

(b)

	amd	**amd-dual**	**hiroyuki**
Distribution	RedHat 8	RedHat 9	Redhat 9
Kernel	2.4.18-14smp	2.4.20-8smp	2.4.20-8
gcc	gcc-3.2-7	3.2.2-5	3.2.2-5
glibc	2.3.2-4.80.8	2.3.2-27.9.7	2.3.2-27.9.7


HPC Cluster Information - Microsoft Internet Explorer
High Performance Computing Lab
Summary
Grid
Task Submit
Task Status
FTP Login
Old Page
search
Refresh Data

HPC Cluster Information

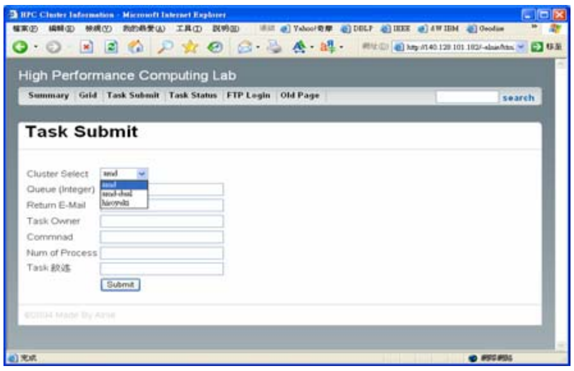
Total 3 Units connected, 12 Hosts total, 10 Hosts running, (18 CPUs, 31333 MHz), 2 Host down

amd Cluster Total 4 Hosts 4 Hosts running (8 CPUs [16800 MHz]) 0 Host down | Legend
Total Load Average 0.00 | Memory 5035 MBytes | 155 processes running

amd-dual Cluster Total 4 Hosts 4 Hosts running (8 CPUs [12267 MHz]) 0 Host down | Legend
Total Load Average 0.00 | Memory 2005 MBytes | 160 processes running

hiroyuki Cluster Total 4 Hosts 2 Hosts running (2 CPUs [2265 MHz]) 2 Host down | Legend
Total Load Average 0.00 | Memory 435 MBytes | 116 processes running

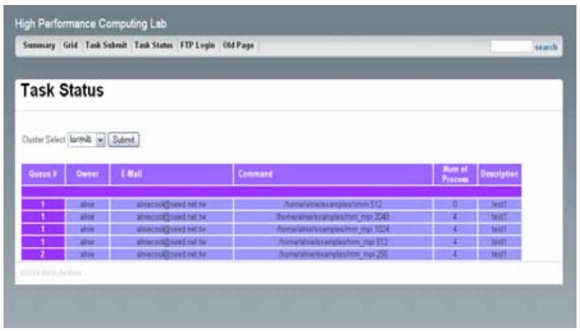
Fig. 6. Clusters view page


HPC Cluster Information - Microsoft Internet Explorer
High Performance Computing Lab
Summary
Grid
Task Submit
Task Status
FTP Login
Old Page
search
Submit

Task Submit

Cluster Select: amd (amd, amd-dual, hiroyuki)
Queue (Integer)
Return E-Mail
Task Owner
Command
Num of Process
Task 敘述

Fig. 7. Add new job interface

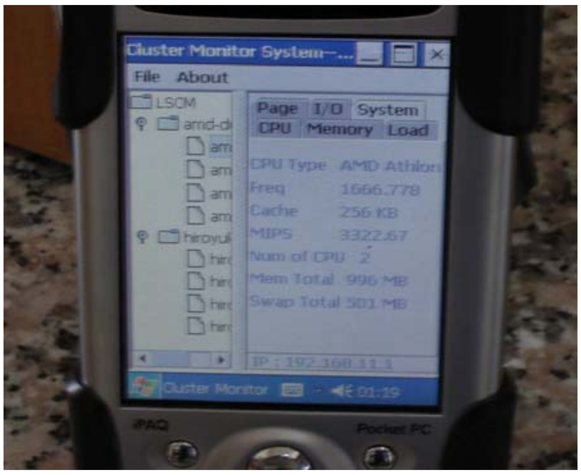


Task Status

Cluster Select: hiroyuki

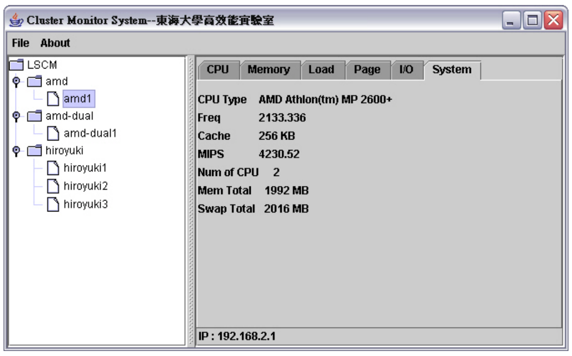
Queue #	Owner	E-Mail	Command	Num of Process	Description
1	alvie	alviecool@seed.net.tw	/home/alvie/examples/smm 512	0	test1
1	alvie	alviecool@seed.net.tw	/home/alvie/examples/mm_mpi 2048	4	test1
1	alvie	alviecool@seed.net.tw	/home/alvie/examples/mm_mpi 1024	4	test1
1	alvie	alviecool@seed.net.tw	/home/alvie/examples/mm_mpi 512	4	test1
2	alvie	alviecool@seed.net.tw	/home/alvie/examples/mm_mpi 256	4	test1

Fig. 8. The status of submitted tasks.



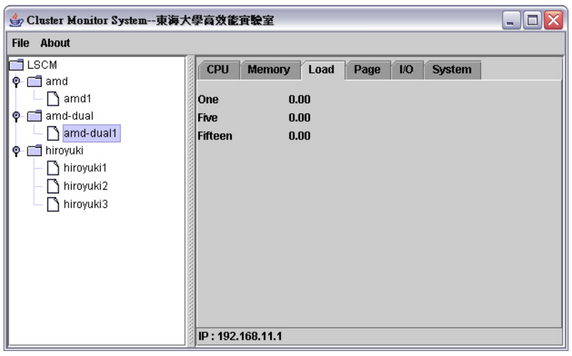
```
CPU Type    AMD Athlon
Freq        1666.778
Cache       256 KB
MIPS        3322.67
Num of CPU  2
Mem Total   996 MB
Swap Total  501 MB
IP : 192.168.11.1
```

Fig. 9. Monitoring screen on PDA


Cluster Monitor System--東海大學高效能實驗室
File About
LSCM
amd
amd1
amd-dual
amd-dual1
hiroyuki
hiroyuki1
hiroyuki2
hiroyuki3
CPU Memory Load Page I/O System

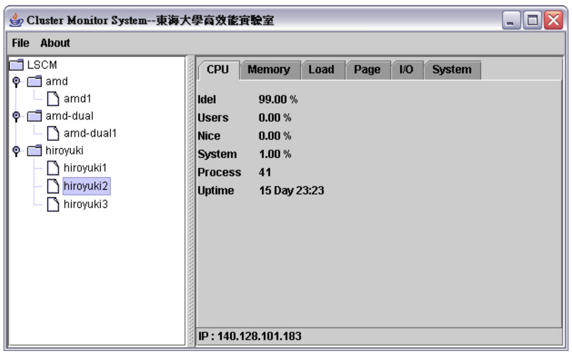
```
CPU Type    AMD Athlon(tm) MP 2600+
Freq        2133.336
Cache       256 KB
MIPS        4230.52
Num of CPU  2
Mem Total   1992 MB
Swap Total  2016 MB
IP : 192.168.2.1
```

(a)


Cluster Monitor System--東海大學高效能實驗室
File About
LSCM
amd
amd1
amd-dual
amd-dual1
hiroyuki
hiroyuki1
hiroyuki2
hiroyuki3
CPU Memory Load Page I/O System

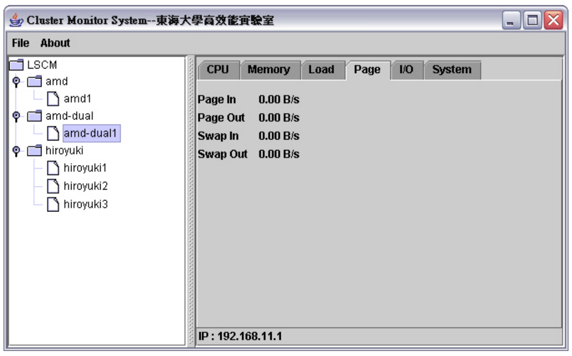
```
One       0.00
Five      0.00
Fifteen   0.00
IP : 192.168.11.1
```

(b)


Cluster Monitor System--東海大學高效能實驗室
File About
LSCM
amd
amd1
amd-dual
amd-dual1
hiroyuki
hiroyuki1
hiroyuki2
hiroyuki3
CPU Memory Load Page I/O System

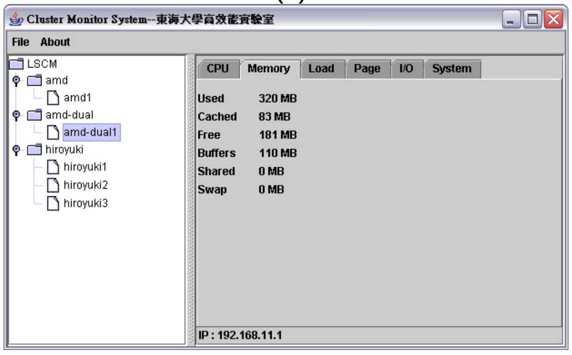
```
Idel      99.00 %
Users     0.00 %
Nice      0.00 %
System    1.00 %
Process   41
Uptime    15 Day 23:23
IP : 140.128.101.183
```

(c)


Cluster Monitor System--東海大學高效能實驗室
File About
LSCM
amd
amd1
amd-dual
amd-dual1
hiroyuki
hiroyuki1
hiroyuki2
hiroyuki3
CPU Memory Load Page I/O System

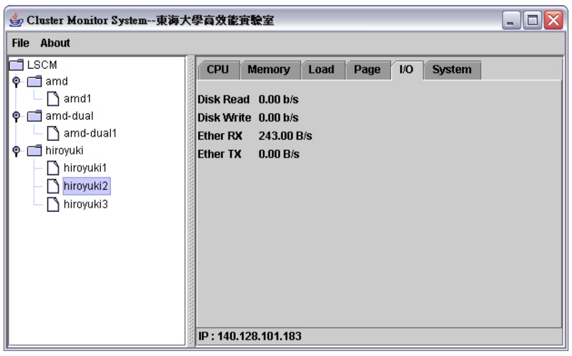
```
Page In    0.00 B/s
Page Out   0.00 B/s
Swap In    0.00 B/s
Swap Out   0.00 B/s
IP : 192.168.11.1
```

(d)


Cluster Monitor System--東海大學高效能實驗室
File About
LSCM
amd
amd1
amd-dual
amd-dual1
hiroyuki
hiroyuki1
hiroyuki2
hiroyuki3
CPU Memory Load Page I/O System

```
Used      320 MB
Cached    83 MB
Free      181 MB
Buffers   110 MB
Shared    0 MB
Swap      0 MB
IP : 192.168.11.1
```

(e)


Cluster Monitor System--東海大學高效能實驗室
File About
LSCM
amd
amd1
amd-dual
amd-dual1
hiroyuki
hiroyuki1
hiroyuki2
hiroyuki3
CPU Memory Load Page I/O System

```
Disk Read    0.00 b/s
Disk Write   0.00 b/s
Ether RX     243.00 B/s
Ether TX     0.00 B/s
IP : 140.128.101.183
```

(f)

Fig. 10. Monitoring screen from JAVA client (a) System information (b) Loading (c) CPU information (d) Page information (e) Memory information (f) I/O status

5. Conclusions

In this paper, a software system that allows the centralized administration of a generic Beowulf cluster is proposed. This system also provides web service and application to monitor for large scale clusters with task scheduling. We introduced an original algorithm to arbitrate threads between job executive and job listener, as also a novel signal programming technique is used to notify the main job executive, when the job is submitted and the scheduler modify to keep thread-safe for synchronicity. We also examined the special properties of threads in Linux operating system in our implementation.

References

[1] G. Pfister, *In Search of Clusters*, Prentice Hall PTR, ISBN: 0138997098; 2nd edition, January 1998.

[2] A. Geist, Cluster Computing: The Wave of the future, Springer Verlag, Lecture Notes in Computer Science, May 1994.

[3] The Beowulf Project, http://www.beowulf.org

[4] T. Anderson, D. Culler, and D. Patterson, "A Case for Network of Workstations," *IEEE Micro*, 15(1):54-64, Feb. 95. http://now.cs.berkeley.edu/

[5] SCMS, http://www.opensce.org/

[6] MPI, http://www.mpi-forum.org/

[7] PVM, http://www.csm.ornl.gov/pvm/pvm_home.html

[8] D. J. Becker, T. Sterling, D. Savarese, E. Dorband, U.A. Ranawake and C. V. Packer, "BEOWULF: A Parallel Workstation for Scientific Computation", *Proc. International Conference on Parallel Processing (ICPP)*, pp 11-14, 1995.

[9] M.R. Guarracino, G. Laccetti and U. Scafuri,"Beowulf Project at CPS-CNR", *Proc. of PC-NETS99*, L'Aquila (I), 1999. See also http://pixel.dma.unina.it/beowulf.html

[10] B. Saphir, P. Bozeman, R. Evard and P. Beckman, "Production Linux Clusters", *Supercomputing '99* Tutorial, Portland (OR), 1999.

[11] T. Sterling and D. Savarese, "A Coming of Age for Beowulf-Class Computing", *Proc. of Euro-Par '99, Lecture Notes in Computer Science*, no. 1685, pp 78-88, Springer, 1999.

[12] P. Uthayopas and A. Rungsawang, "SCMS: An Extensible Cluster Management Tool for Beowulf Clusters", Supercomputing 99, Portland (OR), 1999.

[13] University of California San Diego. *The Network Weather Service Homepage*. http://nws.npaci.edu/NWS.